

SpyGlass[®] DFT ADV

Rules Reference Guide

Version N-2017.12-SP2, June 2018



Copyright Notice and Proprietary Information

©2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>. All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Report an Error

The SpyGlass Technical Publications team welcomes your feedback and suggestions on this publication. Please provide specific feedback and, if possible, attach a snapshot. Send your feedback to spyglass_support@synopsys.com.

Contents

Preface	31
About This Book	31
Contents of This Book	32
Typographical Conventions	33
Understanding the SpyGlass DFT ADV Product	35
Key Concepts	36
RTL Design for Test	37
SpyGlass DFT ADV Design Constraints	47
Operating Modes	56
Types of Flip-Flops	63
Types of Latches	68
Identifying Clock Gating Cells	74
Identifying Synchronizer	86
Types of Faults	89
Support For Clock Shaper with Scannable Flip- Flops	95
Inserting RTL Testpoints	97
Impact of Different Path Types on Fanin/Fanout Cone Traversal	102
Design Impact	107
Scannability	107
Improvements to Fault and Test Coverage.....	108
Detecting Structures Leading to Non-Robust Tests.....	116
Identifying Test Points To Reduce Random Resistance	116
Using AutoFix and Selective AutoFix	119
Suggested SpyGlass DFT ADV Operation	122
Using the Design Constraints.....	123
Making the RTL scan ready	125
Complying with the SpyGlass DFT ADV Best Practices	128
Adding Test Points	129
Working with Scan Chains.....	130
Checking Block-level Test Requirements.....	133
Making the RTL Ready for Atspeed Test.....	133
Performing Conditional Connectivity Checks	143

Rules in SpyGlass DFT ADV	153
Asynchronous Rules	155
Overview	155
Async_01 : Do not use flip-flops with asynchronous set or reset unless disabled in shift mode.....	157
Async_02_capture : Ensure that the flip-flop set or reset fan-in cones do not contain -flip-flops, latches, or black boxes in capture mode .	161
Async_02_shift : Ensure that the flip-flop set or reset fan-in cones do not contain flip-flops, latches, or black boxes in shift mode	171
Async_03 : Ensure that the active phase of all set and reset pins connected to the same root level pin are at the same level	176
Async_04 : Ensure that the flip-flops are not used with both asynchronous set AND reset	180
Async_05 : Only one unblocked path allowed from a primary input to a flip-flop set or reset pin.	186
Async_06 : Ensure that the set and reset lines on the same flip-flop are not simultaneously active.	190
Async_07 : Ensure that the asynchronous set/reset sources are inactive during shift mode.....	195
Async_07Lssd : Ensure that the asynchronous set/reset sources are inactive during shift mode.	201
Async_08 : Ensure that the asynchronous set/reset pins of all the flip-flops are fully controllable during capture.....	205
Async_09 : Ensure that the set and reset pins of scan flip-flops are controllable when circuit is in capture mode.....	210
Async_10 : Ensure that the user designated pins control all sets and resets.	214
Async_11 : Avoid using set/reset signal as data signal in capture mode. ...	219
Async_12 : Reports flip-flops with data pins connected to the set/reset pins of the same flip-flops during capture mode.....	224
Async_13 : Ensure that the asynchronous set/reset pins of scan flip-flops are controllable to their inactive state during capture	227
Async_15 : Separate control must be maintained for asynchronous set/reset pins of flip-flops and latches	231
Async_16 : Ensure that the set or reset sources are disabled or controllable by PI	239
Async_17 : Report all the sources, which drive asynchronous preset, clear	

and clock pins.	246
Atspeed Test Rules.....	250
Overview.....	250
Atspeed_01 : Ensure that the source clock for all scannable flip-flops pass through a PLL.....	252
Atspeed_03 : Ensure that asynchronous logic in the functional mode does not interact synchronously in the capture at-speed mode	257
Atspeed_04 : Do not clock the synchronously interacting logic in the functional mode by asynchronous clocks in the capture at-speed mode	264
Atspeed_05 : Ensure that all false paths or multi-cycle paths in the functional mode are blocked in the capture at-speed mode..	269
Atspeed_06 : Ensure that all paths crossing asynchronous clock domains are blocked.	277
Atspeed_07 : Use a separate signal, such as scan_enable, for clock gating each domain in the capture at-speed mode	283
Atspeed_08 : Clock tree in the clock generation logic should not have different logic along paths which need to be balanced	287
Atspeed_09 : Ensure that the data pin of scan flip-flop is fully controllable	291
Atspeed_10 : Only valid clock sources allowed	295
Atspeed_11 : Ensure that all clock sources are controlled by an at-speed clock	300
Atspeed_12 : A clock-pin receiving no frequency or multiple frequencies is a violation.	305
Atspeed_13 : Expected frequencies must be achieved.....	307
Atspeed_14 : Test clocks must not be used as data signals	310
Atspeed_15 : Combinational loops are not allowed in capture mode....	315
Atspeed_16 : This rule has been deprecated.....	317
Atspeed_17_capture : clockshaper cell must be enabled in capture mode	318
Atspeed_17_captureatspeed : Clockshaper cell must be enabled in captureatspeed mode.....	320
Atspeed_17_shift : clockshaper cell must be enabled in scanshift mode ..	322
Atspeed_19 : Latches should be transparent in capture mode	324
Atspeed_20 : Asynchronous set/reset pins of all the flip-flops should be fully controllable during capture atspeed mode.	331
Atspeed_22 : No "ANDing" (Merging) of atspeed test clocks in capture	

atspeed mode.	333
Atspeed_23 : The clock pin of memories must pass through a pll.	335
Atspeed_24 : Clock should not be shaped more than once.	341
Atspeed_25 : Reports the presence of combinational re-convergence to flip-flops asynchronous pins	344
Atspeed_26 : Reports if an atspeed clock is applied on the sensitized fan- out cone of another atspeed clock	350
Atspeed_27 : Reports if there is a convergence at the async pins of a flip- flop	355
Atspeed_29 : Detects supply of bad clock during atspeed test.	359
Atspeed_30 : Reports the presence of combinational re-convergence to flip-flop clock pin	363
Atspeed_31 : Ensure that each clock shaper clock pin is directly controlled via PLL in the capture atspeed mode.....	368
Atspeed_32 : Reports the presence of cascaded reset re-convergence to asynchronous pins of flip-flop	373
Atspeed_33 : All latch inputs should be driven by controllable sources in the capture-atspeed mode.....	378
Atspeed_34 : Reports flip-flops having feedback from q-terminal to d- terminal.....	381
BIST Rules	384
Overview	384
BIST_01 : Restrict input cone width for BIST	385
BIST_02 : Limit the number of gate inputs.....	391
BIST_03 : Checks for unknown state after initialization sequence.....	395
BIST_04 : Ensure that fan-in cones of the scannable flip-flops do not have unknown values at observable points.....	398
BIST_05 : Ensure that in scan capture mode TIE-X cells outputs are bypassed	403
Clock Rules	406
Overview	406
Clock_01 : Ensure that only one clock is specified and that clock is driven from a top-level pin.....	408
Clock_02 : Ensure that both the edges of a clock are not switched on..	413
Clock_03 : Reports multiple clock domains.....	418
Clock_04 : Do not use clock signals as data signals.....	423
Clock_05 : Ensure that all opposite edge flip-flops are used as retiming flip-flops	431
Clock_06 : Ensure that only one clock port is available for each block..	436

Clock_08 : Ensure that merged testclocks are not present in the design ...	440
Clock_09 : Ensure that the clock and data pins do not have common logic	446
Clock_10 : Ensure that each clock domain has a dedicated test clock ..	449
Clock_11 : Ensure that all clock sources are testclock controlled in shift mode.	458
Clock_11_capture : Ensure that all clock sources are testclock controlled in capture mode	470
Clock_13 : This rule has been deprecated.....	480
Clock_14 : Ensure that the negedge registers are only driving primary outputs and no other flip-flop	481
Clock_16 : Ensure that the flip-flops capturing on the falling (rising) edge must not have any data paths from flip-flops capturing on the rising (falling) edge of the same rtz (rto) clock source	484
Clock_17 : Ensure that the capture clocks are not gated by flip-flops capturing on the same clock.....	488
Clock_18 : Ensure that the flip-flops are not triggered on the negative edge of their clocks.	493
Clock_21 : Ensure that clocks do not drive flip-flop set or reset pins....	498
Clock_22 : Ensure that each clock source should be combinationaly driven by one testclock only.....	502
Clock_23 : Ensure that the scan flip-flops must only trigger on the positive edge	506
Clock_24 : Ensure that the test clocks do not change shift mode signals. ...	509
Clock_25 : Ensure that the test clock must not drive flip-flops greater than that specified in -fflimit or -fflimit_percentage field(s) of the clock constraint.	514
Clock_26 : Reports if a test clock is applied on the sensitized fan-out cone of another test clock	518
Clock_27 : Detects edge inconsistency between CGC and driven flip-flops.	523
Clock_28 : Reports the presence of combinational re-convergence to flip-flop clock pin.....	529
Clock_29 : Ensure that all clock sources of memories/hard macros are testclock controlled in shift/capture mode	533
Clock_30 : All clock pins in a memory / hard macro should get the same root level clock	539

Clock Gating Rules.....	542
Overview	542
CG_01_atspeed : Ensure that the clock gating cell enables are controllable to on state in atspeed mode	544
CG_01_capture : Ensure that the clock gating cell enables are controllable to on state in capture mode	548
CG_01_shift : Ensure that the clock gating cells are enabled in shift mode	552
CG_02_atspeed : Ensure that the CGC enables are controllable to off state in atspeed mode	555
CG_02_capture : Ensure that the CGC enables are controllable to off state in capture mode	558
CG_03_atspeed : Ensure that the system enable pins on clock gating cell are observable in atspeed mode.....	561
CG_03_capture : Ensure that the system enable pins on clock gating cell are observable in capture mode	564
CG_04 : Test enable pins of clock gating cells should be driven by the specified gatingcell_enable ports	567
CG_05 : Test enable pins of clock gating cells in the same level should be driven by the same test enable ports	570
CG_06 : Flip-flops in the fan-in cone of a CGC may adversely affect coverage if they capture wrong value	572
CG_07 : Detects edge inconsistency between CGC and driven flip-flops	576
CG_08 : System enable pin of a CGC, which is driving the clock pin of a flip-flop, should not be driven by the same flip-flop.	582
CG_consistency : Behavior model of clock gating cell should be consistent with the constraint applied on it	586
CG_generateReport : Generate a text report with details of all clock gating cells in design.....	588
Connection Rules.....	592
Conn_01 : Ensure that the expected node value is achieved	593
Conn_02 : Ensure that the paths between user-specified nodes exist... ..	598
Conn_07 : Checks the structure between the user-specified nodes	612
Conn_08 : Checks the path between the user-specified nodes	619
Conn_09 : Path between user-specified nodes should not exist	637
Conn_10 : Reports nets with illegal node values	643
Conn_11 : Node must satisfy the specified constraint message tag expression	652
Conn_12 : Node must not have the specified constraint message tag	

expression	657
Conn_14 : Ensure that specified nets are having stable values under specified condition	662
Conn_15 : Check required pulse pattern at specified node.	671
Diagnostic Rules	676
Overview	676
Diagnose_02 : Ensure that the at-speed paths are not blocked by testmode signals	677
Diagnose_03 : Ensure that the faults are not blocked by false paths or multi-cycle paths	680
Diagnose_04 : Identifies the faults in paths crossing clock domains that cannot be tested.....	683
Information Rules	686
Overview	686
Coverage_audit : Analyze coverage for circuit.	690
CreateDebugSGDC : Create an SGDC file for all the user-specified instances.	695
Diagnose_testclock : Display instances that block the testclock propagation.	699
Diagnose_testmode : Display instances that block the testmode propagation.	705
Info_atpg_conflict : Reports testpoints to reduce ATPG conflicts.	708
Info_addFault : Reports all the ports and pins which are specified as add fault.....	710
Info_atSpeedClock : Displays at-speed test clock propagation	714
Info_atspeedClockSynchronization : Provides information on interacting atspeed clock domain pairs and non-interacting atspeed clock domain groups	718
Info_atSpeedCoverage : This rule has been deprecated.	723
Info_atSpeedDomain : Displays at-speed test clock domain propagation. 724	
Info_atSpeedFrequency : Displays at-speed frequency propagation ..	726
Info_atSpeedFrequency_EnableConflict : This rule has been deprecated. Use the dftDsmConstraintCheck_08 rule as a replacement for the Info_atSpeedFrequency_EnableConflict rule. 728	
Info_blackboxDriver : Reports black box drivers.....	729
Info_coverage : Estimate fault and test coverage	735
Info_dBist : Display simulation values of design in DBIST mode.	745

Info_define_tag : Show system state for a given tag.	748
Info_dft_deprecated : Displays info messages for all the rules, parameters and constraints which will be deprecated future but are enabled/used in current run.....	752
Info_DftDebugData : Reports dft debug data (schematic and excel sheet) 758	
Info_dftmax_configuration : Enables the planning and configuration of DFTMAX Ultra in an SoC.....	767
Info_enabledFlops : Reports the flip-flops in the design, sorted by their respective clock domains	781
Info_forcedScan : Displays all registers and flip-flops specified as 'force_scan'.....	784
Info_freqAssignTable : Generates a frequency assignment report	790
Info_inferredNoScan : Displays all flip-flops that have been inferred as no scan by SpyGlass DFT ADV.....	792
Info_IP_Report : Generates boundary report on the IPs specified using ip_block SGDC command.	798
Info_Latch : Reports status of latches in the design	802
Info_latchMapping : Reports the transparency and lockup status of all latches along with their respective test clock domain in the shift and capture mode	806
Info_logicalRedundant : Displays logically redundant faults	809
Info_memories : Reports on memory instances present in a design. ..	815
Info_memoryforce : Back annotate the memoryforce simulation results onto the structural view.	820
Info_memorywritedisable : Displays memorywritedisable propagation data	824
Info_noAtspeed : Displays all registers and flip-flops specified as 'no_atspeed'.	828
Info_noFault : Reports all the ports and pins which are specified as no fault.	830
Info_noScan : Display all registers and flip-flops specified as 'force_no_scan'	834
Info_noscanFlopsTextReport : The Info_noscanFlopsTextReport rule has been deprecated. Use the <u>Info_inferredNoScan</u> rule instead of this rule.....	841
Info_path : Display blocked path information	842
Info_potDetectable : Display faults that are potentially detectable. ...	846
Info_pwrGndSim : Display the power-ground simulation results.	849

Info_random_resistance : Estimates random pattern fault and test coverage for design.....	852
Info_schain : Displays all properly stitched scan chains	864
Info_scanwrap : Reports scanwrap related information	869
Info_self_gating_logic : Recognizes self-gating cells and identifies the test-points in self-gating logic.....	876
Info_selective_testpoint : Inserts selective testpoints	881
Info_soft_error_propagation : Generates the Soft Error related metrics 883	
Info_stilFile : Generates a STIL file	889
Info_stil_to_sgdc : Generates an SGDC file corresponding to a Stil file. .. 893	
Info_synthRedundant : Displays pins (for an RTL design) that are likely to be absent in an optimized netlist. Displays faults (for a netlist design) which fall in either TIED or BLOCKED category.....	901
Info_testclock : Display test clock propagation.....	908
Info_testCounts : This rule has been deprecated.	914
Info_testmode : Display testmode simulation results	915
Info_testmode_conflict_01 : Displays conflict in user specified testmode with respect to simulation result of fan-in cone.	920
Info_Top_SGDC_Report : This rule does bottom up hierarchical migration of block level constraints	925
Info_transitionCoverage : Evaluate upper bound of transition coverage for design	930
Info_transitionCoverage_audit : Analyze transition coverage for circuit. 938	
Info_uncontrollable : Show nets with imperfect controllability	943
Info_undetectedCause : Display undetectable fault information.....	948
Info_unobservable : Show all unobservable pins.....	952
Info_untestable : Display untestable faults caused by test mode	958
Info_unstable_testmode_registers : Report flip-flops, intended to be used as a configuration register, which are unable to retain their respective initialized 0/1 value during scan shift operation and create unstable test mode conditions.....	965
Info_unused : Display faults having no effect on system function.....	969
Info_x_sources : Reports all the x-sources related information for the design.....	973
Latch Rules	978

Overview	978
Latch_01 : Do not use latches unless transparent during power ground mode.	979
Latch_02 : Ensure that no combinational loops exist from transparent latches	983
Latch_04 : Ensure that there are no synchronous latches in the design	987
Latch_05 : This rule has been deprecated.	990
Latch_06 : Ensure that the cascaded latches have clock phase inversion ...	991
Latch_08 : Ensure that the latches are transparent	995
Latch_10 : Ensure that all latch enables must only be combinational driven through primary ports.	1003
Latch_15 : Ensure that all slave LSSD latches are fed directly by only one master LSSD Latch.	1007
Latch_16 : Ensure that all latches except retiming ones are transparent in Logic DBIST mode.	1010
Latch_18 : All latch inputs should meet scannability criterion in scan-shift and should be driven by controllable sources in the capture mode	1013
Latch_19 : Ensure that all non-transparent and non-shadow latches meet the scannability criterion	1016
PLL Rules	1020
Overview	1020
PLL_01 : Ensure that the reference clock of PLL is directly controlled by external signal.	1021
PLL_02 : Ensure that the reset pin of a PLL is directly controlled by external signal	1027
PLL_03 : PLL reference clocks should not directly reach to any flip-flop.	1034
RAM Rules	1036
Overview	1036
RAM_01 : Memory shadow logic must be less than a user specified %	1037
RAM_02 : Ensure that the memory outputs are registered to scannable flip-flops	1041
RAM_03 : Memory inputs should be scannable.	1045
RAM_04 : Memory outputs must be forced to known values in shift mode .	1049
RAM_05 : Disable RAM write enables.	1053
RAM_06 : Ensure that the Top level pins must drive memory read/write pins	

	1057
RAM_07 : Ensure that the memory outputs must not affect scannable flip-flops during capture.	1061
RAM_08 : Ensure that the write controls to register files/memories are ATPG controlled during capture mode	1065
RAM_09 : Ensure that the Memory inputs must feed scannable flip-flops during capture.....	1069
RAM_10 : Ensure that the memory or black box input and output terminals are scanwrapped or bypassed.....	1073
RAM_11 : Ensure that there are no combinational loops through memories.	1078
Scan Rules	1082
Overview.....	1082
Scan_06 : All scan chains should operate in parallel	1084
Scan_07 : Ensure that there are no internal test_mode signals.....	1087
Scan_08 : Ensure that all registers and flip-flops are scannable	1096
Scan_11 : Ensure that the scan ratio must exceed threshold	1101
Scan_16 : Ensure that all flip-flops, except retiming flip-flops, are scannable	1104
Scan_17 : Ensure that the sequential depth does not exceed limit between any two scan points.	1109
Scan_18 : Ensure that all top level inputs are registered.....	1114
Scan_19 : All top level outputs must be controlled by scan flip-flops..	1118
Scan_20 : Ensure that outputs of bypassed devices do not affect any scan flip-flop data & inputs are observable	1122
Scan_21 : Keeper, pullUp, and pullDown enables must be controlled by scan flip-flops.....	1128
Scan_22 : Ensure that the scan chains have lockup latches at domain crossing.	1134
Scan_23 : Ensure that the module bypass conditions are satisfied.....	1140
Scan_24 : Ensure that all flip-flops are part of some scan chain.	1143
Scan_25 : Ensure that the scan chains do not contain inverters in scan path.	1149
Scan_26 : Ensure that the scan chains contain lockup latch at chain end...	1152
Scan_27 : Validate the connectivity of scan data input pin of a scan flip-flop	1156
Scan_28 : Validate the desired value at scan enable pin of scan flip-flop...	1163

Scan_29	: Ensure that the scan chains do not have inversions in scan path	1166
Scan_30	: Reports observable flip-flop list.....	1169
Scan_31	: Reports flip-flops with constant value data pin.	1171
Scan_32	: Reports scanout ports that are not output ports during scan mode	1174
Scan_33	: Reports scanin ports that are not inputs during scan mode	1177
Scan_34	: Reports top test signals not named as per dedicated parameters	1180
Scan_35	: Ensure that the scan chains do not exceed maximum specified length	1183
Scan_36	: Ensure that the scan chains are balanced in length, that is, they should not deviate more than allowed from the expected length	1187
Scan_38	: Ensure that the scan chains start with a positive edge flip-flop..	1191
Scan_39	: Ensure that each scan flip-flop in a design is part of only one scan chain.....	1195
Scan_40	: Ensure that the scan chains have a 'copy flip-flop' at positive to negative clock edge crossing	1198
Scan_41	: Scanout port should be driven by pad terminal of pad cells	1202
Sanity Check Rules	1209
Overview	1209
dftParamCheck_02	: Sanity check to ensure that parameter rme_selection points to a valid directory.....	1210
dsmParamSanity_00	: Performs sanity checks on the defined parameter values	1212
dftDsmConstraintCheck_01	: Performs sanity checks on the 'atspeed_clock_frequency' and 'clock_shaper constraints...	1214
dftDsmConstraintCheck_02	: Performs SGDC sanity check on the clock constraint.	1216
dftDsmConstraintCheck_04	: Performs a sanity check as to whether the expect_frequency constraint is specified on a module that is a clock_shaper.	1218
dftDsmConstraintCheck_05	: Performs a sanity check as to whether the expect_frequency constraint is specified on a module that is a clock_shaper.	1219
dftDsmConstraintCheck_06	: gating_cell constraint should have equal width for -clkoutTerm, -enTerm and -obsTerm.....	1220

dftDsmConstraintCheck_07 : Do not apply the abstract_port constraint on non-black box cells.....	1222
dftDsmConstraintCheck_08 : Report conflict of simulation value with enabling value given in the constraint file.....	1226
dftDsmConstraintCheck_ComplexCell : Constraint complex_cell is obsolete and should be replaced by the clock_shaper constraint	1227
Scan Enable Rules	1228
Overview.....	1228
SE_Sanity_01 : User specified scan_enable_source must feed scan enable pin of a scan flip-flop.....	1229
SE_Sanity_02 : When -active_value is applied on user specified scan_enable_source, it must bring all scan flip-flop in mask-mode (scan chain mode)	1231
SE_Sanity_03 : In atspeed mode, user specified scan_enable_source should not get a value other than specified -active_value in atspeed mode	1234
SE_Sanity_04 : Avoid specifying scan_enable_source in fan-out of another scan_enable_source.....	1237
SE_Sanity_05 : scan_enable_source with -clock should be actually driven by logic clocked on specified clock.....	1240
SE_01 : Scan Enable cannot be driven by combinational logic	1243
SE_02 : Scan Enable cannot be driven by sequential logic	1246
SE_03 : Scan Enable cannot be used as data input to a flip-flop	1249
SE_04 : Scan Enable cannot feed primary output	1252
SE_05 : Scan enable source must feed flip-flops with the same clock as is specified in the -clock field	1255
SE_06 : All scannable flip-flops must have their scan enable pins driven by a designated scan_enable_source	1258
Scan Power Rules.....	1263
Overview.....	1263
SP_01 : No voltage domain crossing in a scan chain.....	1264
SP_02 : A PMU Test Control cell (PMUWR) must be inserted directly in the fan-in of the PMU (Power Management Unit).....	1266
SP_03 : Control signal of the ISO cells, Power Switches and Retention cells must be driven by a PMU Test Control cell (PMUWR).....	1268
SP_04 : A PMU Test Control Cell (PMUWR) should only drive control signal of ISO cells, Power Switches and Retention cells.	1270
SP_05 : A scan chain containing flip-flops in power management test control	

modules should be stand alone	1272
SoC Rules	1273
Overview	1273
Soc_00 : Generates an abstract view of a design	1274
Soc_04 : Show system state for a given tag.	1280
Soc_05 : Performs IP verification for mode, clock, controllability and observability in the SoC flow.....	1284
Soc_06 : Provides the hierarchical names of all instances appended with pin name.	1292
Testability Analysis Rules.....	1296
Overview	1296
TA_01 : Add test points to improve controllability.....	1297
TA_02 : Select test points to improve observability.....	1307
TA_05 : This rule has been deprecated.	1315
TA_06 : Generate testcoverage when all SpyGlass DFT ADV suggested test points are selected.	1316
TA_07 : Ensure that the testmode signals that only control asynchronous set or reset pins are unrestricted during capture.	1324
TA_08 : Reports test enable/enable pins of clock gating cells that are tied to an undesired value.....	1329
TA_09 : Reports cause of uncontrollability or unobservability and estimates the number of nets whose controllability/ observability is impacted	1333
TA_10 : Add test points to improve controllability and observability ...	1344
Test Compression Rules	1351
Overview	1351
TC_01 : Pipeline depth for head registers should equal head_register_depth 1352	
TC_02 : Head registers must be driven by unique primary inputs.....	1354
TC_03 : Head registers to the same decompressor must use the same clock and edge.	1356
TC_04 : Pipeline depth for tail registers should equal tail_register_depth ... 1358	
TC_05 : Tail registers from the same compressor must be driven by the same clock and edge	1360
Topology Rules.....	1362
Overview	1362
Topology_01 : Ensure that the design does not contain combinational loops in the capture mode.	1363

Topology_02 : No asynchronous pin to pin paths	1369
Topology_03 : Avoid sequentially derived asynchronous signals with common clock	1373
Topology_04 : Reports direct connection between two registers	1378
Topology_05 : Ensure that the design does not have a Wire-OR and Wire- AND	1382
Topology_07_flat : Do not use parallel drivers	1390
Topology_07_rtl : Do not use parallel drivers	1394
Topology_09 : Reports the existence of combinational re-convergent fan- out	1398
Topology_10 : Avoid long logic paths	1401
Topology_11 : Ensure that there are no combinational paths from a clock pin to a root level port in the capture mode	1406
Topology_12 : Ensure that all top level inputs are connected to a pullup or pulldown device	1409
Topology_13 : Reports the presence of combinational re-convergence to flip-flops asynchronous pins	1412
Topology_14 : Reports if there is a convergence at the async pins of a flip- flop	1418
Topology_15 : Ensure that the top-level output/inout ports, except scanout ports, are controllable	1422
Topology_16 : Topology_17 : Reports undriven terminals and nets.	1429
Tristate Rules	1431
Overview	1431
Tristate_01 : Reports inferred tristate components	1433
Tristate_03 : Tri-state enables should be disabled during scan shifting	1436
Tristate_04_capture : Ensure that inout ports are forced to outputs only in capture mode	1440
Tristate_04_shift : Ensure that the inout ports are forced to outputs only in shift mode	1445
Tristate_05 : Tristate buses should have a pull-up or pull-down connection	1450
Tristate_06 : Ensure that the tristate bus enables are fully decoded so that exactly one driver is active at any time	1454
Tristate_07_capture : Ensure that all Inout ports are inputs only in capture mode	1461
Tristate_07_shift : All Inout ports are inputs only in shift mode	1465
Tristate_08_capture : Reports inout ports not forced to only input or only	

output direction in the capture mode	1469
Tristate_08_shift : Reports inout ports not forced to only input or only output mode in the shift mode	1473
Tristate_09 : Ensure that the tristate buses are designed to prevent bus contention and floating during scan shift	1476
Tristate_10 : Reports tristate bus enables that are driven by flip-flops where capture clock causes x-condition.....	1482
Tristate_11 : Ensure that only one tri-state enable is active in shift mode.	1487
Tristate_12 : Bus enables should not be forced in scan shift.	1490
Tristate_13 : Generates bus width report.	1493
Tristate_14 : Ensure that the testclock are not connected to the enable pin of tristate buffer	1495
Tristate_15 : Ensure that the primary bidirectional pin is only driven by a tristate gate	1498
Tristate_16 : Associate a bus keeper to each tri-state	1501
Tristate_17 : Reports bidirectional ports that do not have fixed direction in DBIST mode.....	1504
Tristate_18 : Reports pins of bus keeper that are not fitted with a controllable pullup.....	1508
Integrity Checks	1512
Overview	1512
BlackBoxDetection : This rule has been deprecated.	1515
dftMandatory_Constraint_Check : Reports inadequacy of the user-specified mandatory constraints.....	1516
dftMultiplyDrivenPowerRail : Reports all the power rails that have external drivers as well.....	1518
dftOptional_Constraint_Check : Reports whether or not the optional constraint requirements of SpyGlass DFT ADV rules are met	1521
dftSetup : Performs some pre-DFT run initial setup.	1523
dftParamCheck_01 : Ensure that the dftSetLatchFedByTCIKAsT parameter is not used along with the dft_scannable_latches parameter	1525
dftSGDCExistence_00 : Ensures that constraints are specified for all top modules	1527
dftSGDCSTX_000 : Performs implementation level sanity checks on the SGDC constraints.....	1530
dftSGDCSTX_051 : Ensure that there is no duplicate declaration of a node	

for the same scalar tag.....	1532
dftSGDCSTX_053 : Reports incorrectly merged tags.....	1534
dftSGDCSTX_054 : Ensure that the merge tag specification do not span multiple lines.	1537
dftSGDCSTX_055 : Ensure that there are no conflicting values for the same node when used in a merged tag	1539
dftSGDCSTX_057 : Reports mismatches in node widths.....	1541
dftSGDCSTX_058 : Ensure that the testmode and testclock are separate ports.....	1543
dftSGDCSTX_059 : Ensure that the testmode and testclock are separate ports.....	1545
dftSGDCSTX_060 : Reports memory_type constraint defined for a module which is not a black box.....	1547
dftSGDCSTX_061 : Reports gating_cell constraint having different width for -clkoutTerm, -enTerm and -obsTerm	1549
dftSGDCSTX_062 : Checks for the existence of the 'module_pin' constraint in the design	1552
dftSGDCSTX_064 : Performs sanity checks for macros	1554
dftSGDCSTX_069 : Ensure that the user-specified design state and scan_chain constraint are consistent	1559
dftSGDCSTX_070 : Ensure that the user-specified set_case_analysis and test_mode -functional are consistent	1561
dftSGDCSTX_071 : User-specified 'clock_shaper' 'complex_cell' and 'pll' should apply to at least an instance, in case of 'clock_shaper - register, or to a module.....	1563
dftSGDCSTX_072 : Performs sanity check for the require_structure constraint	1566
dftSGDCSTX_073 : Performs sanity checks for module_bypass	1568
dftSGDCSTX_074 : Performs sanity checks for -constraint_message_tag field of the require_constraint_message_tag and illegal_constraint_message_tag constraints	1571
dftSGDCSTX_075 : Performs sanity check for the clock_shaper constraint	1574
dftSGDCSTX_076 : Performs sanity check for the missing required fields of the require_path, illegal_path, require_strict_path, require_value, illegal_value, require_frequency, require_constraint_message_tag and illegal_constraint_message_tag constraints	1576

dftSGDCSTX_077 : Performs sanity check for usages of instance based macros in require_path, illegal_path, require_strict_path, require_value, illegal_value, require_frequency, and require_pulse constraint	1578
dftSGDCSTX_078 : Sanity check for usage of macros in require_value and illegal_value constraints when -value_type field is used.....	1580
dftSGDCSTX_080 : Performs sanity checks on design-object-type/conditional-checks(filtering) based fields	1582
dftSGDCDefineMacroCheck_01 : The define_macro constraint sanity checks	1591
dftSGDCDefineMacroCheck_02 : The define_macro constraint sanity checks	1593
dftUserMacroSanityCheck_01 : Performs sanity check for usages of user macros in the require_path, illegal_path, require_strict_path, require_value, illegal_value, require_frequency, and require_pulse constraints	1595
dftVSTOPDUSetup : This is a set up rule.....	1597
Diagnose_ScanChain : Ensure that the scan chain must exist between the user-specified scanin and scanout points	1600
dumpBlackBox : Identify black boxes in a design and dump corresponding 'scan_wrap' constraints in a file	1606

Reports in the SpyGlass DFT ADV Product 1609

add_fault	1611
atpg_conflict_testpoints	1613
bist_ready_summary	1615
dft_connectivity_check_summary	1618
Scan Clock Tree Report	1620
dft_ff_set_reset_active	1623
dft_ff_set_reset_sequential_in_capture	1625
dft_ff_set_reset_sequential_in_shift	1625
dft_ff_X_source_for_tristate_enable	1626
dft_latch_enable	1629
dft_mandatory_sgdc	1631
dft_memory_report	1635
dft_multibit_flipflops	1636
dftmax_ultra_configuration	1637
dftmax_ultra_configuration_command	1639
dft_self_gating_logic_summary	1639

dft_self_gating_ff	1641
dft_self_gating_test_points.....	1643
no_fault	1646
dft_optional_sgdc	1647
dft_summary	1649
dft_tristate	1651
dft_initialized_ffs	1652
forced_scan	1654
inferred_no_scan_ffs.....	1655
dft_x_source_report	1656
dft_x_capture_report	1658
no_scan	1659
observable_ffs.....	1660
potentially_detected_faults	1660
scan_chain	1661
scan_wrap.....	1664
soc_06_rpt	1668
soft_error_metrics	1668
soft_error_registers_spfm.....	1669
stil_file.....	1670
stuck_at_faults	1671
stuck_at_coverage.....	1676
stuck_at_coverage_audit.....	1682
stuck_at_coverage_collapsed	1687
test_points_selected	1691
test_points_selected_2.....	1693
undetected_faults.....	1695
atspeed_03_rpt.....	1696
atspeed_04_rpt.....	1699
atspeed_07_rpt.....	1700
atspeed_08_rpt.....	1701
atspeed_clock_synchronization.....	1701
dft_dsm_clock_frequency	1706
dft_dsm_clock_gating_cell.....	1706
dft_dsm_constr-err-file	1710
dft_dsm_enabled_flipflops	1710
dft_dsm_ip_report.....	1715
ff_async_reconvergence	1719
ff_clock_reconvergence	1721
no_atspeed.....	1727
random_pattern_coverage	1728

random_pattern_dc_testpoints	1729
random_pattern_node_distribution.....	1732
random_pattern_testpoints	1735
transition_coverage.....	1737
transition_coverage_audit	1742
transition_coverage_clockdomain.....	1746
transition_coverage_collapsed	1750
transition_faults	1752

Customizing the SpyGlass DFT ADV Product 1761

Common SpyGlass DFT ADV Rule Parameters..... 1761

SpyGlass DFT ADV Rule Parameters 1765

ATPG_credit	1765
busWidthHigh.....	1765
busWidthLow	1766
captureClockCount	1766
captureClockEnd.....	1767
captureClockStart	1767
checkPLLSourceForAllNonFFCellClockPins	1767
debugInst.....	1769
dftmax_si_so_pin_number	1769
dftmax_is_occ_present.....	1770
dftmax_allow_asymmetric_pin_allocation.....	1770
dftmax_config_lookup_file	1771
dft_conn_check_allow_trace_through_async	1771
dft_conn_check_allow_trace_through_clock_shaper	1772
dft_allow_clock_merge_at_mux_via_data	1772
dft_allow_inactive_resets_from_scan_ffs	1774
dft_allow_sequential_propagation_during_clock_simulation	1775
dft_allow_path_from_enable_to_cgc_clkout	1776
dft_allow_single_flipflop_driver_in_Async_02_capture	1776
dft_all_scan_chains_defined.....	1777
dft_agnostic_cgc_flipflop_deadlock_check.....	1777
dft_atpg_conflict_tp_count.....	1778
dft_autofix_testmode_signal	1778
dft_autofix_testclock_signal	1779
dft_block_unstable_value_trace_on_no_clock.....	1779
dft_cgc_pre_scan_stitched_treatment_only_to_tied_test_enable	1780
dft_check_clock_merge_in_shift_as_well	1781
dft_check_latch_transparency_in_shift	1781

dft_check_path_name_for_instance_suffix	1782
dft_check_path_name_for_register_suffix	1782
dft_check_test_mode_conflicts	1783
dft_clock_end_point_types_for_Clock_02	1784
dft_collapse_equivalent_faults	1784
dft_conn_check_allow_non_x_value_on_sensitizable_path	1786
dft_conn_check_handle_rtl_negedge	1786
dft_conn_treat_pass_as_strict_pass	1787
dft_conn_treat_fail_as_strict_fail	1787
dft_conn_check_treat_endpoint_as_stoppoint	1788
dft_conn_check_rp_rsp_unified_flow	1788
dft_coverage_report_depth	1789
dft_detect_shadow_latches	1789
dft_enable_checks_for_all_cgc	1791
dft_ignore_constant_latches_in_rule_checking	1792
dft_ignore_no_scan_latches_in_rule_checking	1793
dft_ignore_inout_ports_as_driver	1793
dft_identify_equivalent_faults	1794
dft_identify_lockup_latch_only_through_buffers	1794
dft_infer_flop_enable	1795
dft_infer_sequential_propagation_as_no_scan	1796
dft_ignore_x_sources	1796
dft_ignore_x_sources_for_bist	1797
dft_ignore_state_holding_flipflop_driver_in_Async_02_capture	1799
dft_ignore_unate_reconvergence	1799
dft_insert_ta_tp	1800
dft_infer_tp_clock	1800
dft_insert_rrf_tp	1800
dft_internal_testmode_nodes_stability_check	1801
dft_list_latches_of_type	1801
dft_maximum_number_of_messages_with_spreadsheet	1802
dft_max_flops_in_diagnose_scan_chain_violation_schematic	1803
dft_min_scannability_ratio_for_test_points	1803
dft_report_all_paths_between_reconvergence_start_and_end	1804
dft_report_cgc_centric_deadlock	1805
dft_require_path_fail_limit	1805
dft_require_path_invalid_limit	1806
dft_require_path_pass_limit	1806
dft_rrf_display_limit	1806
dft_rrf_generate_fault_report	1807
dft_rrf_tp_count	1808

dft_rrf_tp_count_for_cutoff_incremental_gain	1808
dft_rrf_tp_cutoff_incremental_gain	1809
dft_rrf_tp_effort_level	1809
dft_rrf_tp_generate_dc_report	1809
dft_rrf_tp_ignore_generated_nets	1810
dft_rrf_tp_thread_count	1810
dft_rrf_tp_target_test_coverage	1811
dft_rrf_tp_type	1811
dft_rrf_tp_use_multiple_threads	1811
dft_scan_chain_report_redundant_lockup_latch	1812
dft_set_scan_wrap_on_memory	1813
dft_share_atpg_conflict_testpoint_at_branch	1813
dft_show_rule_name_in_audit	1813
dft_show_scan_clock_tree	1814
dft_show_schematic_info_in_coverage_audit	1816
dft_show_spreadsheet_path_in_message	1816
dft_show_unused_define_tag	1817
dft_soc_strict_boundary_validation	1817
dft_soc_unstable_value_sources	1819
dft_stable_value_pessimistic_check	1820
dft_state_holding_ff_identification_effort_level	1821
dft_store_tcl_query_data	1821
dft_stop_simulation_at_mux_with_unknown_select	1822
dft_support_flip_flop_bank	1826
dft_ta_tp_for_port	1826
dft_ta_tp_count	1827
dft_ta_tp_type	1827
dft_ta_tp_criteria	1829
dft_target_random_pattern_fault_coverage	1829
dft_target_random_pattern_test_coverage	1829
dft_target_stuck_at_fault_coverage	1830
dft_target_stuck_at_test_coverage	1830
dft_target_transition_fault_coverage	1831
dft_target_transition_test_coverage	1831
dft_treat_latches_with_X_on_enable_as_combinational_for_soc_path_checks	
1831	
dft_treat_primary_inputs_as_x_source	1832
dft_treat_primary_outputs_as_unobservable	1833
dft_treat_primary_port_as_valid_clock_point_for_cgc_check	1833
dft_use_n_cycle_capture	1834
dft_use_old_Topology_10	1835

dft_use_source_centric_Async_02_capture	1835
dft_use_stable_simulation_conditions.....	1836
dft_use_specified_and_inferred_clocks	1836
dft_use_new_Atsspeed_19.....	1837
dft_x_sources_display_limit	1837
dft3BitClock	1838
dft_use_cumulative_fault_status	1839
dftAllowMergeOfSameClock.....	1839
dftAllowNonXValueAtStartOfSensitizedPathInSoc_02.....	1840
dftAllowedScanChainLengthDeviation	1840
dftAutoFix	1841
dftDebugData	1841
dftCaptureCriteria.....	1843
dftClockEdgeFaultAnalysis.....	1844
dftClockEdgeType	1844
dftCPMechanism.....	1845
dftDesignState	1845
dftDoInferredNoScanAnalysis	1849
dftDoLogicalRedundancyCheck	1849
dftDsmConvergingPathsLimit.....	1850
dftEnablePinNameForCGC	1851
dftExpectedScanChainLength	1851
dftFindCombLoopThruSetResetToQ.....	1852
dftGenerateStuckAtFaultReport.....	1852
dftFunctionalClockPropagation	1853
dftGenerateTextReport	1853
dftIdentifyTestPoints	1854
dftIgnoreConstantOrUnusedFlipFlops.....	1854
dft_generate_no_fault_and_add_fault_report.....	1855
dft_generate_robustness_audit_report	1855
dft_max_files_in_a_directory	1856
dft_maximum_number_of_rows_with_schematic.....	1856
dft_min_scannability_ratio_for_test_points	1857
dft_pattern_count.....	1858
dft_ignore_bb_inout_term_for_topology_05.....	1858
dft_ignore_constant_supply_flip_flops.....	1859
dft_ignore_no_scan_driver_in_Async_02_capture.....	1860
dft_infer_clock_gating_cell	1860
dft_infer_clock_gating_cell_test_enable.....	1861
dft_scannable_latches.....	1861
dft_treat_suffix_as_pattern.....	1862

dft_use_simulation_based_unblocked_path_check_for_scan_chain_tracing	1863
dftInferredDriverControl	1863
dftMaxScanChainLength	1864
dftPOSDetect_credit	1864
dftReportIfUsedAsClock	1866
dftReportOnlyBBForBIST	1866
dftReportWDriversInTriBus	1866
dftSkipLogicForTestPoints	1867
dftSkipTestPointsOnImprovementLessThan	1867
dftTagAsync07Clock11Scan08	1868
dftScanDataConnectivityCheck	1868
dftScanEnablePinValue	1869
dftSENames	1869
dftSetAllBBScanwrapped	1870
dftSetAllLatchT	1870
dftSetAllTriEnableObs	1871
dftSetCGCEnablePinsValue	1871
dftSetEffortLevel	1872
dftSetLatchFedByTCIkAsT	1872
dftSetTMTClkNodesAsUnObs	1873
dftShowDataInversionInScanChain	1873
dftShowForcedValues	1873
dftShowSourceCentricViolation	1874
dftShowWaveForm	1875
dftSINames	1875
dftSkipNonXValueNetsForTA	1875
dftSkipPwrGndNetsForTA	1876
dftSkipTAMsgCount	1876
dftSkipUnObservableNetsForTA	1877
dftSONames	1877
dftSortTAMsgOnFaultCount	1877
dftStrictTestClkDomains	1878
dftTargetFaultCoverage	1878
dftTargetTestCoverage	1879
dftTestclockCycles	1879
dftTestEnablePinNameForCGC	1880
dftTreatControllableLatchTransparentForLoop	1880
dftTreatFlipFlopsAsScan	1881
dftTreatLatchesAsTransparent	1881
dftUseOffStateOfClockInClockPropagation	1882

dftTreatBBoxAsScanwrapped	1882
dftTMPropThruFF	1883
dftUseOffStateOfClockInClockPropagation	1883
dftUUMarking	1885
dsm_apply_migrated_constraints.....	1885
dsm_check_unblocked_mcp.....	1885
dsm_count_false_path_transition_faults	1886
dsm_gen_hiersgdc.....	1888
dsm_infer_clock_domain_at_clock_shaper_boundary	1888
dsm_launch_method.....	1889
dsm_max_fanin_paths_for_Atspeed_12	1889
dsm_memory_clock_check	1889
dsm_report_domain_for_faults.....	1891
dsm_use_cumulative_fault_status	1891
dsmAtspeedClockSynchronizationRange	1892
dsmGenerateTransitionFaultReport	1892
dsmIgnoreControlLineFaults	1893
dsmLimitMigrationReportFanin	1893
dsmReportAllCGCCEFaninFlops	1893
dsmReportEnabledFlops.....	1895
dsmUsePIForAtSpeed	1895
dsmUsePOForAtSpeed	1895
dftXPropForScanFF	1896
faultAnnotation	1896
flopInFaninCount	1898
gateInputCount	1898
head_pipeline_clock_phase.....	1898
head_pipeline_stages.....	1899
ignoreBlackBoxDuringSimulation	1900
ignoreBlackBoxDuringTestability	1900
incrementalTA	1901
limitFaninPorts	1901
mergeN.....	1902
monitorFlow.....	1902
noBlackBoxReporting	1903
pathDepth	1904
reconvergenceMinDepth	1904
ser_control_sequential_depth.....	1904
ser_ignore_safety_mechanism_register_lreg.....	1905
ser_load_enable_n_cycle_persistence_adjustment	1905
ser_observe_sequential_depth	1906

ser_observe_nsr_initial_value.....	1906
ser_blackbox_observe_probability.....	1907
ser_blackbox_control_0_probability.....	1907
ser_propagation_difference_threshold.....	1908
ser_register_report_top_contributors.....	1908
schematicForAllBits.....	1908
sequentialDepth.....	1909
showAllReconvergence.....	1909
showPath.....	1910
showPowerGroundValue.....	1910
tail_pipeline_clock_phase.....	1911
tail_pipeline_stages.....	1911
useFirstSource.....	1911

Debugging Violations in SpyGlass DFT ADV Product..... 1913

Diagnosing a Problem.....	1914
Understanding Back Annotation.....	1915
Viewing Results in Fault Browser.....	1924
Viewing Undetectable Faults.....	1926
Organizing Report Columns.....	1928
Configuring Legend.....	1929
Searching an Instance.....	1931
Setting Search Preferences.....	1932
Scanning a Clock Tree.....	1933
Clock Browser window.....	1933
Scan Clock Tree Report.....	1936
Performing an On-Demand Trace.....	1937
Trace to Clock in Shift.....	1937
Trace to Clock in Capture.....	1939
Trace to Clock in Atspeed.....	1939
Trace to Uncontrollable in Atspeed (Before Launch).....	1939
Trace to Uncontrollable in Atspeed (After Launch).....	1941
Trace to Uncontrollable Sources in Capture.....	1941
Trace to Unobservable Sources in Capture.....	1941
Viewing Flat Data Hierarchically.....	1943
Understanding the structure of the hierarchy tree.....	1944
Configuring instance count threshold.....	1945
Toggling the instance view.....	1946

SpyGlass IEEE1500 Wrapper Debug Flow..... 1951

Introduction.....1951

Related TCL Commands.....	1951
dsm_assign_ieee1500_pins : Identifies IEEE1500 wrapper pins.....	1952
dsm_reset_ieee1500 : Initialize and reset the wrapper.....	1954
dsm_load_ieee1500_instruction : Load the instruction in the instruction register.	1956
dsm_load_ieee1500_data : Loads the data in the data register.....	1958
dsm_show_ieee1500_sequence : Shows the sequences generated on the pins.	1960
dsm_assert_value : Defines a check that requires a logic value to be established.	1962
dsm_assert_path : Defines the connectivity check for a path.	1966
dsm_assign_clock_pulse : Enables or disables clock pulse for specified clock pin.....	1970
dsm_assign_pin_value : Specifies a value for a pin.....	1972
dsm_reset_assertions : Resets all the assertion commands.	1974
dsm_check_assertions : Runs SpyGlass to check all assertions specified in the flow.	1975
dsm_capture_ieee1500_data : Captures the data from the wrapper into the output data register.....	1976
dsm_capture_ieee1500_instruction : Captures the instruction from the wrapper into the output instruction register.	1978
dsm_read_ieee1500_data : Reads the data from output data register. ..	1980
dsm_read_ieee1500_instruction : Reads the data from output instruction register.....	1983
Example Flow.....	1986

JTAG Based SoC DFT Debug 1989

Introduction.....1989

Related TCL commands	1990
dsm_jtag_get_current_state : Get current state of TAP controller..	1991
dsm_set_current_jtag_state : Move TAP controller to specified state	1992
dsm_get_jtag_pins : Gets the JTAG/TAP pins	1993
dsm_set_jtag_pins : Sets the JTAG/TAP pins.....	1994
dsm_reset_jtag : Resets the TAP	1996

dsm_load_jtag_instruction : Loads the specified bit sequence as a TAP instruction	1997
dsm_load_jtag_data : Loads the specified bit sequence into a TAP data register	1999
dsm_goto_jtag_state : Takes the TAP controller to the specified state	2000
dsm_hold_jtag_state : Holds the current JTAG/TAP state for the specified number of cycles	2001
dsm_show_jtag_sequence : Print jtag sequence	2002
Example Flow	2003

Appendix:

CPF and UPF Commands	2009
CPF Commands	2010
create_power_domain :	2011
UPF Commands	2014
create_power_domain :	2016
create_power_switch :	2018
set_isolation :	2020
set_isolation_control :	2023
set_retention :	2025
set_retention_control :	2027

Appendix:

SGDC Constraints	2029
SGDC Concepts	2029
SpyGlass Design Constraints	2030

Appendix:

RME Parameters	2033
Overview	2033
RME Parameters	2034

Glossary	2035
-----------------------	-------------

Preface

About This Book

The SpyGlass® DFT Rules Reference describes the SpyGlass rules that check the designs for Design-For-Testability.

Contents of This Book

The SpyGlass® DFT ADV Rules Reference consists of the following sections:

Section	Description
<i>Understanding the SpyGlass DFT ADV Product</i>	Introduces the key concepts of the SpyGlass DFT ADV Product.
<i>Rules in SpyGlass DFT ADV</i>	Provides detailed description of the rules in the SpyGlass DFT ADV product.
<i>Customizing the SpyGlass DFT ADV Product</i>	Provides detailed description of the parameters in the SpyGlass DFT ADV product.
<i>Reports in the SpyGlass DFT ADV Product</i>	Provides detailed description of the reports in the SpyGlass DFT ADV product.
<i>Debugging Violations in SpyGlass DFT ADV Product</i>	Provides detailed description on debugging the violations in the SpyGlass DFT ADV Product.
<i>SpyGlass IEEE1500 Wrapper Debug Flow</i>	Describes the Related TCL Commands used to set and run the IEEE1500 Wrapper debug flow
<i>JTAG Based SoC DFT Debug</i>	Describes the Related TCL commands used to set and run the JTAG/TAP based SoC DFT debug
<i>Appendix: CPF and UPF Commands</i>	Describes the CPF Commands and UPF Commands supported by the SpyGlass DFT ADV solution
<i>Appendix: SGDC Constraints</i>	Lists the SGDC commands used by SpyGlass DFT ADV product
<i>Appendix: RME Parameters</i>	Lists the RME parameters used by the SpyGlass DFT ADV product

Typographical Conventions

This document uses the following typographical conventions:

To indicate	Convention Used
Program code	OUT <= IN;
Object names	OUT
Variables representing objects names	<sig-name>
Message	Active low signal name '<sig-name>' must end with _X.
Message location	OUT <= IN;
Reworked example with message removed	OUT_X <= IN;
Important Information	NOTE: This rule...

The following table describes the syntax used in this document:

Syntax	Description
[] (Square brackets)	An optional entry
{ } (Curly braces)	An entry that can be specified once or multiple times
(Vertical bar)	A list of choices out of which you can choose one
. . . (Horizontal ellipsis)	Other options that you can specify

Understanding the SpyGlass DFT ADV Product

The SpyGlass[®] DFT product contains a variety of Design-For-Test-related rules. The performance of many SpyGlass DFT ADV rules depends on whether various user-supplied information is available.

NOTE: *The SpyGlass DFT ADV product requires standard SpyGlass be installed and licensed. A separate license feature line is also required for the SpyGlass DFT ADV product.*

This section covers test-related issues for using the SpyGlass DFT ADV product and explains the following topics:

- *Key Concepts*
- *Operating Modes*
- *Identifying Test Points To Reduce Random Resistance*
- *Support for Multi-Bit Flip-Flop Cells*
- *Support For Clock Shaper with Scannable Flip- Flops*
- *Design Impact*
- *Suggested SpyGlass DFT ADV Operation*

Key Concepts

Rules for SpyGlass DFT ADV are classified into two broad categories: **topology-related** rules and **functionally-dependent** rules.

The **topology-related** rules are similar to a number of rules in various SpyGlass products in that they depend only on part type, part pin, and interconnection. Rules in this category include such rules as combinational feedback detection and port-to-port path connections. The essential characteristic of these rules is that the netlist contains all the information necessary for them to operate.

The **functionally-dependent** rules involve not only the topology information but also take logical function into account. Rules in this category include such rules as latch transparency, clock by-passing, and scannability. These rules require information that is not normally available in the netlist. This extra information must be supplied by the user in the form of a separate SpyGlass Design Constraints file.

This section explains the following DFT concepts:

- *RTL Design for Test*
- *SpyGlass DFT ADV Design Constraints*
- *Operating Modes*
- *Types of Flip-Flops*
- *Types of Latches/Identifying Clock Gating Cells*
- *Identifying Clock Gating Cells*
- *Types of Faults*
- *Support for Multi-Bit Flip-Flop Cells*
- *Support For Clock Shaper with Scannable Flip- Flops*
- *Using AutoFix and Selective AutoFix*
- *Identifying Test Points To Reduce Random Resistance*
- *Impact of Different Path Types on Fanin/Fanout Cone Traversal*

RTL Design for Test

Traditionally, for full scan based circuits, the efficacy of the testability analysis of a design at RTL was performed using the following two approaches:

- Adopting DFT aware RTL coding guidelines or best practices and
- Relying on feedback from downstream ATPG tool analysis of the scanned netlist with help from synthesis and DFT teams.

However, both approaches tend to have misses and late feedback. Therefore, it has become inevitable today for EDA vendors to provide automatic testability analysis tools at RTL.

In the normal process, after synthesis is complete, gate-level tools are typically used for replacing scannable flip-flops with equivalent scan flip-flops and stitching the scan flip-flops into scan chains. Designing the RTL with some appropriate considerations for these downstream steps can significantly improve the success of these operations as well as their impact on performance. In particular, this means designing the RTL with *test mode* and specifying *test clocks*.

While SpyGlass normally requires no additional information (beyond the RTL source) to check rules, this is not the case for SpyGlass DFT ADV. SpyGlass DFT ADV includes a built-in cycle simulator and testability analyzer engine, and these require a minimal amount of additional information beyond the RTL source in order to understand how the design is meant to behave in test mode. For this, you need to specify the `test_mode` and clock constraints using an `sgdc` file.

For the purpose of this document, **test_mode** refers to the `test_mode` constraint. However, **test mode** refers to a state of the circuit.

Test mode

Circuits designed for test typically operate in both a system or intended mission mode and in a test mode for scan and capture. The logic to switch out of system mode can be as simple as setting a single value on a single top-level pin. When such a pin is held low (high), the circuit performs its intended function. When that pin is high (low), the circuit is configured for ATPG tools and for manufacturing testing.

The test mode may be more complex so that a particular sequence of

values on a set of signals may be necessary to enter the test mode.

Two important considerations for scan design are to provide a test mode so that when such a mode is active, the clock to each register, intended for scan, is controlled by means of an external test clock, and to ensure that any asynchronous set and reset pins are inactive. This means that any internally generated clocks must be controlled by test clocks and any asynchronous set or reset signals must be forced inactive in test mode.

You can use one of the following methods to identify the resets in the design:

- Use the `Reset_info01` rule from the SpyGlass CDC product. This rule creates the `autoresets.sgdc` file that lists the resets in the design.
- Alternatively, use the `Design Resets` step of `dft_setup` goal.

This section describes various cases of `test_mode` propagation:

- *Explicit Propagation*
- *Implicit Propagation*
- *No Propagation*

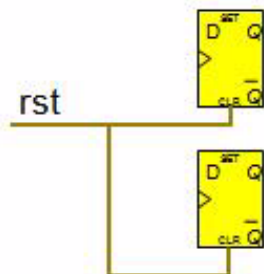
Explicit Propagation

Consider the following sample line in the `autoresets.sgdc` file:

```
reset -name rst -value 1
```

It indicates that the signal `rst` is a high-active reset as shown in the following figure:

Consider the following figure:



Convert the above reset, `rst`, to the following *test_mode* constraint:

```
test_mode -name rst -value 0 -scanshift
```

This means that you need to hold the `rst` signal to 0 during scanshift mode. This satisfies one of the key test setup requirement.

The test mode is also used to make latches transparent and to break combinational feedback loops.

SpyGlass DFT ADV product considers the following two cases of the test mode:

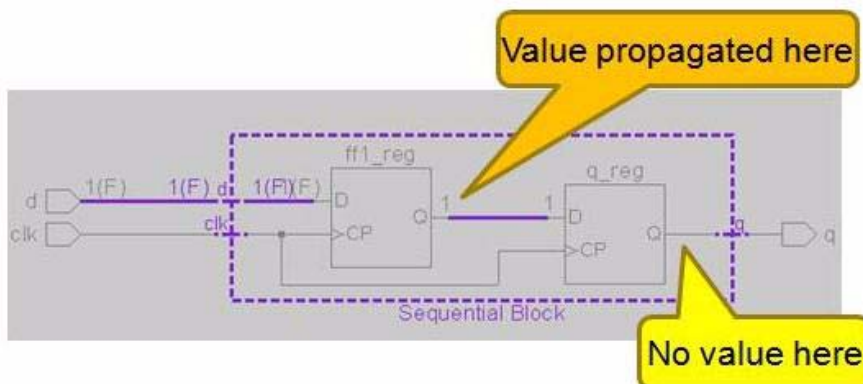
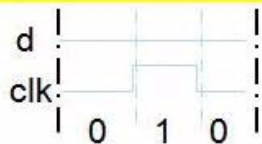
- The *scanshift mode* when the design is in the scan shifting mode.
- The *capture mode* when the scan-shifted values are captured (that is, read).

Most rules in the SpyGlass DFT ADV product work in the scanshift mode and some rules like the `RAM_08` rule work in both the scanshift mode and the capture mode.

NOTE: *When 'testmode' is used in this document, it indicates that the description is applicable for both shift mode and capture mode.*

Following figure shows an example of a sequential test mode setup on a design with two flip-flops in a shift register configuration when one pulse is applied:

```
// test.sgdc
test_mode -name d -value 1
test_mode -name clk -value 010x
```

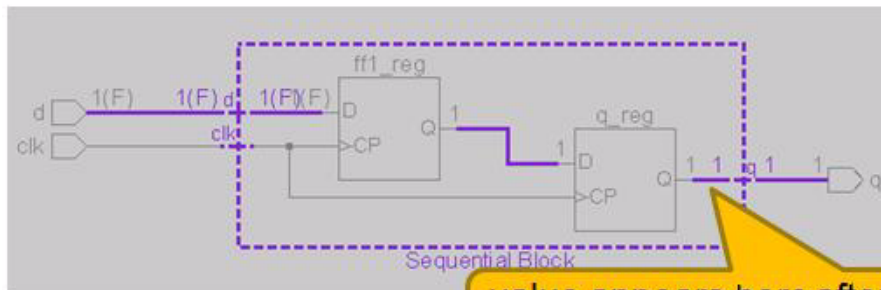


The set of test_mode constraints shown above illustrate the mechanism of simulating one pulse on the clock line. In the above example, the input data propagates to the output of the first flip-flops but does not propagate to the output of the second flop. The waveform diagram shows how the constant values are simulated during each simulation cycle.

The following figure shows the same circuit when two pulses are applied:

Key Concepts

```
// test.sgdc
test_mode -name d -value 1
test_mode -name clk -value 010010x
```



value appears here after two clock pulses

The set of *test_mode* constraints shown above illustrate the mechanism of simulating two pulses on the clock line. This causes the input data to propagate to the output of the second flop.

The *-scanshift* *test_mode* argument is an important option of the *test_mode* constraint. It is used to constrain a signal to the specified constant value only during the scan chain shifting. For more information, refer to the Example 5 of the *test_mode* constraint.

Implicit Propagation

Consider the following example where a flip-flop output pin gets a non-X value because of presence of active reset when clock is in its off state:

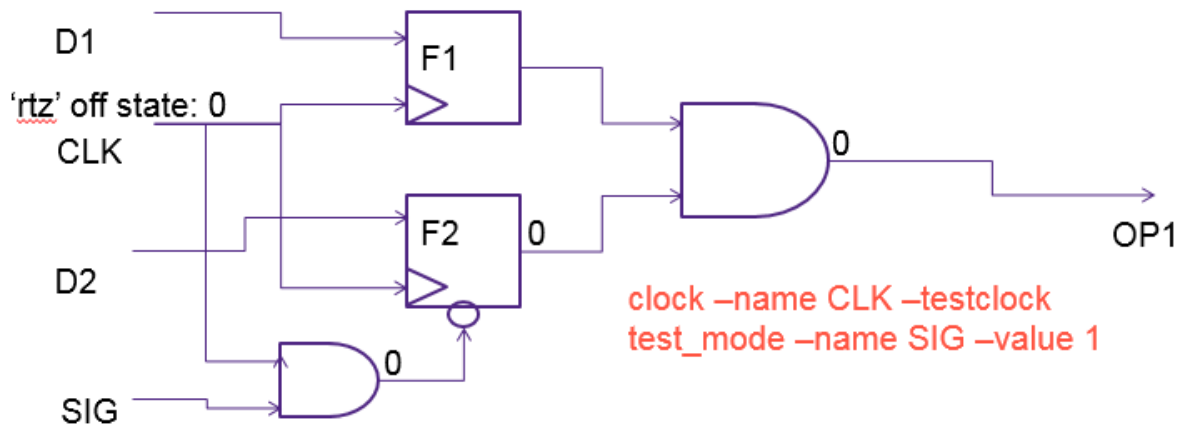


FIGURE 1. Clock Propagation: Active Reset When Clock is in Off State

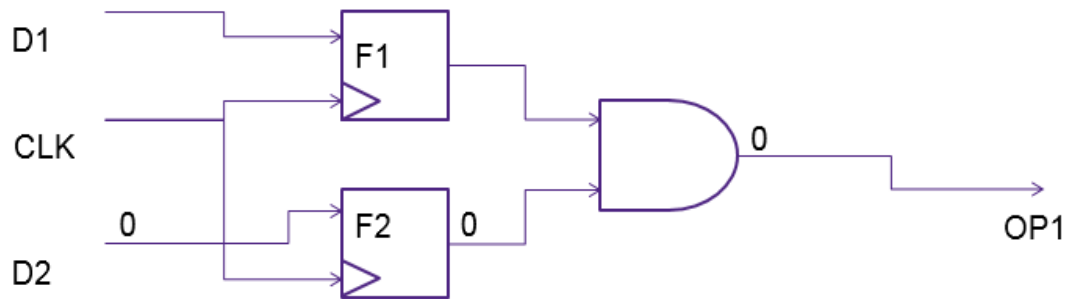
In the above example, the output of flip-flop, F2, is at 0 during both scan shift and capture operation because at the end of shift cycle, clock goes to its off state, that is, 0. Then, F2-Reset becomes active and the output of F2, Q, goes to 0.

First clock edge which comes after scan shift operation is the capture pulse so it should capture the effect of F2-Q-pin being at 0.

Therefore, F1-Q pin is blocked in capture via AND-Gate.

Also, consider the following case where a flip-flop output gets a value where a data-pin has a fixed value and the value of the [dft_infer_sequential_propagation_as_no_scan](#) parameter and the [dftTMPPropThruFF](#) parameter is set to on:

Key Concepts



```

clock -name CLK -testclock
test_mode -name D2 -value 0 -scanshift
set_parameter dft_infer_sequential_propagation_as_no_scan on
set_parameter dftTMPPropThruFF on

```

FIGURE 2. Clock Propagation: Data Pin Has a Fixed Value

In the above example, no explicit clock pulse is specified.

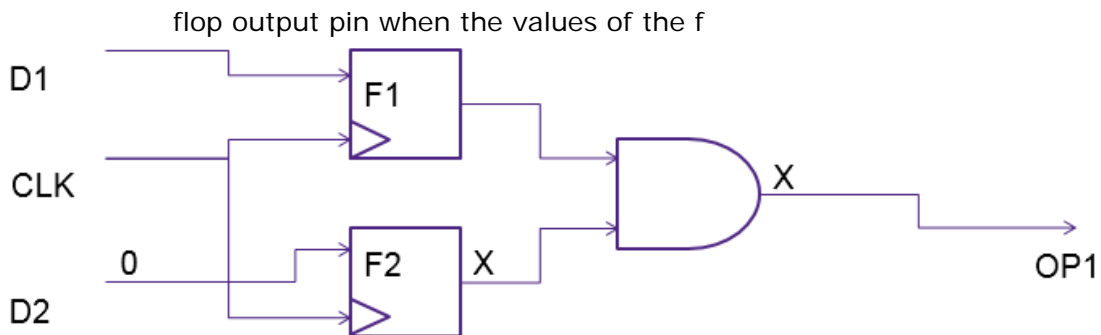
Also, F2 is inferred a no_scan flip-flop and its output is at 0 during both scan shift and capture operation.

It retains the value '0' even when its D-pin is left free (for capture operation) because the first clock edge, which comes after scan shift operation, is the capture pulse so it should capture the effect of F2-Q-pin being at 0.

So, F1-Q-pin is blocked in capture via AND-Gate.

No Propagation

Consider the following example where no values are propagated to the flip-



```

clock -name CLK -testclock
test_mode -name D2 -value 0 -scanshift
set_parameter dft_infer_sequential_propagation_as_no_scan off
set_parameter dftTMPPropThruFF off

```

FIGURE 3. No Propagation

In the above example, F2 is inferred as scannable. Therefore, its output is at X during both scan shift and capture modes. Also, F1-Q-pin is not blocked in capture via AND-Gate.

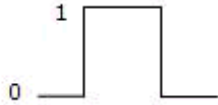
Test clocks

Test clock pins may be dedicated top-level pins or they may be shared with top-level system pins. SpyGlass DFT ADV accommodates both.

SpyGlass DFT ADV is designed to provide assistance with this design-for-test effort but to get the maximum benefit requires that the means to enter test mode be defined and the test clocks be declared.

The following are the types of clocks used by the SpyGlass DFT ADV product:

- Return To Zero (RTZ):** This implies that the rising edge of the root level clock is used for capturing the resultant value of the shifted vector. Zero is the off state of the root clock. In general, `rtz` is used during DFT analysis. Thus, the default value of clock in SpyGlass DFT ADV product is also `rtz`. The following figure illustrates the RTZ clock.



- **Return To One (RTO):** This implies that the falling edge of the root level clock is used for capturing the resultant value of the shifted vector. One is the off state of the root clock.



The performance of all the rules with `test_mode` or `test clock` declared in the user input section of the rule will depend on having either a `test_mode` or a `test clock` or both defined.

Identifying Testclocks

Testclocks in the SpyGlass DFT ADV analysis are classified as:

- **User-specified:** Can be specified using the `clock -testclock` constraint.
- **Inferred:** Output of clock gating cell (CGC), that meets the pipeline depth, are used as inferred test clocks for the dft analysis.

If you want to ignore user-specified testclocks and consider only inferred test clocks, specify the following constraint in the `*.sgdc` file:

```
gating_cell_enable -type phased_clock_enable
```

To use both user-specified and inferred test clocks, use the [dft_use_specified_and_inferred_clocks](#) parameter.

The `dft_inferred_test_clocks.rpt` report, generated by the [Info_testclock](#) rule lists the information on inferred clocks. Also, the violation message reported by the rule also reports information on the clock inference.

For example, consider the following violation message reported by the

Info_testclock rule:

Testclock 'top2.tclk1 (inferred with cgc-en: top2.cgc_en1 and pipeline_depth: 3)' propagation in shift mode. It hits 1 flip-flop(s)

The following figure shows the corresponding Incremental Schematic along with the debug data highlighting the pipeline_depth preceding the test clock inferred point:

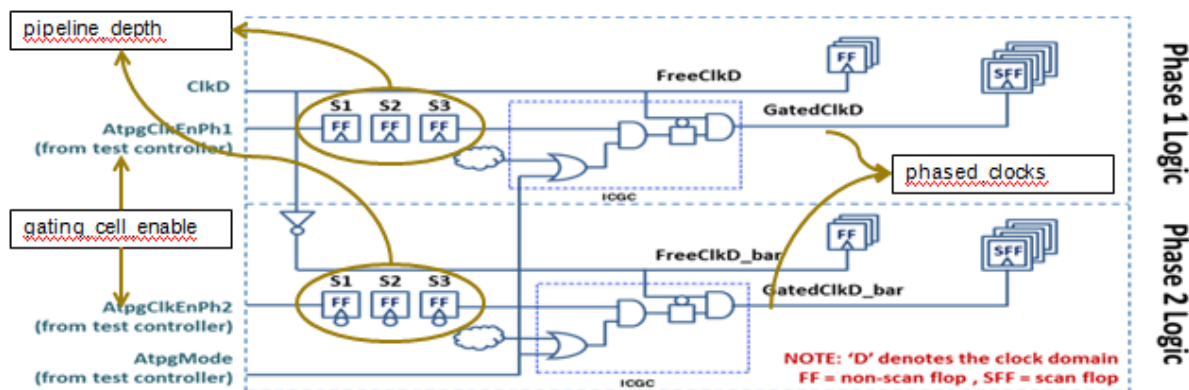


FIGURE 4. Inferred Testclock

SpyGlass DFT ADV Design Constraints

You can specify additional information to the SpyGlass DFT ADV product using a SpyGlass Design Constraints (SGDC) file.

Refer to SpyGlass Consolidated Constraints AppNote for more information on the SGDC constraints defined for the SpyGlass DFT ADV product.

This section explains the following topics:

- [Working with SGDC Files](#)
- [SpyGlass DFT ADV Constraints Usage Features](#)
- [SpyGlass DFT ADV Constraints File Examples](#)

For more details on working with the SGDC files, refer to the *SpyGlass Explorer User Guide*.

Working with SGDC Files

An SGDC file can be a text file of any extension. However, it is recommended to save such files with the .sgdc extension.

This section explains writing the SGDC file for [Verilog](#) and [VHDL](#).

The [Running SpyGlass with constraint files](#) section explains how to run SpyGlass using the constraints file in the batch mode and in the GUI.

Verilog

Assume the top-level module for a Verilog design is:

```
module myTOP (...)  
  ...  
endmodule
```

The SpyGlass DFT ADV constraints are specified in a separate file called `<any name>.sgdc` first specifies the design file that this constraint file corresponds to followed by the constraint information.

VHDL

Constraint files for VHDL are the same as constraint file for Verilog except that the `current_design` line becomes:

```
current_design <entity-name>.<arch-name>
| <entity-name>
```

Running SpyGlass with constraint files

For batch mode runs, add the following command in the SpyGlass project file:

```
read_file -type sgdc mySGCD.sgdc
```

In Atrienta Console GUI, specify the SGDC files by using the Add Files(s) option under the Add Design Files tab.

SpyGlass DFT ADV Constraints Usage Features

SpyGlass Design Constraints (SGDC) provides additional design information that is not apparent in an RTL.

This section explains the following topics:

- [Specifying Values for Vector Signals](#)
- [Using Wildcards](#)
- [Using Macros in SGDC](#)

Specifying Values for Vector Signals

You can specify single-bit/multi-bit values for vector signals in the following ways:

- Single-bit value assigned to each bit (direct assignment)

Consider the following bus:

```
BUSA : InOut STD_LOGIC_VECTOR (5 down to 0);
```

Now, you want to specify the following testmode requirements on the above bus:

```
BUSA [5] =1
BUSA [4] =1
BUSA [3] =1
BUSA [2] =0
BUSA [1] =0
BUSA [0] =0
```

Then, specify the `test_mode` constraints as follows:

```
test_mode -name BUSA[5:3] -value 1
test_mode -name BUSA[2:0] -value 0
```

- Single-bit value assigned to each bit (valueset as a based number)

To specify the values for all bits of a vector signal, you can specify the value as a Binary/Decimal/Hexadecimal number enclosed in curly brackets as in the following example:

```
test_mode -name BUSB[7:0] -value {b 00001111}
```

The above specification can also be written with different value bases as follows:

```
test_mode -name BUSB[7:0] -value {d 15}
test_mode -name BUSB[7:0] -value {h 0F}
test_mode -name BUSB[7:0] -value {b 00001111}
```

Here, `b` represents the base name (binary).

SpyGlass always assigns from lower bit to higher bits and higher bits are always padded, if needed. For example,

```
test_mode -name BUSA[5:0] -value {b 0101}
```

is equivalent to the following bit-value assignments:

```
BUSA[0] = 1
BUSA[1] = 0
BUSA[2] = 1
BUSA[3] = 0
BUSA[4] = 0
BUSA[5] = 0
```

Also, consider the following example:

```
test_mode -name BUSA[0:5] -value {b 0101}
```

This is equivalent to the following bit-value assignments:

```
BUSA[0] = 0
```

```
BUSA[1] = 1
BUSA[2] = 0
BUSA[3] = 1
BUSA[4] = 0
BUSA[5] = 0
```

- Specifying values for all bits of a vector

To specify a value for all bits of a vector, use the name of the vector without using any index.

```
test_mode -name BUS -value {b 0101}
```

where BUS is declared as:

```
input [0:3] BUS;
```

- Multi-bit value assigned to each bit (direct specification)

Assume that a 5-bit bus is required for the test mode specification. If the same values or sequence of values are required on all bits of the bus then the following `test_mode` constraint specification can be used:

```
test_mode -name BUSA[4:0] -value 0001111
```

This is equivalent to:

```
BUSA[4] =0001111
BUSA[3] =0001111
BUSA[2] =0001111
BUSA[1] =0001111
BUSA[0] =0001111
```

- Multi-bit value assigned to each bit (using wildcards)

Use '*' to specify the value as follows:

```
test_mode -name BUSA[4:0] -value <3*0><4*1>
```

See [Using Wildcards](#) for more details.

NOTE: *Spaces within the value sequences are ignored.*

- Specifying the same value to all bits

Consider the following example where 1 is to be applied to all four bits of `tm`:

```
test_mode -name tm[3:0] -value 1
```

This is equivalent to:

```
test_mode -name tm[3] -value 1
test_mode -name tm[2] -value 1
test_mode -name tm[1] -value 1
test_mode -name tm[0] -value 1
```

A sequence can be applied to all four bits of a vector. Suppose 0110 is required on all 4 bits of tm as shown below:

```
test_mode -name tm[3:0] -value 0110
```

This is equivalent to:

```
test_mode -name tm[3] -value 0110
test_mode -name tm[2] -value 0110
test_mode -name tm[1] -value 0110
test_mode -name tm[0] -value 0110
```

- Specify different values to different bits.

Consider the following example:

```
test_mode -name tm[3:0] -value {b 0111}
```

This is equivalent to:

```
test_mode -name tm[3] -value 0
test_mode -name tm[2] -value 1
test_mode -name tm[1] -value 1
test_mode -name tm[0] -value 1
```

Consider the following example where sufficient bits are not present and therefore padding happens with 0.

```
test_mode -name tm[3:0] -value {b 1}
```

This is equivalent to:

```
test_mode -name tm[3] -value 0
test_mode -name tm[2] -value 0
test_mode -name tm[1] -value 0
test_mode -name tm[0] -value 1
```

- Specifying sequence of different values for different bits.

Consider the following example:

```
test_mode -name tm[3:0] -value {b 0111 1101 0001}
```

This is Equivalent to:

```
test_mode -name tm[3] -value 010
test_mode -name tm[2] -value 110
test_mode -name tm[1] -value 100
test_mode -name tm[0] -value 111
```

Using Wildcards

You can use '*' with the `-value` argument while specifying the `test_mode` and `define_tag` constraints.

You can specify the sequences of ones, zeros, and x's using the '*' wildcard character that indicates the number of times the particular value on its left is to be applied.

Consider the following `define_tag` specification:

```
define_tag -tag tag1 -name top.rst1 -value 0 0 1 1
```

The above specification can be rewritten using the wildcard character as follows:

```
define_tag -tag tag1 -name top.rst1
  -value <2*0> <2*1>
```

Please note the following:

- You must enclose the values containing wildcards in `<>`.
- The format to use the wildcard is `<N*V>` (no spaces) where `V` is the value to be applied and `N` is the number of times the value is to be repeated. The value `V` can be 0, 1, x, or X and `N` can be any non-zero positive integer number but cannot be zero).
- Each set of operations with the wildcard character must be separated by a blank space from the next operation. For example:

```
-value <3*1> 000 <50*x> 101
```

Here, the sequence applied will be three 1's, three 0's, fifty x's, then sequence 101.

Using Macros in SGDC

Macros enable you to define logical names for a collection of generic design

Key Concepts

objects, such as, flip-flop, latch, and mux, for which you want to perform a particular check through a set of SGDC commands.

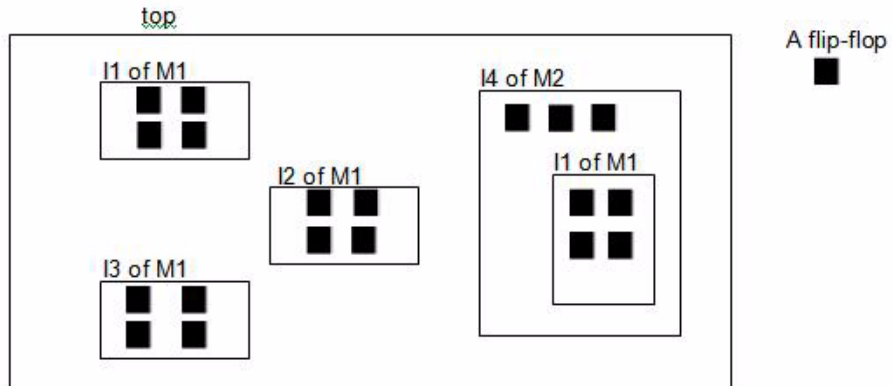
Supported Macros

The Following lists the macros supported by the SpyGlass DFT ADV and SpyGlass MBIST products.

PLL_CLOCK_IN	PLL_CLOCK_OUT	PLL_RESET	DIVIDER_CLOCK_IN
DIVIDER_CLOCK_OUT	DIVIDER_RESET	CLOCK_SHAPER_CLOCK_IN	CLOCK_SHAPER_CLOCK_OUT
CLOCK_SHAPER_RESET	SCAN_FLIP_FLOP_CLOCK / FLIP_FLIP_CLOCK	SCAN_FLIP_FLOP_DATA / FLIP_FLOP_DATA	SCAN_FLIP_FLOP_OUT / FLIP_FLOP_OUT
SCAN_FLIP_FLOP_RESET / FLIP_FLOP_RESET	SCAN_FLIP_FLOP_SET / FLIP_FLOP_SET	SCAN_FLIP_FLOP_ENABLE / FLIP_FLOP_ENABLE	LATCH_ENABLE
LATCH_RESET	LATCH_SET	LATCH_DATA	LATCH_OUT
MUX_SELECT	CGC_CLOCK_IN	CGC_CLOCK_OUT	CGC_SYSTEM_ENABLE
CGC_TEST_ENABLE	BLACK_BOX	BLACK_BOX_INPUT	BLACK_BOX_OUTPUT
INPUT_PORTS	OUTPUT_PORTS	INOUT_PORTS	ALL_PORTS
TIED	TIED_0	TIED_1	TIED_SGDC
TIED_0_SGDC	TIED_1_SGDC	TESTMODE	TESTMODE_SCANSHIFT
TESTMODE_CAPTURE	TESTMODE_ATSPEED	TESTMODE_FUNCTIONAL	CLOCK
CLOCK_SCANSHIFT	CLOCK_CAPTURE	CLOCK_TESTCLOCK	CLOCK_ATSPEED
CLOCK_FUNCTIONAL	SYNCHRONIZER_OUT	SYNCHRONIZER_DATA	SYNCHRONIZER_CLOCK
SYNCHRONIZER_RESET	SYNCHRONIZER_RESET	SYNCHRONIZER_ENABLE	

Example

Consider the following figure:



Now, consider the following macro assignments:

- `FLIP_FLOP_DATA`: Signifies all D-pin flip-flops in the current design unit, `top`.
- `top:FLOP_FLOP_DATA`: Signifies all D-pin flip-flops in the current design unit, `top`.
- `M1:FLIP_FLOP_DATA`: Signifies all D-pin flip-flops in the hierarchy of `M1` module. This implies that if `M1` is instantiated four times in the `top` design unit, that is, `top.I1`, `top.I2`, `top.I3`, and `top.I4.I1`, then all flip-flops in these instantiations are represented by this macro.
- `top.I4.I1:FLIP_FLOP_DATA`: Signifies all D-pin flip-flops in the hierarchy of `top.I4.I1`.

Related Commands

Following SGDC commands support the macro feature:

- [*require_path*](#)
- [*require_strict_path*](#)
- [*require_pulse*](#)
- [*expect_frequency*](#)
- [*require_value*](#)
- [*illegal_path*](#)
- [*illegal_value*](#)

SpyGlass DFT ADV Constraints File Examples

You can specify an SGDC constraint for both Verilog and VHDL designs. This section explains the following topics:

- [Constraint file for Verilog](#)
- [Constraint file for VHDL](#)

Constraint file for Verilog

Consider a Verilog module `mydesign` that requires three signals, `tm1=1`, `tm2=0` and `te=1`, to enter the test mode. Then, the corresponding constraints file is as follows:

```
current_design mydesign
  test_mode -name tm1 -value 1
  test_mode -name tm2 -value 0
  test_mode -name te -value 1
```

Test clock pin and phase information are also specified by constraints.

For the same module `mydesign`, assume that has two test clocks, named `testclock1` and `testclock2`. Then, the updated constraints file is as follows:

```
current_design mydesign
  test_mode -name tm1 -value 1
  test_mode -name tm2 -value 0
  test_mode -name te -value 1
  clock -name testclock1 -testclock -value rto
  clock -name testclock2 -testclock -value rtz
```

For `testclock1`, the actual pattern is `1 => 0 => 1`.

For `testclock2`, the actual pattern is `0 => 1 => 0`.

Constraint file for VHDL

The constraint file for a VHDL design unit (entity named `dft_testIn` and architecture named `logic`) with three test clocks and one testmode signal is shown below.

```
current_design dft_testIn.logic
```

```
clock -name tc1[3:0] -testclock -value rtz
clock -name tc2[4] -testclock -value rtz
clock -name tclk -testclock -value rtz
test_mode -name tm -value 1
```

In the current_design dft_testIn.logic statement, dft_testIn is the entity name of the architecture where the SpyGlass DFT ADV constraints are set and logic is the name of that architecture.

NOTE: When specifying bit or bus, no space is allowed between the bus-name and its index (part-select).

Operating Modes

Following operating modes are available under SpyGlass DFT ADV product:

- *Power Ground*
- *Functional*
- *Shift*
- *Scanshift*
- *Capture*
- *Capture (atspeed)*
- *intialize_for_bist*
- *Dbist*
- *Define_tag*

Power Ground

The Power Ground mode defines the circuit state reached, after simulating the power-ground condition. This mode:

- does not depend on any SGDC command.
- is used when you want to perform some structural check.

Functional

The Functional mode defines the circuit state reached when the functional testmode (`testmode -functional`) constraints are applied and simulated on top of the power-ground circuit state. This mode:

- simulates the `test_mode -functional` SGDC command.
- is used when you want to perform the rule checking in the functional mode.

Shift

The Shift mode defines the circuit state reached, when shift mode (`testmode -scanshift`) constraints are applied and simulated on top of the power-ground circuit state. This mode:

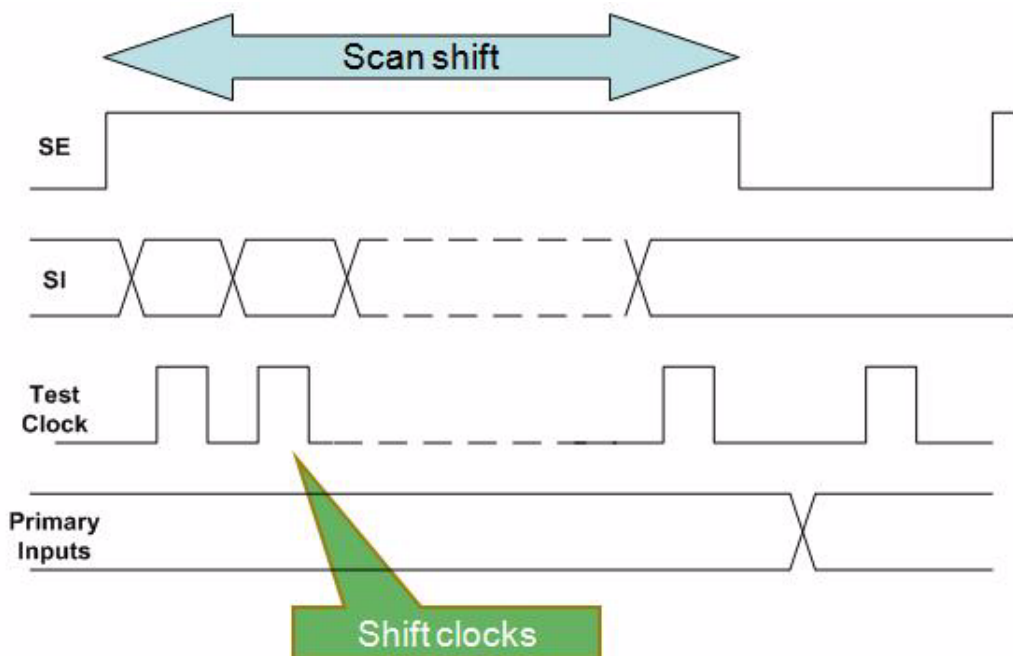
- simulates the following SGDC commands:
 - `testmode` (without any modifier): Specified value is applied.
 - `testmode -scanshift`: Specified value is applied.
 - `testmode -invertInCapture`: Specified value is applied.
- is used when you want to perform rule checking in shift mode, that
- is, in mode when scan chain is formed and shift operation happens.

Scanshift

The scanshift mode defines the circuit state reached, when the scanshift mode (`testmode -scanshift`) constraints are applied and simulated on

top of the power-ground circuit state. This mode:

- simulates the following SGDC commands:
 - `test_mode` (without any modifier): Specified value is applied.
 - `test_mode -scanshift`: Specified value is applied.
 - `test_mode -invertInCapture`: Specified value is applied.
 - is used when you want to perform rule checking in scanshift mode, that
 - is, in mode when scan chain is formed and shift operation happens.
- Consider the following which illustrates the state of operation during the shift phase of scan testing:



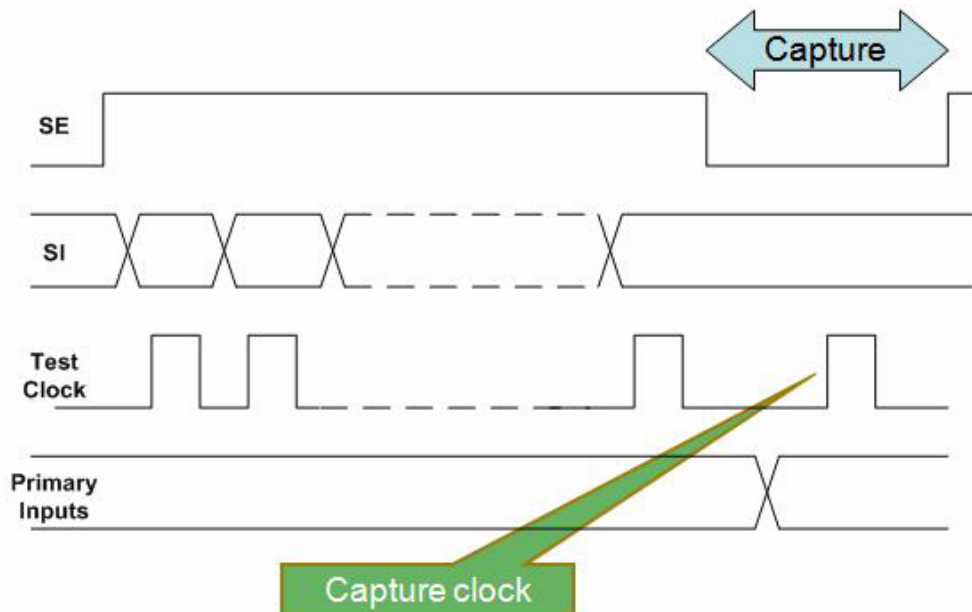
In the above example, the Scan Enable signal (SE) is held high during the scanshift phase. The Scan Input (SI) signal is fed with the required scan data to be inserted in the chain.

Capture

The Capture mode refers to the Capture-static mode. This mode defines the circuit state reached, when the capture mode (`testmode -capture`) constraints are applied and simulated on top of the scanshift mode circuit state. This mode:

- simulates the following SGDC commands:
 - `test_mode -scanshift`: These signals are left free during capture by putting 'X' on them.
 - `test_mode -capture`: Specified value is applied.
 - `test_mode -invertInCapture`: Inverted specified value is applied.
- is used when you want to perform the rule checking in capture mode, that is, in the condition when a capture pulse is fired.

Consider the following figure that illustrates the state of operation during the capture phase of testing:



In the above example, the Scan Enable signal (SE) is held low at the beginning of the capture phase. During this time a single capture clock pulse is applied to the scan flip-flops. This allows the response of the system logic to be captured in the scan flip-flops. Immediately after this capture phase, the shift phase is initiated. In this case, the SE signal goes high. Also, the captured data shifts out through the scan chain.

This process of configuring the design in alternate *shift* and *capture* phases is carried out until all the test vectors are applied and all the system responses are shifted out for comparison with the required response of the circuit.

Capture (atspeed)

The Capture (atspeed) mode defines the circuit state reached, when the atspeed mode (testmode -captureATspeed) constraints are applied and simulated on top of the shift mode circuit state. This mode:

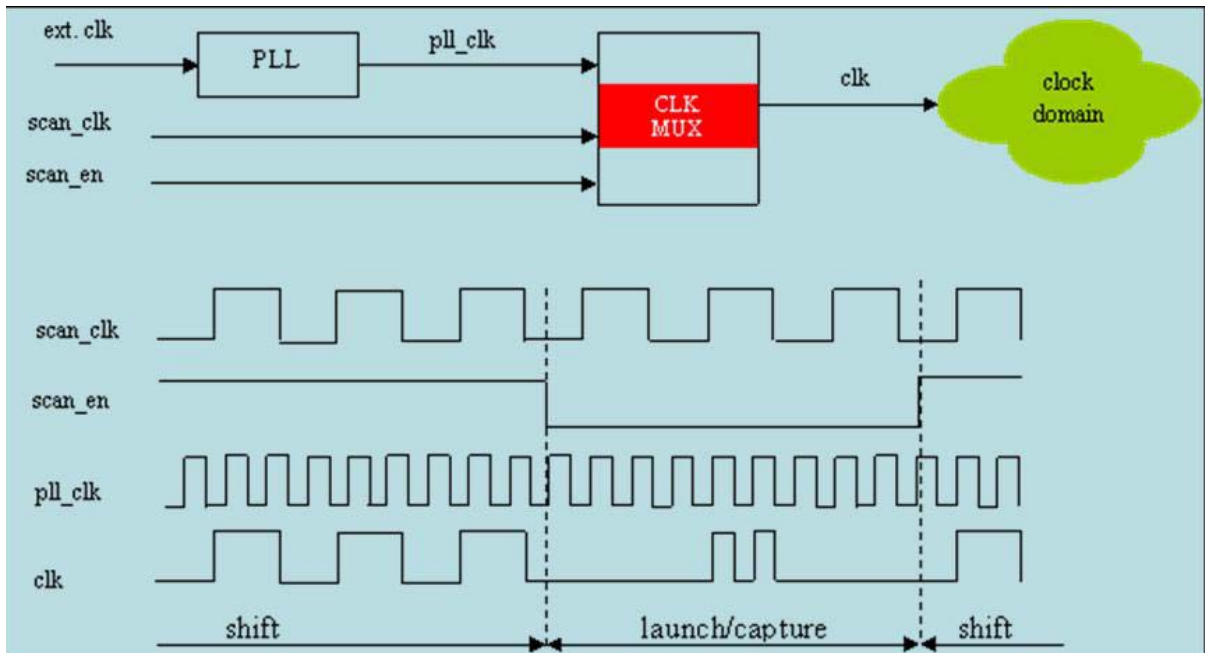
- Simulates the following SGDC commands:
 - testmode -scanshift: These signals are left free during atspeed by putting 'X' on them.
 - testmode -captureATspeed: Specified value is applied.
 - testmode -invertInCapture: These signals are left free during atspeed by putting X on them.
- is used when you want to perform the rule checking in atspeed mode, that is, when atspeed launch and capture pulses are fired.

The atspeed test clocking is complex compared to stuck-at test scenario. In stuck-at testing all the test clocks are driven from the tester directly and are at slower speed than the system clocks. Typical test clocks on the tester are 50-100Mhz.

For Transition ATPG, the most common at-speed clocking uses the double-clock launch/capture technique. The other technique called launch-on-last-shift or LOS scheme needs additional DFT (pipelining) on the scan enable signal and is not currently supported by SpyGlass DFT ADV.

Key Concepts

Consider the following figure:



In the above example, the test clocks are now shared with the functional clocks from the PLL, `pll_clk`. The CLK MUX circuitry selects the slow scan clock, `scan_clk`, in shift mode and the fast launch/capture system clock from the PLL in the capture mode for at-speed testing.

intialize_for_bist

Specifies the mode in which BIST is initialized. This mode:

- simulates the `test_mode -intialize_for_bist` command.
- is simulated on top of the *Power Ground* mode.

Dbist

Specifies the *dbist* constraint. This mode:

- simulates the `dbist` command.

- is simulated on top of the Power Ground mode.

Define_tag

Specifies the mode in which the user-specified condition is simulated. This mode:

- simulates the *define_tag* command.
- is simulated on top of the Power Ground mode.

Types of Flip-Flops

SpyGlass DFT ADV solution processes the following types of flip-flops:

- [Scannable Flip-Flops](#)
- [No-Scan Flip-Flops](#)
- [Unscannable Flip-Flops](#)
- [Synthesis Redundant \(SR\) Flip-Flops](#)

This section also explains the [Difference Between no_scan and Unscannable Flip-Flops](#).

For back annotation data generated by the [Info_DftDebugData](#) rule for various flip-flops, see [Understanding Back Annotation](#).

Scannable Flip-Flops

The scannable flip-flops are of the following types:

- **Forced:** These are the flip-flops that you can specify using the [force_scan](#) constraint. Use the [Info_forcedScan](#) rule to identify such flip-flops.
- **Inferred:** These are the flip-flops that meet the following scannability criteria:
 - For RTL designs: If a flip-flop does not have the [Clock_11](#) and [Async_07](#) violations
 - For Netlist designs: If a flip-flop does not have the [Clock_11](#) and [Async_07](#) violations and is part of valid scan chain. If the value of the [dft_all_scan_chains_defined](#) parameter is set to no, then "**is part of a valid scan chain**" condition is not required. In this case, scannability requirements for both rtl and netlist designs become the same.

No-Scan Flip-Flops

The no-scan flip-flops are of the following types:

- **Forced:** These are the flip-flops for which you specify the [force_no_scan](#) constraint.
The [Info_noScan](#) rule reports such flip-flops.

- **Inferred:** These are the flip-flops that meet the following criteria:
 - ❑ A forced value is present at the output of a flip-flop during the shift simulation
 - ❑ A forced test_clock is present on the output of a flip-flop
 - ❑ A flip-flop is not part of a valid scan chain in the scan-stitched design, that is, the `dftDesignState` parameter is set to `post_scan_stitched` and the `dft_all_scan_chains_defined` parameter is set to `yes`
 - ❑ The output of a flip-flop is getting a non-X value due to propagation of the shift condition, the `dft_infer_sequential_propagation_as_no_scan` parameter is set to `on`, and the asynchronous pins of the flip-flops are not active.
- The [Info_inferredNoScan](#) rule reports such flip-flops.

Unscannable Flip-Flops

The unscannable flip-flops are the flip-flops that are not declared as scannable.

These flip-flops may go on a `scan_chain` if you correct the clock and asynchronous pin controllability ([Async_07](#) and [Clock_11](#)).

Synthesis Redundant (SR) Flip-Flops

The synthesis redundant flip-flops are the flip-flops that have a constant value in the power-ground mode.

By default, flip-flops are not marked as SR even if their D-pin have constant value in power-ground mode.

Consider the following schematic:

Key Concepts

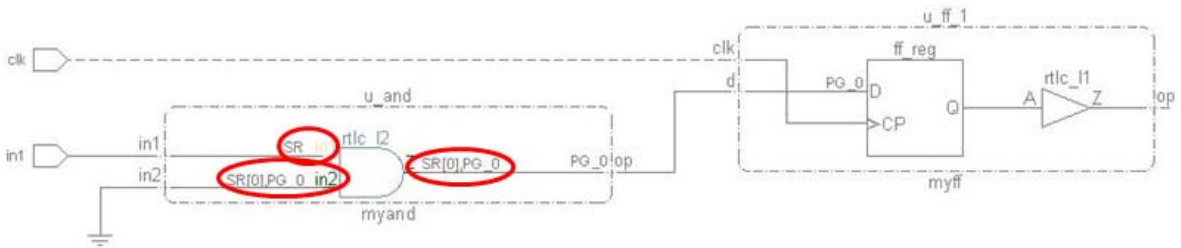


FIGURE 5. SR Faults

In the above example:

- Three pins are marked as SR.
- Six faults are considered SR.
- SR denotes that the pin is blocked in the PG_MODE (Power Ground Mode)
- SR[0] denotes that the pin has value 0 in PG_MODE (PG_0)
- SR[1] denotes that the pin has value 1 in the PG_MODE (PG_1)

However, to allow constant propagation through flip-flops and to mark them SR if required conditions are met, set the value of:

- the *dft_ignore_constant_supply_flip_flops* parameter to yes
- the *enable_const_prop_thru_seq* command to yes.

Consider the following example:

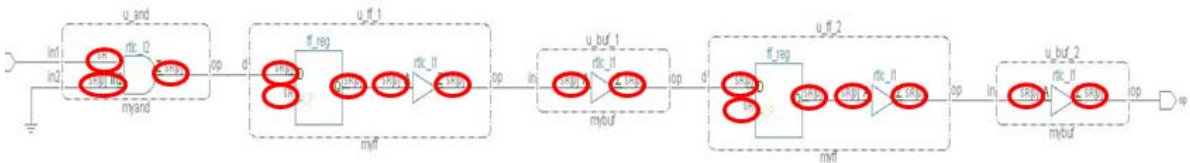


FIGURE 6. SR Faults When Parameters are On

In the above example:

- 17 pins are marked as SR.

- 34 faults are considered SR.
- SR denotes that the pin is blocked in the PG (Power Ground Mode)
- SR[0] denotes that the pin has value 0 in PG (PG_0)
- SR[1] denotes that the pin has value 1 in the PG (PG_1)

Difference Between no_scan and Unscannable Flip-Flops

Following points highlight the difference between the no_scan and unscannable flip-flops:

- no_scan: These flip-flops are not intended to be on scan_chain. See [No-Scan Flip-Flops](#) section for more information on this type of flip-flop.
- **Unscannable:** These flip-flops may go on scan_chain if we correct the clock and async pin controllability (Async_07 and Clock_11).

Support for Multi-Bit Flip-Flop Cells

A flip-flop bank refers to multi-bit flip-flops that are identified by presence of the `ff_bank` keyword in their `.lib` cell description.

SpyGlass DFT ADV supports multi-flop cell, if following conditions are satisfied:

- Corresponding control signals, that is, clock, reset, and set, of flip-flops inside cell are tied together and controlled from a top-level pin of a cell
- Library compiler is able to create a functional view for the cell

You can enable/disable the support for flip-flop banks by setting the value of the `dft_support_flip_flop_bank` parameter to on. By default, the support for flip-flop bank is enabled.

[Figure 7](#) illustrates a sample flip-flop bank (`ff_bank`) description in the `.lib` cell file:

```

227         ff_bank(IQ,IQN,2) {
228             clocked_on : "CLK";
229             next_state : "D";
230         }
231     pin (CLK) {
232         direction : "input" ;
233     }
234     pin (SE) {
235         direction : "input" ;
236         signal_type : "test_scan_enable";
237     }

```

FIGURE 7. ff_bank description

The following rules in the SpyGlass DFT ADV product support flip-flop banks:

Async_01	Async_02_capture	Async_02_shift	Async_03
Async_05	Async_07	Async_08	Async_09
Async_13	Atspeed_01	Atspeed_03	Atspeed_04
Atspeed_05	Atspeed_06	Atspeed_09	Atspeed_10
Atspeed_11	Atspeed_12	Atspeed_13	Atspeed_14
Atspeed_15	Atspeed_19	Atspeed_20	Atspeed_22
Atspeed_25	Atspeed_27	Atspeed_30	Atspeed_33
BIST_01	CG_03_atspeed	Clock_02	Clock_04
Clock_08	Clock_09	Clock_11	Clock_11_capture
Clock_21	Clock_25	Clock_28	Coverage_audit
Diagnose_02	Diagnose_03	Diagnose_04	Diagnose_ScanChain
Diagnose_testclock	Diagnose_testmode	Info_atSpeedClock	Info_atSpeedDomain
Info_atSpeedFrequency	Info_coverage	Info_DftDebugData	Info_forcedScan
Info_inferredNoScan	Info_noAtspeed	Info_noFault	Info_noScan
Info_pwrGndSim	Info_scanchain	Info_scanwrap	Info_synthRedundant

Info_testmode	Info_transition Coverage	Info_transitionCoverage_audit	Info_untestable
Info_unused	Info_random_resistance	Latch_08	Latch_17
Latch_18	PLL_03	Scan_20	Scan_22
Scan_24	Scan_25	Scan_26	Scan_27
Scan_28	Scan_29	Scan_31	Scan_35
Scan_36	Scan_38	Scan_39	Scan_40
Scan_41	Soc_09	Topology_01	Topology_02
Topology_03	Topology_10	Topology_11	Topology_13
Topology_14	Tristate_10		

Types of Latches

The SpyGlass DFT ADV solution reports the following categories of latches:

- *Transparent*
- *Shadow*
- *Scannable*
- *Retiming or Lockup Latches*

Transparent

A latch is considered transparent, if one of the following conditions hold true:

- Its enable pin gets an active value in shift, capture, or atspeed mode
- Its enable pin gets an active value when the test clocks are simulated to their off state, that is, 0 for rtz (010, and 1 for rto (101)

NOTE: *A latch may become active when the off state of a clock is simulated but the latch may not necessarily belong to that clock domain as proper clock pulse may fail to get to the latch enable.*

- Its enable pin is controllable to the active value

Following examples describe different scenarios in which a latch is

considered a transparent latch.

Example 1

Consider the following design where the enable pin of the latch is getting an active value in the shift, capture, or atspeed mode:

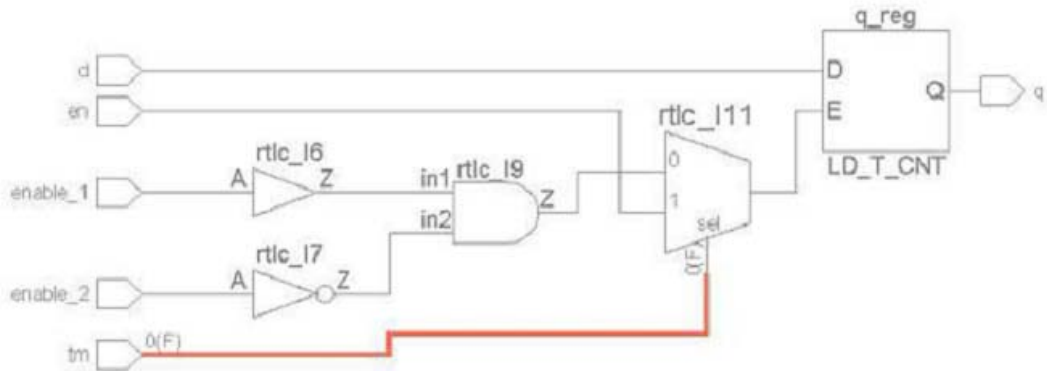


FIGURE 8. Enable pin getting active value

In the above example, the enable pin of the latch, q_reg is getting an active value in a given mode (shift, capture or atspeed). Therefore, this latch is considered transparent.

Example 2

Consider the following example where the enable pin of the latch is getting an active value:

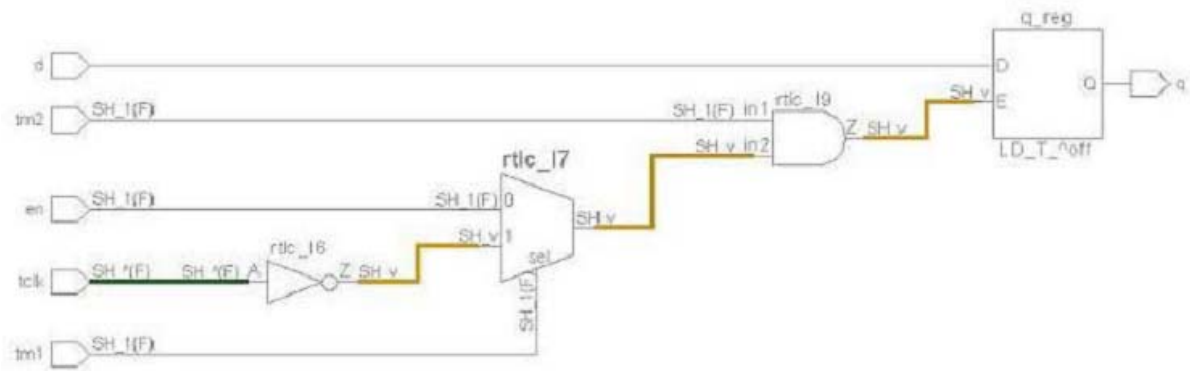


FIGURE 9. Enable pin getting a test clock

In the above example, the enable pin of the latch, q_reg, is getting a test clock, tclk. The enable pin of q_reg is in its active state when tclk is simulated in its off state. Therefore, this latch is considered as transparent

Shadow

A latch is considered as a shadow latch, if all of the following conditions hold true:

- The D-pin of a latch is driven by exactly one scan flip-flop
- The EN-pin of a latch is clocked by the same clock and in the same phase as that of the scan flip-flop, which is driving its D-pin

The output of a shadow latch is considered fully controllable because it mirrors the controllability of the scan flip-flop, which is driving its D-pin. The shadow latch analysis is done in the shift mode.

The detection of shadow latches is controlled by the [dft_detect_shadow_latches](#) parameter. By default, the value of the dft_detect_shadow_latches parameter is set to on. Therefore, the shadow latches are identified and treated accordingly. Set the value of the parameter to off to [dft_detect_shadow_latches](#) treat all the latches normally.

See [dft_detect_shadow_latches](#), to view the related examples and

understand the latch behavior when the *dft_detect_shadow_latches* parameter is set to on or off.

Scannable

A latch is considered scannable if the *dft_scannable_latches* parameter is set to on and all the following conditions hold true:

- Latch is not marked as a transparent or shadow latch using the following parameters:
 - dft_detect_shadow_latches*
 - dftSetLatchFedByTCIkAsT*
 - dftTreatLatchesAsTransparent*
- Latch enable pin is receiving test clocks through the *sensitized* path in the scanshift mode

A latch is not considered scannable, if the *dft_scannable_latches* parameter is set to off.

The following table lists the conditions when a latch is marked as a scannable latch:

Parameter	Value	Clock Type
<i>dft_scannable_latches</i>	ON	Inferred Clock
<i>dft_use_specified_and_inferred_clocks</i>	OFF	
<i>dft_scannable_latches</i>	ON	User-specified clock
<i>dft_use_specified_and_inferred_clocks</i>	ON	

Control and Observe seeding for scannable latches is the same as for the scannable flip-flops.

The scannable latches are denoted as *LD_SCN_I* in the back-annotation (BA) data.

Retiming or Lockup Latches

Retiming or lockup latches are categorized under following categories:

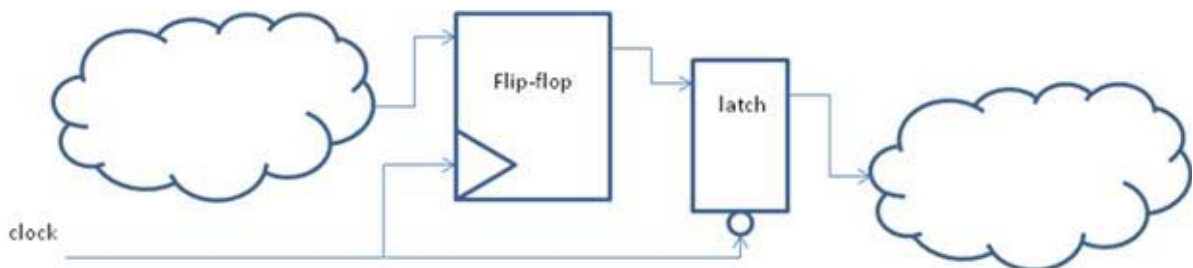
- [Source Retiming Latches](#)
- [Destination Retiming Latches](#)

Source Retiming Latches

Source retiming latches are recognized by:

- The latch data-pin fan-in cone comes from the edge-triggered flip-flops that all use the same clock source (CS) and phase.
- The latch enable is also from CS.
- The latch enable active state is 0 if the CS is rising edge and 1 if the CS is falling edge.
- The set and reset pins on a retiming latch must not be used. The term, **Not Used**, means that if the retiming latch has either a set or reset pin or both, these pins must be tied off to the static 1 or 0 level that is the **inactive** level for that set or reset.

Following figure illustrates a source retiming latch:



Destination Retiming Latches

Destination retiming latches are recognized by:

- The latch q-pin fan-out cone goes to edge triggered flip-flops that all use the same clock source and phase (call this source CD)
- The latch enable is also from CD
- The latch enable active state is 1 if CD is rising edge and 0 if CD is falling edge
- The set and reset pins on a retiming latch must not be used. "Not used" means that if the retiming latch has either a set or reset pin or both,

Key Concepts

these pins must be tied off to a static 1 or 0 level that is the 'inactive' level for that set or reset.

The following figure illustrates a destination retiming latch:

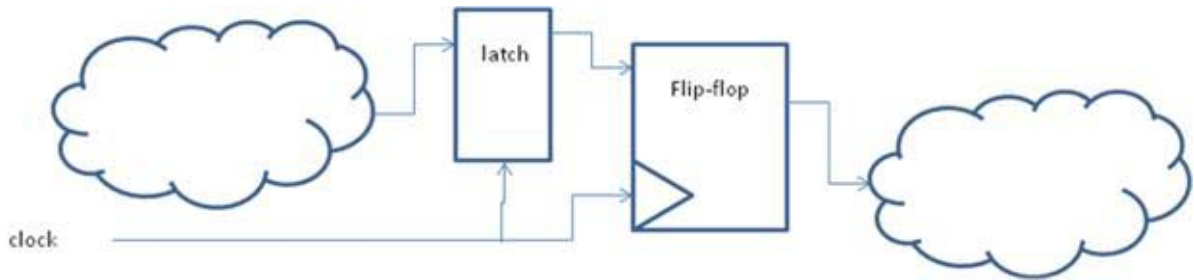


FIGURE 10. Destination Retiming Latch

Identifying Clock Gating Cells

SpyGlass DFT ADV identifies Clock Gating Cells (CGCs) using the following methods:

- *.lib attributes*
- *User-specified gating_cell SGDC command*
- *Inference in the RTL code*
- *Inferring Flat CGC*

Additionally, SpyGlass DFT ADV reports the data related to the identified CGCs using the following ways:

- Following reports generate data related to the identified CGCs:
 - Clock Gating Report*
 - Reports Generated by the Info_dftDebugData Rule*
- *Back Annotation (BA) data for identified CGCs*

.lib attributes

SpyGlass DFT ADV product checks for the keyword, *clock_gating_integrated_cell*, in the .lib file.

Consider the following example:

```
cell ( my_cgc ) {
    clock_gating_integrated_cell : "latch_posedge";
    statetable ( " CLK CEN ", "IQ " ) {
        table : " L L : - : L , \
                L H : - : H , \
                H - : - : N ";
    }
}
pin (GCLK) {
    direction : output ;
    state_function : "CLK * IQ "
}
.
.
}
```

In the above example, SpyGlass DFT ADV product searches for the keyword, `clock_gating_integrated_cell`. Here, the identified CGC name is `latch_posedge`.

User-specified gating_cell SGDC command

You can specify a CGC through the `gating_cell` sgdc command.

Refer to the `gating_cell` section in the *Consolidated Constraints Application Note* for more information on the syntax and usage of the `gating_cell` command.

Consider the following example:

```
gating_cell -name wb_cg_1 -clkinTerm clkin -clkoutTerm clkout
-enTerm en -enValue 1 -testenTerm te -testenValue 1
```

In the above example, the CGC name is `wb_cg_1`.

Inference in the RTL code

To support structural inference of CGC for both stuck-at and transition analysis, SpyGlass DFT ADV product infers CGCs in the RTL code itself. However, parameterized modules or modules with vector ports are ignored from CGC inference.

To infer a CGC, SpyGlass DFT ADV product checks for the following conditions:

- Entire gating structure should be in a module or entity boundary
- It should either have one or two enable pins
- It can have any number of unconnected ports.
- It follows standard naming convention. Following are the supported test enable and system enable names:
 - **System Enable Name:** EN, en, E, e, ENABLE, enable, SE, se,
 - **Scan/Test Enable Name:** TE, te, TEST_ENABLE, test_enable, SE, se, SCAN_ENABLE, scan_enable, TEN, ten

Using any other names for the system-enable and test-enable modules, apart from the above listed standards, then DFT will report such modules as potential CGCs.

- It should have simple or regular CGC structures. For OR gate for enables, following inferences are made:
 - Latch with inverted clock connected to enable followed by AND-Gate with latch output and clock, is inferred as Inferred Positive Edge CGC (WB_iCGC_POS). Following figure shows a sample Positive Edge CGC:

```

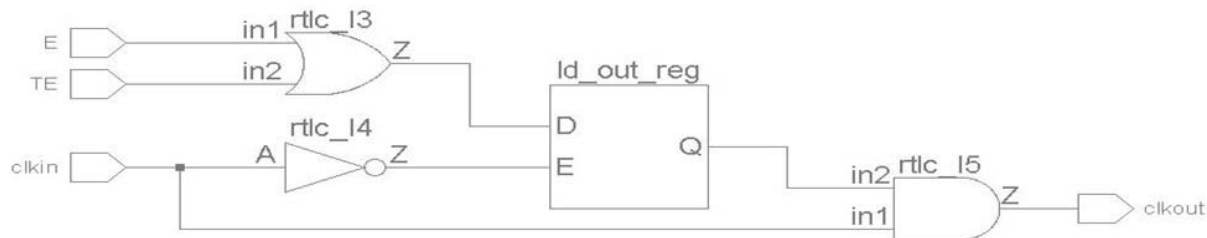
11 module icg_pos_correct (output clkout, input clkkin, E, TE);
12     wire ld_in;
13     reg ld_out;
14
15     assign ld_in = E | TE;
16     always @(ld_in or clkkin) begin
17         if (!clkkin) ld_out <= ld_in;
18     end
19     assign clkout = clkkin & ld_out;
20 endmodule

```

Schematic: icg_pos_correct(top.u_cgc_pos_correct)

Edit View Help

Find



```

#Inferred
# gating_cell -name "icg_pos_correct" -clkkinTerm "clkkin" -clkoutTerm "clkout" -cgcEdgeType "positive" -enTerm "E" -enValue 1 -testenTerm "TE" -testenValue 1
#cgc_instance_count : 1 ,potential affected flip-flops : 1
#top.u_cgc_pos_correct - 1

```

- Latch with same phase clock connected to enable followed by OR-Gate with inverted latch output and clock, is inferred as Inferred Negative Edge CGC (WB_iCGC_NEG). Following figure shows a sample Negative Edge CGC:

Key Concepts

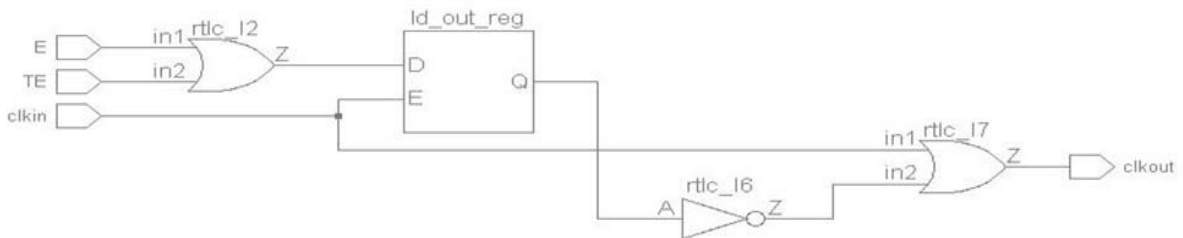
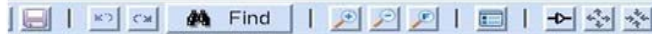
```

28
29 module icg_neg_correct (output clkout, input clkkin, E, TE);
30     wire ld_in;
31     reg ld_out;
32
33     assign ld_in = E | TE;
34     always @(ld_in or clkkin) begin
35         if (clkkin) ld_out <= ld_in;
36     end
37     assign clkout = clkkin | !ld_out;
38 endmodule

```

Schematic : icg_neg_correct (top.u_cgic_neg_correct)

e Edit View Help



#Inferred

```

# gating_cell -name "icg_neg_correct" -clkkinTerm "clkkin" -clkoutTerm "clkout" -cgcEdgeType "negative" -enTerm "E" -enValue 1 -testenTerm "TE" -testenValue 1
#cgc_instance count : 1 ,potential affected flip-flops : 1
#top.u_cgic_neg_correct - 1

```

Inferred or potential CGCs are reported in the [Clock Gating Report](#).

For inferred CGCs, [gating_cell](#) behavior is applied in the same run. However, for potential CGCs, review the generated SGDC command before applying the [gating_cell](#) behavior in the subsequent runs.

Consider the following examples showing the successful and unsuccessful CGC inferences:

Example 1

```
module cgc_len_ok (output clkout, input clkin, en);  
    reg ldout;  
  
    always @(clkin or en) begin  
        if (!clkin) ldout <= en;  
    end  
    assign clkout = clkin & ldout;  
endmodule
```

Inference **Successful**

```
module cgc_len_not_ok (output clkout, input clkin, en);  
    reg ldout;  
  
    always @(clkin or en) begin  
        if (!clkin) ldout <= en;  
    end  
    assign clkout = clkin | ldout;  
endmodule
```

Inference **Unsuccessful**

```
module cgc1_2en_ok (output clkout, input clkin, en, te);  
    reg ldout;  
    wire ldin;  
  
    assign ldin = en | te;  
    always @(clkin or ldin) begin  
        if (!clkin) ldout <= ldin;  
    end  
    assign clkout = clkin & ldout;  
endmodule
```

Inference **Successful**

Example 2

```

module cgc2_2en_ok (output clkout, input clkkin, EN, TE);
  reg ldout;
  wire ldin;

  assign ldin = EN | TE;
  always @(clkkin or ldin) begin
    if (!clkkin) ldout <= ldin;
  end
  assign clkout = clkkin & ldout;
endmodule
module cgc3_2en_ok (output clkout, input clkkin, EN, SE);
  reg ldout;
  wire ldin;

  assign ldin = EN | SE;
  always @(clkkin or ldin) begin
    if (!clkkin) ldout <= ldin;
  end
  assign clkout = clkkin & ldout;
endmodule
module cgc1_2en_not_ok (output clkout, input clkkin, EN, SE);
  reg ldout;
  wire ldin;

  assign ldin = EN & SE;
  always @(clkkin or ldin) begin
    if (!clkkin) ldout <= ldin;
  end
  assign clkout = clkkin & ldout;
endmodule

```

Inference **Successful**

Inference **Successful**

Inference **Unsuccessful**

Example 3

```

module cgc1_2en_partial_ok (output clkout, input clkin, XX, YY);
  reg ldout;
  wire ldin;

  assign ldin = XX | YY;
  always @(clkin or ldin) begin
    if (!clkin) ldout <= ldin;
  end
  assign clkout = clkin & ldout;
endmodule

```

Inference **partially** successful because software is not able to distinguish system-enable vs test-enable pins based on names

Inferring Flat CGC

In flat CGC's, gating logic is not necessarily enclosed in a module. To infer a flat CGC, SpyGlass DFT ADV product checks for the following conditions:

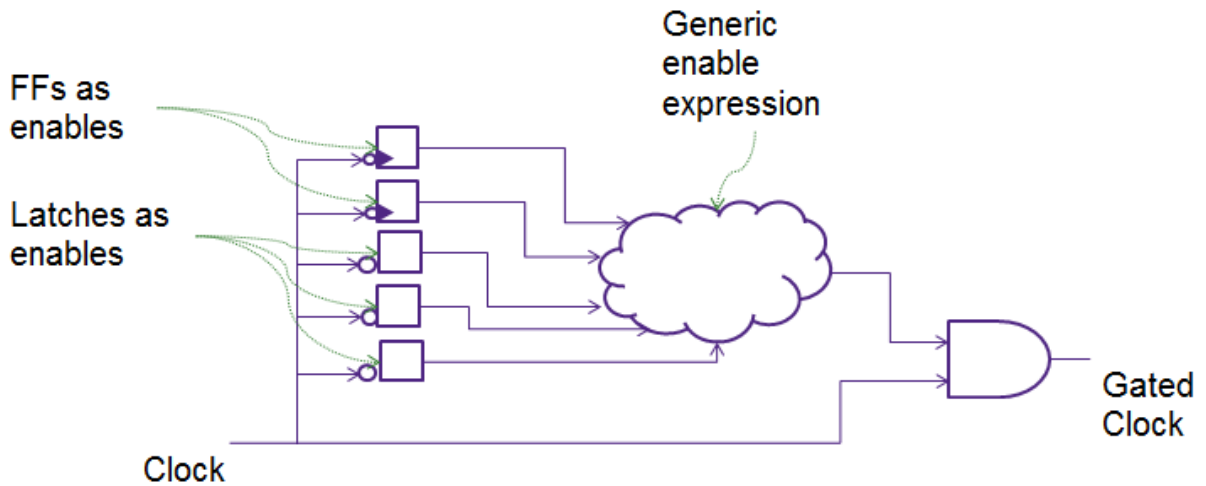
- *Positive CGC*
- *Negative CGC*

Positive CGC

An AND gate is considered a positive CGC with 2 inputs (clock in and system enable) when the following conditions are met:

- Enable logic driving gating 'and' gate is driven by any generic combination expression, which in turn, is driven by flop/latch outputs
- Clock source for enable pin of latches (calculated by traversing back through buffer, inverters and CGC's) and other input of gating 'and' gate should be same but with a phase inversion.

Figure 11 illustrates a positive CGC:



Note: Generic enable expression can be any combinational logic

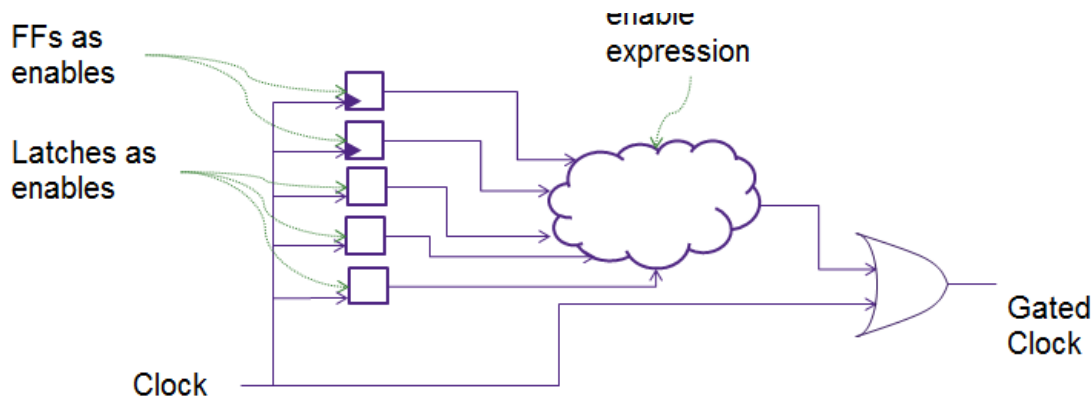
FIGURE 11. Positive CGC

Negative CGC

An OR gate is considered a negative CGC with 2 inputs (clock in and system enable) when the following conditions are met:

- Enable logic driving gating 'or' gate is driven by any generic combinational expression, which in turn, is driven by flop/latch outputs.
- Clock source for enable pin of latches (calculated by traversing back through buffer, inverters and CGC's) and other input of gating 'or' gate should be same and in phase.

Figure 12 illustrates a negative CGC:



Note: Generic enable expression can be any combinational logic

FIGURE 12. Negative CGC

Clock Gating Report

The `CG_generateReport` rule generates the following `dft_dsm_clock_gating.rpt` report:

```
current_design "top"
# Section I:

gating_cell -name "cgc1_2en_partial_ok" -clkInTerm "clkIn" -clkOutTerm "clkOut" -cgcEdgeType "positive" -enTerm "YY"
-enValue 1 -testenTerm "XX" -testenValue 1
# gating_cell -name "cgc1_2en_partial_ok" -clkInTerm "clkIn" -clkOutTerm "clkOut" -cgcEdgeType "positive" -enTerm "XX"
-enValue 1 -testenTerm "YY" -testenValue 1
#cgc_instance_count : 1 ,potential affected flip-flops : 1
#top.u7 - 1

#Inferred
# gating_cell -name "cgc1_1en_ok" -clkInTerm "clkIn" -clkOutTerm "clkOut" -cgcEdgeType "positive" -enTerm "en" -enValue 1
#cgc_instance_count : 1 ,potential affected flip-flops : 1
#top.u1 - 1

#Inferred
# gating_cell -name "cgc1_2en_ok" -clkInTerm "clkIn" -clkOutTerm "clkOut" -cgcEdgeType "positive" -enTerm "en" -enValue 1
#cgc_instance_count : 1 ,potential affected flip-flops : 1
#top.u3 - 1
```

In the above sample report, following information is displayed:

Key Concepts

- Potential CGCs: Potential CGCs are reported as proper `gating_cell` command. Report provides all combinations of system enable and test enable pins captured in the `sgdc` but only one will be un-commented. Therefore, inference is partially successful.
- Inferred CGCs: Inferred CGCs are captured as commented `gating_cell` `sgdc` command. Therefore, inference is successful.

Reports Generated by the Info_dftDebugData Rule

To get the debug data for CGC, the [Info_DftDebugData](#) rule generates the `dft_data_01.csv` spreadsheet containing all CGCs, categorized according to their type.

Following is a sample `dft_data_01.csv` spreadsheet report:

	B	C
	Design Characteristic: top	<u>C</u> ount
1 <input type="checkbox"/>	Flip-flop	0
2 <input type="checkbox"/>	Latch	0
3 <input type="checkbox"/>	Black-Box	1
4 <input type="checkbox"/>	Clock Gating Cells (CGCs)	4

Messages: Displayed: 4 Total: 4

Click on the CGC row to open the *dft_data_05.csv* report, which displays the type of identified CGCs, as shown below:

Spreadsheet Viewer - dft_data_05.csv (ReadOnly)

File View Tools Help

Show Header

-1 value=

	B	C
	Clock Gating Cell (CGC) Name	Type
1	top.cgc4	BB_CGC_POS
2	top.cgc2	WB_iCGC_POS
3	top.cgc1	WB_CGC_POS
4	top.cgc3	LIB_CGC_POS

Messages: Displayed: 4 Total: 4

Back Annotation (BA) data for identified CGCs

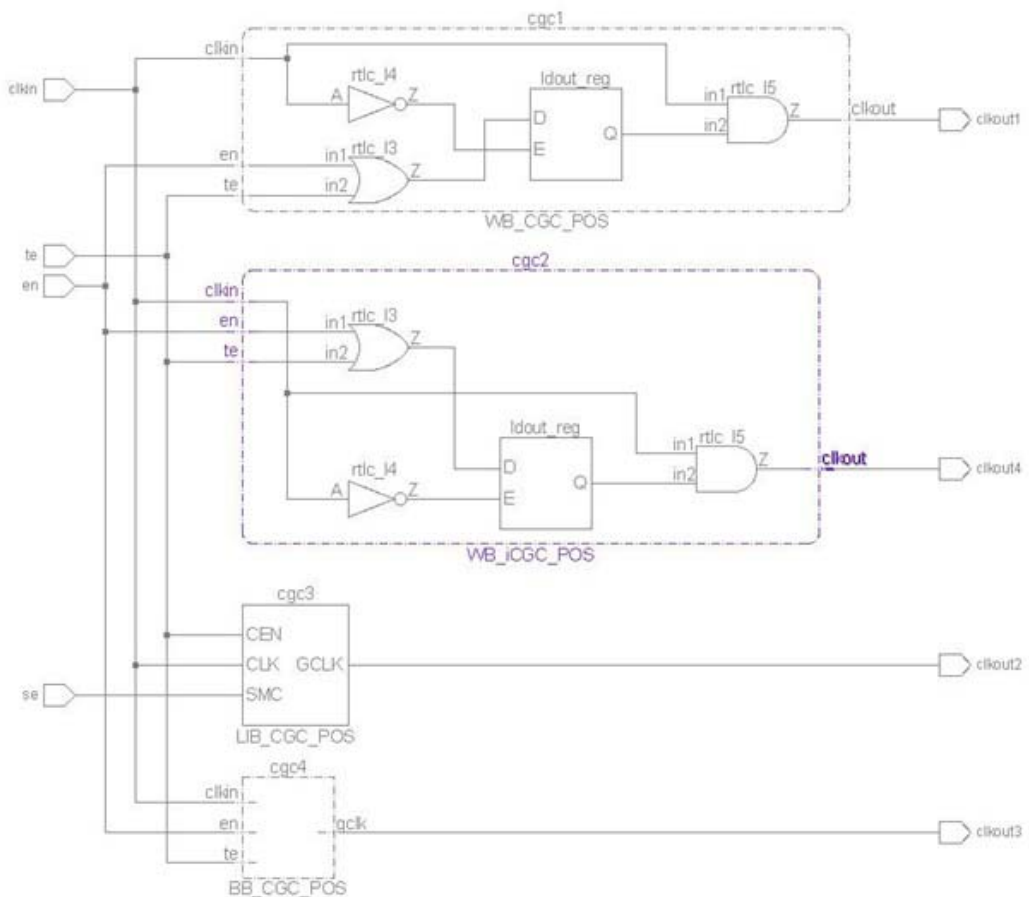
The SpyGlass DFT ADV product categorizes the Back Annotation (BA) data for CGCs as described below:

- **WB_CGC_POS/ WB_CGC_NEG:** Represents a white box specified as CGC, using the *gating_cell* sgdc command. The identified CGC either has a positive or a negative edge.
- **WB_iCGC_POS/ WB_iCGC_NEG:** Represents a white box inferred as a CGC (*gating_cell* sgdc command). The inferred CGC either has a positive or a negative edge.

Key Concepts

- **BB_CGC_POS/ BB_CGC_NEG** - Represents a black box specified as clock gating cell using the *gating_cell* sgdc command. The specified CGC either has a positive or a negative edge.
- **LIB_CGC_POS/ LIB_CGC_NEG** - Represents a lib cell acting as a CGC. The specified CGC either has a positive or a negative edge.
- **FLAT_iCGC_POS/ FLAT_iCGC_NEG** - Represents a flat inferred CGC. The inferred CGC either has a positive or a negative edge.

The following figure illustrates the identified CGCs along with the generated BA data in the incremental schematic:



Identifying Synchronizer

SpyGlass DFT ADV identifies synchronizer using the following methods:

- Inference in RTL code
- .lib cell

A synchronizer:

- should consist of identical flip-flops with same clock and asynchronous signals.
- can cascade n flip-flops within the same hierarchy.
- is considered a lib cell only if synchronizer connections starts and ends on the boundary of the lib cell.
- can not be part of other synchronizers.

Also, only buffer can exist between consecutive cells of a synchronizer.

Examples

3-Flip-Flop Synchronizer

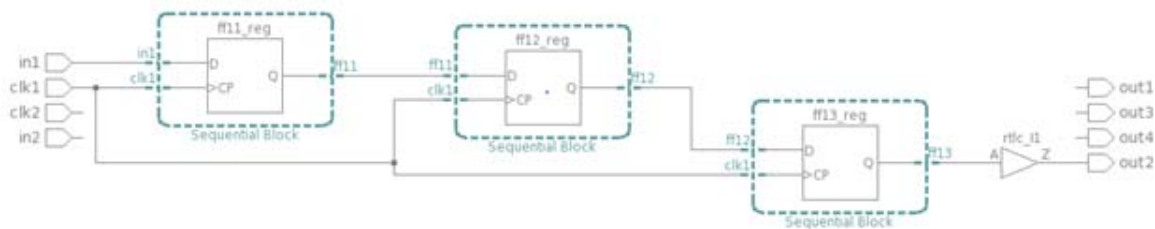


FIGURE 13. 3-Flip-Flop Synchronizer

Branching Out From Flip-Flop's Q-Pin

Key Concepts

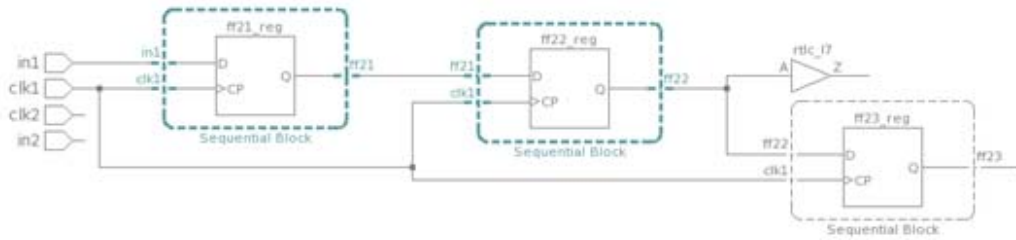


FIGURE 14. Branching Out From Flip-Flop's Q-Pin

Multiple Driver

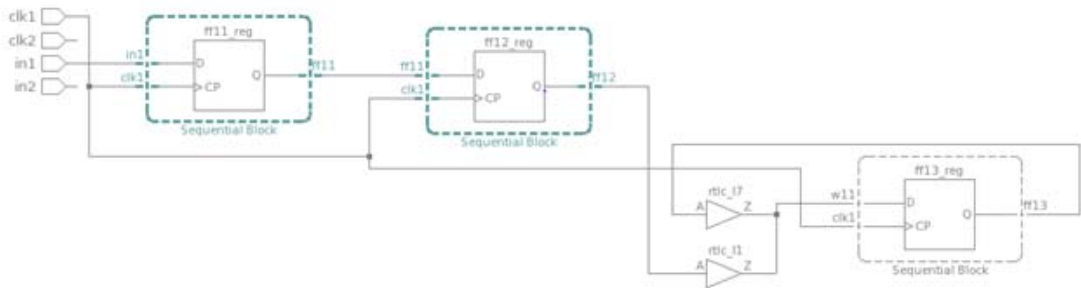


FIGURE 15. Multiple driver

Cascaded 2 Flip-Flops with .lib Flip-Flop

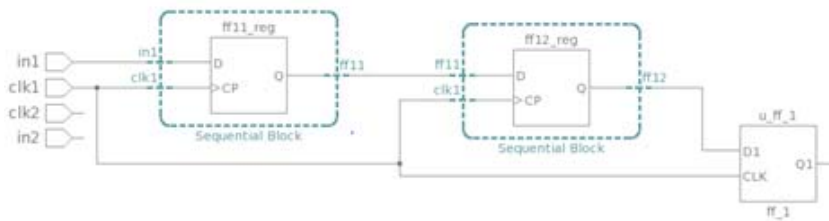


FIGURE 16. Cascaded 2 Flip-Flops with .lib Flip-Flop

Identical Flip-Flops

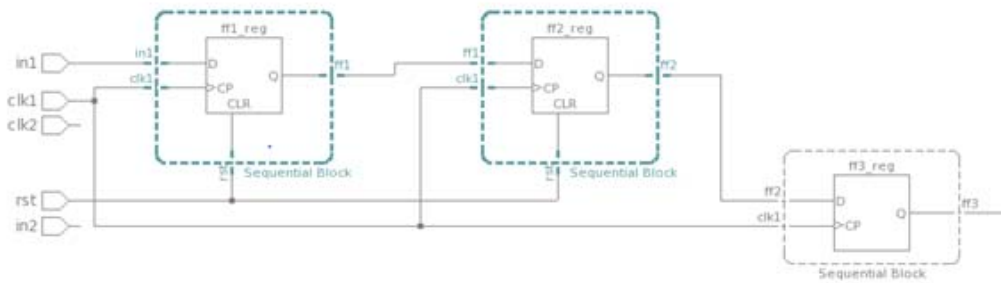


FIGURE 17. Identical Flip-Flops

Types of Faults

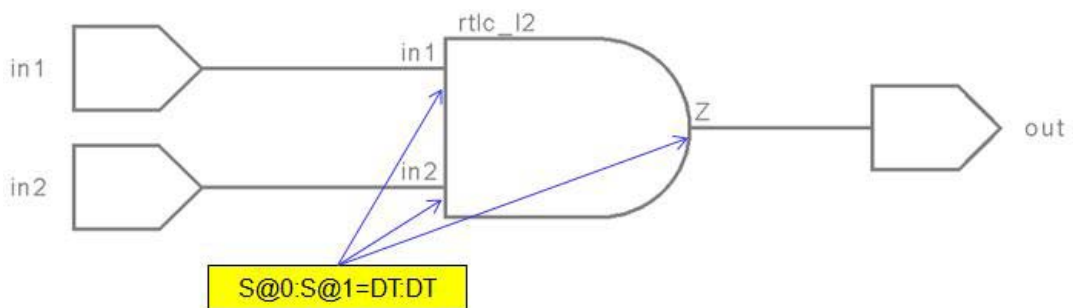
The SpyGlass DFT ADV solution reports the following categories of faults:

- *Detectable (DT)*
- *Un-Testable (UT)*
- *Equivalent (EQ)*
- *Un-Detectable (ND)*
- *Potentially Detectable (PT)*
- *Synthesis Redundant (SR)*
- *Blocked (BL)*
- *Tied (TI)*
- *Un-Used (UU)*
- *Logical Redundant (LR)*

Detectable (DT)

A fault is considered detectable, if the node under consideration is controllable and observable.

Consider the following figure where all faults are marked as DT:



For the AND gate shown in the above figure, both inputs are controllable and the output is observable. Therefore, all the faults are marked as DT.

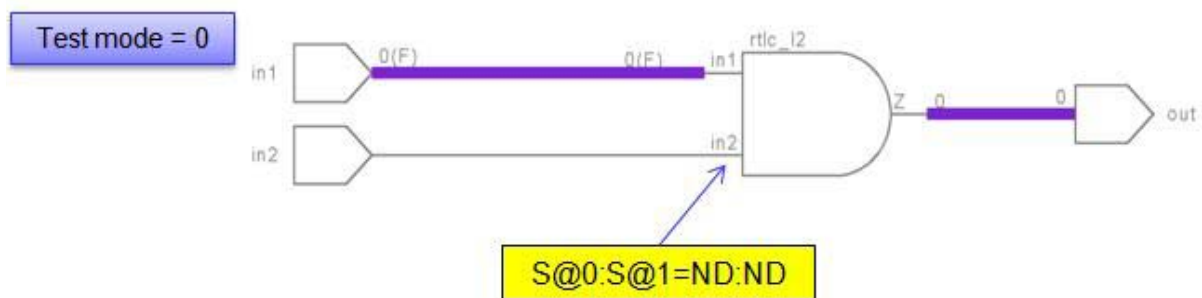
Un-Detectable (ND)

A fault is considered undetectable, if the node under consideration is uncontrollable and / or unobservable in the test mode.

Un-Detectable (ND) faults are reported as ATPG Untestable (AU) in ATPG.

Presence of this fault in the design negatively effects the functionality of the circuit, that is, it may lead to chip malfunctioning.

Consider the following figure with the faults marked as ND:



For the AND gate shown in the above figure, input, in1 is supplied with value 0 in the capture mode. Therefore, faults at the other input of the AND gate are marked as ND.

Equivalent (EQ)

The following table lists the gates and the corresponding conditions that are checked to consider a fault in the stuck-at-fault model in the SpyGlass DFT ADV product as an equivalent fault:

Gate	Condition
Buffer	<ul style="list-style-type: none"> Input s@0 is equivalent to output s@0 Input s@1 is equivalent to output s@1
Inverter	<ul style="list-style-type: none"> Input s@0 is equivalent to output s@1 Input s@1 is equivalent to output s@0
AND	Output s@0 is equivalent to any input s@0
NAND	Output s@1 is equivalent to any input s@0
OR	Output s@1 is equivalent to any input s@1

Key Concepts

Gate	Condition
NOR	Output s@0 is equivalent to any input s@1
Net between single output pin and single input pin	<ul style="list-style-type: none"> Output pin s@0 is equivalent to input pin s@0 Output pin s@1 is equivalent to input pin s@1

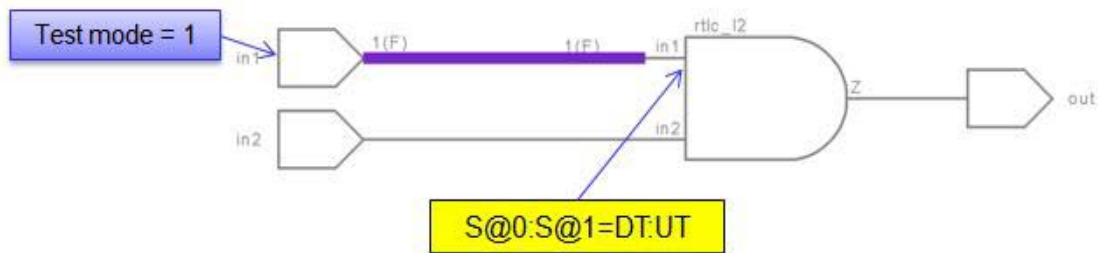
Un-Testable (UT)

A fault is considered untestable if the node under consideration has non-x simulation value in test mode. Therefore, an ATPG tool is unable to produce a test pattern to exercise this fault.

Un-Testable (UT) faults are reported as ATPG Untestable (AU) in ATPG.

NOTE: *The faults with non-X sim value in PG mode are classified as TI faults.*

Consider the following figure with the faults marked as UT:

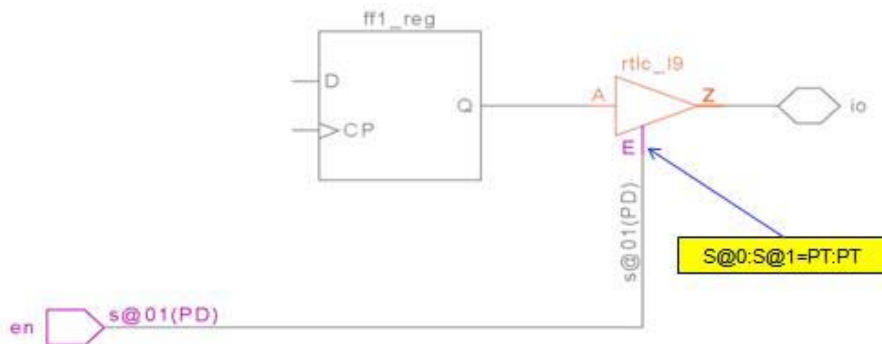


For the AND gate shown in the above figure, input, in1 is supplied with test mode value 1 in the capture mode. Therefore, S@1 faults at the in1 input of AND gate are marked as UT.

Potentially Detectable (PT)

A fault is considered potentially detectable, if a test on the fault site could produce an unknown (Z or X) value at the observable point. A fraction of potentially detectable faults can be considered as detectable by specifying the fraction with the ATPG_credit constraint.

Consider the following figure with the faults marked as PT:



In the above figure, both the faults at the enable pin of the Tri-state buffer are candidates of potential detectable fault.

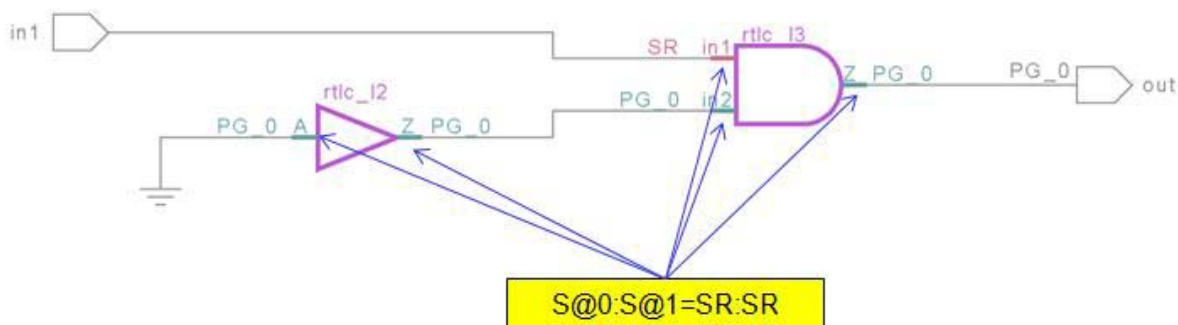
Synthesis Redundant (SR)

A fault is considered synthesis redundant, if the design is still at the RTL level and a node is either tied to a fixed value or is blocked because of the tied value.

Note that the following design elements are not marked as SR:

- Ports are never marked as SR as they are retained during synthesis
- Flip-flops, by default, are not marked as SR even if their D-pin has constant value in the power-ground mode.

Consider the design shown in the following figure where 10 faults over 5 nodes are identified as SR:



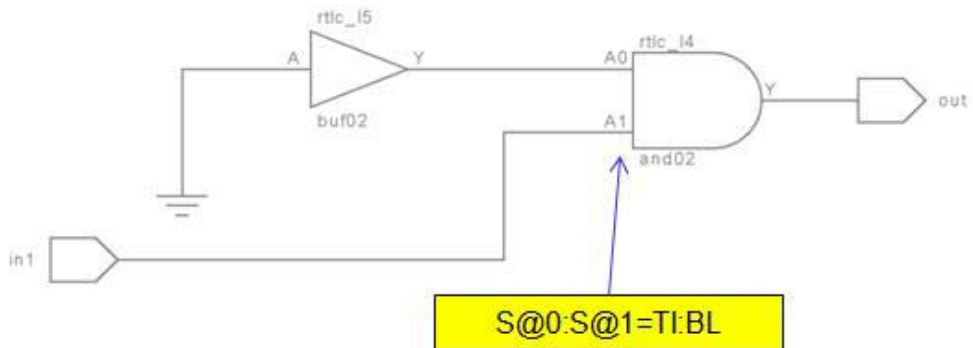
Blocked (BL)

A fault is considered blocked, if the node under consideration is blocked in power ground mode due to a net being tied to 0 (Low) or 1(High)

A fault is not marked as BL, if the design is still in the RTL stage, before synthesis. Instead, these faults are marked as SR. However, BL faults are marked in the netlist designs.

Presence of the BL fault in a design circuit does not affect the functionality of the circuit.

Consider the design in the following figure where faults are marked as BL:



```

1 module test (in1, out);
2 input  in1;
3 output out;
4 wire  w1, w2;
5
6     and02  rtlc_I4 (.A0(w1), .A1(in1), .Y(out));
7     buf02  rtlc_I5 (.A(1'b0), .Y(w1));
8 endmodule
9

```

In the above figure, the input of the AND gate is tied to 0. This makes the faults in the fan-in cone of the other input as BL.

Tied (TI)

A fault is considered tied, if the node under consideration has non-x

simulation value in the PG mode

A fault is not marked as TI if the design is still in the RTL stage, before synthesis. Instead, these faults are marked as SR. However, TI faults are marked in the netlist designs.

Presence of a TI fault in a design circuit does not affect the functionality of the circuit. That is, TI faults do not affect the test coverage.

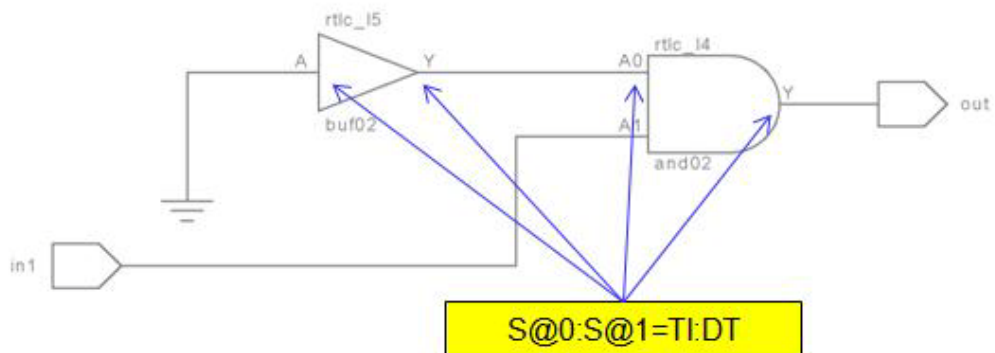
Consider the design in the following figure where faults are marked as TI:

```

1 module test (in1, out);
2 input  in1;
3 output out;
4 wire  w1, w2;

    and02  rtlc_I4 (.A0(w1), .A1(in1), .Y(out));
    buf02  rtlc_I5 (.A(1'b0), .Y(w1));
endmodule

```

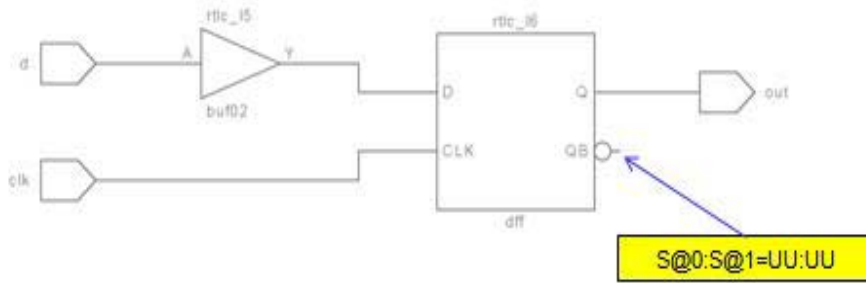


In the above figure, the input of the AND gate is tied to 0. This makes the S@0 faults in the input and output of the AND gate as TI.

Un-Used (UU)

A fault is considered unused if the node under consideration has no connection to an observable point. Consider the design in the following figure:

Key Concepts



In the above figure, faults at the QB pin of the flip flop are marked as UU.

Logical Redundant (LR)

A fault is considered logically redundant, if it is present in the fanin of a node whose value is constant, irrespective of the simulation values applied at its fanin.

Consider the design in the following figure where faults are marked as LR:



For the above design circuit, 6 faults over 5 nodes are identified as LR. To observe these faults with the Info_logicalRedundant rule, set the value of the dftDoLogicalRedundancyCheck parameter to on.

Support For Clock Shaper with Scannable Flip-Flops

Clock shapers, specified through the *clock_shaper* constraint, are treated as black box by SpyGlass DFT ADV and no rule checking is done for logic inside them. If a clock shaper contains scannable flip-flops, specify the

clock shaper pins that drive async and clock pins of flip-flops using the -scan_clock, -scan_set, and -scan_reset arguments of the *clock_shaper* constraint. Also, PLL / DIVIDER / CLOCK_SHAPER are identified using the following:

- clock_shaper -pll
- clock_shaper -divider
- clock_shaper

The following rules in the SpyGlass DFT ADV product support clock shaper with scannable flip-flops:

SpyGlass DFT ADV Product		
<i>Async_02_capture</i>	<i>Async_02_shift</i>	<i>Async_03</i>
<i>Async_05</i>	<i>Async_07</i>	<i>Async_08</i>
<i>Async_09</i>	<i>Async_13</i>	<i>Scan_24</i>
<i>Clock_02</i>	<i>Clock_04</i>	<i>Clock_08</i>
<i>Clock_11</i>	<i>Clock_11_capture</i>	<i>Clock_21</i>
<i>Clock_25</i>	<i>Topology_03</i>	<i>Topology_13</i>
<i>Topology_14</i>	<i>Conn_09</i>	<i>Diagnose_testclock</i>
<i>Info_forcedScan</i>	<i>Info_testclock</i>	<i>Info_DftDebugData</i>
<i>Info_noScan</i>	<i>Info_noFault</i>	<i>Info_addFault</i>
<i>Info_synthRedundant</i>	<i>Info_pwrGndSim</i>	<i>Info_untestable</i>
<i>Info_coverage</i>	<i>Coverage_audit</i>	<i>Info_testmode</i>
<i>Info_testmode</i>	<i>Atspeed_30</i>	<i>Clock_28</i>

Inserting RTL Testpoints

RTL testpoint insertion allows you to insert testpoints reported by the Info_random_resistance and TA_10 rules.

This section explains the following topics:

- Features
- Inputs
- Output
- Second Pass
- Default Testpoint Modules

Features

The following are the features of the testpoint insertion:

- Provides support for inferring testpoint clock
- Enables overriding clock connection, if required
- Provides capability to provide user-specified testpoint modules
 - You can specify custom testpoint module name using the following arguments of the rme_config constraint:
 - ◆ obs_tp_module_name
 - ◆ cnt_tp_module_name
 - ◆ cnt0_tp_module_name
 - ◆ cnt1_tp_module_name

NOTE: *The interface for custom testpoints should be the same as the one for the default testpoints.*

- Provides ability to disable individual test points in synthesis, if they impact timing. To disable an inserted testpoint, connect it's en pin to ground.

Inputs

Specify the following inputs to insert RTL testpoints:

- Mandatory inputs:
 - Set parameter `rme_active` to 1
 - Enable testpoint insertion for corresponding rule:
 - ◆ For `Info_random_resistance` testpoints, set the `dft_insert_rrf_tp` parameter to on
 - ◆ For `TA_10` testpoints, set the `dft_insert_ta_tp` parameter to on.
- Optional inputs
 - Set the value of the `dft_autofix_testmode_signal` parameter to name of the testmode signal. This signal can be an existing design node or a new port to be added by SpyGlass DFT ADV.
 - By default, the clock for newly inserted flops is inferred. To disable clock inference, set the `dft_infer_tp_clock` parameter to off. If clock inference is off or SpyGlass is unable to infer clock, then clock signal specified using the `dft_autofix_testclock_signal` parameter is used.
- The `Info_random_resistance` and `TA_10` rules generate an Info message for listing testpoints.
- Double-click on the violation message to open the spreadsheet. The following figure shows a sample spreadsheet:

	C	D	E	F	G	H	I
	Fix	Testpoint Node	Testpoint Type	Proposed Root Clock Net	Proposed Leaf Clock Net	Specified Root Clock Net	Specified Leaf Clock Net
1	Y	test_mod_i1_w3	Control	test clk2	test_mod_i1_clk2_w1	-	-
2	Y	test_mod_i1_w11	Control	test clk2	test_mod_i1_clk2_w1	-	-
3	Y	test_mod_i1_w2	Observe	test clk1	test_mod_i1_clk1_w1	-	-
4	Y	test_mod_i1_w10	Observe	test clk1	test_mod_i1_clk1_w1	-	-

FIGURE 18. TA_10 Spreadsheet Report

Output

After testpoint insertion, following source files are generated inside the `spyglass_reports/rme/dft/atrenta_modified_rtl_files` directory:

- `atrenta_generated_sources.f`

- <top_name>.f

You can use these files to test the syntax, functionality, and consistency of the modified RTL with respect to the original RTL

Second Pass

Use second pass if you want to modify testpoint clock connection or disable some testpoints.

Disable Testpoints

To **disable testpoints**, open the spreadsheet generated by the Info_random_resistance and TA_10 rules. Now, for the testpoints with Y in column C (Fix), which you do not want to be inserted, change the corresponding record to N and save the spreadsheet.

Change TestPoint Clock Connection

To **change the testpoint clock connection**, open the spreadsheet generated by the Info_random_resistance or TA_10 rules and specify:

- root clock net in column H (Specified Root Clock Net).
- leaf clock net in column I (Specified Leaf Clock Net).

Setup Process

The following steps explain the **setup process** in the second pass:

1. Save the spreadsheet after updating
2. Specify the path of directory containing the spyglass_report directory of the first pass as a value to the rme_selection parameter.
3. Set the value of the rme_active parameter to 1.
4. Ensure that no rule is enabled for the second pass.

NOTE: *Ensure that the files generated by SpyGlass during first pass are not overwritten using the following methods:*

- 📄 *Project working directory (projectwdir) for second pass run should be different from the projectwdir created during the first pass.*
- 📄 *You can edit and save the spreadsheet report. However, do not change the report location.*

Output

After testpoint insertion, following source files are generated inside the

spyglass_reports/rme/dft/atrenta_modified_rtl_files directory:

- atrenta_generated_sources.f
- <top_name>.f

You can use these files to test the syntax, functionality, and consistency of the modified RTL with respect to the original RTL.

If any clock connection was changed, the *Info_selective_test_point* rule generates an Info message for listing the testpoints.

Double-click the violation message to open the spreadsheet showing testpoints and their clock connection as shown in the following figure:

	C	D	E	F	G
	Fix	Testpoint Node	Testpoint Type	Root Clock Net	Leaf Clock Net
1	Y	test.op2	Observe	test.clk2	test.clk2_w1
2	Y	test.ip1	Control	test.clk1	test.clk1_w1
3	Y	test.w3	Observe	test.clk2	test.mod2_i2.clk_mod2_int
4	Y	test.w6	Control-1	atrenta_generated_port_tclk	atrenta_generated_port_tclk
5	Y	test.w4	Control-0	test.clk1	test.clk1_w1

FIGURE 19. Info_selective_testpoint report

Default TestPoint Modules

The following are the default testpoint module definitions:

```

module snps_obs_tp(clk, tp_i,en, tm, tp_o);
input clk, tp_i, en, tm;
output tp_o;
reg tp_o;
wire tm_i;
assign tm_i = en & tm;
always @ (posedge clk or negedge tm_i)

```

Key Concepts

```
begin
  if (~tm_i) tp_o <= 1'b0;
  else tp_o <= tp_i;
end
endmodule

module snps_cnt_tp(tp_o, tp_i,clk, tm,en);
  input tp_i, clk, tm, en;
  output tp_o;
  wire tm_i;
  assign tm_i = en & tm;
  reg tp_q;
  always @(posedge clk or negedge tm_i)
  begin
    if (~tm_i) tp_q <= 1'b0;
    else tp_q <= tp_i;
  end
  assign tp_o = (tm_i) ? tp_q : tp_i;
endmodule

module snps_cnt1_tp(clk, tp_i, en, tm,tp_o);
  input clk, tp_i, en, tm;
  output tp_o;
  wire tm_i;
  reg tp_q;
  assign tm_i = en & tm;
  assign tp_o = tp_i | tp_q;
endmodule
```

```
always @ (posedge clk or negedge tm_i)
begin
    if (~tm_i) tp_q <= 1'b0;
    else tp_q <= tp_q;
end
endmodule
```

```
module snps_cnt0_tp(clk, tp_i,en,tm,tp_o);
input clk, tp_i, tm, en;
output tp_o;
wire tm_i;
reg tp_q;
assign tm_i = en & tm;
assign tp_o = tp_i & tp_q;
always @ (posedge clk or negedge tm_i)
begin
    if (~tm_i) tp_q <= 1'b1;
    else tp_q <= tp_q;
end
endmodule
```

Impact of Different Path Types on Fanin/Fanout Cone Traversal

The `require_path` and `illegal_path` constraints support the following path types:

- *Topological*
- *Sensitizable*

Key Concepts

- *Sensitized*
- *Buffered*
- *Direct*

The following sections explain the impact of different path types on the fanin/fanout cone traversal. See [Example](#) section for an illustration of the same.

Topological

The following is the impact on the fanin/fanout cone traversal when the `path_type` is topological:

Non-X value on path	No impact. Traversal continues along such nodes.
Blocked path	No impact. Traversal continues along such nodes.
Presence of a black box	Stops the traversal. This is because the connection inside the black box is unknown.
Presence of a flip-flop or a latch	Traversal continues for as long as the sequential depth is within the user-specified limit.

Additionally, the following are the implications on the operating mode (`-tag` / `-use_shift` / `-use_capture` / `-use_atspeed`):

- Since the traversal is structural, it is not impacted by the specified mode.
- If a mode is specified, it is ignored when you specify path type as topological.

Sensitizable

The following is the impact on the fanin/fanout cone traversal when the `path_type` is sensitizable:

Non-X value on path	No impact. Traversal continues along such nodes.
Blocked path	Stops the traversal.
Presence of a black box	Stops the traversal.
Presence of a flip-flop or a latch	Traversal (sensitizable) continues for as long as the sequential depth is within the user-specified limit.

Additionally, blockage is caused by a constant (0 or 1) value, which is governed by the specified mode.

Sensitized

The following is the impact on the fanin/fanout cone traversal when the path_type is sensitized:

Non-X value on path	No impact. Traversal continues along such nodes.
Blocked path	Stops the traversal.
Presence of a black box	Stops the traversal.
Presence of a flip-flop or a latch	Stops the traversal.

Additionally, the following are the implications on the operating mode (-tag / -use_shift / -use_capture / -use_atspeed):

- Blockage is caused by a constant (0 or 1) value, which is governed by the specified mode.
- Enabler on a path is also a constant (0 or 1) value, which is governed by the specified mode.

Buffered

The following is the impact on the fanin/fanout cone traversal when the path_type is buffered:

Non-X value on path	No impact. Traversal continues along such nodes.
Blocked path	Stops the traversal.
Presence of a black box	Stops the traversal.
Presence of a flip-flop or a latch	Stops the traversal.
Presence of a gate other than a buffer and inverter	Stops the traversal.

Additionally, the following are the implications on the operating mode (-tag / -use_shift / -use_capture / -use_atspeed):

- Since the traversal is structural, it is not impacted by the specified mode.
- If a mode is specified, it is ignored when you specify path type as buffered.

Direct

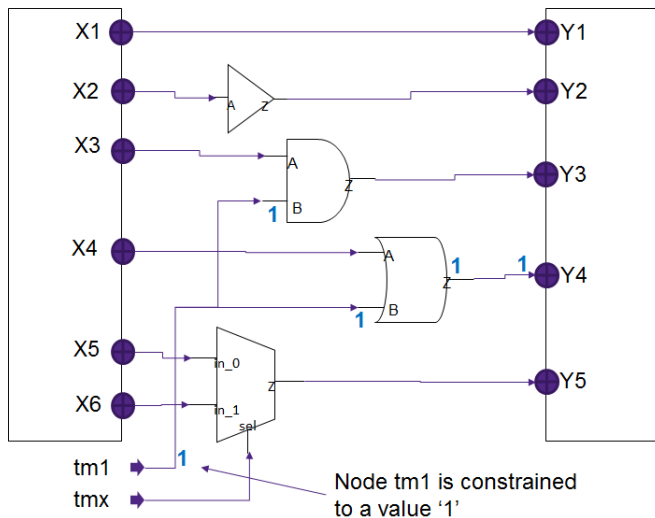
No traversal is involve when the path type is direct. Presence of any logic between the source and destination blocks the connection.

Additionally, the following are the implications on the operating mode (-tag / -use_shift / -use_capture / -use_atspeed):

- Since the traversal is structural, it is not impacted by the specified mode.
- If a mode is specified, it is ignored when you specify path type as direct.

Example

Consider the following sample schematic and the possibility of traversal when different path types are considered for the nodes in the schematic:



From Node	To Node	Path Types				
		topological	sensitizable	sensitized	buffered	direct
X1	Y1	Y	Y	Y	Y	Y
X2	Y2	Y	Y	Y	Y	N
X3	Y3	Y	Y	Y	N	N
X4	Y4	Y	N	N	N	N
X5	Y5	Y	Y	N	N	N
X6	Y5	Y	Y	N	N	N
tm1	Y3	Y	Y	N	N	N
tm1	Y4	Y	Y	N	N	N
tm1	Y5	N	N	N	N	N

FIGURE 20. Traversals for Different Path Types

Design Impact

This section explains the following topics:

- [Scannability](#)
- [Improvements to Fault and Test Coverage](#)
- [Detecting Structures Leading to Non-Robust Tests](#)

Scannability

SpyGlass defines a scannable flip-flop as a flip-flop that satisfies the following two requirements when the circuit state is in scanshift mode:

- The flip-flop's clock input pin is controlled by a testclock pin.
- The flip-flop's asynchronous set and reset pins are not used or are inactive in the scanshift mode.

How these requirements are checked depends on the available user-supplied constraints. In all cases, a simulation of power and ground and any available testmode (shift mode and capture mode) conditions is performed.

This section explains the following topics:

- [Scannability in Different Modes](#)
- [Scannability-Dependent Rules](#)

Scannability in Different Modes

Scannability is mode sensitive as follows:

- Default mode (assumes combinational ATPG tools will be used) sets scannable flip-flop's q-pins as controllable and d-pins as observable. Non-scannable flip-flop's q-pins are never allowed to become controllable in this mode and their inputs are never allowed to become observable.
- The seq_atpg mode uses the same scannability definition as the default mode.
- The noscan mode assumes all flip-flops are not scannable.

The SpyGlass DFT ADV rules that involve testability automatically change their behavior according to the amount of user-supplied information.

Scannability-Dependent Rules

A number of SpyGlass DFT ADV rules are adaptive with respect to scannability. As more SpyGlass DFT ADV structures are designed into the logic and described with the appropriate input, then increasing amounts of the circuit will become controllable and/or observable. The behavior of SpyGlass DFT ADV now tracks the intuitive expectation that as testability improves, fewer nodes are uncontrollable or unobservable.

Improvements to Fault and Test Coverage

Testability analysis has been upgraded to be a function of user-defined modes and the scannability determination. The intention is to more closely match the behavior of combinational and sequential ATPG tools especially for fault coverage estimates.

In all cases, testability analysis is initialized with all input pins set to controllable and all output pins set to observable. In addition, scannable flip-flops have their outputs set to controllable and their d-inputs set to observable.

This section explains the following methodologies:

- *Fault Coverage and Test Coverage*
- *Determine Coverage Percentage and Back-annotate Untestable Faults*
- *Determine the Number of Undetected Faults and Back-annotate Cause Information*
- *Testability Analysis*
- *Scan-Ready Design Conditions for Testability Rules*

Fault Coverage and Test Coverage

SpyGlass DFT ADV includes rules for quickly estimating fault coverage and test coverage. The intent is to provide estimates as close as possible to the results expected from either TetraMax or FastScan. Those tools decompose

complex functions such as adders into much smaller gates and then consider faults on the resulting gates. Therefore, the actual fault space will generally be greater than the space considered by SpyGlass DFT ADV.

If SpyGlass DFT ADV declares that the output of a 32-bit wide adder is not testable because of unobservability then we would conclude that all 64 output faults and all 128 input faults (2x32 inputs) cannot be detected for a total of 192 faults. The adder might be actually synthesized into many parts having a larger number of pins and therefore may have more faults than 192. However, both the numerator and the denominator for the ATPG results would move in the same direction so that coverage percent may still be close to the estimate from SpyGlass DFT ADV.

$$\text{FaultCovPercent} = \frac{\text{DT} + (\text{PT} \times \text{POSDETECT_credit})}{(2 \times \text{TC}) - \text{SyRd}} \times 100$$

$$\text{TestCovPercent} = \frac{\text{DT} + (\text{PT} \times \text{POSDETECT_credit})}{(2 \times \text{TC}) - \text{SyRd} - \text{TI} - \text{BL} - \text{UU} - (\text{ATPG_credit} \times \text{UT}) - \text{LR}} \times 100$$

The symbols used in the above formulae are described in the following table:

Symbol	Description
ATPG_credit	Value specified with the <i>ATPG_credit</i> rule parameter (default is 0)
BL	Total number of blocked faults
DT	Total number of detectable faults
LR	Total number of logically-redundant faults when the <i>dftDoLogicalRedundancyCheck</i> parameter (default is 0) is specified
PT	Number of potentially detectable faults
SyRd	Total number of synthesis-redundant faults (that is, 2 times the number of synthesis-redundant primary ports and terminals)
TC	Total number of primary ports and terminals
TI	Total number of tied faults

Symbol	Description
UT	Total number of untestable faults
UU	Total number of unused faults (that is, 2 times the number of primary ports and terminals that do not have a path to the primary outputs)

All these values are calculated and reported for each top-level design unit.

NOTE: *PT* fault or *potentially detectable faults* refer to the faults in which the difference between the good circuit result and bad circuit result produces any of the following detected faults:

Good Circuit Result	Bad Circuit Result
0	z
0	x
1	z
1	x

Determine Coverage Percentage and Back-annotate Untestable Faults

The [Info_untestable](#) rule lists an estimate of the fault coverage and test coverage. Double-clicking the message will back annotate all undetected faults onto the structural view.

The display format is as follows:

- 0 if the *pin stuck at 0* fault cannot be detected.
- 1 if the *pin stuck at 1* fault cannot be detected.
- 01 if both the *pin stuck at 0* fault and the *pin stuck at 1* fault cannot be detected.

Determine the Number of Undetected Faults and Back-annotate Cause Information

The [Info_undetectedCause](#) rule lists number of undetected faults. Double-clicking the message will back annotate controllability and observability

information for all undetected faults onto the structural view. This information provides further details regarding why a specific fault is not detected.

The display format is defined in the following table:

Controllable to 0	Controllable to 1	Observability of pin	Structural Display
n	n	n	nn-(n)
n	n	y	nn-(y)
n	y	n	ny-(n)
n	y	y	ny-(y)
y	n	n	yn-(n)
y	n	y	yn-(y)
y	y	n	yy-(n)

Testability Analysis

Testability analysis plays a central role in many SpyGlass DFT ADV rules. The analysis is sensitive to any user supplied test mode and test clock data and to the type of ATPG and scan environment planned. The default mode assumes that combinational ATPG tools will be used so that non-scannable registers and non-transparent latches will always be treated as not controllable even if their inputs are controllable.

If a sequential ATPG tool is planned, then use of the [seq_atpg](#) constraint allows non-scannable flip-flops and non-transparent latches to become controllable whenever their inputs are controllable. In this mode, scannable flip-flops will also have their data input pins declared as observable since, for ATPG purposes, values on those inputs can be captured and subsequently scanned out. The outputs of scannable flip-flops will be initialized to controllable to both a logic low and a logic high since arbitrary values can be scanned in. Controllability to high impedance will be maintained to "no."

The noscan mode causes only primary inputs to be initialized to yyy controllability and only primary outputs to be initialized as observable. All internal nodes will become controllable only if the inputs to those nodes are controllable with no regard to scannability. Similarly, internal nodes will only become observable only if they are observable with respect to primary

outputs without regard for scannability.

This data is summarized below.

- In `seq_atpg` mode, controllability analysis is allowed to compute good controllability for non-transparent latches and non-scannable flip-flops
- In `noscan` mode, scannability information is not used in testability analysis. When this mode is specified, testability analysis will initialize all internal nodes to `nnn`, which means not controllable to 0, not controllable to 1 and not controllable to z. Internal nodes can become controllable only if the direct inputs to these become appropriately controllable. Only primary outputs are seeded as observable.
- If a `scanwrap` constraint is specified on a module, then all ports on all instances of that module type are declared controllable and observable.

TABLE 1 Operating Modes vs. Process Steps

	(Default mode)	seq_atpg	noscan
SIMULATE STEP	Simulate power and ground and all available testmode conditions	Simulate power and ground and all available testmode conditions	Simulate power and ground and all available testmode conditions
INITIALIZE STEP	Any node with a non-x value will have its controllability set to ynn if sim value is LOW Controllability set to nyn if sim value is high Controllability set to nny if sim value is z	Any node with a non-x value will have its controllability set to ynn if sim value is LOW Controllability set to nyn if sim value is high Controllability set to nny if sim value is z	Any node with a non-x value will have its controllability set to ynn if sim value is LOW Controllability set to nyn if sim value is high Controllability set to nny if sim value is z

TABLE 1 Operating Modes vs. Process Steps (Continued)

	(Default mode)	seq_atpg	noscan
CONTROLLABILITY COMPUTATION STEP Note that a non-scannable device that has its output forced by testmode will still retain a C# according to the initialize step	In this mode, non-scannable flip-flops never become controllable or observable.	In this mode, all scannable flip-flops are set to yyn and d-pins set to obs. Non-scannable flip-flops may become controllable if their inputs are sufficiently controllable. Their inputs may become observable, if sufficient controllability exists.	Equivalent to the original algorithm where device outputs are controllable only if the inputs are sufficiently controllable and device inputs are obs if the output is obs and sufficient input controllability exists.
Primary Inputs	Controllability set to yyy	Controllability set to yyy	Controllability set to yyy
Primary Outputs	obs set to y	obs set to y	obs set to y
scannable flip-flop	Ctrl(q) = yyn Obs of all inputs are "y"	Ctrl(q) = yyn Obs of all inputs are "y"	If default mode, then q is not nnn and all inputs are not obs. If seq_atpg mode, then q is controllable according to input controllability and inputs are obs according to input controllability and if q is obs

TABLE 1 Operating Modes vs. Process Steps (Continued)

	(Default mode)	seq_atpg	noscan
Non-scannable flip-flop	q's remain nnn unless the asynchronous set and reset pins can be used to force the controllability of the q-pin. All inputs are not obs.	q is controllable according to input controllability & inputs are obs according to input controllability and if q is obs	If default mode, then q is not nnn and all inputs are not obs. If seq_atpg mode, then q is controllable according to input controllability & inputs are obs according to input controllability and if q is obs
Retiming flip-flops	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability
Transparent latches (enables active in the shift mode)	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability
Latches with active enable when testclocks are off	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability
Retiming latches	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability	q controllability equals d controllability; d observability equals q observability

TABLE 1 Operating Modes vs. Process Steps (Continued)

	(Default mode)	seq_atpg	noscan
All other latch configurations	q's remain nnn d's not obs	q is controllable according to input controllability d is obs only if q is obs and clock is -y-	noscan has no effect on latches.
Black box outputs	Controllability set to yyy if module named in scanwrap, otherwise nnn	Controllability set to yyy if module named in scanwrap, otherwise nnn	Controllability set to nnn
Black box inputs	obs if module named in scanwrap, otherwise not obs	obs if module named in scanwrap, otherwise not obs	not obs

Scan-Ready Design Conditions for Testability Rules

The steps to make/ a design scan-ready are as follows:

1. All flip-flops are scannable (unless forced/inferred noscan)
2. All latches are transparent (unless used for retiming)
3. All black boxes are scanwrapped (unless bypassed using the [module_bypass](#) constraint)

The design may not be scan-ready before running the [TA_09](#) rule. This may lead to an increased run time and also a large number of test-insertions points may be reported. To avoid this, you can set `dftTreatFlipFlopsAsScan`, `dftTreatLatchesAsTransparent`, and `dftTreatBBoxAsScanwrapped` parameters, which tells SpyGlass to make the design full-scan-ready (by default) by executing the following steps:

1. Make the output of unscannable flip-flop (unless forced/inferred noscan) controllable

2. Make the input of unscannable flip-flop (unless forced/inferred noscan) observable
3. Make the output of non-transparent latch (unless retiming) controllable
4. Make the input of non-transparent latch (unless retiming) observable
5. Make the output pins of all black box (unless bypassed using the `module_bypass` constraint or scanwrapped) controllable
6. Make the input pins of all black box (unless bypassed using the `module_bypass` constraint or scanwrapped) observable

Once the design is made scan-ready, run the TA_09 rules under "testmode + scan-ready" conditions.

Detecting Structures Leading to Non-Robust Tests

Non-robust tests may create a false sense of security caused by a high coverage result from ATPG. This high percentage coverage result does not guarantee good results on the test machine, that is, it may result in failing good chips and passing bad chips.

The following rules in the SpyGlass DFT ADV product check the design robustness:

Async_02_capture	Async_11	Clock_04	Clock_08
Clock_16	Clock_17	Clock_21	Clock_27
Clock_28	Scan_07	Scan_22	Topology_03
Topology_05	Topology_13	Tristate_06	

Identifying Test Points To Reduce Random Resistance

Identifying test points to reduce random resistance improves the probability of fault detection.

The [Info_random_resistance](#) rule of the SpyGlass DFT ADV product enables you to identify design locations at which inserting a test-point will decrease the random resistance of the design for test. The rule suggests test-points based on the probability of fault detection at a particular design node by estimating the improvement in probability of fault detection in its vicinity.

Exclusions to RRF Test Point Selection

The following design nodes are not considered as potential candidate for RRF test point analysis:

- Undriven nets
- Clock nets in the Shift or Capture modes
- Nets inside a CGC
- Nets in Clock fan-in cone
- Nets with a non-x value
- Nets inside a ``celldefine` module
- Generated nets (this is controlled by the `dft_rrf_tp_ignore_generated_nets`)
- Nets specified inside the module/instance on which the `no_test_point` constraint is applied
- Nodes with `stuck@0` and `stuck@1` faults
- Nets having `force_probability` constraint defined
- Nets directly driven by scan cells

Algorithm

The following algorithm is used by the [Info_random_resistance](#) rule to identify test points to reduce random resistance:

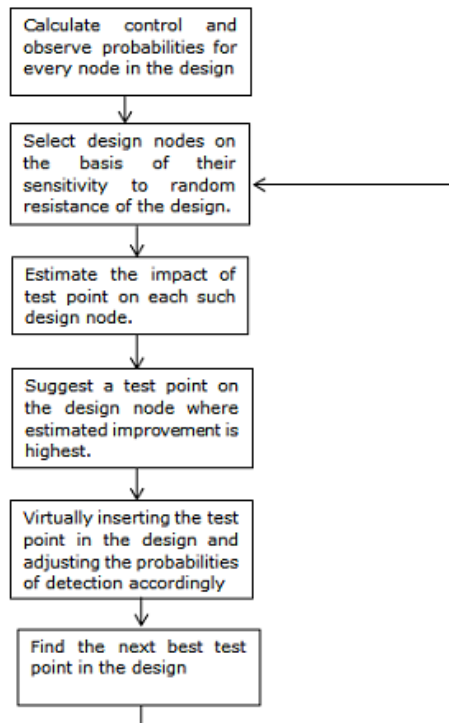


FIGURE 21. Identifying TestPoints to Reduce Random Resistance

Reports Generated

The following reports are generated by the v rule in order to identify test points to reduce random resistance of the design:

- *random_pattern_testpoints*: This report lists all the test points suggested along with approximate improvement in the random pattern coverage of the design. The syntax of this report is SGDC compatible. Therefore, you can also use this report as an SGDC input to recover or continue from the earlier run.
- *random_pattern_dc_testpoints*: This report lists test points suggested by SpyGlass. The syntax of this report is Design Compiler compatible. Therefore, you can specify this report as an input to DC for test point insertion.

Using AutoFix and Selective AutoFix

The AutoFix feature enables you to automatically fix the violation by modifying the RTL. The feature is supported by the [Info_random_resistance](#) rule.

Enabling the AutoFix feature generates an Info message and a spreadsheet, which contains the AutoFix summary. Following figure shows a sample spreadsheet generated for the AutoFix feature:

	C Fix	D Testpoint Node	E Testpoint Node Type
1	Y	sample_block.rtlc_l25.in1	Control
2	Y	sample_block.block_c.pt[127].Z	Observe
3	Y	sample_block.block_b.comp0_inst.and11.rtlc_l2.in2	Control
4	Y	sample_block.block_b.comp0_inst.and11.b.Z	Observe
5	Y	sample_block.block_b.comp0_inst.and5.rtlc_l2.in2	Control
6	Y	sample_block.block_b.comp0_inst.and5.b.Z	Observe

FIGURE 22. Selective AutoFix Spreadsheet

In the above spreadsheet, Column C, named Fix, shows whether SpyGlass DFT ADV can fix a specific condition. The possible values are Y or Disabled. In the case of complex RTL descriptions, where AutoFix cannot modify the RTL, this field is specified as Disabled. In all other cases, it is taken as Y, meaning yes.

All the cases which can be fixed by SpyGlass are fixed in the current run

and corresponding files are available at the following location:

```
<dir>/spyglass_reports/rme/dft/
```

NOTE: *Currently, the AutoFix feature is available only for the rules specified above.*

You may not want to fix all of the possible case. To do so, you can select the cases, which you want to ignore while modifying RTL using Autofix.

Flow of the Selective Autofix Feature

The Selective AutoFix feature requires two runs of SpyGlass. The first run generates the autofix data in a spreadsheet, where you can select the DFT opportunities you want to fix. In the second run, DFT AutoFix feature fixes the selected opportunities.

To run selective AutoFix, perform the following steps:

1. Enable the AutoFix feature by selecting the *dftAutoFix* parameter *rme_active* parameter.
2. The above specified rules generate a violation message about the opportunities available for fixing by the AutoFix feature.
3. Double-click the violation to open the spreadsheet, which lists the possible opportunities that you can selectively fix.

For the opportunities marked as Y in column C (Fix), which you do not want to be fixed by the AutoFix feature, change the corresponding record to N and save the spreadsheet.

All the opportunities marked as Y in the spreadsheet will be fixed by the Selective AutoFix feature in the next step.

NOTE: *The spreadsheet currently does not run any sanity check. If the Column C is set to a value other than Y, N, or Disabled, unexpected results may occur.*

4. Run SpyGlass again, by specifying the following command in the project file:

```
set_parameter rme_selection <dir_path>
```

Here, <dir_path> signifies the directory path to the goal results.

The *rme_selection* parameter should be set to the path of the previous run where the spreadsheet is saved. The information of the power reduction opportunities to be modified is picked up from the spreadsheets saved in this directory.

This run creates the modified RTL at the following path:

<wdir-path of Step 3 run>/spyglass_reports/rme/dft>

Files Generated

At the end of the Selective AutoFix run, the following source files are generated inside the spyglass_reports/rme/dft/atrenta_modified_rtl_files directory:

- atrenta_generated_sources.f
- <top_name>.f

These files can be used to test the syntax, functionality, and consistency of the modified RTL with respect to the original RTL.

Suggested SpyGlass DFT ADV Operation

The process to create RTL depends on a variety of elements including the nature of the functions to be implemented, the available tools and corporate standards and products. This section assumes an understanding of scan design and outlines a typical method for using SpyGlass DFT ADV.

The recommended method of running SpyGlass DFT ADV rules is to partition the rules into stages and to run each stage separately.

One approach is to partition the rules into categories called “goals.” Refer to the *Atrenta Console User Guide* for description of the SpyGlass goal files.

SpyGlass DFT ADV includes a set of standard goals that are organized for the following steps:

TABLE 2 SpyGlass DFT ADV Goals

Order	Goal Name	Comment
1	dft_setup	Checks for presence of pre-requisites (valid constraints)
2	dft_best_practices	Ensures compliance to DFT Best Practices
3	dft_scan_ready	Make registers scannable
4	dft_latches	Make latches transparent
5	dft_test_points	Improve controllability and observability
6	dft_scan_chain	Performs scan chain related checks
7	dft_block_check	Checks unit-level test requirements at the block level

Each goal has specific rules useful for that stage. A number of the rules automatically adapt to available test logic if appropriate test constraint information is available.

See the *SpyGlass Methodologies User Guide* for more details of the SpyGlass DFT ADV goals.

Suggested SpyGlass DFT ADV Operation

This section describes the following topics:

- [Using the Design Constraints](#)
- [Making the RTL scan ready](#)
- [Complying with the SpyGlass DFT ADV Best Practices](#)
- [Adding Test Points](#)
- [Working with Scan Chains](#)
- [Checking Block-level Test Requirements](#)
- [Making the RTL Ready for Atspeed Test](#)

Using the Design Constraints

SpyGlass DFT ADV Design Constraints play a critically important role in DFT rule-checking and analysis including scannability and fault coverage estimation. This section describes the following topics:

- [Reporting Missing Constraints](#)
- [Working with Black Boxes](#)
- [Checking for Presence of Valid Constraints](#)

Reporting Missing Constraints

SpyGlass DFT ADV product also generates missing SpyGlass Design Constraints information for the following rules:

Async_01	Async_07	Async_08	Async_09	Clock_10
Clock_11	Latch_01		RAM_05	Scan_16
TA_01	TA_02	TA_04	Tristate_03	Tristate_04
Tristate_07	Tristate_08			

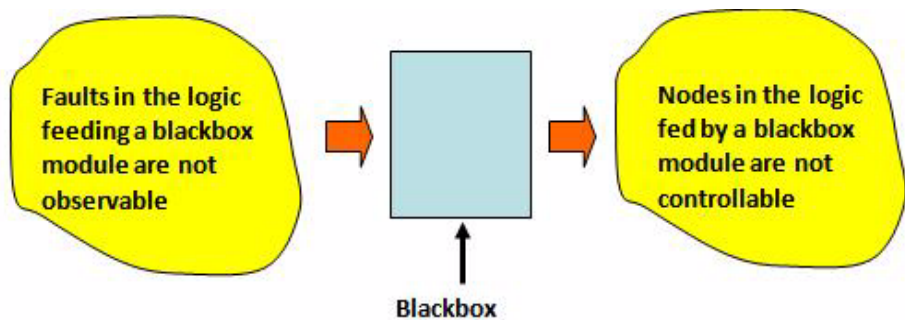
NOTE: *The TA_01 and TA_02 rules do not generate dftCntrlTestPt.sgdc and dftObsTestPt.sgdc files respectively.*

Working with Black Boxes

If you are analyzing a Verilog design that contains black box instances, without any associated RTL description in either the source files or .lib files, you may get false messages for some SpyGlass DFT ADV rules. The reason for these false messages is that all ports of black boxes are assumed to be inout ports.

Use the SpyGlass `set_option inferblackbox yes` command to avoid these false rule messages. This feature heuristically determines the port directions based on their connectivity. Refer to the *Atrenta Console Reference Guide* for details of using the `set_option inferblackbox yes` command.

The black box inputs are unobservable and black box outputs are uncontrollable as shown in the following figure:



Above situation negatively impacts the coverage. Therefore, the designer should try to eliminate the black boxes or try to find a workaround for them.

You can use the following ways to fix black boxes in the design:

- [Use the scan_wrap constraint](#)
- [Use the module_bypass constraint](#)

Checking for Presence of Valid Constraints

The `dft_setup` goal includes the following rules that ensure that valid constraints are present:

- The *Diagnose_testmode* rule displays the devices that block testmode signal propagation in shift and capture modes. If a `test_mode` constraint is present then the combinational instances that block testmode signals in shift and capture modes are displayed.
- The *Diagnose_testclock* rule displays the devices that block testclock signal propagation. Testmode simulation is performed on any available testmode constraints.
- The *Info_testmode* rule displays testmode simulation results.
- The *Info_testclock* rule displays the propagation of each testclock through the design in shift, capture and runflow mode.
- The *Scan_23* rule checks for the bypass conditions.
- The *dumpBlackBox* rule provides a list of all possible 'scanwrap' constraints for a given top module. The generated file can then be passed in the second run along with any other constraint file.

Making the RTL scan ready

SpyGlass DFT ADV may be run on RTL designs without any built-in test logic or on designs with test mode controls and test clocks. In the first case, SpyGlass will identify the system logic that does not comply with SpyGlass DFT ADV requirements. In the second case, SpyGlass DFT ADV will verify that the design-for-test logic is performing correctly as well as identify where additional design-for-test logic may be required or useful.

This “scan ready” step is designed to ensure that as many registers in the RTL as possible can easily be replaced with scan equivalents either during logic synthesis or in a post-synthesis step. SpyGlass DFT ADV does not do the scan insertion or scan chain stitching but rather deals with the clocks and the asynchronous signals to ensure compliance with scan replacement requirements.

There are three primary tasks for scannability:

1. Test clock control for all registers

SpyGlass has two rules, *Clock_10* and *Clock_11* that are both designed for ensuring that registers are properly controlled by a test clock. *Clock_11* checks that each internal clock is by-passed with a testclock. *Clock_10* checks that each internal clock is by-passed with a dedicated testclock. The `dft_scan_ready` goal uses the simpler *Clock_11* rule. Either rule will identify registers whose clocks are not under test clock control.

In many situations, these rules will highlight the RTL statement where an internal clock that is not by-passed is defined and display, in the structural view, how test clocks and test mode signals are currently operating. The designer's task is to modify the identified internal clock, using test mode signals and a test clock, to force the original clock to be controlled by a test clock in test mode. If necessary, root level test pins may have to be added to the design and routed through the hierarchy to the current module. Appropriate additions should also be made to the constraint file for this design.

The *Info_testclock* rule may be useful if this design already has some SpyGlass DFT ADV controls and test clocks designed in. The rule shows how these controls and clocks function that may facilitate modifications to the current clock. The structural view is back-annotated for nets with a testclock pulse and all nets with fixed values. For each displayed net, the value for that net will be back annotated at the source and at all sinks as follows:

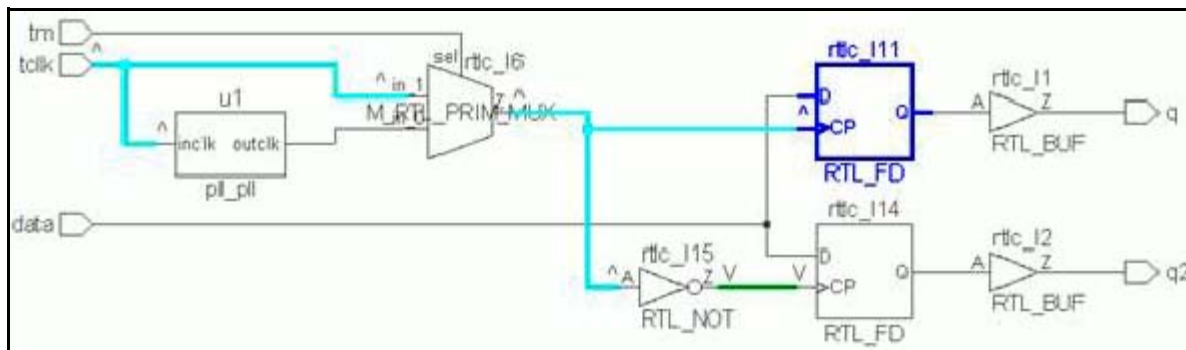
if simulation result = 1 then display "1" on the net

if simulation result = 0 then display "0" on the net

if simulation result is a positive pulse then display " ^ " on the net

if simulation result is a negative pulse then display "V" on the net

The following figure shows an example of such a displayed net:



In the above figure, constraints have designated `tm` as a testmode signal with value 1 and `tc1k` is a testclock. The pll instantiated at `u1` is a black box.

2. Disable all sets and resets in test mode

The [Async_07](#) rule is designed for this task and operates by simulating power, ground and any available test mode conditions and then reports the asynchronous sources that are active. This rule differentiates active high sets and resets from active low sets and resets driven by the same source. In such a case, only the active phase will be reported.

The [Info_testmode](#) displays all signals controlled in the scanshift mode and the capture mode and may be useful in diagnosing the Async_07 rule messages.

3. All registers should be scannable

The [Scan_08](#) rule may be used as a final check that as many registers as possible are scannable.

SpyGlass has two rules, [Scan_08](#) and [Scan_16](#), that are both designed for ensuring that registers are scannable. The Scan_08 rule checks that each register is scannable and the Scan_16 rule checks that each register is scannable or used for retiming. The `dft_scan_ready` goal uses the simpler Scan_08 rule. Either rule may be used.

If there are no [Clock_11](#) (or [Clock_10](#)) and no [Async_07](#) rule messages, then all registers should be scannable. An alternative approach is to use the Scan_08 rule since that requires both clock and asynchronous conditions. We suggest use of Clock 11 and Async_07 rules because often clocks and asynchronous signals are common to multiple registers so therefore it is generally more efficient to fix the common problems rather than focusing on the individual registers. However, in some cases, timing or other considerations may prevent making changes to the common clock source or the common asynchronous source so that clock and asynchronous violations remain. In that case, the Scan_08 rule messages list the individual registers.

Identifying clocks

Before starting to design-in SpyGlass DFT ADV, no test clocks have been defined. Since system clocks are or should be known, they are relatively easy to specify in a SpyGlass Design Constraints file and will be used to do initial checks by SpyGlass DFT ADV including scannability.

Use of testclocks

As SpyGlass DFT ADV logic is added to the design, then the test clock constraints (clock constraints with the `-testclock` argument) should be added as well. These constraints may have the same pin names as system clock constraints (clock constraints without the `-testclock` argument). In that case, the clock constraints are ignored but do not need to be removed.

Use of testmode

Testmode signals are signals used for both clocking as well as asynchronous functions. Clocking functions include controls for by-pass or clock enabling logic. Such controls remain active for both scan operations as well as for capture operations. Therefore, they remain in their active state throughout all testing. This use of testmode requires the syntax:

```
test_mode -name <pin-name> -value 0 | 1
```

where `<pin-name>` is a top level pin and with active value as specified.

The asynchronous use of testmode includes signals to render flip-flop sets and resets inactive during scan. When scan is complete, such signals may then be set to arbitrary values as required to achieve desired fault coverage. This use of testmode requires the following syntax:

```
test_mode -name <pin-name> -value 0 | 1 -scanshift
```

Complying with the SpyGlass DFT ADV Best Practices

Numerous rules are included with SpyGlass DFT ADV that may or may not apply to every application. The rules in the [Table 2, "SpyGlass DFT ADV Goals"](#) are suggested as possible candidates for ensuring compliance with best practices design standards. These rules may also be augmented with various other rules included with SpyGlass. Please see the *Atrenta Console User Guide* for instructions for creating and modifying goal files.

This goal may be skipped if desired. However, if rigorous test design guidelines exist then it may make sense to create a goal of basic rules with a focus on test issues other than the specifics of scan insertion and ATPG. Such a step helps to establish a design framework on which the more scan related rules operate, focuses the design activity and better manages the violation report. This section explains the following topics:

- [Make Latches Transparent](#)
- [Make The Design Robustness Ready](#)

Make Latches Transparent

The [Latch_08](#) rule is designed for ensuring that latches are transparent in capture modes. The rule checks that each latch is transparent when capture mode conditions are simulated with testclocks at their “return to” state.

The [Info_testmode](#) rule displays how the test mode signals, if any, propagate starting at the top-level of the design. Drill down into the hierarchal path toward the module containing the rule-violating latch. If at any stage, test mode is lost, then that is where some design rework is necessary to properly pass test mode to the rule-violating latch.

Make The Design Robustness Ready

Refer to the [Detecting Structures Leading to Non-Robust Tests](#) section.

Adding Test Points

The identified signals are the places where test points should be added. One way to add an observation test point is to create a dummy module (a black box) as follows:

```
module dummy_bb (testpoint, testclock, sout);
  input in, testclock;
  output sout;
endmodule
```

Add the following line to the constraint file:

```
scanwrap -name dummy_bb
```

For each signal listed as unobservable signal `<yy>`, instantiate a `dummy_bb` module:

```
dummy_bb dummy_bb1 (<yy>, <clock>, <soutxx>);
```

where `<clock>` is any available clock signal that is controlled by a

testclock and `<coutxx>` is a unique but arbitrary name. The `dummy_bb` module will create what will appear as a system flip-flop but in fact is only used to provide observability to signal `<yy>`.

The `scanwrap <module-name>` constraint directs SpyGlass DFT ADV to consider the inputs to `<module-name>` as observable and the outputs to `<module-name>` as controllable. Later, the `dummy_bb` can be replaced with a more complete module that eventually will be replaced with a scannable flip-flop and stitched onto a scan chain. This method allows the designer to quickly add an observation test point and to see the effect on fault coverage.

NOTE: *You can also specify a library cell with the `scanwrap` constraint. Such library cells are not black boxes but at the same time, are unknown to SpyGlass DFT ADV's controllability and observability analysis. Therefore, any analysis on them gives the same result as it would for a non-scanwrapped black box. This usually means that fault/test coverage estimates can be improved. You should therefore be able to make such cells 'scan-wrapped'.*

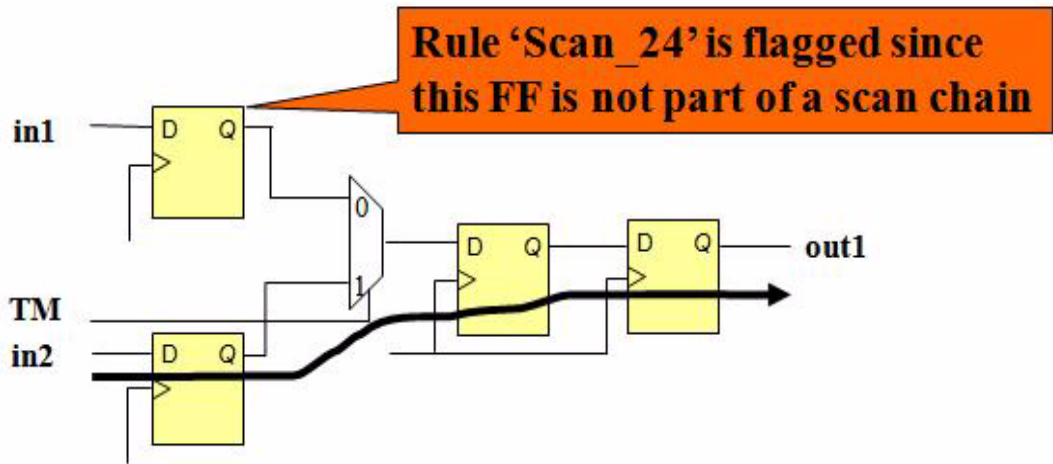
Working with Scan Chains

The `dft_scan_chain` goal includes scan chain related rules. Scan chains must have lockup latches at domain crossings and at chain end. All flip-flops should be part of some scan chain. Scan chains must not contain inverters in scan path.

For a list of scan rules, see [Scan Rules](#).

Consider the following figure:

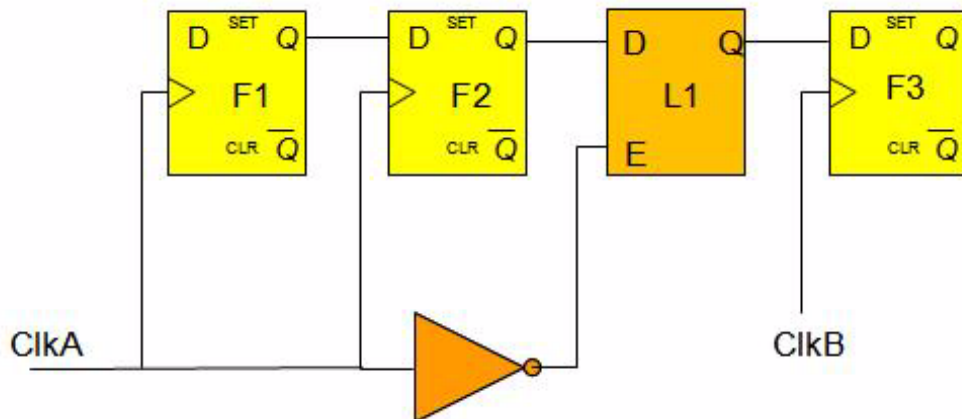
Suggested SpyGlass DFT ADV Operation



In the above example, an attempt is made to create a scan chain between `in1` and `in2`. However, due to the `test_mode` constraint, the output of the flip-flop connected to `in1` is not selected through the multiplexer. This results in a `Scan_24` rule violation because this flip-flop is not part of any scan chain.

In case of a scan chain with multiple clocks, data latching across domains is dependent on the arrival time of the two clock edges. A lockup latch at the place of domain crossing ensures reliable shifting because the data in the lockup latch retains the data in the prior scan flip-flop so that proper shifting occurs.

Consider the following figure:



In the above example, double-shifting can occur if one of the following conditions is true:

- If ClkB rises before ClkA, then old F2 => F3 and then old F1 => F2
- If ClkB rises after ClkA then old F1 => F2 and then old F1 => F3

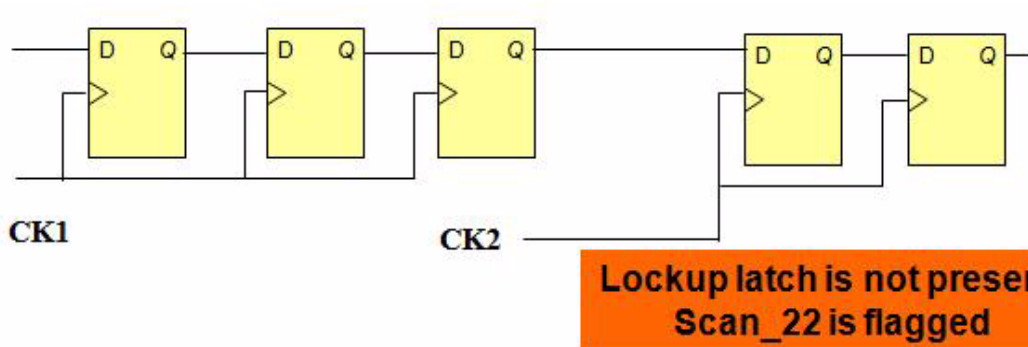
To ensure proper data transfer, a lock up latch is inserted. The latch enable, in this case, is driven by inverted ClkA.

Lockup latch shift or double shifting cannot occur, if one of the following conditions is true:

- If ClkB rises before ClkA, then old F2 => F3 and then old F1 => F2
- L1 retains old F2 and if ClkB rises after ClkA then old F1 => F2 and then L1 (old F2) => F3

If a scan chain does not have a lockup latch in the place where the data is crossing from one clock domain to another, the [Scan_22](#) rule violation is reported as shown in the following figure:

Suggested SpyGlass DFT ADV Operation



Checking Block-level Test Requirements

The `dft_block_check` goal checks unit level test requirements at the block level. The goal includes the following rules:

The [Soc_05](#) rule verifies `test_mode` and `testclock` constraints specified at lower level module.

The [Info_testmode](#) rule displays the testmode simulation results.

Making the RTL Ready for Atspeed Test

This section explains the following topics

- [Testmode Considerations for At-speed Testing](#)
- [At-Speed Capture Clock Discussion](#)
- [At-Speed \(Transition Delay Fault\) Coverage Definitions](#)
- [Process Flow](#)

Testmode Considerations for At-speed Testing

High coverage at-speed tests require as many flip-flops as possible be made scannable. This requirement poses some extra considerations for the

rule `Atspeed_01`. Refer to the circuit in Figure 6.

If no constraints have been supplied and the SpyGlass DFT ADV "Scan Ready" is run then a `Clock_11` violation is reported on the OR gate. As a result, the flip-flop will not be declared scannable. If `Atspeed_01` is selected in the same run as Scan Ready, there will not be an `Atspeed_01` violation since it only flags cases where a scannable flip-flop is not clocked by a PLL clock.

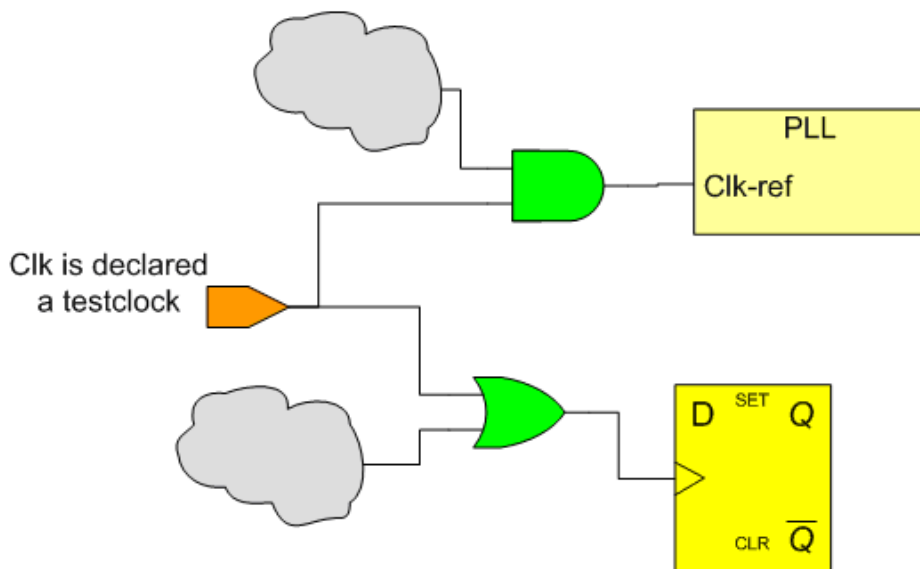


FIGURE 23. Clock_11 Violation

The circuit in Figure 7 has been modified so the `Clock_11` violation is removed and the flip-flop is declared scannable. The `Atspeed_01` rule will have a violation because the flip-flop is scannable but its clock is not derived from a PLL clock.

Suggested SpyGlass DFT ADV Operation

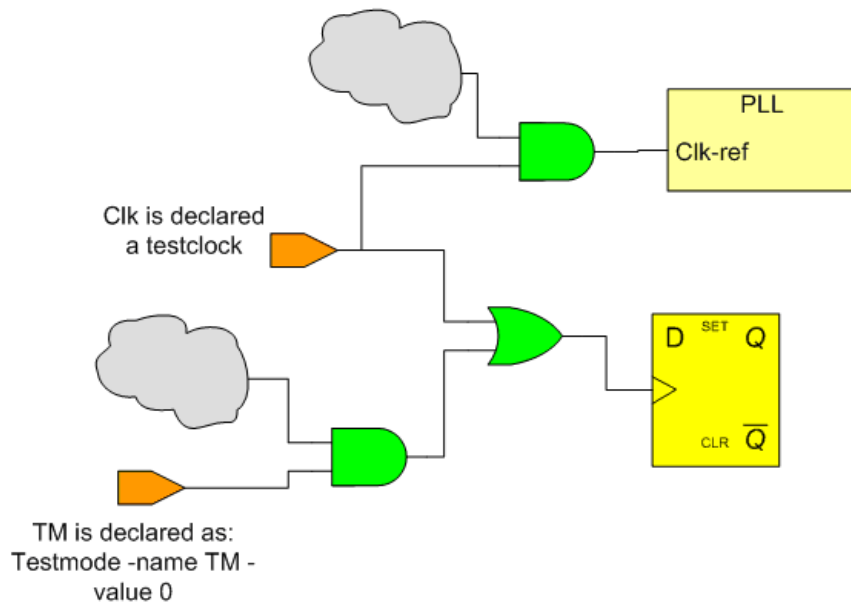


FIGURE 24. Clock_11 Fix

You can use the following modes with the `testmode` constraint:

- `-scanshift`
Use this option for scannability determination
- `-invertInCapture`
Use this option for scannability determination
- `-capture`
- `-captureATspeed`
This is optional
- No option specified
Use this option for scannability and stuck-at capture. It will only be used for at-speed if no `-at-speed` option is present

NOTE: If the `-captureATspeed` option is not specified for the at-speed rules, testmode with `-capture` condition will be assumed for those rules. This includes `testmode -capture`, `testmode -invertInCapture` and

testmode with no qualifier. However, if the testmode -captureATspeed is specified, all the at-speed rules will use testmode -captureATspeed mode. This also includes the testmode with no qualifier.

If Sig1 is declared as an at-speed testmode then the circuit in the figure below will pass the Atspeed_01.

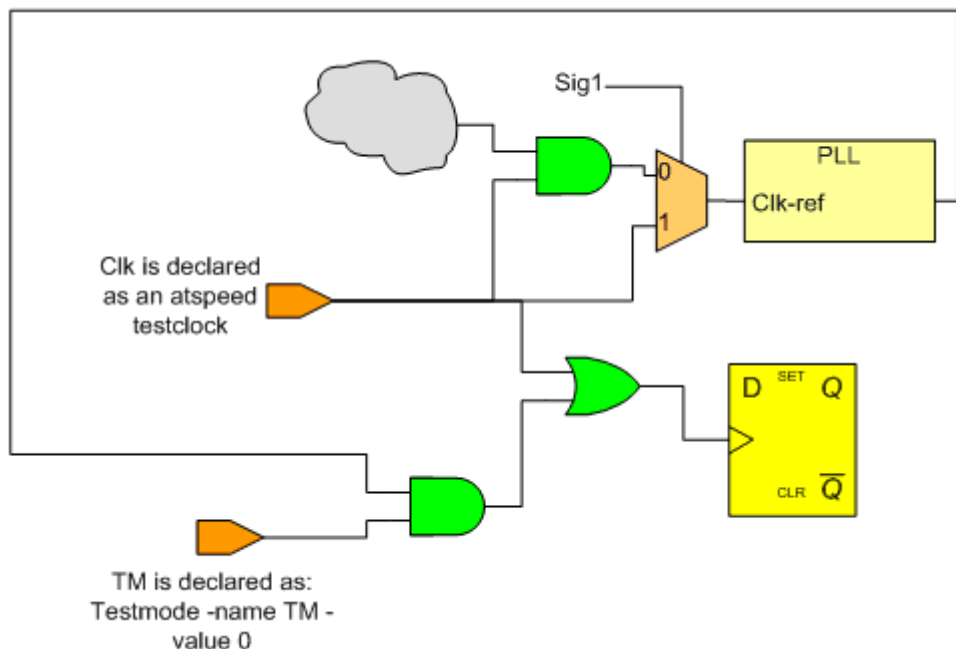


FIGURE 25. Scan FF Clocked by PLL

At-Speed Capture Clock Discussion

Generation of At-speed clocks

Two methods are typically used to generate at-speed test clocks.

■ ATE Generated Clocks

In this method, the at-speed test clocks are generated by the test machine. Therefore, this method is similar to the current SpyGlass DFT ADV (stuck-at test) behavior where PLLs are by-passed so that scan

shifting and test capture can be performed by the test machine.

The problem with this method is that only expensive test machines are capable of generating the precision, high speed clocks necessary for at-speed testing. Moreover if a design has large number of asynchronous system clocks, it may not be practical to generate the at-speed test clocks from the test equipment.

■ On-Chip Clock Generation

In this method, on-chip PLLs are designed with the necessary clock rates and controls to function as the at-speed test clocks. The advantages are:

- Avoids expensive test machines
- The functional clock distribution can be used for the at-speed test clock

At-speed test clock domain

At-speed test clocks must be defined by a clock constraint using the `-at_speed` option. All flip-flops clocked by such clocks will be defined as an at-speed test clock domain.

For example, as shown in the Figure 1, if C1, C2, C3 & C4 are declared as 4 test clocks for at-speed then all flip-flops in D_i are in the domain of C_i for $i = 1$ to 4 so that 4 domains are created.

On the other hand, if the `ref_clock` input to the PLL is declared as an at-speed test clock then all the flip-flops in D1, D2 and D3 will be in the same domain.

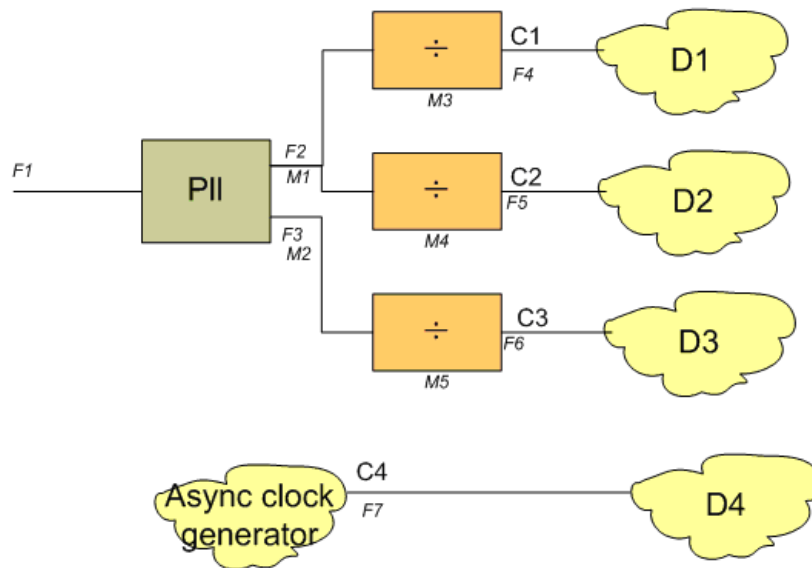


FIGURE 26. On-chip Clock Generation Example

At-Speed (Transition Delay Fault) Coverage Definitions

■ Transition Delay Fault

A slow to rise or slow to fall condition on a device pin that is not SR where SR are pins that will be deleted by logic synthesis is an at-speed transition fault. For example floating pins on combinatorial gates that do not influence any output function will be classified as SR.

■ Number of Transition Delay Faults

TC (terminal count) is defined as the number of device pins minus the number of SR pins.

The total fault space for transition faults is $2 \times TC$

■ Transition Delay Fault Launch Point

An at-speed transition launch point is a place where transition tests can be triggered and are of two types

- Scannable flip-flop q-pin

Suggested SpyGlass DFT ADV Operation

- ❑ Primary inputs can be used as launch points if the `dsmUsePIForAtSpeed` parameter is set to on
- Transition Delay Fault Capture Point

An at speed transition capture point is a location where transition test results can be captured (for subsequent interrogation by a test machine) and are of two types:

 - ❑ Scannable flip-flop d-pin
 - ❑ Primary outputs can be used as capture points if the `dsmUsePOForAtSpeed` parameter is set to on
- Transition Delay Fault Test Example

A typical example of a transition delay test is illustrated in the figure below. If all the flip-flops are scannable, then all flip-flops are launch points. U3 is the capture point for the cloud of combinational logic. To launch a rising edge transition delay test from U1, the scan operation loads a 0 into U1 and a 1 into U4. The first capture clock causes U1 to change from 0 to 1. The second capture clock captures the result in U3.

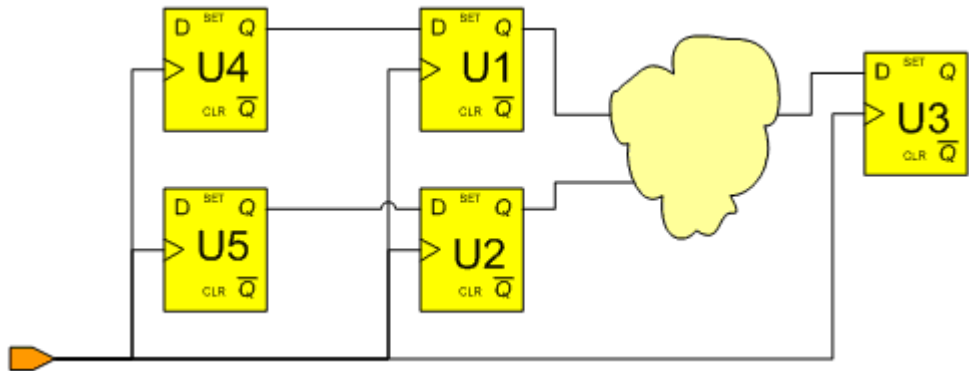


FIGURE 27. Simple Example of At-speed (Transition Delay Fault) Test Circuit

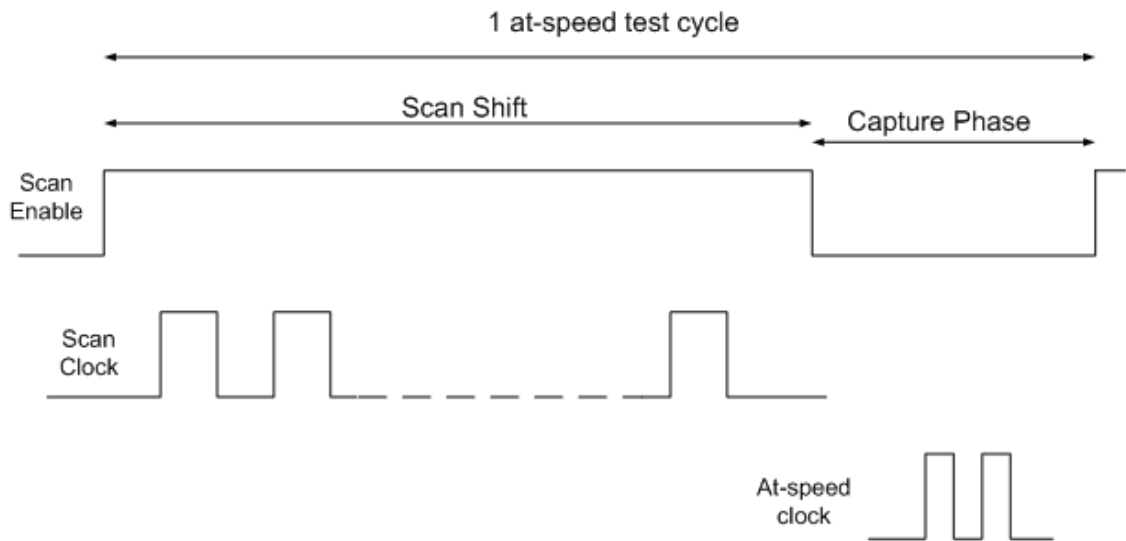


FIGURE 28. Transition Delay Fault Test Waveform

■ Detectable Transition Delay Fault

A slow to rise (slow to fall) fault at a pin is at-speed detectable if the following conditions are satisfied:

- ❑ A rising (falling) edge can be driven on to this pin from
 - ◆ a single flip-flop, called the launch flip-flop or,
 - ◆ if `dsmUsePIForAtSpeed` is on, from a primary input.
- ❑ The pin is observable on a flip-flop in the same domain as the flip-flop launching the edge
 - ◆ If `dsmUsePOForAtSpeed` is on then if the pin is observable on a primary output.

■ Fault Coverage

Transition delay fault coverage is defined as follows:

Fault Coverage = Number of detectable transition delay faults / (2 * (TC-SR))

■ Test Coverage

Suggested SpyGlass DFT ADV Operation

Transition delay test coverage is defined as follows:

$$\text{Test Coverage} = \frac{\text{Number of detectable transition delay faults}}{2 * (\text{TC} - \text{UU} - \text{LR} - \text{FP} - \text{SR})}$$

Where UU is the unused logic, LR is redundant logic and FP are pins on false paths and multi-cycle paths.

Process Flow

Setup at-speed clocks

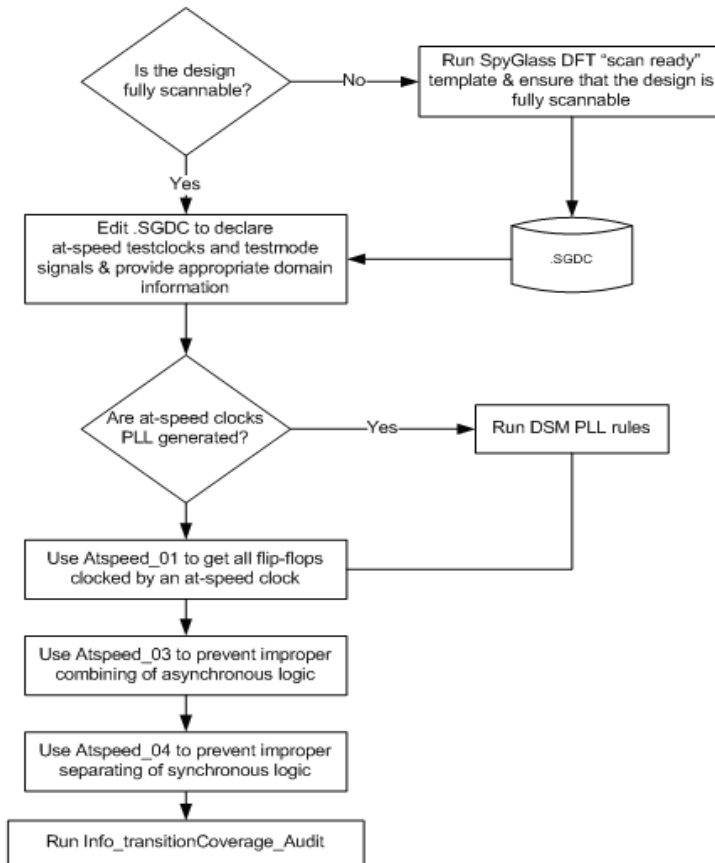


FIGURE 29. SpyGlass DFT ADV Start-up

Clock setup

At-speed testing, as described earlier requires scannable designs. This requires use of `testmode -scanshift` option. In addition, at least one clock must be declared as `-atspeed`.

NOTE: *If you do not use `-atspeed` option with the clock constraint for the at-speed rules, the transition coverage calculation will not operate correctly.*

Improve Coverage

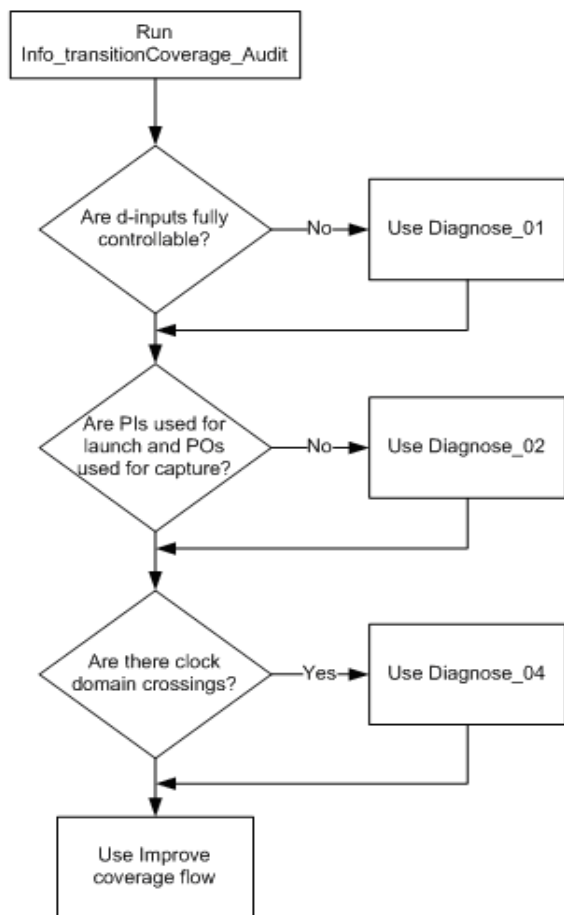


FIGURE 30. Flow to Improve Coverage

Performing Conditional Connectivity Checks

SpyGlass DFT ADV provides mechanism for doing conditional connectivity checks on design nodes. Such conditional connectivity checks are dependent on some other connectivity checks.

You can perform the conditional connectivity checks using one of the following methods:

- [Tcl-Based User-Defined Macros \(UDMs\)](#)
- [SGDC-Based Flow](#)
- [Example](#)

Tcl-Based User-Defined Macros (UDMs)

SpyGlass DFT ADV allows you to perform conditional connectivity check using TCL-based UDMs. For more information on the SpyGlass DFT ADV Tcl commands, see [SpyGlass Tcl Shell Interface User Guide](#). In the UDM-based approach, you need to write TCL procs to create the UDMs at the end of the first SpyGlass run and then re-invoke SpyGlass. Since this approach requires multiple SpyGlass runs, runtime increases, thereby, impacting the performance.

Following are the drawbacks of the Tcl-based UDM approach:

- Many connectivity check-related violation messages are reported, causing noise.
- High runtime because the check is performed on the complete set.
- Dependency on the [Conn_11/Conn_12](#) conditional connectivity checks.

SGDC-Based Flow

Due to the limitation with the TCL-based UDMs approach, SpyGlass DFT ADV provides and SGDC-based approach where filtered connectivity checks may happen within SpyGlass in a single run. Such conditional connectivity checks are performed on the filtered design nodes, that is, the nodes that have PASSED/FAILED the earlier connectivity checks.

For example, consider the following constraint description, which checks if the `sig1` signal is driving the clock-pin of a flip-flop, then its reset-pin is

not driven by sig2:

```
require_path -from sig1 -to_type FLIP_FLOP_CLOCK -
constraint_message_tag CLOCK_CHECK
illegal_path -from sig2 -to_type FLIP_FLOP_RESET -
instance_filter_in_cmt_to "CLOCK_CHECK:PASS"
```

In the above example, the `to` set selects only those reset pins where the corresponding instance has passed the `CLOCK_CHECK`.

This approach uses the following SpyGlass DFT ADV elements:

- [Constraints](#)
- [Rules](#)
- [The constraint_message_tag Modifiers](#)

Constraints

The following table lists the various constraints that are used to perform the filtered connectivity checks using the SGDC-based approach:

TABLE 3 Constraints for the SGDC-Based Approach

Constraints Name	Argument Name	Argument Description
require_value	-filter_in_cmt <constraint_message_tag _expression>	Filters the specified node when the <code>constraint_message_tag_expression</code> holds TRUE on the node itself.
	-instance_filter_in_cmt <constraint_message_tag _expression>	Filters the specified node when the <code>constraint_message_tag_expression</code> holds TRUE on the associated instance of node.

TABLE 3 Constraints for the SGDC-Based Approach (Continued)

Constraints Name	Argument Name	Argument Description
<i>require_path</i>	[-filter_in_cmt_from <constraint_message_tag _expression],	Filters the starting-point when the <code>constraint_message_tag_expression</code> holds TRUE for the specified node.
	[-filter_in_cmt_to <constraint_message_tag _expression>]	Filters the starting-point when the <code>constraint_message_tag_expression</code> holds TRUE for the specified node.
	- instance_filter_in_cmt_from <constraint_message_tag _expression>	Filters the starting-point when the <code>constraint_message_tag_expression</code> holds TRUE for the associated instance of specified node.
	- instance_filter_in_cmt_to <constraint_message_tag _expression>	Filters the end-point when the <code>constraint_message_tag_expression</code> holds TRUE for the associated instance of specified node.

TABLE 3 Constraints for the SGDC-Based Approach (Continued)

Constraints Name	Argument Name	Argument Description
<i>illegal_value</i>	-filter_in_cmt <constraint_message_tag _expression>]	Filters the specified node when the <code>constraint_message_tag_expression</code> holds TRUE on the node itself.
	-instance_filter_in_cmt <constraint_message_tag _expression>	Filters the specified node when the <code>constraint_message_tag_expression</code> holds TRUE on the associated instance of node.

TABLE 3 Constraints for the SGDC-Based Approach (Continued)

Constraints Name	Argument Name	Argument Description
<i>illegal_path</i>	filter_in_cmt_from <constraint_message_tag_expression>	Filters the starting-point when the <code>constraint_message_tag_expression</code> holds TRUE for the specified node.
	-filter_in_cmt_to <constraint_message_tag_expression>	Filters the end-point when the <code>constraint_message_tag_expression</code> holds TRUE for the specified node.
	-instance_filter_in_cmt_from <constraint_message_tag_expression>	Filters the starting-point when the <code>constraint_message_tag_expression</code> holds TRUE for the associated instance of specified node.
	-instance_filter_in_cmt_to <constraint_message_tag_expression>	Filters the end-point when the <code>constraint_message_tag_expression</code> holds TRUE for the associated instance of specified node.

NOTE: Do not use the `filter_in_cmt` argument and its corresponding `instance_filter_in_cmt` argument together.

Rules

The SGDC-based approach uses the following rules to perform the filtered connectivity checks:

- *dftSGDCSTX_080*: This rule performs sanity checks for the following constraint arguments:
 - **The *require_value* and *illegal_value* constraints**: Performs check on the `filter_in_cmt` and `-instance_filter_in_cmt` arguments.
 - **The *require_path* and *illegal_path* constraints**: Performs check on the `-filter_in_cmt_from`, `-filter_in_cmt_to`, `-instance_filter_in_cmt_from`, and `-instance_filter_in_cmt_to` arguments.
- *dftSGDCSTX_064*: This rule reports a violation when the following constraint arguments result in 0 object selection:
 - **The *require_value* and *illegal_value* constraints**: Performs check on the `-filter_in_cmt` and `-instance_filter_in_cmt` arguments.
 - **The *require_path* and *illegal_path* constraints**: Performs check on the `-filter_in_cmt_from`, `-filter_in_cmt_to`, `-instance_filter_in_cmt_from`, and `-instance_filter_in_cmt_to` arguments.

The `constraint_message_tag` Modifiers

By default, the value of *dft_conn_treat_pass_as_strict_pass* is on and the value of the *dft_conn_treat_fail_as_strict_fail* parameter is off. In this case, the SpyGlass DFT ADV provides the following modifiers:

- PASS: presence of at least one pass tag and no fail tag will result in a tag being passed, that is, 'pass and no fail' or 'strict pass'
- FAIL: at least one fail tag will result in a tag being failed, that is, 'fail'

However, based on the value of the *dft_conn_treat_pass_as_strict_pass* and *dft_conn_treat_fail_as_strict_fail* parameters, the following modifiers are available:

- PASS_AND_NO_FAIL: Specifies that the presence of at least one pass tag and no fail tag will result in a tag being passed (that is, strict pass, which is same as PASS when the *dft_conn_treat_pass_as_strict_pass* parameter is on.)
- PASS: Specifies that at least one pass tag will result in a tag being passed

Suggested SpyGlass DFT ADV Operation

- FAIL_AND_NO_PASS: Specifies that the presence of at least one fail tag and no pass tag will result in a tag being failed (that is, strict fail)
- FAIL: Specifies that at least one fail tag will result in a tag being failed (that is, fail, which is same as FAIL, when the [dft_conn_treat_fail_as_strict_fail](#) parameter is on.)

Example

Consider the following RTL specification:

```

module mux41 (output op, input in0, in1, in2, in3, s0, s1);
    assign op = s1 ? (s0 ? in3:in2) : (s0 ? in1 : in0);
endmodule

module mux21 (output op, input in0, in1, s0);
    assign op = s0 ? in1 : in0;
endmodule

module top (output op0, op1, op2, op3, op4, op5, input in0,
in1, sel0, sel1, tm);
    wire w1;
    assign w1 = sel0 & tm;
    mux21 u0(op0, in0, in1, w1);
    mux21 u1(op1, in1, in0, sel1);
    mux21 u2(op2, in1, in0, sel1);
    mux21 u3(op3, in1, in0, sel1);
    mux21 u4(op4, in1, in0, sel1);
    mux21 u5(op5, in1, in0, sel1);
endmodule

```

The intent is to find the muxes in the design whose:

- MUX-data(in0) pin is driven by primary port in0
- MUX-select(s0) pin is driven by primary port sel0

Tcl-Based User-Defined Macros (UDMs) Flow

Consider the following constraint specifications:

```
define_tag -tag initState -name tm -value 1
```

```
require_path -tag initState -from in0 -to "mux*::in0"
```

```
-path_type sensitized -constraint_message_tag MUX_DATA_CHECK
require_path -tag initState -from sel0 -to "mux*::s0"
-path_type sensitized -constraint_message_tag
MUX_SELECT_CHECK
```

```
require_constraint_message_tag -type MUX -
constraint_message_tag_expression "MUX_DATA_CHECK:PASS &&
MUX_SELECT_CHECK:PASS"
```

The following violations are reported:

- The constraint_message_tag, MUX_DATA_CHECK, reports violations (1 success + 5 failures) based on the following status:
 - ❑ PASSED: -from in0 -to top.u0.in0
 - ❑ FAILED: -from in0 -to top.u1.in0 top.u2.in0 top.u3.in0 top.u4.in0 top.u5.in0
- The constraint_message_tag, MUX_SELECT_CHECK, reports violations (1 success + 5 failures) based on the following status:
 - ❑ PASSED: -from sel0 -to top.u0.s0
 - ❑ FAILED: -from sel0 -to top.u1.s0 top.u2.s0 top.u3.s0 top.u4.s0 top.u5.s0
- The constraint require_constraint_message_tag checks whether the constraint_message_tag_expression is TRUE and reports violations (1 Success + 5 failures) based on the following status:
 - ❑ PASSED : top.u0
 - ❑ FAILED : top.u1 top.u2 top.u3 top.u4 top.u5

Therefore, total of 18 (6 for MUX_DATA_CHECK, 6 for MUX_SELECT_CHECK, 6 for require_constraint_message_tag) violations are reported.

SGDC-Based Approach

Consider the following constraint specifications:

```
define_tag -tag initState -name tm -value 1
```

```
require_path -tag initState -from in0 -to_type MUX_DATA -
```


Suggested SpyGlass DFT ADV Operation

```
path_type sensitized -constraint_message_tag MUX_DATA_CHECK

require_path -tag initState -from sel0 -to_type MUX_SELECT -
path_type sensitized -constraint_message_tag
MUX_SELECT_CHECK -instance_filter_in_cmt_to MUX_DATA_CHECK
```

The following violations are reported:

- The Constraint_message_tag, MUX_DATA_CHECK, reports violations (1 success + 5 failures) based on the following status:
 - ☐ PASSED: -from in0 -to top.u0.in0
 - ☐ FAILED: -from in0 -to top.u1.in0 top.u2.in0 top.u3.in0 top.u4.in0 top.u5.in0
- The constraint_message_tag, MUX_SELECT_CHECK, filters the instance on which the MUX_DATA_CHECK has passed. Therefore, it reports one violation for the PASSED status for the following:
 - from sel0 -to top.u0.s0

Therefore, total of 7 violations (6 for MUX_DATA_CHECK and 1 for MUX_SELECT_CHECK) are reported using this approach.

This approach reduces the required number of connectivity checks, thereby, reducing the noise.

Rules in SpyGlass DFT ADV

The SpyGlass[®] DFT product has the following types of rules:

- *Asynchronous Rules*
- *Atspeed Test Rules*
- *BIST Rules*
- *Clock Rules*
- *Clock Gating Rules*
- *Connection Rules*
- *Diagnostic Rules*
- *Information Rules*
- *Latch Rules*
- *PLL Rules*
- *RAM Rules*
- *Scan Rules*
- *Sanity Check Rules*
- *Scan Enable Rules*
- *Scan Power Rules*

- *SoC Rules*
- *Testability Analysis Rules*
- *Test Compression Rules*
- *Topology Rules*
- *Tristate Rules*
- *Integrity Checks*

Asynchronous Rules

Overview

The Asynchronous rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>Async_01</i>	Flip-flops with asynchronous set or reset that are not disabled in the shift mode
<i>Async_02_capture</i>	Flip-flops, latches, or black boxes that are driving flip-flop set/reset pins in capture mode
<i>Async_02_shift</i>	Flip-flops, latches, or black boxes that are driving flip-flop set/reset pins in shift mode
<i>Async_03</i>	Flip-flops where asynchronous set/reset pins are buffer-connected to the same root-level pin but have different active levels
<i>Async_04</i>	Flip-flops with both asynchronous set and asynchronous reset descriptions
<i>Async_05</i>	Flip-flops where the set or reset pins are not directly driven by primary ports
<i>Async_06</i>	Flip-flops where both set and reset lines are simultaneously active
<i>Async_07</i>	Flip-flops where the asynchronous set or reset sources are not inactive in the shift mode
<i>Async_07Lssd</i>	Latches where the asynchronous set or reset sources are not inactive in the shift mode
<i>Async_08</i>	Asynchronous set/reset pins of all the flip-flops are fully controllable during capture
<i>Async_09</i>	Scan flip-flops where the set or reset pins are not controllable in the capture mode
<i>Async_10</i>	Set/reset signals that are not controlled from the user-specified asynchronous reset port
<i>Async_11</i>	Flip-flops whose data pins are connected to the set/reset pin of any flip-flop during capture mode

Rule	Description
<i>Async_12</i>	Flip-flops whose data pins are connected to the set/reset pin of the same flip-flop during capture mode
<i>Async_13</i>	Asynchronous set/reset pins of Scan flip-flops that are not controllable to their inactive state during capture
<i>Async_15</i>	Maintain separate control for asynchronous set/reset pins of flip-flops and latches
<i>Async_16</i>	Ensure that the set or reset sources are disabled or controllable by PI
<i>Async_17</i>	Report all the sources, which drive asynchronous preset, clear and clock pins.

Async_01

Do not use flip-flops with asynchronous set or reset unless disabled in shift mode

When to Use

Use this rule to identify the flip-flops with asynchronous set or reset pins that are not disabled in shift mode.

Description

The Async_01 rule flags those flip-flops with asynchronous set or reset that are not disabled in the shift mode.

Rule Exception

The Async_01 rule does not report a violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode shift conditions. If any flip-flop has a non-inactive set or reset pin, report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Modes

Scanshift

Messages and Suggested Fix

The following violation message is displayed for the Async_01 rule:

```
[WARNING] Flip-flop '<flip-flop-name>' has non-inactive '<pin-type>' pin
```

Arguments

- Name of the flip-flop output net, *<flip-flop-name>*
- Type of the pin, can be SET or RESET, *<pin-type>*

Potential Issues

A violation is reported because of incomplete or incorrect *test_mode* constraint description.

Consequences of Not Fixing

To operate correctly, all flip-flop set and reset pins must be guaranteed to be inactive to not corrupt scan data.

How to Debug and Fix

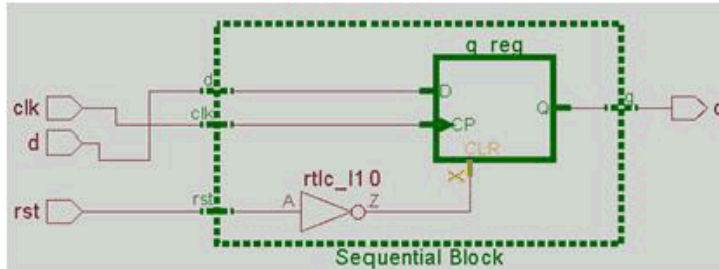
You can view the violation for the Info_testmode (under shift condition) rule along with the violation of the Async_01 rule in the Incremental Schematic window. To do this, double-click the violation for the Async_01 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Async_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_01 rule.

To fix the violation, if any flip-flop uses set or reset functions, then the logic that generates these signals must be modified to be inactive during scan shift.

Example Code and/or Schematic

Consider the following schematic:

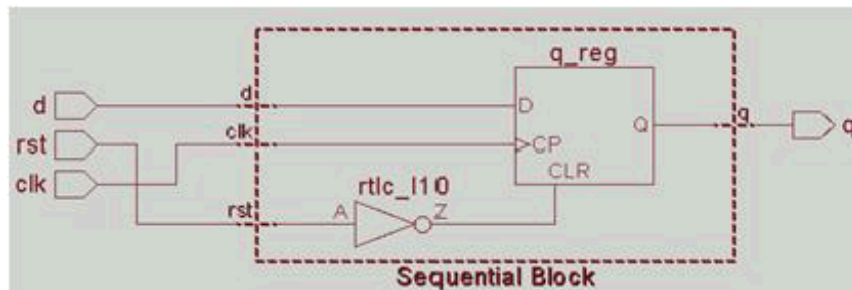


In the above schematic, the Async_01 rule reports the following violation message for the above schematic:

Async_01[1]: Do not use flip-flops with asynchronous set or reset unless disabled in shift mode.
 Flip-flop 'test.q' has non-inactive RESET ('CLR') pin, test.v, 9

To resolve the above violation, constraint the `rst` port as shown below:

```
testmode -name rst -value 1 -scanshift
```



Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

No related reports or files.

Async_02_capture

Ensure that the flip-flop set or reset fan-in cones do not contain - flip-flops, latches, or black boxes in capture mode

When to Use

Use this rule to identify path from the invalid object to the affected flip-flop's set/reset pin.

Description

The Async_02_capture rule reports a violation for those flip-flops whose set/reset pins are driven by sequential elements, that is, flip-flop, latch, or black box, in the capture mode.

NOTE: *The Async_02_capture rule reports violations for non-transparent latches only. For transparent and controllable-transparent latches, the rule does not report a violation.*

Rule Exceptions

The Async_02_capture rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode (without scansift) conditions. Walk the unblocked fan-in cones for scan flip-flop set and reset sources. If any flip-flop, non-transparent latch or black box without scanwrap are reached, declare a message.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_allow_single_flipflop_driver_in_Async_02_capture](#): Default value is off. Set the value of the parameter to on to allow a single flip-flop as driver of an asynchronous sources in the Async_02_capture rule.

- *dft_use_source_centric_Async_02_capture*: Default value is on. Set the value of the parameter to off to switch back to the old behavior of the Async_02_capture rule.
- *dft_ignore_no_scan_driver_in_Async_02_capture*: Default value is off. Set the value of the parameter to on to ignore no_scan flip-flops.
- *dft_ignore_state_holding_flipflop_driver_in_Async_02_capture*: Default value is on. Set the value of the parameter to off to consider the flip-flops, which can retain their states in capture mode, as drivers of asynchronous sources for the Async_02_capture rule.
- *dftGenerateTextReport*: The default value is none. Set the value of the parameter to rule name to generate a text report, which lists the flip-flops whose set/reset pins are driven by sequential elements or black boxes in the capture mode.
- *dft_state_holding_ff_identification_effort_level*: The default value of the parameter is 4. Set the value of the parameter to any natural number to change the effort level for identifying the state-holding flip-flops.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

dft_use_source_centric_Async_02_capture=off

The following violation message is displayed for the *Async_02_capture* rule

when the value of the *dft_use_source_centric_Async_02_capture* parameter is set to off:

[WARNING] <obj -type> '<obj -name>' is affecting '<pin-type>' pin of <num> flip-flop(s) (e.g. '<flip-flop-name>')

Arguments

- Type of the invalid object — Flip-Flop(s), Latch(es), Blackbox(es). (<obj-type>)
- Name of the invalid object. (<obj-name>)
- Type of the pin affected — set or reset. <pin-type>
- Number of the flip-flops affected. (<num>)
- Name of one of the flip-flops affected. (<flip-flop-name>)

Potential Issues

A violation is reported when the path from flip-flop to async pin of scan flip-flop is not blocked in capture mode.

Consequences of Not Fixing

If sets and resets are combinational functions of primary inputs, then they can be held active/inactive with system pins rather than imposing additional routing requirements for separate testmode signals.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic displays the flip-flop driving asynchronous pins of some flip-flops. Overlay (Auxiliary violation mode) the Info_testmode rule under the capture mode. Start probing the fan-out for the flip-flop until it hits asynchronous pins of some flip-flop.

You can also view the violations for the Info_testmode (under capture condition) and rules along with the violation of the Async_02_capture rule in the Incremental Schematic window. To do this, double-click the violation for the Async_02_capture rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Async_02_capture rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_02_capture rule.

To fix the violation, modify either test_mode or design.

dft_use_source_centric_Async_02_capture=on

The following violation message is displayed for the *Async_02_capture* rule when the value of the *dft_use_source_centric_Async_02_capture* parameter is set to on:

```
[WARNING] [<affected_set_reset_pin_count>] Async source  
'<source-name>' is driven by { <drivers-breakup> }
```

Arguments

- Async Source Net Name, <source-name>
- Count and type of drivers, that is, flip-flops, latches, black boxes, or hanging terminals/ nets, driving the source, <drivers-breakup>
- Count of affected async set and reset pins, <affected_set_reset_pin_count>

Potential Issues

A violation is reported when the path from flip-flop, latch, black box, or a hanging terminal/ net to async pin of scan flip-flop is not blocked in capture mode.

Consequences of Not Fixing

If sets and resets are combinational functions of primary inputs, then they can be held active/inactive with system pins rather than imposing additional routing requirements for separate testmode signals.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic displays the flip-flop driving asynchronous pins of some flip-flops. Overlay (Auxiliary violation mode) the Info_testmode rule under the capture mode. Start probing the fan-out for the flip-flop until it hits asynchronous pins of some flip-flop.

The Async_02_capture rule also generates two spreadsheet reports for each message, listing all the drivers and all affected set/reset pins, respectively.

You can also view the violations for the Info_testmode (under capture condition) and rules along with the violation of the Async_02_capture rule in the Incremental Schematic window. To do this, double-click the violation for the Async_02_capture rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Async_02_capture rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_02_capture rule.

To fix the violation, modify either test_mode or design.

Example Code and/or Schematic

Example 1

Consider the following incremental schematic for the Async_02_capture rule:

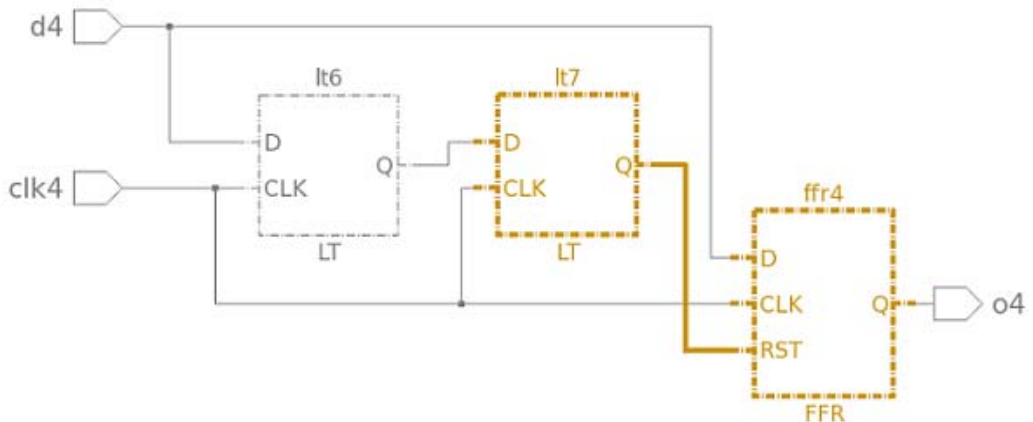


FIGURE 1. Async_02_capture schematic

In the above design, the reset pin of ffr4 flip-flop is connected to the output of the lt7 latch such that one of the asynchronous pins of ffr4 flip-flop is driven by a latch. Therefore Async_02_capture rule reports a violation stating that the top.wrst4 net is driven by a latch, as shown in [Figure 2](#).

Consider the following violation message generated by the Async_02_capture rule:

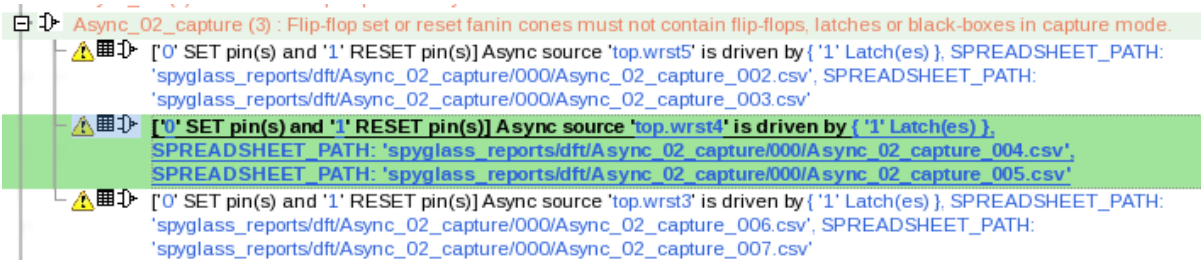


FIGURE 2. Async_02_capture violations

Asynchronous Rules

Clicking on the violation message displays two spreadsheet reports as shown in *Figure 3* and *Figure 4*:

	B "X-source drivers"	C "Type"
1	top.lt7.Q_reg	LATCH

FIGURE 3. The Async_02_capture_004.csv report

The above spreadsheet report states that the driver source is included in the spreadsheet attached to the violation message.

	B	C	D
	"Source-Net"	"Pin Name"	"Type"
1	top.wrst4	top.fpr4.Q_reg.CLR	RESET

FIGURE 4. The Async_02_capture_005.csv report

The above spreadsheet report lists the source net, pin name of the affected pin, and the type of the asynchronous pin.

Example 2

Consider an example when the combinational fanin cones for a set or reset contain unblocked paths to sequential devices. This may cause glitches that may corrupt the capture operation as shown in the following figure:

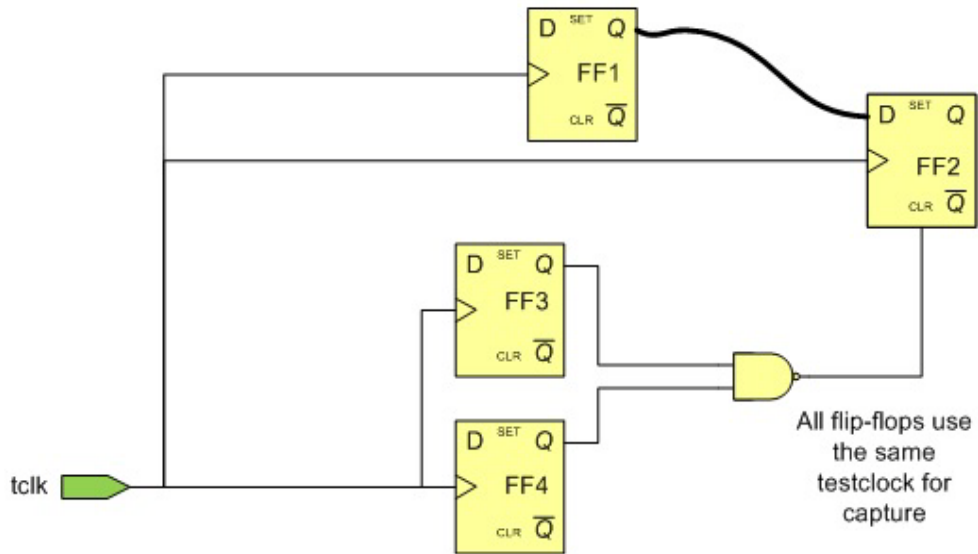


FIGURE 5. Flip-flops feeding async pins

In the above example, the reset pin of FF2 is driven by FF3 and FF4. The testclock, tclk, acts as a capture clock to FF2, which acts as a clock pulse to FF2 and FF3. Since both FF3 and FF4 may change state because of the capture, a reset pulse is sent to FF2. This can cause unexpected capture results.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

[dft_ff_set_reset_sequential_in_capture](#) Report

NOTE: *To generate this report, ensure that the [dftGenerateTextReport](#) parameter is set.*

Async_02_shift

Ensure that the flip-flop set or reset fan-in cones do not contain flip-flops, latches, or black boxes in shift mode

When to Use

Use this rule to identify set or reset pins that are driven by sequential elements in the shift mode.

Description

The Async_02_shift rule reports violation for the flip-flops whose set/reset pins are driven by sequential elements, such as, flip-flop, latch, or black box, in the shift mode.

Rules Exceptions

The Async_02_shift rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode scanshift conditions. Walk the unblocked fan-in cones for scan flip-flop set and reset sources. If any flip-flop, non-transparent latch or black box without scanwrap are reached, declare a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftGenerateTextReport*: The default value is none. Specify a space-separated rule name list as input to the parameter to generate a text report for the specified rules.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_wrap* (optional): Specifies black box design units or instances that will be designed with scan wrappers
- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they qualify as scannable.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] <obj -type> ' <obj -name>' is affecting ' <pin-type>' pin of <num> flip-flop(s) (e. g. ' <flip-flop-name>')

Arguments

- Type of the invalid object - Flip-Flops, Latches, Black boxes, <obj-type>
- Name of the invalid object, <obj-name>
- Type of the pin affected - set or reset, <pin-type>
- Number of the flip-flops affected, <num>
- Name of one of the flip-flops affected, <flip-flop-name>

Potential Issues

A violation is reported when the path from flip-flop to async pin of a scan flip-flop is not blocked in capture.

Consequences of Not Fixing

If sets and resets are combinational functions of primary inputs, then they can be held active/inactive with system pins rather than imposing additional routing requirements for separate testmode signals.

How to Debug and Fix

To debug the violation, double-click the violation message. The Incremental Schematic window highlights the path from the invalid object to the affected flip-flop's set/reset pin.

Asynchronous Rules

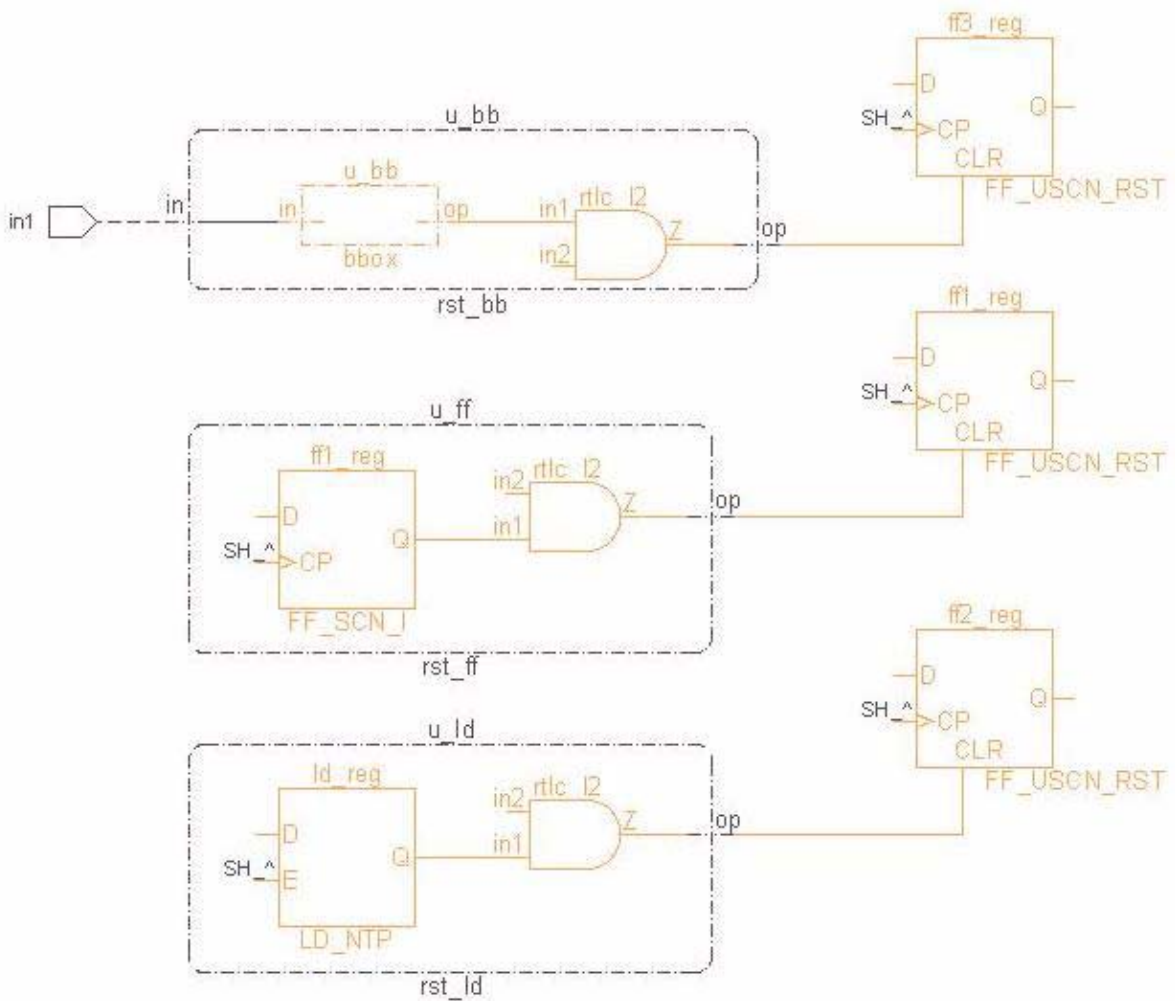
You can also view the violations for the Info_testmode (under shift condition) and Info_path rules along with the violation of the Async_02_shift rule in the Incremental Schematic window. To do this, double-click the violation for the Async_02_shift rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Async_02_shift rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_02_shift rule.

To fix the violation, flip-flop asynchronous pins must be held inactive during scan operations. One method is to force them inactive with a testmode signal. However, if sets and resets are combinational functions of primary inputs, then they can be held inactive with system pins rather than imposing additional routing requirements for separate testmode signals.

Example Code and/or Schematic

Consider the following figure:



In the above figure, the *Async_O2_shift* rule reports violation because the set/reset pins of the flip-flops are driven by the following sequential elements, respectively:

- Black box
- Flip-flop
- Latch

Asynchronous Rules

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

[dft_ff_set_reset_sequential_in_shift](#) Report

NOTE: *To generate this report, ensure that the [dftGenerateTextReport](#) parameter is set.*

Async_03

Ensure that the active phase of all set and reset pins connected to the same root level pin are at the same level

When to Use

Use this rule to identify paths from the root-level pin to the set/reset pins of flip-flops that are connected at different phase with respect to the root-level pin.

Description

The Async_03 rule reports violation for flip-flops where asynchronous set/reset pins are connected to the same root-level pin but do not have the same phase inversion.

The Async_03 rule is often necessary in scan techniques to guarantee that all asynchronous pins can go to their inactive state at the same time.

Rule Exceptions

The Async_03 rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

10

Language

Verilog, VHDL

Method

Walk the unblocked combinational fan-in cone for all flip-flop set and reset pins. For each root-level pin, walk the combinational fan-out cone. The active phase for all flip-flops driven by this pin must be the same.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Set/Reset source '<pin-name>' is being used at different levels: 'same phase (count = <num1>)' (eg. '<flip-flop1-name>'), opposite phase (count = <num2>)' (eg. '<flip-flop2-name>'), unknown phase (count = <num3>)' (eg. '<flip-flop3-name>'). These phases are wrt source

Arguments

- Name of the source pin, <pin-name>
- Number of flip-flops using same phase as the source, <num1>
- Name of one flip-flop using same phase as the source, <flip-flop1-name>
- Number of the flip-flops using the opposite phase with respect to the source, <num2>
- Name of one flip-flop using the opposite phase with respect to the source, <flip-flop2-name>
- Number of the flip-flops with unknown phase, <num3>
- Name of one flip-flop with unknown phase, <flip-flop3-name>

Potential Issues

A violation is reported when the synchronous set/reset pins are connected to the same root-level pin but do not have the same phase inversion.

Consequences of Not Fixing

Fixing the violation is necessary in scan designs to guarantee that all async set/reset pins can go to their active state at the same time.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic displays the flip-flops and the paths from the asynchronous pin of flip-flop to a source pin. The highlighted path is the reason for the violation.

You can also view the violations for the Info_testmode (under shift condition) and Info_path rules along with the violation of the Async_03 rule in the Incremental Schematic window. To do this, double-click the violation for the Async_03 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Async_03 rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_03 rule.

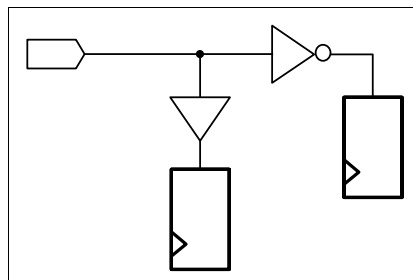
To fix the violation, separate the flip-flops by phase of the set and reset pins so that either dedicated root level pins are used or the flip-flops are connected through the phase inversion logic controlled by a shift mode signal.

NOTE: *Details of a particular category are reported only when such flip-flops are detected.*

Example Code and/or Schematic

Example 1

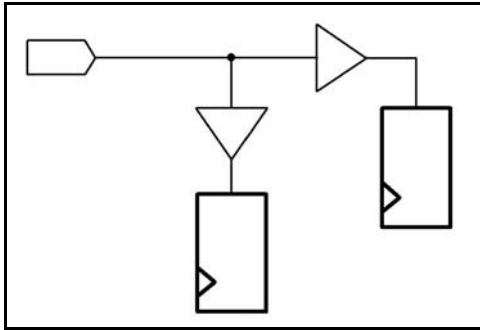
Consider the following figure:



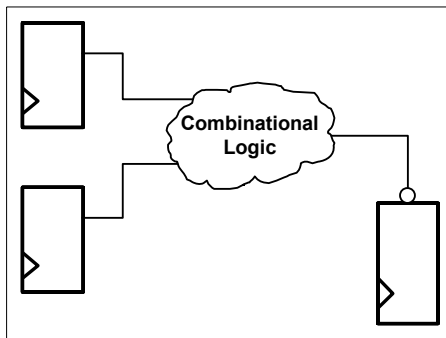
In the above figure, the rule reports a violation because the flip-flops have asynchronous set/reset pins, which are connected to the same root-level pin but do not have the same phase inversion.

The following figure demonstrates a converted good design example:

Asynchronous Rules

**Example 2**

Consider the following figure:



The above figure demonstrates a good design scenario because the flip-flops have asynchronous set/reset pins, which are connected to the same root-level pin and have the same phase inversion.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

No related reports or files.

Async_04

Ensure that the flip-flops are not used with both asynchronous set AND reset

When to Use

Use this rule to avoid flip-flops with both asynchronous set and reset pins.

Description

The Async_04 rule reports violation for flip-flops with both asynchronous set and asynchronous reset descriptions.

Scan designs require that flip-flops are operated as one or more serial shift registers. In such a mode, all flip-flop set and reset pins must be inactive to avoid corrupting the scan data. System logic driving asynchronous defeats this ability. This is a topological check.

Rule Exceptions

The Async_04 rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Method

Simulate power and ground. If any flip-flop has non-inactive set and reset pins, report a message.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Flip-flop '<flip-flop-name>' has both asynchronous pins (set and reset) active

Potential Issues

The violation message appears if a flip-flop has both asynchronous pins in the active state.

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the flip-flop that has both asynchronous pins in the active state.

To fix the violation, ensure that only one pin is active.

Message 2

[WARNING] Flip-flop '<flip-flop-name>' has both asynchronous pins (set and reset) unknown and possibly active

Potential Issues

This violation message appears if both pins of a flip-flop have unknown simulation values.

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click

Consequences of Not Fixing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights both the asynchronous and possibly active set and reset pins of the flip-flops.

To fix the violation, ensure that only one pin is active.

Message 3

[WARNING] Flip-flop '<flip-flop-name>' has reset pin active and set pin unknown (possibly active)

Potential Issues

The violation message appears if the reset pin of a flip-flop has non-x simulation value and the set pin of the same flip-flop has unknown simulation value.

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights active or possibly active set and reset pins of the flip-flops.

To fix the violation, ensure that only one pin is active.

Message 4

[WARNING] Flip-flop '<flip-flop-name>' has set pin active and reset pin unknown (possibly active)

Potential Issues

The violation message appears if set pin has non-X simulation value and the reset pin has unknown simulation value.

Consequences of Not Fixing

Not fixing the violation may potentially corrupt the result in testing in shift and capture mode. It can also cause unpredictable behavior.

Connections to both the set and the reset pins on the same flip-flop can cause unpredictable behavior when both pins are active or when these pins simultaneously transition to the inactive state. Furthermore, since the pins have a common root, modifying the design for proper scan operation

without causing race conditions may be complicated.

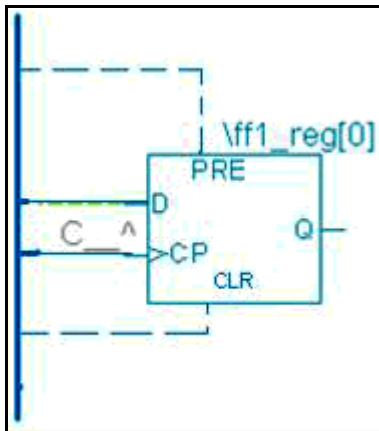
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights active or possibly active set and reset pins of the flip-flops.

To fix the violation, ensure that only one pin is active.

Example Code and/or Schematic

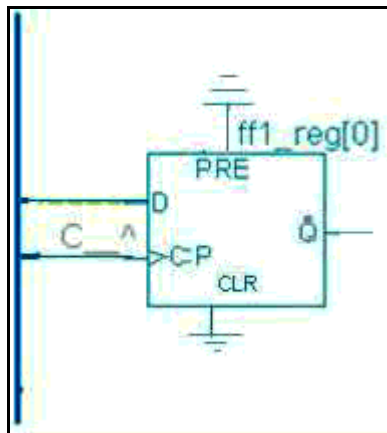
Consider the following figure:



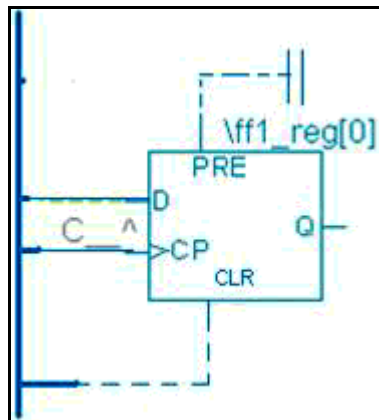
The Async_04 rule reports violation for the above design because both the set and the reset pins of the flip-flop are active.

To fix the violation illustrated above, select from one of the following options:

- Make both the set and the reset pins inactive as shown in the following figure:

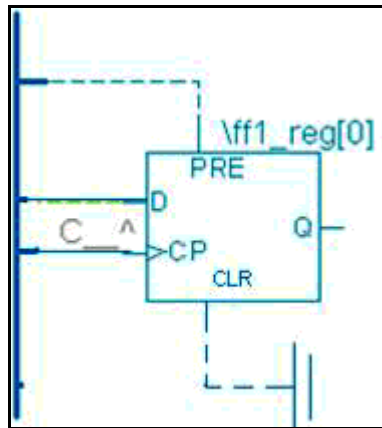


- Ground the set pin of the flip-flop while reset pin is not active as shown in the following figure:



- Ground the reset pin while the set pin is still not active as shown in the following figure:

Asynchronous Rules

**Default Severity Label**

Warning

Rule Group

Asynchronous

Reports and Related Files

No related reports or files.

Async_05

Only one unblocked path allowed from a primary input to a flip-flop set or reset pin.

When to Use

Use this rule to identify the flip-flops whose set or reset pins are not directly driven by single primary ports.

Description

The Async_05 rule reports violation for flip-flops where the set or reset pins are not directly driven by single primary ports.

The Async_05 rule does not require the same root-level pin to control all flip-flops set or reset pins.

The Async_05 rule uses the *test_mode* constraint. Use of testmode conditions has the effect of reducing the effective gate size for devices fed by testmode signals. A 2x1 mux with the selector pin forced to a logic 1, for example, qualifies as a 1-input device with the IN1 pin as the active connection.

NOTE: *The Async_05 rule is a special case of the [Async_02_shift](#) rule.*

Rules Exceptions

The Async_05 rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode scan shift conditions. Walk the unblocked path for every flip-flop set and reset input pin. If the walk terminates at any internal node, report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] The 'set | reset' pins of flip-flop '<flip-flop name>' has '<n>' drivers (Primary port: <m>, Internal: <p>)

Arguments

- Number of primary ports are driver, <m>
- Number of internal (flip-flop, latch, multiple-driver, bbox, undriven net/pin) drivers, <p>
- Number of total drivers, <n> = <m> + <p>

Potential Issues

A violation is reported when the set or reset pins of a flip-flops are driven by more than one unblocked path from the primary ports.

Consequences of Not Fixing

Scan design requires that flip-flops be operated as one or more serial shift registers. In such a mode, all flip-flop set and reset pins must be guaranteed to be inactive to not corrupt scan data.

How to debug and Fix.

See [How to Debug and Fix](#)

Message 2

[WARNING] The 'set | reset' pin of flip-flop '<flip-flop name>' has 'active' value

Potential Issues

The violation message is reported when async pin has an active value and is driven by some logic instead of being directly driven from a primary port.

Consequences of Not Fixing

Scan design requires that flip-flops be operated as one or more serial shift registers. In such a mode, all flip-flop set and reset pins must be guaranteed to be inactive to not corrupt scan data. This can easily be done if they are controlled from the root level ports.

How to Debug and Fix

To debug the violation, view the Incremental Schematic. The Incremental Schematic window highlights the following:

- Flip-flops
- Set or reset pin of the flip-flops which has no or more than one unblocked path from the top level pins.
- Top level ports in case of multiple unblocked path to the set/reset pin.

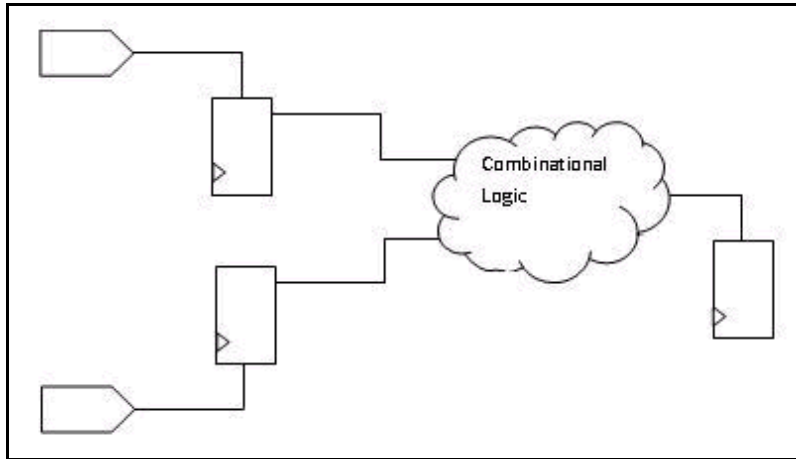
You can also view the violations for the Info_testmode (under shift condition) and Info_path rules along with the violation of the Async_05 rule in the Incremental Schematic window. To do this, double-click the violation for the Async_05 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Async_05 rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_05 rule.

To fix the violation, scan design requires that the set and reset pins be held inactive in order to achieve reliable scan shifting operations. An often-used method is to force such pins inactive with a common testmode signal. Any flip-flop with asynchronous pins directly controllable from top-level pins need not necessarily be modified with a testmode signal since direct access already exists.

Example Code and/or Schematic

Consider the following figure:



For the above design, the Async_05 rule reports a violation because the set or reset pins of the flip-flop are not directly driven by single primary ports.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

No related reports or files.

Async_06

Ensure that the set and reset lines on the same flip-flop are not simultaneously active.

When to Use

Use this rule to identify flip-flops whose both set and reset pins are active simultaneously.

Description

The Async_06 rule flags flip-flops where both set and reset lines are active simultaneously.

Rule Exceptions

The Async_06 rule ignores rule checking for the following:

- Flip-flops where the sum of the fan-in count of the set pin and reset pin is more than or equal to the limit specified with the *limitFaninPorts* rule parameter.
- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Method

Simulate power and ground and any available testmode conditions.

For every flip-flop, skip if set or reset is inactive. Walk the unblocked combinational fan-in cone of either the set or the reset. Stop the walk at non-X nodes.

Let S be the set of primary inputs and flip-flop outputs reached in the walk back.

Simulate all possible legal input conditions on S, while preserving testmode.

If any combination makes both set and reset become active or one of them x, then report a message on this flip-flop.

Language

Verilog

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *limitFaninPorts*: The default value is 12. Set the value of the parameter to any natural number to set the maximum limit on the number of objects (ports, latches, flip-flops, and black box instances) in the fan-in cone of a net.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they so qualify.
- *define_legal_input_values* (optional): Considers the specified input values for a set of inputs as legal. All other input values not specified for that set of inputs are considered illegal.
- *define_illegal_input_values* (optional): Considers the specified input values for a set of inputs as illegal. All other input values not specified for that set of inputs are considered legal.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Fanin Count for set/reset pins of flop '<flip-flop-name>' is '<num>' (>= \$limitFaninPorts) hence rule checking ignored

Arguments

- Name of the flip-flop, <flip-flop-name>
- Sum of the fan-in count of the set pin and reset pin, <num>

Potential Issues

A violation message is reported if the count of the fan-in cone drivers on set and reset pins of a flip-flop is more than the limit specified with the [limitFaninPorts](#) parameter. SpyGlass DFT ADV product does not analyze set/reset cone of such flip-flop.

Consequences of Not Fixing

The violation is reported and the SpyGlass DFT ADV product does not evaluate that specific flip-flop. Therefore, potential design problems may or may not be detected.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the flip-flop whose count of the fan-in cone drivers on set and reset pins is more than the limit specified with the [limitFaninPorts](#) parameter.

To fix the violation, either put additional constraints or modify the design connectivity.

Message 2

[INFO] Async_06 report file <file-name> is generated

Arguments

Name of the report file, <file-name>

Potential Issues

A violation message is reported when the Async_06 rule generates the [dft_ff_set_reset_active](#) report that lists the flip-flops and the input conditions that make both set and reset lines simultaneously active.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required.

Message 3

[WARNING] Both the asynchronous pins (set and reset) of flip-flop '<flip-flop-name>' are active in capture mode

Arguments

Name of the flip-flop, <flip-flop-name>

Potential Issues

A violation is reported when both the set and reset pins of a flip-flop are active in capture mode.

Consequences of Not Fixing

Results, during capture, may get corrupted when both the pins get activated during random testing.

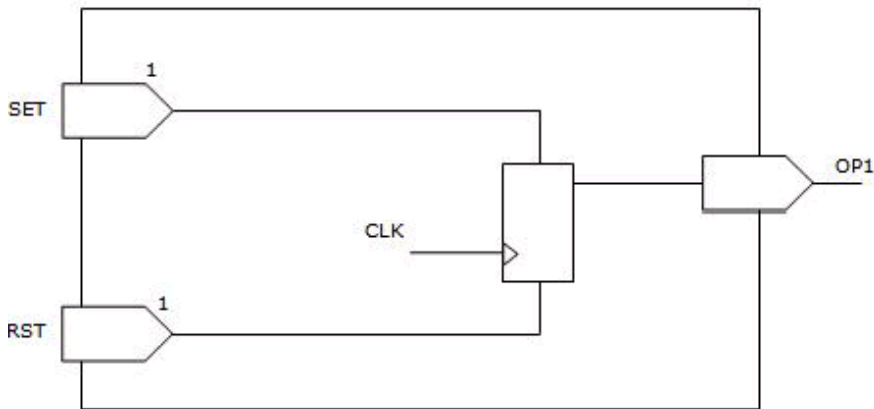
How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the flip-flop whose set and reset pin count is more than the limit specified with the [limitFaninPorts](#) parameter.

To fix the violation, either put additional constraints or modify the design connectivity.

Example Code and/or Schematic

Consider the following figure:



The above figure demonstrates a bad design example as both the set and reset pins of a flip-flop are simultaneously active.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

[*dft_ff_set_reset_active*](#): Lists the flip-flops and the input conditions that make both set and reset lines simultaneously active.

Async_07

Ensure that the asynchronous set/reset sources are inactive during shift mode.

When to Use

Use this rule to identify path from asynchronous (set/reset) source to the set/reset pin of the flip-flop.

Description

The Async_07 rule reports violation for flip-flops where the asynchronous set/reset source is active during the shift mode. The rule reports asynchronous source and the number of flip-flops, fed by this source, that are not inactive.

This rule verifies that all the sources of asynchronous sets and resets are disabled in the shift mode. Since one source often drives multiple registers, this rule and the Clock_11 rule are efficient means to qualify a design for scannability.

Use [Info_testmode](#) rule to display how shift mode signals are distributed in the design.

When the Async_07 rule is run in conjunction with the [Scan_08](#) rule, a tag can be added in the violation messages by setting the value of the [dftTagAsync07Clock11Scan08](#) rule parameter to on. This enables the correlation of the cause of unscannability.

Rule Exceptions

The Async_07 rule does not report violation for following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

The Async_07 rule supports [dftAutoFix](#) spy-parameter. So, if the AutoFix feature is on, a new RTL code is generated with all the Async_07 rule violations fixed.

The flip-flops marked as no-scan flip-flops are not counted. In case, all flip-flops controlled by the internally generated async set or reset are marked as no-scan flip-flops, the async set or reset is not considered as a rule-violating async set or reset and no message is generated.

Default Weight

10

Language

Verilog, VHDL

Method

Find all asynchronous sets and resets sources. (An asynchronous source is an input pin, black box, or the first device with more than one input found when traversing the set input or reset input of a flip-flop.)

Simulate power and ground and any available testmode scan shift conditions.

For every flip-flop, skip if set or reset is inactive. Walk the unblocked combinational fan-in cone of either the set or the reset pin. If testmode information is not supplied, report any asynchronous source that is not a root level pin. If testmode information is available, simulate testmode conditions. Verify that each flip-flop set or reset pin, connected to this asynchronous source is at its inactive state. Otherwise, report a message.

NOTE: *The `Async_01` rule is flip-flop-centric whereas the `Async_07` rule is asynchronous source-centric. Therefore, it is usually more efficient to use the `Async_07` rule.*

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*

- *dftAutoFix*: The default value of the parameter is as follows:

```
"{+RULES[nothing]+TMPOR[T[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}"
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
"{+RULES[<rulelist>]+TMPOR[tmport]+TCLKPORT[tclkport]}"
```

- *dft_autofix_testmode_signal*: The default value of this parameter is `atrenta_generated_port_tm`. Set the value of the parameter to a testmode signal name to specify testmode signal for autofix and RTL testpoint insertion.
- *dft_autofix_testmode_signal*: The default value of this parameter is `atrenta_generated_port_tm`. Set the value of the parameter to a

Asynchronous Rules

testmode signal name to specify testmode signal for autofix and RTL testpoint insertion.

- *dft_autofix_testclock_signal*: The default value of this parameter is `atrenta_generated_port_tclk`. Set the value of the parameter to a testclock signal name to specify testclock signal for autofix and RTL testpoint insertion.
- *dftTagAsync07Clock11Scan08*: The default value of the parameter is off. Set the value of the parameter to `on` to add a unique tag for every asynchronous source (set, reset, or clock) that is visible in the violations of Async_07, Clock_11, and Scan_08 rules.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to `off` so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *dont_touch*: Use this constraint to specify the modules/nets that are not considered for AutoFix.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *reset_pin* (optional): Use this constraint to specify black box pins that should be assumed to be reset pins.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.
- *set_pin* (optional): Use this constraint to specify black box pins that should be assumed to be set/reset pins.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

The following violation message is displayed for the Async_07 rule:

```
[WARNING] <tag> Async signal (<type>) [in '<du-name>'] '<net-name>' is not disabled for <num> flip-flop(s) in test-mode [stops at '<tobj-type>']
```

Arguments

- Asynchronous source tag in the form DFT_ASYNC_CLR_id, DFT_ASYNC_SET_id, <tag>
- Type of signal — Set or Reset, <type>
- Name of the design unit, <du-name>
- Name of the internally generated asynchronous set or reset source, <net-name>
- Number of flip-flops with a set or reset connected to this source without phase inversion, <num>
- Terminating object type — Port, Hanging net, <gate-name> gate, <tobj-type>

Potential Issues

A violation is reported when either test_modes are not properly specified or async pins are not bypassed properly.

Consequences of Not Fixing

Not fixing the violation may result in flip-flops not participating in scan chain, which will result in low coverage.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the flip-flop on which the asynchronous control is not deactivated. Overlay (Auxiliary violation mode) the [Info_testmode](#) rule in shift mode. This helps in identifying the blocked values.

Double-click the schematic and start probing the fan-in cone of the failing asynchronous source. Identify the path on the basis of the signal name. The probed signal is annotated with a shift mode value, which is either 0 or 1. Absence of any value implies that neither 0 nor 1 has reached to this point. Presence of X as a shift mode value may imply potential contention.

Asynchronous Rules

You can also double-click on the violation message to view the spreadsheet that lists the details of the flip-flops for which the async signal is not disabled in the test mode.

To fix the violation, either fix the test mode or the async signal.

Message 2

[INFO] Suggested async pins for autofix

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

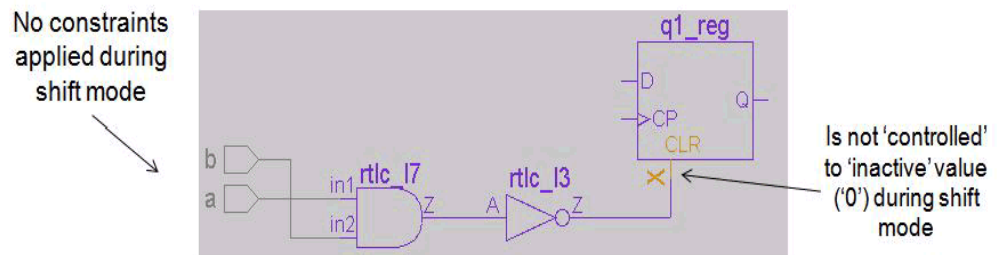
This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

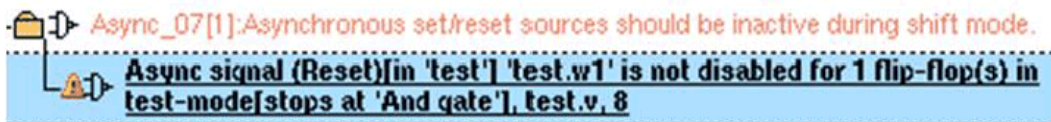
See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic

Consider the following schematic:



The Async_07 rule reports the following violation message for the above scenario:



To fix the above violation, apply `test_mode` constraint on a and b to drive

inactive value to the `clr` pin of the flip-flop as shown below:

```
test_mode -name a -value 1
```

```
test_mode -name b -value 1
```

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

No related reports or files.

Async_07Lssd

Ensure that the asynchronous set/reset sources are inactive during shift mode.

When to Use

Use this rule to avoid shift vector corruption and for reliable capture and shift operation.

Description

The Async_07Lssd rule reports latch instances with asynchronous set/reset source not inactive during the shift mode.

This rule verifies that all the sources of asynchronous sets and resets are disabled in the shift mode. Since one source often drives multiple registers, this rule is an efficient tool that helps to qualify a design for scannability.

Use [Info_testmode](#) rule to display how the shift mode signals are distributed in the design.

Rule Exceptions

The Async_07Lssd rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftAutoFix](#): Use this parameter to generate a new RTL code with all the Async_07Lssd rule violations fixed.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *set_pin* (optional): Use this constraint to specify black box pins that should be assumed to be set/reset pins.
- *reset_pin* (optional): Use this constraint to specify black box pins that should be assumed to be reset pins.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN). Use this constraint's `-lssd` argument for the `Async_07Lssd` rule.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Lssd Latch has set and reset not disabled during scan shift

Potential Issues

The violation message appears, if set and reset pins for Lssd latches are not disabled during the scan shift.

Consequences of Not Fixing

Not fixing the violation may result in shift vector corruption.

How to Debug and Fix

To know how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Lssd Latch has set not disabled during scan shift

Potential Issues

The violation message appears when the set pin for Lssd latch is not

disabled.

Consequences of Not Fixing

Not fixing the violation may result in shift vector corruption.

How to Debug and Fix

To know how to debug and fix the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Lssd Latch has reset not disabled during scan shift

Potential Issues

The violation message appears when the reset pin for Lssd latch is not disabled.

Consequences of Not Fixing

Not fixing this violation may result in shift vector corruption.

How to Debug and Fix

To debug the violation, double-click the violation message. The Incremental Schematic window highlights the path from asynchronous (set/reset) source to the set/reset pin of the latch.

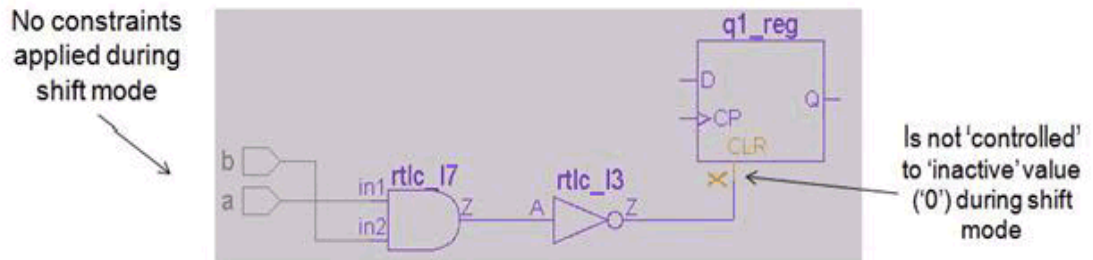
To fix the violation, ensure that the asynchronous pin is disabled.

To view the violation for the Info_testmode rule under shift condition, along with the violation of the Async_07Lssd rule in the Incremental Schematic window, double-click the violation for the Async_07Lssd rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Async_07Lssd rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_07Lssd rule.

Example Code and/or Schematic

Consider the following schematic:



In the above example, asynchronous set/reset is not inactive in the shift mode. Therefore, the Async_07Lssd rule reports a violation in this scenario.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

None

Async_08

Ensure that the asynchronous set/reset pins of all the flip-flops are fully controllable during capture

When to Use

Use this rule to identify testmode signals that only control asynchronous set/reset pins of flip-flops and these set or reset pins are not tied off and are also not testable in the capture mode.

Description

The Async_08 rule reports violation for testmode signals that only control asynchronous set/reset pins of flip-flops and these set or reset pins are not tied off and are also not testable in the capture mode.

The following table lists the difference between the Async_08 and [Async_09](#) rules:

TABLE 1 Difference between Async_08 and Async_09 rules

	Async_08	Async_09
Checks all flip-flops	✓	
Checks only scannable flip-flops		✓

Rule Exceptions

The Async_08 rule does not report violations for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Method

Simulate power and ground conditions

Simulate capture conditions and then do controllability. If a set or reset is yy, assume it is ATPG controllable otherwise report a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUUMarking*: The *dftUUMarking* parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode. For the Async_08 rule, use this constraint without the `-scanshift` argument.
- *force_no_scan* (optional): Exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix**Message**

[WARNING] The source net '<net-name>' feeding '<pin-type>' pin of '<num>' flops is not ATPG controllable

Arguments

- Name of the set/reset source net, <net-name>
- The pin type as set or reset, <pin-type>
- Number of flip-flops whose set/reset pin is driven the set/reset source net, <num>
- Name of the one flip-flop whose set/reset pin is driven the set/reset source net, <flip-flop-name>

Potential Issues

A violation is reported when the synchronous set/reset pins of the flip-flops are ATPG controllable during capture.

Consequences of Not Fixing

Not fixing the violation results in faults on the data line and set/reset lines of all the flip-flops as ATPG is unable to generate vectors.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the violating flip-flop and its set/reset pin.

You can also view the violations for the Info_testmode (under capture condition) and Info_path rules along with the violation of the Async_08 rule in the Incremental Schematic window. To do this, select the violation for the Async_08 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Async_08 rule in the Incremental Schematic window. This is useful in debugging the violation for the Async_08 rule.

You can also double-click on the violation message to view the spreadsheet that lists the details of the flip-flops whose source net feeding pin is not ATPG controllable.

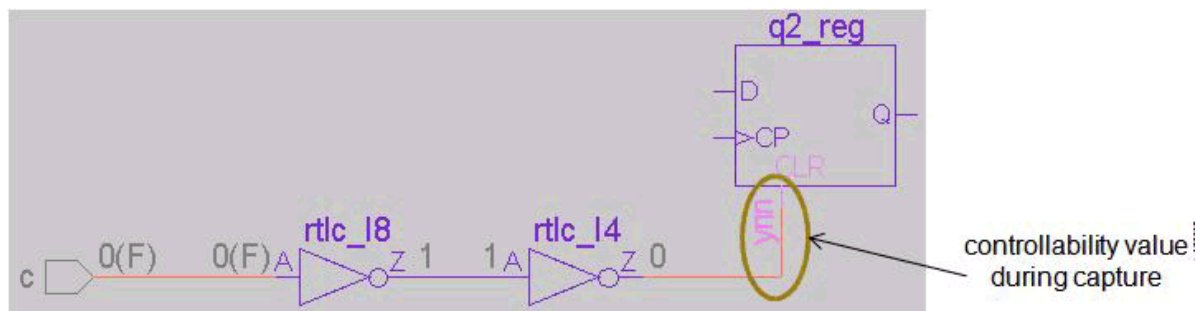
To fix the violation, the logic driving the set or reset pins should be designed so that the sets and resets can be controlled from a single root-level pin. This allows root level pins to be operated as testresets, after a scan is complete, by the ATPG tool. If the set/reset logic is controlled by a single testmode pin, and that pin only controls sets and reset pins, then consider adding scanshift to that pin. Otherwise, the design may have to be changed to get maximum test coverage.

One solution is to add the `-scanshift` argument to the `test_mode`

constraint to indicate that the constraint only applies during shifting and is a don't care otherwise.

Example Code and/or Schematic

Consider the following schematic:



Also, consider the following *test_mode* constraint definition for the above schematic:

```
test_mode -name c -value 0
```

For the above design scenario, the Async_08 rule reports the following violation message:



To fix the above violation, define the *test_mode* constraint as shown below:

```
test_mode -name c -value 0 -scanshift
```

Default Severity Label

Warning

Rule Group

Asynchronous Rules

Asynchronous Rules

Reports and Related Files

No related reports or files.

Async_09

Ensure that the set and reset pins of scan flip-flops are controllable when circuit is in capture mode.

When to Use

Use this rule to identify scan flip-flops where the set or reset pins are not controllable in the capture mode.

Description

The Async_09 rule reports violation for scan flip-flops where the set or reset pins are not controllable in the capture mode.

To view the difference between the [Async_08](#) and Async_09 rules, see [Table 1](#).

Rule Exceptions

The Async_09 rule does not report violations for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Method

Simulate power and ground and capture mode conditions (testmode without -scanshift argument). If any flip-flop, not marked as "noscan", has a non X simulation value on set or reset pin (other than the ones tied), report a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- [dft_maximum_number_of_rows_with_schematic](#): Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_type* (optional): Use to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Async signal (<type>) '<net-name>' is non-controllable for <num> flip-flop(s) in capture mode

Arguments

- Signal type (Set or Reset), <type>
- Set or reset signal, <net-name>
- Number of flip-flops where the set pin or reset pin is not controllable in the capture mode, <num>

Potential Issues

A violation is reported when the synchronous set/reset pins of the flip-flops are ATPG controllable during capture.

Consequences of Not Fixing

Not fixing the violation results in faults on the data line and set/reset lines of all the flip-flops as ATPG is unable to generate vectors.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the violating flip-flop and the path from asynchronous (set/reset) source to the set/reset pin of the flip-flop.

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Async_09 rule in the Incremental Schematic window. To do this, select the violation for the Async_09 rule and open the Incremental Schematic window.

The violation message for the `Info_testmode` rule overlaps the violation message for the `Async_09` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Async_09` rule.

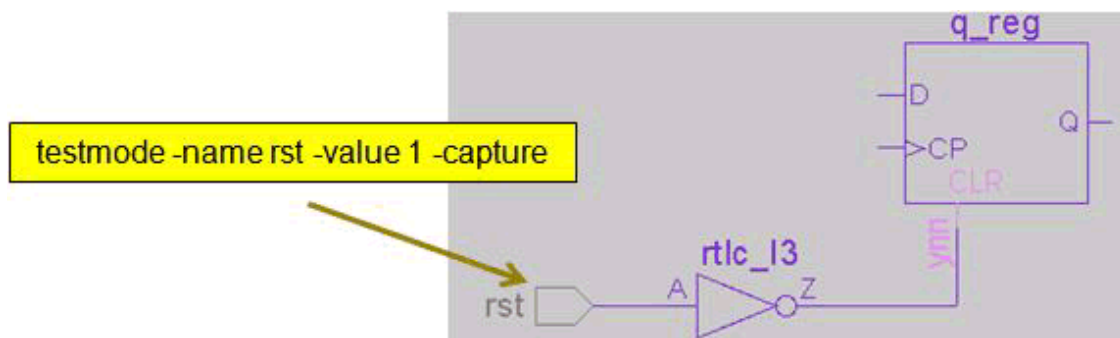
You can also double-click on the violation message to view the spreadsheet that lists the details of the scan flip-flops for which the async signal is not ATPG controllable in the capture mode.

To fix the violation, the logic driving the set or reset pins should be designed so that the sets and resets can be controlled from a single root-level pin. This allows root level pins to be operated as testresets, after a scan is complete, by the ATPG tool. If the set/reset logic is controlled by a single testmode pin, and that pin only controls sets and reset pins, then consider adding `scanshift` to that pin. Otherwise, the design may have to be changed to get maximum test coverage.

One solution is to add the `-scanshift` argument to the `test_mode` constraint to indicate that the constraint only applies during shifting and is a don't care otherwise.

Example Code and/or Schematic

Consider the following schematic:



In the above example, since the `q_reg` is a scannable flip-flop, the `Async_09` rule reports a violation. However, the `Async_08` rule does not report a violation in this case.

Default Severity Label

Warning

Asynchronous Rules

Rule Group

Asynchronous Rules

Reports and Related Files

No related reports or files.

Async_10

Ensure that the user designated pins control all sets and resets.

When to Use

Use this rule for faster and reliable scan of design.

Description

The Async_10 rule reports set/reset signals that are not controlled by user-specified asynchronous reset port.

The Async_10 rule ensures that all flip-flop sets and resets are controlled to behave as inactive by one reset pin. This helps to ensure that ATPG generates tests for the asynchronous logic.

Prerequisites

Specify one reset pin that controls all sets and resets for ATPG to generate tests for asynchronous logic.

Rule Exceptions

The Async_10 rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *reset -async* (optional): Use to specify asynchronous set or reset pins in testmode.
- *test_mode* (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode. Use this constraint's `-scanshift` argument for the `Async_10` rule.
- *force_no_scan* (optional): Use to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Message Details

Message 1

[WARNING] No 'reset -async' constraint specified for design '<du-name>'

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message appears if the *reset -async* constraint is not specified for a design.

Consequences of Not Fixing

Not fixing this violation may result in low speed and less control over the design.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window displays that the *reset -async* pin is not specified. To know more about debugging the violation, click [How to Debug and Fix](#).

To fix the violation, assign all the user-specified pins.

Message 2

[WARNING] No 'test_mode -scanshift' constraint specified for reset -async '<port-name>'

Arguments

Name of the asynchronous reset port, <port-name>

Potential Issues

The violation message appears when no *test_mode* -scanshift constraint is specified for the *reset -async* constraint.

Consequences of Not Fixing

Not fixing this violation may result in reduced observability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window displays the *reset -async* pin when the matching *test_mode* constraint is not specified. To know more about debugging the violation, click [How to Debug and Fix](#).

To fix the violation, specify the *test_mode* -scanshift constraint for the *reset -async* constraint.

Message 3

[WARNING] No asynchronous source is deactivated by reset -async '*<port-name>*'

Arguments

Name of the asynchronous reset port, <port-name>

Potential Issues

The violation message appears if no asynchronous source is deactivated by *reset -async* pin.

Consequences of Not Fixing

Not fixing this violation may result in reduced observability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window displays that no asynchronous source deactivated by *reset -async* pin.

To know more about debugging the violation, click [How to Debug and Fix](#).

To fix the violation, ensure that *reset -async* constraint deactivates the asynchronous source.

Message 4

[WARNING] Asynchronous source <net-name> was not deactivated by any `reset -async pin`

Arguments

Name of the reset net, <net-name>

Potential Issues

The violation message appears if the asynchronous source is not deactivated by `reset -async pin`.

Consequences of Not Fixing

Not fixing this violation may result in reduced observability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window displays the path from asynchronous (set/reset) source to the set/reset pin of the flip-flop when the set/reset pin is not deactivated by the specified asynchronous reset port.

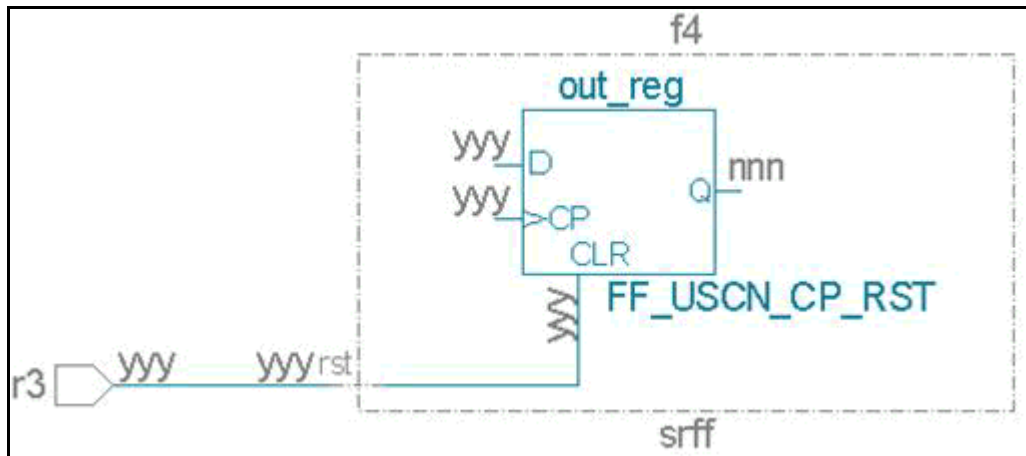
To view the violation under shift condition for the `Info_testmode` rule along with the violation of the `Async_10` rule in the Incremental Schematic window, double-click the violation for the `Async_10` rule and open the Incremental Schematic window.

The violation message for the `Info_testmode` rule overlaps the violation message for the `Async_10` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Async_10` rule.

To fix the violation, ensure that `reset -async` constraint deactivates the asynchronous source.

Example Code and/or Schematic

Consider the following figure:



In the above example, the Async_10 rule reports a violation because the reset -async pin, FF_USCN_CP_RST, is not controlled by the user-specified asynchronous reset port.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

No related reports or files.

Async_11

Avoid using set/reset signal as data signal in capture mode.

When to Use

Use this rule to identify the common path between the set/reset pin and data pin of a flip-flop.

Description

The Async_11 rule reports violation for flip-flops whose data pins are connected to the set/reset pin of any flip-flop during capture mode.

The Async_11 rule reports a violation, if async reset is used as data. In order to test the data line, ATPG toggles data line randomly, which may corrupt the vector through the reset line. On the other hand, if ATPG constraints reset line to a fixed value (inactive value) to avoid vector-corruption, the loss of coverage happens at the flip-flop where it is used as data. The Async_11 rule detects such a scenario.

The Async_11 rule also flags the common path between data and reset on select line of muxes.

Rule Exceptions

The Async_11 rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Async_11 rule:

```
[WARNING] In top design '<top_du>', source '<driver_net>' is driving '<no_async_pin>' set/reset pin(s) and '<no_data_pin>' data pin(s)
```

Arguments

- Top design unit name, <top_du>
- Driver net name, <driver_net>
- Number of asynchronous pins, <no_async_pin>
- Number of data pins, <no_data_pin>

Potential Issues

A violation is reported due to bad RTL.

Consequences of Not Fixing

The Async_11 rule reports a violation, if async reset is used as data. In order to test the data line, ATPG toggles data line randomly, which may corrupt the vector through the reset line. On the other hand, if ATPG constraints reset line to a fixed value (inactive value) to avoid vector-corruption, the loss of coverage happens at the flip-flop where it is used as data. The Async_11 rule detects such a scenario.

How to Debug and Fix

Asynchronous Rules

View the Incremental Schematic for the violation message. The Incremental Schematic displays the flop and the paths from its asynchronous pin of a flip-flop to a source pin. The highlighted path is the reason for the violation.

You can also view the violations for the `Info_testmode` (under shift condition) and `Info_path` rules along with the violation of the `Async_11` rule in the Incremental Schematic window. To do this, double-click the violation for the `Async_11` rule and open the Incremental Schematic window.

The violation messages for the `Info_testmode` and `Info_path` rules overlap the violation message for the `Async_11` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Async_11` rule.

To fix the violation, fix RTL or block one of the paths in RTL.

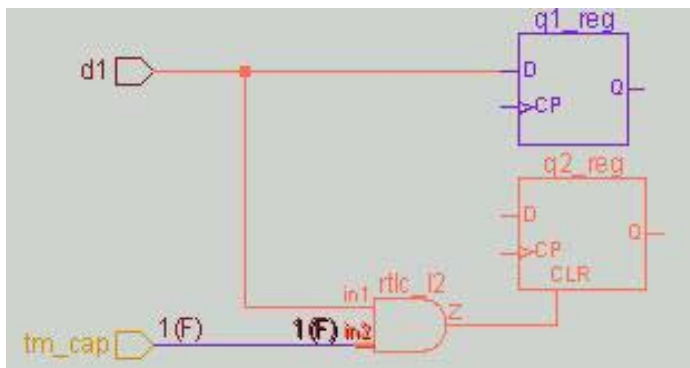
Example Code and/or Schematic

Example 1

Consider the following example:

```
testmode -name tm_cap -value 1 -capture
```

```
Async_11 Data pin of flip-flop 'test.q1' and reset pin of flip-flop 'test.q2' share root 'test.d1'
```



Example 2

This example illustrates conditions when you should avoid using set/reset signal as data signal in capture mode.

Consider the following figure illustrating a problem in a circuit with an Async_11 violation.

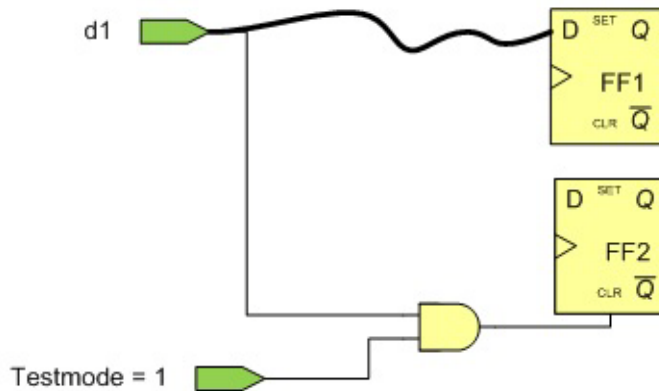


FIGURE 6. Data used as reset

In the above example, if a constraint is placed on d1 to make FF2.clr inactive, then faults on the path from d1 to FF1.d, that require values on d1 in conflict with the constraint on d1, cannot be built.

Now, consider the following figure that illustrates the case of re-convergence from d2 to u1:

Asynchronous Rules

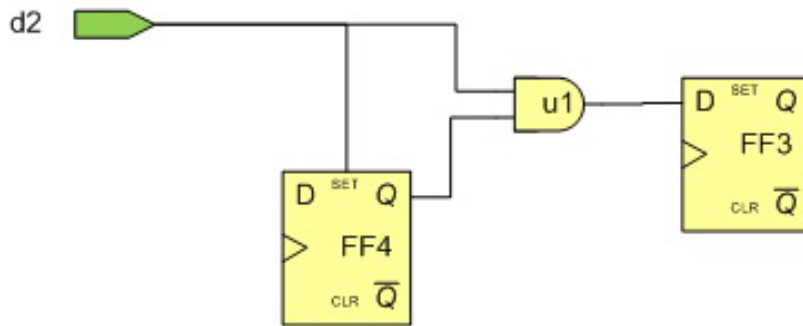


FIGURE 7. Data used as reset in re-convergence

In the above figure, if d2 is unrestricted, then tests that require an active value on d2 will corrupt the scanin data on FF3. As a consequence, the test is invalidated and the detection is decreased.

Default Severity Label

Warning

Rule Group

Asynchronous Rules

Reports and Related Files

No related reports or files.

Async_12

Reports flip-flops with data pins connected to the set/reset pins of the same flip-flops during capture mode

When to Use

Use this rule for reliable data capture.

Description

The Async_12 rule reports violation for flip-flops whose data pins are connected to the set/reset pins of the same flip-flop during capture mode.

NOTE: *The Async_12 rule is switched off by default.*

ATPG tools find it difficult to generate patterns for flip-flops whose data pins are connected to the set/reset pins.

Rule Exceptions

The Async_12 rule does not report violation for the following:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (optional): Use to declare the clock pins used as test clocks. Use this constraint's `-testclock` argument for the `Async_12` rule.
- *force_no_scan* (optional): Use to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Data and reset pin of flip-flop <flip-flop-name> share signal <sig-name>

Arguments

- Name of the flip-flop, <flip-flop-name>
- Name of the set/reset signal, <sig-name>

Potential Issues

The violation message appears, if data and reset pin of a flip-flop has common combinational logic.

Consequences of Not Fixing

Not fixing the violation may result in race condition and unreliable capture.

How to Debug and Fix

To debug, double-click the violation message. The Incremental Schematic window shows the common path between the set/reset pin and the data pin of the flip-flop.

You can also view the violations for the `Info_testmode` (under capture condition) and `Info_path` rules along with the violation of the `Async_12` rule in the Incremental Schematic window. To do this, double-click the violation for the `Async_12` rule and open the Incremental Schematic window.

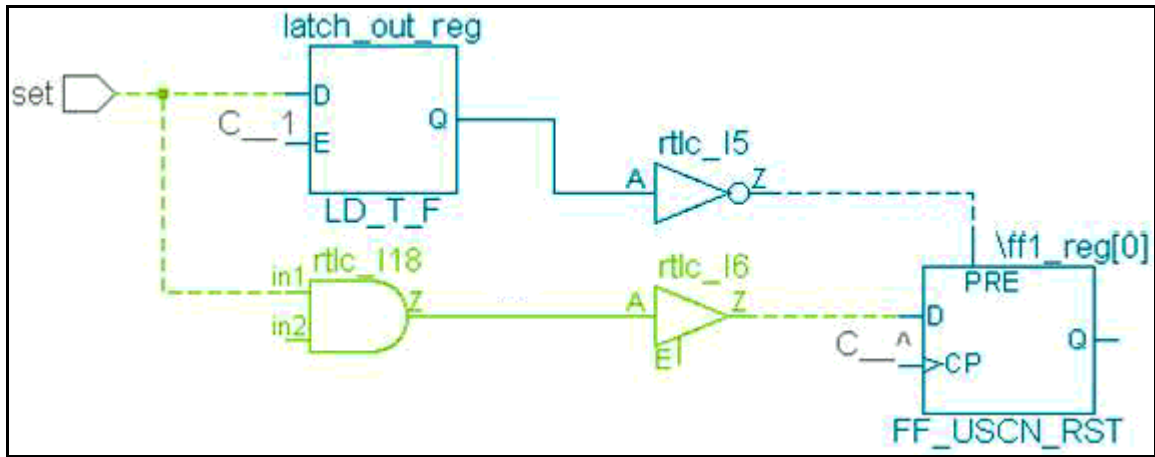
The violation messages for the `Info_testmode` and `Info_path` rules overlap the violation message for the `Async_12` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Async_12` rule.

To fix the violation, use different signals for data and reset pin of the flip-

flop.

Example Code and/or Schematic

Consider the following figure:



In the above example, reset pin of `ff1_reg[0]` is connected to the data pin of `latch_out_reg` and data pin of `ff1_reg[0]` is connected to the data pin of `latch_out_reg`.

Default Severity Label

Warning

Rule Group

Asynchronous

Reports and Related Files

None

Async_13

Ensure that the asynchronous set/reset pins of scan flip-flops are controllable to their inactive state during capture

When to Use

Use this rule to identify set/reset pins of scan flip-flops that are not controllable to their inactive state during capture mode.

Description

The Async_13 rule reports violation for set/reset pins of scan flip-flops that are not controllable to their inactive state during capture.

NOTE: *The Async_13 rule reports violation for scan flip-flops only.*

Rule Exceptions

The Async_13 rule does not report violations for the following cases:

- 'no scan' flip-flops (forced or inferred)
- Synthesis Redundant (SR) set/reset pins

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- [dft_maximum_number_of_rows_with_schematic](#): Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- [test_mode](#) (Optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (Optional): Defines the clocks of the design. For the Async_13 rule, use the constraint's `-testclock` argument.
- *force_no_scan* (Optional): Excludes flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] The source net '`<net-name>`' feeding '`<pin-type>`' pin of '`<num&type-obj>`' is not controllable to inactive state in capture mode

Arguments

- Name of the source net, `<net-name>`
- Type of the pin (set/reset), `<pin-type>`
- Number and type of object, `<num & type-obj>`
- Example of the object, `<obj-name>`

Potential Issues

A violation is reported when the set/reset pin is not controllable to inactive state during capture.

Consequences of Not Fixing

Not fixing the violation may reduce the observability or coverage of faults on data line as reset can not be deactivated.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the path from asynchronous (set/reset) source to the set/reset pin of the flip-flop.

You can also double-click on the violation message to view the spreadsheet that lists the details of the flip-flops whose set/reset pins are not controllable to their inactive state during capture.

To fix the violation, design the logic driving the set or reset pins such that the sets and resets of scan flip-flops may be controllable to their inactive state for proper data capture. This ensures that ATPG will be able to generate vectors to detect faults on data line of scan flip-flops (i.e. the flip-

flop is 'data capture ready').

Example Code and/or Schematic

Consider the following example:

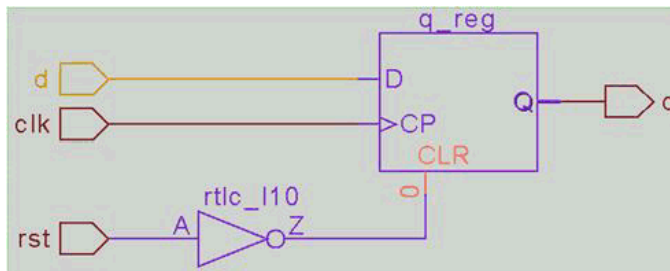
```
module test(d, clk, rst, q);
input d, clk, rst;
output q;
reg q;
```

```
always @(posedge clk or negedge rst)
  if (~rst)
    q <= 1'b0;
  else
    q <= d;
endmodule
```

Also, consider the following constraint definition for the above code:

```
current_design test
  clock -name clk -testclock
  testmode -name rst -value 1 -scanshift -invertInCapture
```

Following figure shows the corresponding schematic for the above design:



Default Severity Label

Warning

Rule Group

Asynchronous Rules

Reports and Related Files

No related reports or files.

Async_15

Separate control must be maintained for asynchronous set/reset pins of flip-flops and latches

When to Use

Use this rule to check whether a source net is driving the asynchronous pins of both flip-flop and latch.

Description

The Async_15 rule reports violation, if a source net is driving the asynchronous pins of both flip-flop and latch.

Having separate control for asynchronous pins of flip-flops and latches makes it easier to control flip-flops and latches separately.

The Async_15 rule works in the shift and capture modes. In each mode, it finds following three types of sources for all asynchronous pins, that is, latch or flip-flops:

- **Tied:** Source net has fixed simulation value in the current mode.
- **Sensitized:** Source net has unique unblocked path (fan-in trace) from asynchronous pins.
- **Sensitizable:** Source net has unblocked path (fan-in trace) from asynchronous pins.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.
- *clock* (optional): Defines the clocks of the design. For the Async_15 rule, use the constraint's `-testclock` argument.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] [`<mode_name>`] Via a sensitizable path, the source net '`<signal_name>`' is driving asynchronous pins of '`<count1>`' flip-flops and '`<count2>`' latches

Potential Issues

A violation is reported when a common source net is found in the sensitizable fanin cone, which is shared by asynchronous pins of both latches and flip-flops.

Consequences of Not Fixing

For more information, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information, see [How to Debug and Fix](#).

Message 2

[WARNING] [`<mode_name>`] Via a sensitized path, the source net '`<signal_name>`' is driving asynchronous pins of '`<count1>`' flip-flops and '`<count2>`' latches

Potential Issues

A violation is reported when a common source net is found in the sensitized fanin cone, which is shared by asynchronous pins of both latches and flip-flops.

Consequences of Not Fixing

For more information, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information, see [How to Debug and Fix](#).

Message 3

[WARNING] Via a tied path, the source net '`<signal_name>`' is driving asynchronous pins of '`<count1>`' flip-flops and '`<count2>`' latches

Potential Issues

A violation is reported when a common tied source net is found, which is shared by asynchronous pins of both latches and flip-flops.

Consequences of Not Fixing

It is difficult to control asynchronous pins of flip-flops and latches separately, which may add difficulties in downstream test tools.

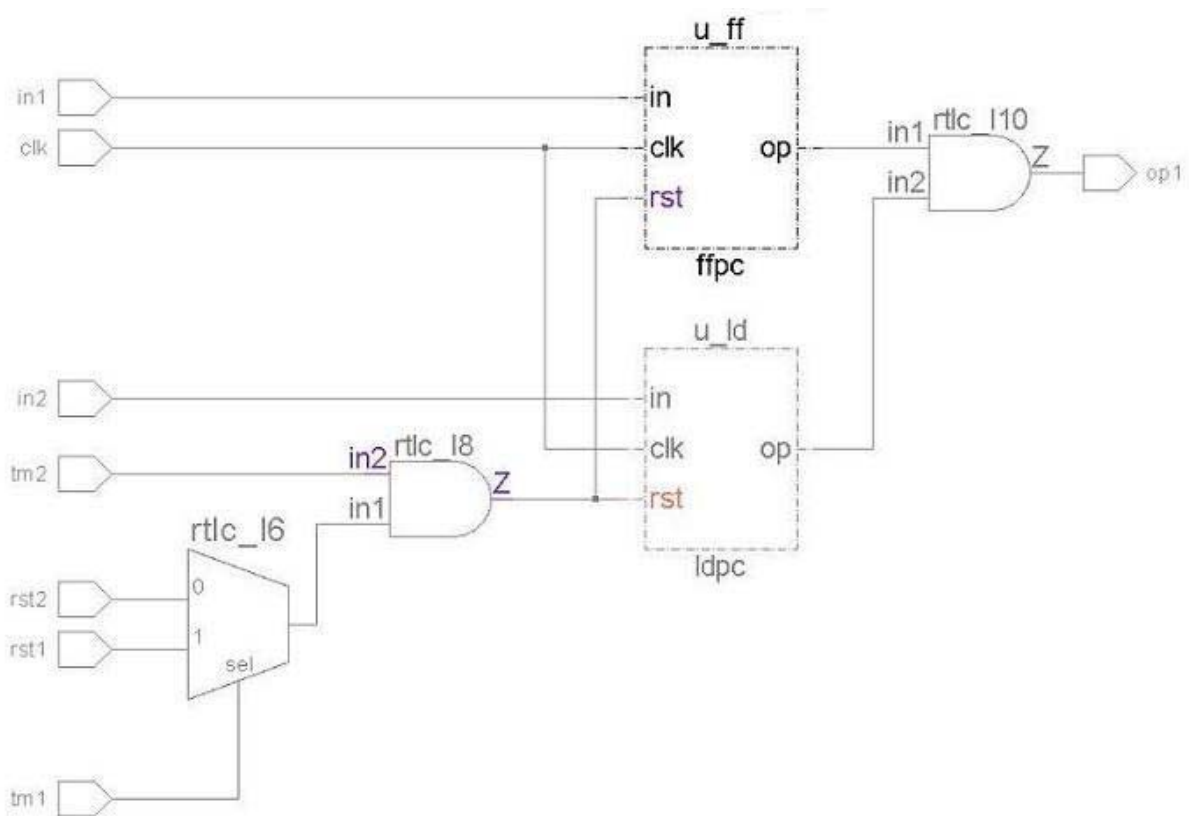
How to Debug and Fix

Check the Incremental Schematic to trace the path.

Modify the RTL to break the common connection. Alternatively, add constraints to block paths to either flip-flops or latches.

Example Code and/or Schematic

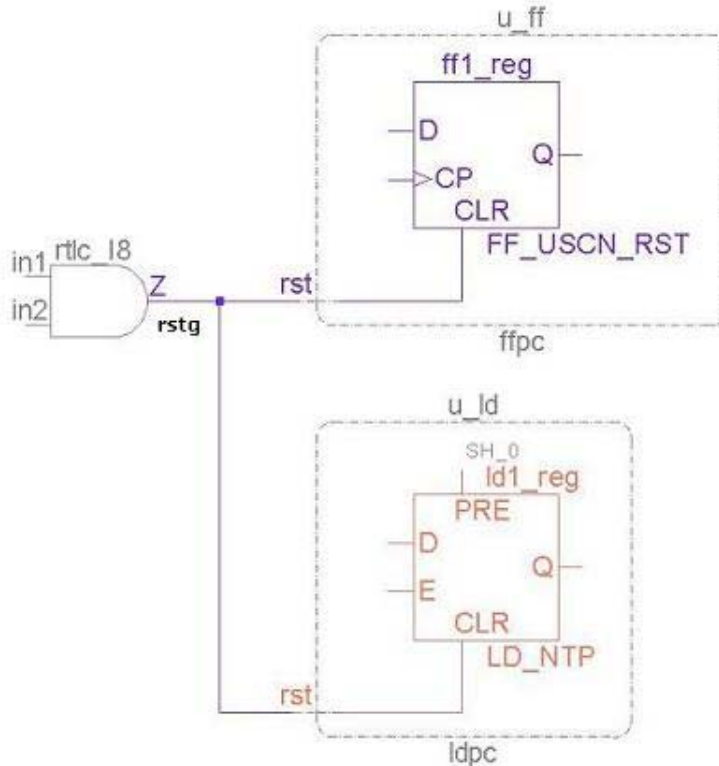
Consider the following sample modular schematic illustrating the three types of sources for all asynchronous pins, that is, sensitized, sensitizable, and tied:



The following examples show the incremental schematics for the above design.

Example 1

Consider the following incremental schematic illustrating the sensitized path:



Now, consider the following SGDC snippet with respect to the above design:

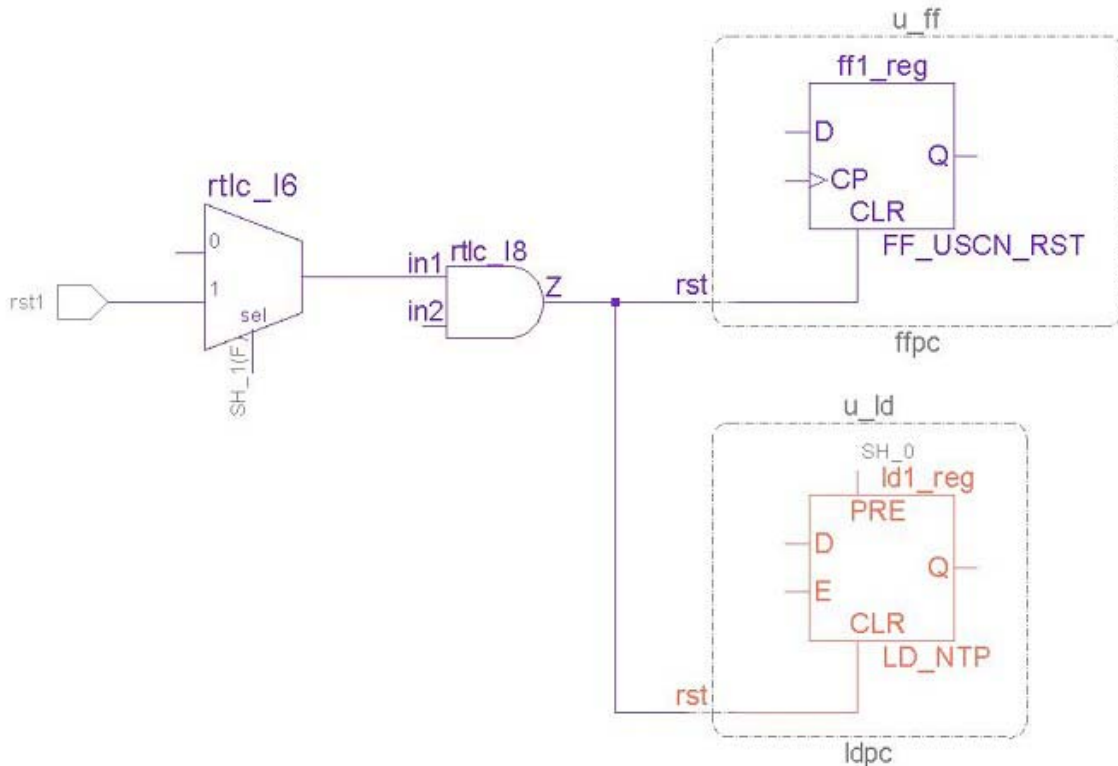
```
current_design top
  clock -name clk -testclock -atspeed
  test_mode -name tml -value 1 -invertInCapture
```

The Async_15 rule reports the following warning message for the above design, when the rule is operating in the shift mode:

[Shift mode] Via a sensitized path, the source net 'top.rstg' is driving asynchronous pins of '1' flip-flops and '1' latches

Example 2

Consider the following schematic:



Now, consider the following SGDC snippet with respect to the above design:

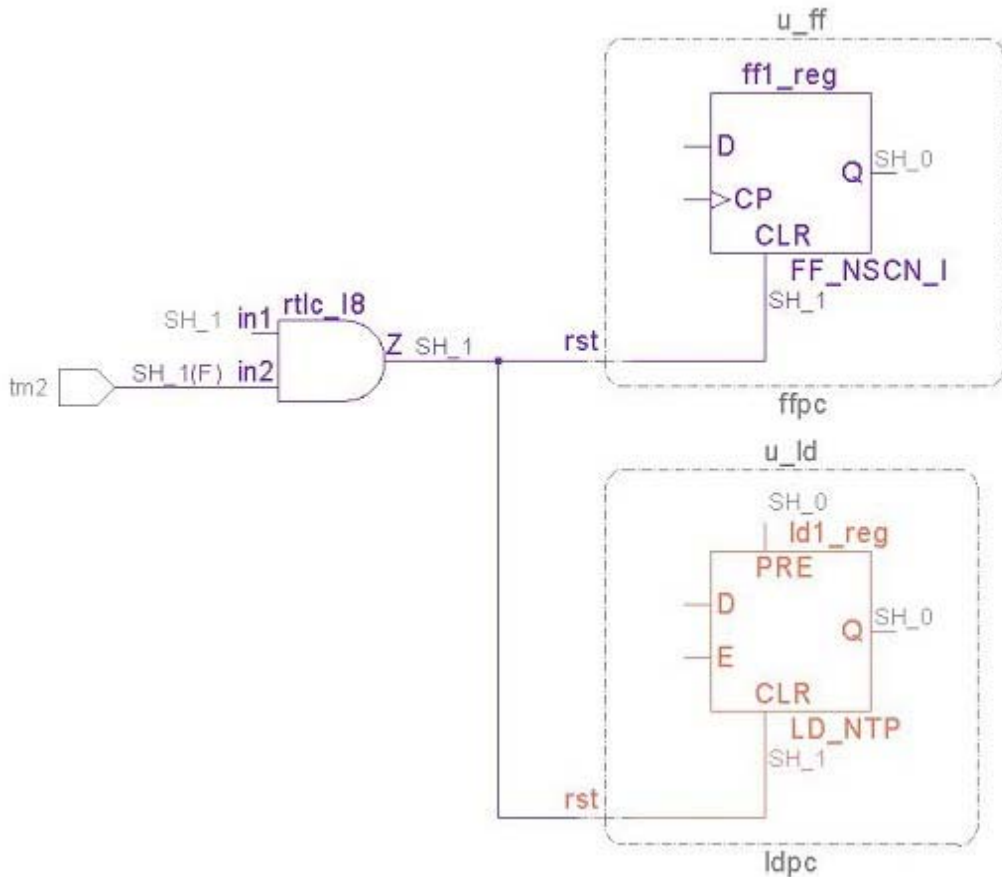
```
current_design top
  clock -name clk -testclock -atspeed
  test_mode -name tml -value 1 -invertInCapture
```

The Async_15 rule reports the following violation message for the above design, when the rule is operating in the shift mode:

[Shift mode] Via sensitizable path, the source net 'top.rst1' is driving asynchronous pins of '1' flip-flops and '1' latches, test.v

Example 3

Consider the following schematic:



Now, consider the following SGDC schematic:

```
current_design top
  clock -name clk -testclock -atspeed
  test_mode -name tm1 -value 1 -invertInCapture
  test_mode -name tm2 -value 1
  test_mode -name rst1 -value 1
  test_mode -name rst2 -value 0
```

The Async_15 rule reports the following violation message for the above design, when the rule is operating in the shift mode:

Via a tied path, the source net 'top.in1' is driving asynchronous pins of '1' flip-flops and '1' latches, test1.v, 1

Default Severity Label

Warning

Rule Group

Asynchronous Rules

Reports and Related Files

No related reports or files.

Async_16

Ensure that the set or reset sources are disabled or controllable by PI

When to Use

Use this rule to check whether the set or reset sources are disabled or controllable by PI.

Description

The Async_16 rule reports a violation for sequential cells, that is, flip-flops and latches, which do not meet the following criterion:

- Asynchronous pins (set/reset) of flip-flops are disabled during the scan-shift ATPG mode
- Asynchronous pins (set/reset) of flip-flops are directly controlled (sensitized) from primary input port.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

None

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.
- *clock* (optional): Defines the clocks of the design. For the Async_16 rule, use the constraint's `-testclock` argument.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] <no_of_pins> <pin_type> pin(s) are floating

Arguments

- Number of set or reset pins that are floating, <no_of_pins>
- Type of pin, whether it is set or reset, <pin_type>

Potential Issues

A violation is reported when set or reset pins are floating.

Consequences of Not Fixing

See [Consequences of Not Fixing](#).

How to Debug and Fix

See [How to Debug and Fix](#).

Message 2

[WARNING] <src_name> is used at both level s and driving
<no_of_level_high_set> level high set term(s)
<no_of_level_high_reset> level high reset term(s)
<no_of_level_low_set> level low set term(s)
<no_of_level_low_reset> level low reset term(s)

Arguments

Node path name, <src_name>

Potential Issues

A violation is reported when set or reset source is driving at both levels.

Consequences of Not Fixing

See [Consequences of Not Fixing](#).

How to Debug and Fix

See [How to Debug and Fix](#).

Message 3

[WARNING] <src_name> is active and driving
<no_of_level_high_set> level high set term(s)
<no_of_level_high_reset> level high reset term(s)
<no_of_level_low_set> level low set term(s)

<no_of_level_low_reset> level low reset term(s)

Arguments

Node path name, <src_name>

Potential Issues

A violation is reported when set or reset source is active.

Consequences of Not Fixing

See [Consequences of Not Fixing](#).

How to Debug and Fix

See [How to Debug and Fix](#).

Message 4

[WARNING] <src_name> is non controllable by Primary Input and driving <no_of_level_high_set> level high set term(s)
 <no_of_level_high_reset> level high reset term(s)
 <no_of_level_low_set> level low set term(s)
 <no_of_level_low_reset> level low reset term(s)

Arguments

Node path name, <src_name>

Potential Issues

A violation is reported for non-active sequential cells that are non-controllable by PI and driving at one level source.

Consequences of Not Fixing

Sets and resets which are derived from complex functional logic are hard to control and it cause ATPG tools to produce low coverage. Disabling the sets and resets during ATPG mode (tied to inactive values) or allowing control from primary inputs ensures sets/resets do not impact scan coverage.

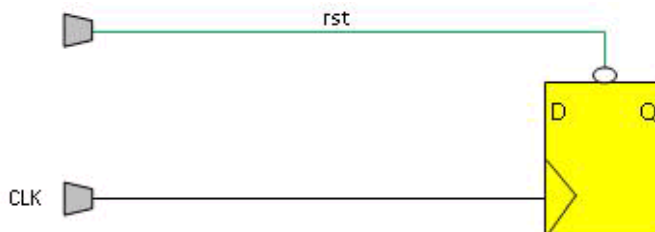
How to Debug and Fix

Check the Incremental Schematic to view the set/Reset sources which are active, driving at both levels, or are non-controllable from primary input along with path to set/resets for latches and flip-flops driven by them.

Example Code and/or Schematic

Example 1

Consider the following figure:



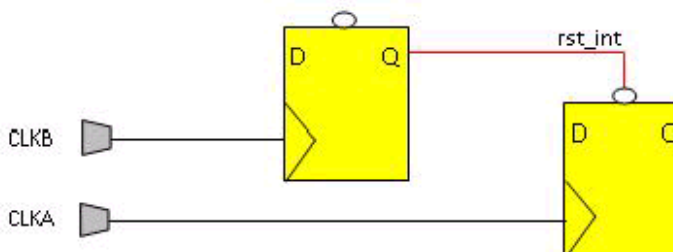
Also, consider the following SGDC setup corresponding to the above figure:

```
current_design top
clock -name top.CLK -testclock
test_mode -scanshift -name top.rst -value 1
```

In the above example, flip-flop's/latch's reset/set pin is driven from a signal coming directly from PI. Therefore, the *Async_16* rule does not report any violation for the reset signal, *rst*.

Example 2

Consider the following figure:



Also, consider the following SGDC setup corresponding to the above figure:

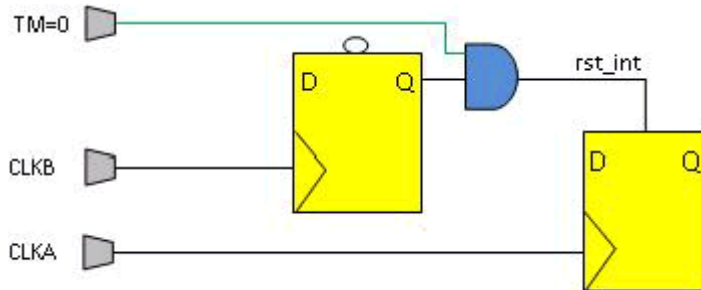
```
current_design top
clock -name top.CLKA -testclock
clock -name top.CLKB -testclock
```

In the above example, flip-flop's/latch's reset/set pin is driven by an

internally generated reset/set signal, `rst_int`. Therefore, the `Async_16` rule reports a violation for `rst_int`.

Example 3

Consider the following figure:



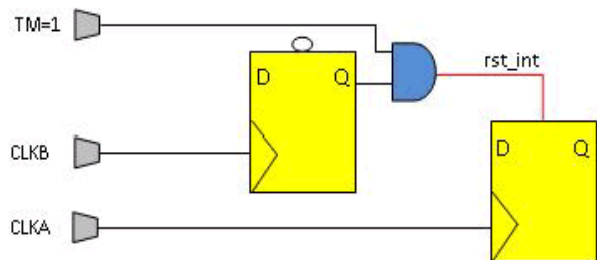
Also, consider the following SGDC setup corresponding to the above figure:

```
current_design top
clock -name top.CLKA -testclock
clock -name top.CLKB -testclock
test_mode -name top.TM -value 0
```

In the above example, flip-flop's/latch's reset/set pin is driven by an internally generated reset/set signal, `rst_int`, which is disabled during scan shift mode. Therefore, the `Async_16` rule does not report any violation for `rst_int`.

Example 4

Consider the following figure:



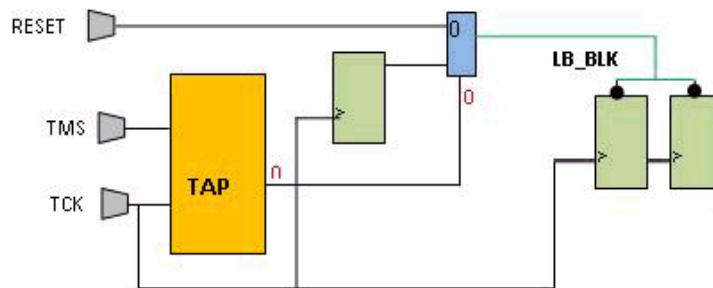
Also, consider the following SGDC setup corresponding to the above figure:

```
current_design top
clock -name top.CLKA -testclock
clock -name top.CLKB -testclock
test_mode -name top.TM -value 1
```

In the above example, flip-flop's/latch's reset/set pin is driven by an internally generated reset/set signal, `rst_int`, which is not disabled during scan shift mode. Therefore, the *Async_16* rule will report violation for `rst_int`.

Example 5

Consider the following figure:



Also, consider the following SGDC setup corresponding to the above figure:

```
current_design top
clock -name top.TCK -testclock
## Define Test Mode/Initialization Condition
## Test Mode signal generated by TAP after 5 clock cycle
test_mode -name top.TCK -value 0101010101
test_mode -name top.TMS -scanshift -value 1111001100
```

In the above example, flip-flop's/latch's reset/set pin is driven by an internally generated reset/set signal, `rst_int`, which is not disabled during scan shift mode. Therefore, the *Async_16* rule will not report any violation.

Default Severity Label

Error

Asynchronous Rules

Rule Group

Asynchronous Rules

Reports and Related Files

No related reports or files.

Async_17

Report all the sources, which drive asynchronous preset, clear and clock pins.

When to Use

Use this rule to detect all sources, which drive asynchronous preset, clear, and clock pins.

Description

For every flip-flop instance, the following is traced:

- fanin cone(s) of PRESET and CLEAR pins
- fanin cone of CLOCK pin

The ports, terminals, or the nets, which are common to both the fan-ins are considered as source points.

The rule iterates over the list of source points and performs fanout.

Also, the rule highlights paths from source point to one of the PRESET pins (if applicable), one of the CLEAR pins (if applicable), and one of the CLOCK pins.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

[WARNING] Signal <net_name> drives <string>. First diverging node: <name_of_diverging_node><Spreadsheet path>

Arguments

- Name of net, <net_name>
- String consisting of "clock-pins" , "preset-pins" and "clear-pins" along with the number of times these are encountered, <string>
- Name of first diverging net, <name_of_diverging_node>

- Spreadsheet path, <spreadsheet_path>

Potential Issues

The violation message is reported when the rule detects a signal, which drives asynchronous pin(s) and clock pin(s).

Consequences of Not Fixing

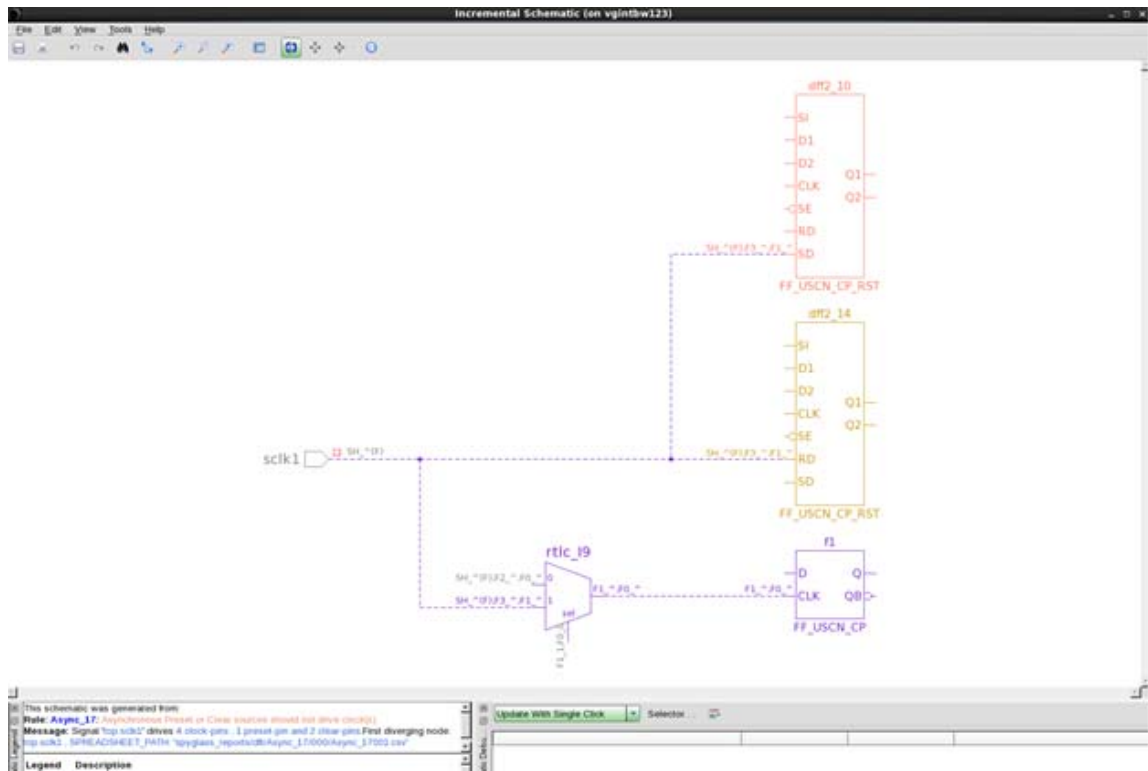
If the reset is used as a clock, it may cause race condition resulting into non-robust test.

How to Debug and Fix

Check the Incremental Schematic and update the logic accordingly.

Example Code and/or Schematic**Example 1**

Consider the following Incremental Schematic:

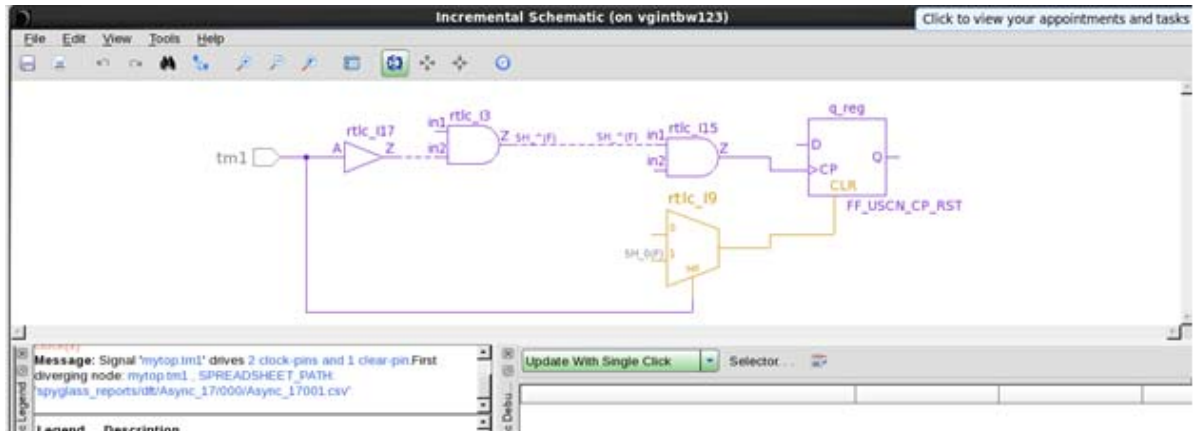


In the above example, the rule reports violation for sclk1 as it is also driving the set/reset pin. This will corrupt the value of the flip-flops.

Example 2

Consider the following incremental schematic:

Asynchronous Rules



In the above example, the rule reports a violation for tm1 as it is shared between and reset and clock.

Default Severity Label

Error

Rule Group

Asynchronous Rules

Reports and Related Files

No related reports or files.

Atspeed Test Rules

Overview

The At-Speed Test rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>Atspeed_01</i>	Scannable flip-flops whose clock is not controlled by a PLL in testmode
<i>Atspeed_03</i>	Combinational paths crossing different functional clock domains that are put in the same test clock domain
<i>Atspeed_04</i>	Combinational paths in the same functional clock domains that are put in different test clock domain
<i>Atspeed_05</i>	Blocked false paths or multi-cycle paths in the functional mode
<i>Atspeed_06</i>	Paths crossing the asynchronous clock domains that are not blocked
<i>Atspeed_07</i>	Pairs of at-speed testclocks that have the same gating conditions
<i>Atspeed_08</i>	Clock generator nets that do not have the same logic in all fan-out cones or are not driving functional clock net
<i>Atspeed_09</i>	Highlights the root cause for bad controllability at the data pin of a scannable flip-flop
<i>Atspeed_10</i>	Invalid clock sources in the design
<i>Atspeed_11</i>	Clock sources that are not controlled by an at-speed clock
<i>Atspeed_12</i>	Flip-flops that are not controlled by a single clock frequency during at-speed testing
<i>Atspeed_13</i>	Expected frequencies that do not reach the terminal of a clockshaper cell
<i>Atspeed_14</i>	Test clocks that are used as data signals
<i>Atspeed_15</i>	Combinational loops detected in capture mode

Atspeed Test Rules

Rule	Description
<i>Atspeed_16</i>	Tristate logic that is not allowed
<i>Atspeed_17_capture</i>	Pins of clockshaper cell that has incorrect enabling condition in the capture mode
<i>Atspeed_17_captureatspeed</i>	Pins of clockshaper cell that has incorrect enabling condition in the captureATspeed mode
<i>Atspeed_17_shift</i>	Pins of clockshaper cell that has incorrect enabling condition in the scanshift mode
<i>Atspeed_19</i>	Non-transparent latches in capture mode
<i>Atspeed_20</i>	Asynchronous set/reset pins of all the flip-flops should be fully controllable during capture atspeed mode.
<i>Atspeed_22</i>	No "ANDing" (Merging) of test clocks in capture atspeed mode.
<i>Atspeed_23</i>	The clock pin of memories must pass through a pll.
<i>Atspeed_24</i>	Clock should not be shaped more than once.
<i>Atspeed_25</i>	Reconverging combinational paths
<i>Atspeed_26</i>	An atspeed clock should not feed another atspeed clock
<i>Atspeed_27</i>	An async pin of a flip-flop should be driven by a single design node
<i>Atspeed_29</i>	Detects supply of bad clock during atspeed test.
<i>Atspeed_30</i>	Reports the presence of combinational reconvergence to flip-flop clock pin
<i>Atspeed_31</i>	Ensure that each clock shaper clock pin is directly controlled via PLL in the atspeed mode.
<i>Atspeed_32</i>	Reports the presence of cascaded reset re-convergence to asynchronous pins of flip-flop
<i>Atspeed_33</i>	All latch inputs should be driven by controllable sources in the capture-atspeed mode
<i>Atspeed_34</i>	Reports flip-flops having feedback from q-terminal to d-terminal

Atspeed_01

Ensure that the source clock for all scannable flip-flops pass through a PLL

When to Use

Use this rule to ensure that the clock inputs of all scannable flip-flops are driven by a PLL, thereby, controlling the frequency.

Description

The Atpspeed_01 rule highlights all the scannable flip-flops where clocks are driven by sources other than PLL.

The rule Atpspeed_01 violates for the following cases:

- The flip-flop is scan (when the test-clock reaches in the shift mode).
- The flip-flop also gets test-clock in the Atpspeed mode.
- The clock coming in Atpspeed mode as in (2), is NOT from a PLL.

Prerequisites

Async_07, Clock_11, Scan_08 of dft for scannability

Rule Exceptions

If no constraints is supplied and the dft_scan_ready goal is run, then a Clock_11 violation is reported on the OR gate. As a result, the flip-flop is not considered scannable. If Atpspeed_01 is selected in the same run as Scan Ready, the Atpspeed_01 rule does not report any violation since it only flags the cases where a scannable flip-flop is not clocked by a PLL clock.

Default Weight

10

Language

Verilog, VHDL

Method

```
Foreach scannable flip-flop
    if it does not receive a clock through PLL report a violation
EndFor
```

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

Message

[WARNING] Clock source '<net-name>' does not get atspeed clock through a PLL (<num> flip-flops affected)

Arguments

- Name of the clock source, *<net-name>*
- Number of flip-flops driven by clock source, *<num>*

Potential Issues

Violation may arise when no PLL is defined or when PLL is not on the clock path.

Consequences of Not Fixing

Flip-flops may not get atspeed (high frequency) clock as atspeed clocks are PLL generated.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the path from flip-flop to a clock source without containing a pll. This path causes the Atspeed_01 rule to report a violation.

To fix the violation, see [Example Code and/or Schematic](#) Section.

Example Code and/or Schematic

Consider the following figure:

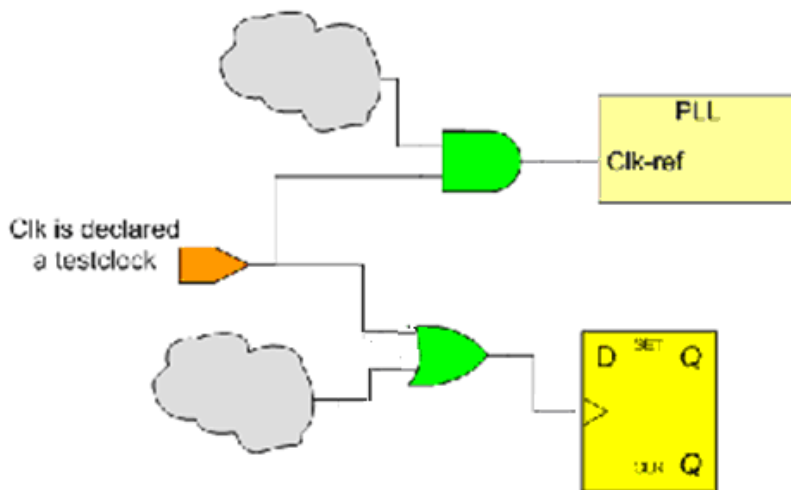


FIGURE 8. Design with Clock_11 Violation

The above figure illustrates a design with Clock_11 violation.

Now, consider the following figure:

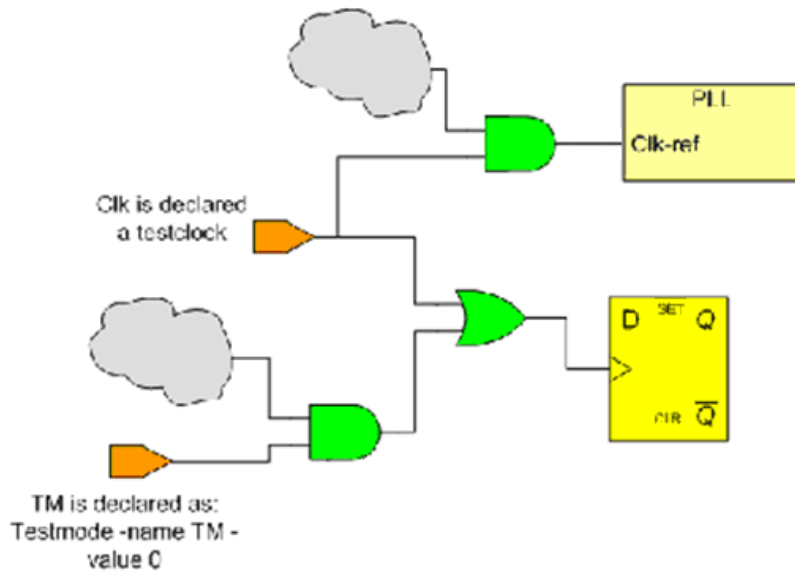


FIGURE 9. Design with Clock_11 Violation Fixed

In the above figure, the Clock_11 violation is removed and the flip-flop is considered as scannable. However, the `Atspeed_01` rule still reports a violation in this case because though the flip-flop is scannable, its clock is not derived from a PLL clock.

Also, consider the following figure:

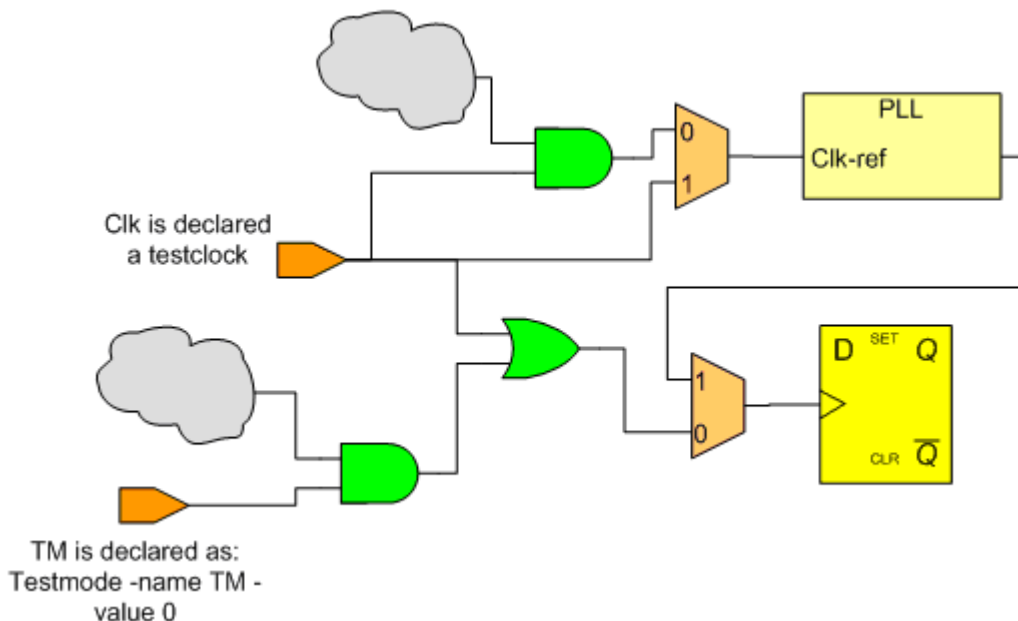


FIGURE 10. Good PLL ref-clock Design

In the above figure, two mux selector pins are declared as captureATspeed testmode with value 1. In this case, the circuit will pass the Atspeed_01 rule because an atspeed clock controls the PLL input, the flip-flop is scannable and is clocked by a PLL clock for atspeed capture.

Default Severity Label

Warning

Rule Group

Atspeed rules

Reports and Related Files

None

Atspeed_03

Ensure that asynchronous logic in the functional mode does not interact synchronously in the capture at-speed mode

When to Use

Use this rule to detect combinational paths that cross functional clock domains, which are in the same atspeed clock domain.

Description

The Atspeed_03 rule reports violation for asynchronous logic in the functional mode that does interact synchronously in the capture at-speed mode.

The Atspeed_03 rule reports violation only when there is data interaction between the domain pair, that is, synchronous and asynchronous domains. In case of no interaction, violation is not reported because non-interacting synchronous-asynchronous domains do not create test issues.

The Atspeed_03 rule generates the [atspeed_03_rpt](#) report, which lists all cross-domain paths between flip-flops that are asynchronous in functional domain and synchronous in atspeed testmode domain.

Prerequisites

Functional mode ([test_mode](#) -functional) must also be defined completely so that the flip-flop gets a unique functional clock.

Rule check is not performed for flip-flops that do not have a functional and atspeed clock.

Rule Exceptions

The Atspeed_03 rule does not report violation if an atspeed clock domain crossing path is specified as a false path.

Default Weight

10

Language

Verilog, VHDL

Method

Find pairs of functional domains whose functional clocks are asynchronous
If their at-speed test clocks are asynchronous then skip this pair

A pre-analysis should be performed to associate each functional clock with exactly one at-speed clock. Any violating domain is not a candidate for this rule checking.

Display the unblocked bath

Otherwise

Find paths between these domains in functional mode

NOTE: *If BB clocks are in a domain, then we use the BB constraint to get the BB ports as starting and ending points for paths*

If paths do not exist then skip this pair

Otherwise // here if we have async pair w/ syn tlcks

Simulate testmode -capture

Find unblocked paths between these domains

If none found then skip this pair

Otherwise // here if paths exist in testmode

Find longest & shortest unblocked between this pair

Print violation message

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks. The supported options are: -name, -testclock, and -atspeed.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *false_path* (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.

Messages and Suggested Fix

The following message is displayed for the Atspeed_03 rule:

[WARNING] Start point '<net-name1>' (functional domain: <net-name2>, atspeed domain: <net-name3>) and end point '<net-name4>' (functional domain: <net-name5>, atspeed domain: <net-name6>) are asynchronous in functional mode but synchronous in atspeed mode

Arguments

- Name of source flip-flop <net-name1>
- Name of functional clock domain of source flip-flop <net-name2>
- Name of atspeed clock domain of source flip-flop <net-name3>
- Name of destination flip-flop <net-name4>
- Name of functional clock domain of destination flip-flop <net-name5>
- Name of atspeed clock domain of destination flip-flop <net-name6>

Potential Issues

Violations may arise due to one of the following reasons:

- Same clock is driving the flip-flops in the functional mode.
- Different atspeed clock are driving the flip-flops in atspeed mode.
-

Consequences of Not Fixing

If the asynchronous logic does not interact synchronously in the capture at-speed mode, extra effort is required to close timing in the test mode compared to the functional mode.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the path from one flip-flop (say A) to another flip-flop (say B), which are driven by same functional and atspeed clock.

Probe the fan-in clock cone of flip-flop A. Overlay (in auxiliary violation) the Info_atSpeedClock violation for the atspeed clock mentioned for flip-flop A to figure out how atspeed clock is driving flip-flop A. Repeat the same for flip-flop B to figure out how the same atspeed clock is driving flip-flop B.

Also, probe the fan-in clock cone of flip-flop A and flip-flop B till the cones

converges to figure to find the functional clock for the flip-flop A and flip-flop B.

To fix the violation, see [Example Code and/or Schematic](#) section.

Example Code and/or Schematic

Example 1

Consider the following example code:

```
module test(d1, d2, fck1, fck2, tck, tm, q1, q2);
input d1, d2;
input fck1, fck2;
input tck;
input tm;
output q1, q2;
reg q1, q2;
wire ck_to_ff1 = (tm)? tck : fck1;
wire ck_to_ff2 = (tm)? tck : fck2;
always @(posedge ck_to_ff1)
    q1 <= d1;
always @(posedge ck_to_ff2)
    q2 <= q1;
endmodule
```

```
// file: Atspeed_03.sgdc
current_design test
clock -name "fck1"
clock -name "fck2"
clock -name "tck" -testclock -atspeed
testmode -name tm -value 1
testmode -name tm -value 0 -functional
```

In this situation, the Atspeed_03 rule violation is rectified, if you allow asynchronous behavior of the logic in the 'at-speed' mode.

The new design format is as shown below:

```

module test(d1, d2, fck1, fck2, tck1, tck2, tm, q1, q2);
input d1, d2;
input fck1, fck2;
input tck1, tck2;
input tm;
output q1, q2;
reg q1, q2;
wire ck_to_ff1 = (tm)? tck1 : fck1;
wire ck_to_ff2 = (tm)? tck2 : fck2;
always @(posedge ck_to_ff1)
    q1 <= d1;
always @(posedge ck_to_ff2)
    q2 <= q1;
endmodule

```

```

// file: Atspeed_03_rectified.sgdc
current_design test
clock -name "fck1"
clock -name "fck2"
clock -name "tck1" -testclock -atspeed
clock -name "tck2" -testclock -atspeed
testmode -name tm -value 1

```

Example 2

Consider [Figure 11](#) and [Figure 12](#):

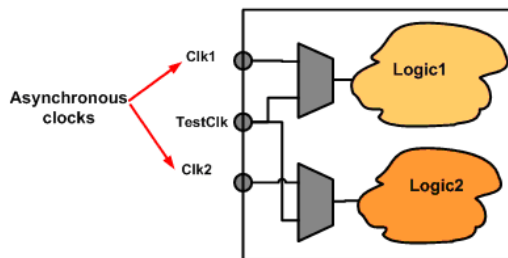


FIGURE 11. Bad Asynchronous Design

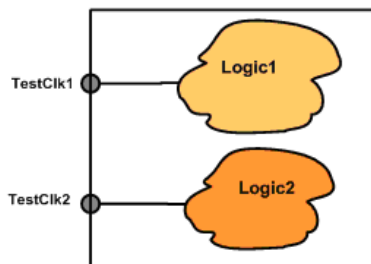


FIGURE 12. Good Asynchronous Design

The above figures illustrates that combining functional domains into the same testclock domain where no paths exist between these domains does not cause a violation.

Example 3

The following figure illustrates the faults on paths that originate at a flip-flop, FF1, clocked in system mode by clk1 and terminates at a flip-flop, FF2, clocked in system mode by clk2:

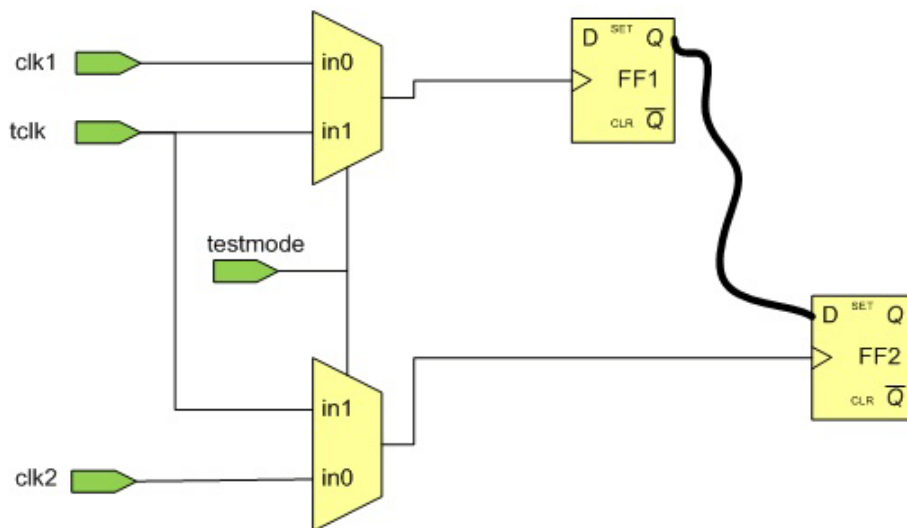


FIGURE 13. Astspeed_03 example

Atspeed Test Rules

When the Atspeed_03 rule reports a violation for the above design, an ATPG tool may test for faults on paths. For such a circuit, the ATPG test will launch a transition at FF1 with tclk and attempt capture at FF2 with the same tclk. Since the path was never intended to transmit data with two pulses of the same clock, such tests are as likely to fail. Therefore, the design should not combine the two system domains into one test domain.

Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

[atspeed_03_rpt](#) Report

Atspeed_04

Do not clock the synchronously interacting logic in the functional mode by asynchronous clocks in the capture at-speed mode

When to Use

Use this rule to prevent excessive test patterns and test time by ensuring that functional domains are not split into multiple test clock domains.

Description

The Atspeed_04 rule reports violation if the synchronously interacting logic in the functional mode is clocked by asynchronous clocks in the capture at-speed mode.

The Atspeed_04 rule reports violation only when there is data interaction between the domain pair, that is, synchronous and asynchronous domains. In case of no interaction, violation is not reported because non-interacting synchronous-asynchronous domains do not create test issues.

The Atspeed_04 rule generates the [atspeed_04_rpt](#) report, which lists all cross-domain paths between flip-flops that are synchronous in functional domain and asynchronous in at-speed testmode domain.

Prerequisites

Functional mode ([test_mode](#) -functional) must also be defined completely so that the flip-flop gets a unique functional clock.

Rule check is not performed for flip-flops that do not have a functional and atspeed clock.

Rule Exceptions

Disable the Atspeed_04 rule, if an asynchronous test clock is designed to partition scan chains/blocks to consume lower power during at-speed test operation.

The Atspeed_04 rule does not report violation if an atspeed clock domain crossing path is specified as a false path.

Default Weight

10

Language

Verilog, VHDL

Method

Find pairs of functional domains whose functional clocks are synchronous

If their at-speed test clocks are synchronous then skip this pair

Any required 'MUST HAVE' analysis where each functional clock is associated with exactly one at-speed clock. Any violating domain is not a candidate for this rule checking.

Otherwise

Find paths between these domains in functional mode

NOTE: *If BB clocks are in a domain, then we use the BB constraint to get the BB ports as starting and ending points for paths*

If paths do not exist then skip this pair

Otherwise // here if we have sync pair w/ async tlcks

Simulate testmode

Find unblocked paths between these domains

If none found then skip this pair

Otherwise // here if paths exist in testmode

Find longest & shortest unblocked between this pair

Print violation message

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *false_path* (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Atspeed_04 rule:

[WARNING] Start point '*<start point name>*' (functional domain: *<func dom name>*, atspeed domain: *<atspeed dom name>*) and end point '*<end point name>*' (functional domain: *<func dom name>*, atspeed domain: *<atspeed dom name>*) are synchronous in functional mode but asynchronous in atspeed mode

Arguments

- Name of source flip-flop *<net-name1>*
- Name of functional clock domain of source flip-flop *<net-name2>*
- Name of atspeed clock domain of source flip-flop *<net-name3>*
- Name of destination flip-flop *<net-name4>*
- Name of functional domain clock of the destination flip-flop *<net-name5>*
- Name of atspeed clock domain of destination flip-flop *<net-name6>*

Potential Issues

Violations may arise due to one of the following reasons:

- Different atspeed clock is used incorrectly in the atspeed mode.
- Same functional clock is used incorrectly in the functional mode.

Consequences of Not Fixing

If the synchronously interacting logic is not clocked in the functional mode by asynchronous clocks in the capture at-speed mode, inter-domain logic between the clocks that have a synchronous interaction in the functional mode are not tested.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental

Schematic highlights the path from one flip-flop (say A) to another flip-flop (say B), which are driven by same functional clock and same atspeed clock. Probe the fan-in clock cone of flip-flop A. Overlay (in auxiliary violation) the Info_atSpeedClock violation for the atspeed clock mentioned for flip-flop A to figure out how atspeed clock is driving the flip-flop. Repeat the same for flip-flop B to figure out how the same atspeed clock is driving flip-flop B. Also, probe the fan-in clock cone of flip-flop A and flip-flop B till the cones converges to figure to find the functional clock for flip-flop A and flip-flop B.

To fix the violation, see [Example Code and/or Schematic](#) section.

Example Code and/or Schematic

Consider the following figure:

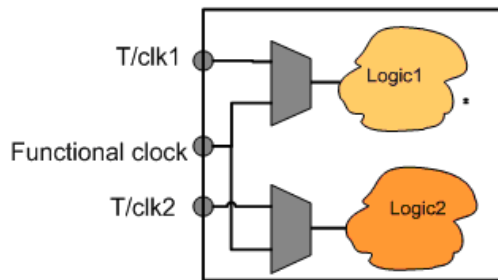


FIGURE 14. Bad Synchronous Design

In the above figure, a functional clock is controlling both Logic1 and Logic2 with two test clocks being used for testing. Hence, the figure depicts a bad design scenario and Atspeed_04 rule reports a violation in this condition.

Now, consider the following figure:

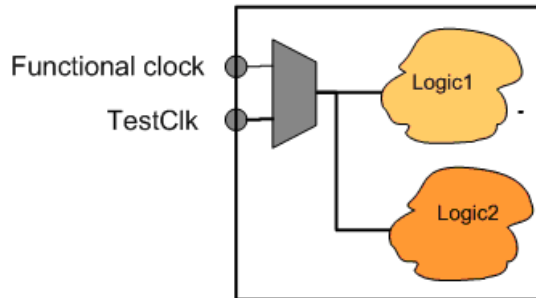


FIGURE 15. Good Synchronous Design

In the above figure, a functional clock is controlling both `Logic1` and `Logic2` and only one testclock, `TestClk`, is used for testing. Hence, the `Atspeed_04` rule does not report a violation in this condition.

Default Severity Label

Warning

Rule Group

Atspeed rules

Reports and Related Files

[atspeed_04_rpt](#)

Atspeed_05

Ensure that all false paths or multi-cycle paths in the functional mode are blocked in the capture at-speed mode

When to Use

Use this rule to detect start and end points in all false or multi-cycle paths in the functional mode that are not blocked in the capture at-speed mode.

Description

The Atpspeed_05 rule reports violation for those false paths or multi-cycle paths in the functional mode that are not blocked in the capture at-speed mode.

NOTE: *The Atpspeed_05 rule reports violation for all unblocked false path faults in the design. However, its in not mandatory that a design will have a false path fault, if the Atpspeed_05 rule reports a violation.*

Also, the Atpspeed_05 rule generates a CSV report for each violation.

Default Weight

10

Language

Verilog, VHDL

Rule Exceptions

All the paths ending at no_scan flip-flops, either forced or inferred, are ignored for rule checking.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dsm_check_unblocked_mcp](#): The default value is off. Set the value of the parameter to on or same_domain to check for the MCPs in the same domain or different domains.

Constraint(s)

- [atspeed_clock_frequency](#) (optional): Use this constraint to specify frequencies associated with a testclock.

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *false_path* (mandatory): Use this constraint to specify false paths that are to be excluded from at-speed testing.
- *sg_multicycle_path*: Use this constraint to specify multi-cycle paths, which you want to exclude from the at-speed testing.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[ERROR] '<number of false paths>' false paths starting from node '<node name>', found in same clock domain, are not blocked in capture at speed mode

Potential Issues

Violation may arise if incorrect or incomplete test_mode is specified that would have otherwise blocked the path.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging, see [How to Debug and Fix](#).

Message 2

[ERROR] '<number of false paths>' false paths terminating at node '<node name>', found in same clock domain, are not blocked in capture at speed mode

Potential Issues

Violation may arise if incorrect or incomplete test_mode is specified that would have otherwise blocked the path.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging, see [How to Debug and Fix](#).

Message 3

[WARNING] <number of false paths>' false paths starting from node '<node name>', found in different clock domains, are not blocked in capture at speed mode

Potential Issues

Violation may arise if incorrect or incomplete test_mode is specified that would have otherwise blocked the path.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging, see [How to Debug and Fix](#).

Message 4

[WARNING] <number of false paths>' false paths terminating at node '<node name>', found in different clock domains, are not blocked in capture at speed mode

Potential Issues

Violation may arise if incorrect or incomplete test_mode is specified that would have otherwise blocked the path.

Consequences of Not Fixing

Unblocked false or multi-cycle paths in the functional mode can result in potential timing violations. This can prohibit the use of test compression techniques using MISRs in signature analysis

How to Debug and Fix

View the Modular Schematic of the violation message. The Modular Schematic highlights the false paths, which are not blocked in the capture-atspeed mode.

To fix the violation, either remove the `false_path/sg_multicycle` path constraint. If it is correct, block the path using the `test_mode` constraint.

Example Code and/or Schematic

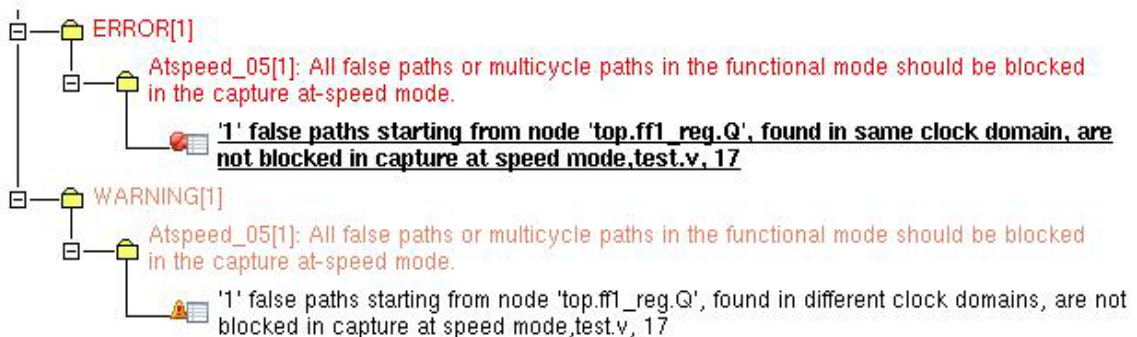
Consider the following SGDC declaration where path from clock, `clk_a` to clock, `clk_b`, is declared as a false path:

```

1  current_design top
2
3  clock -name clk_a -atspeed -testclock -domain a
4  clock -name clk_b -atspeed -testclock -domain a
5  clock -name clk_c -atspeed -testclock
6
7  false_path -from ff1_reg.CP -to ff3_reg.D
8  false_path -from clk_a -to clk_b
9

```

For the above declaration, Atspeed_05 rule reports the following violation messages:



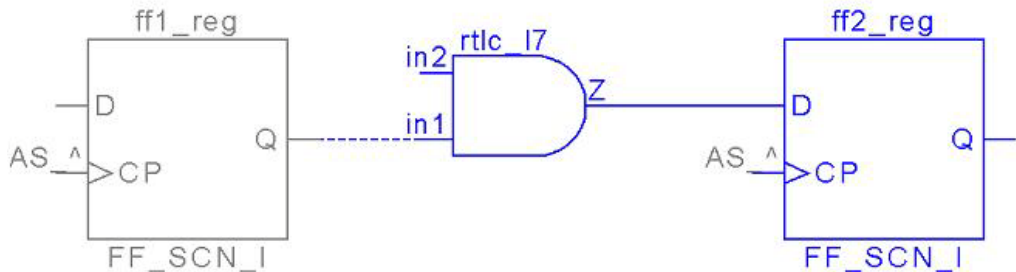
The Atspeed_05 rule reports an error message when the *Clocks defined are in the same clock domain*. Also, the rule reports a warning message when the *Clocks defined are in the different clock domain*.

Clocks defined are in the same clock domain

Clicking on the message displays the following report in the spreadsheet viewer:

	B	C	D	E
	<u>Start Node</u>	Start-Domain	To Node	Dest-Domain
1	top.ff1_reg.Q	a	top.ff2_reg.D	a

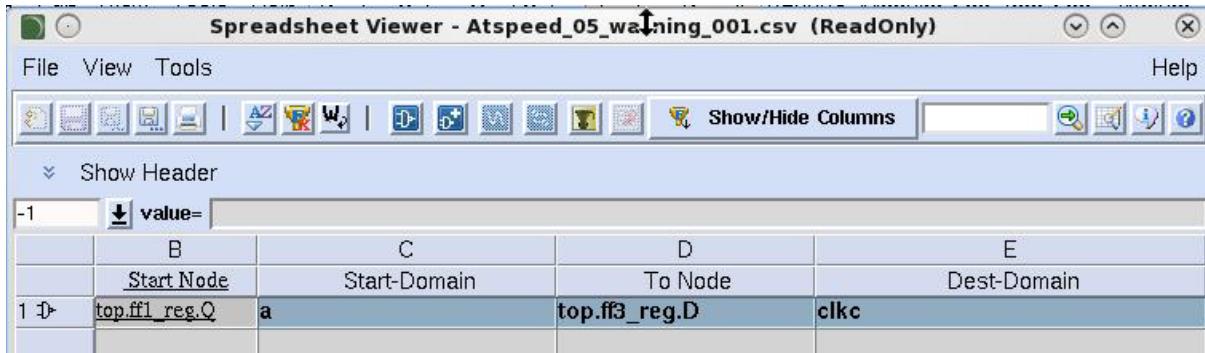
Double-clicking on a row in the report to view the Incremental Schematic as shown in the following figure:



In the above schematic, the path from ff1_reg to ff2_reg is a false path.

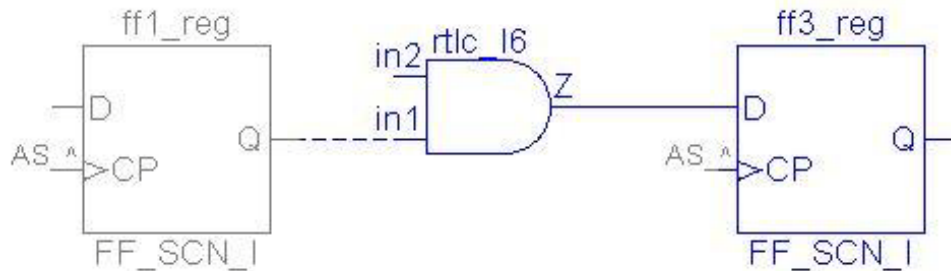
Clocks defined are in the different clock domain

Clicking on the message displays the following report in the spreadsheet viewer:



	B	C	D	E
	<u>Start Node</u>	Start-Domain	To Node	Dest-Domain
1	<u>top.ff1_reg.Q</u>	a	top.ff3_reg.D	clkc

Double-click on a row in the report to view the Incremental Schematic as shown in the following figure:



In the above schematic, the path from ff1_reg to ff3_reg is a false path.

Multi-cycle paths

Assume that the flip-flops and state machines illustrated in the following figure are scanned. For the above example, also assume that the path from FF1, u1, u2 and FF2 is a multi-cycle path controlled by the state machine:

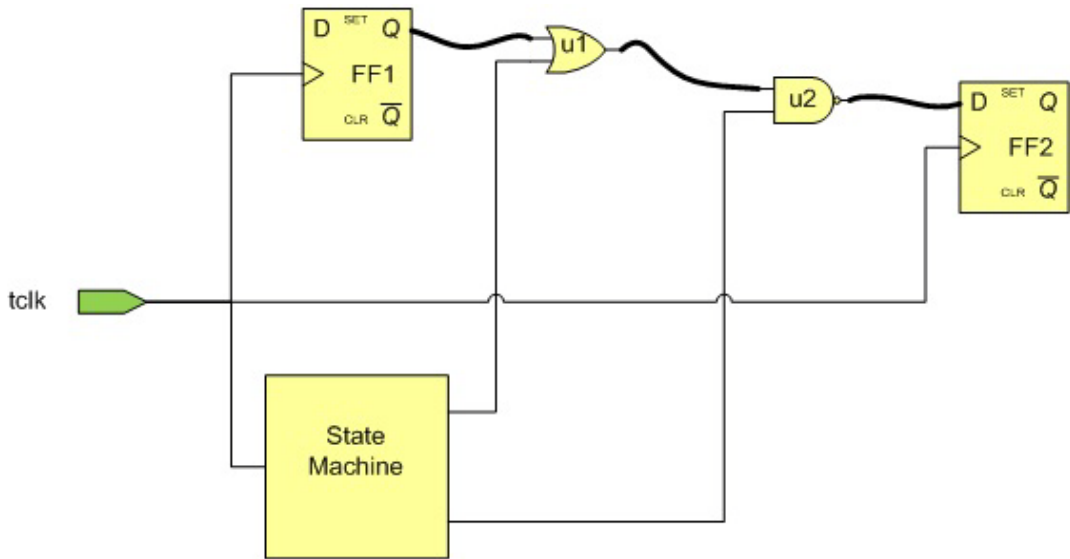


FIGURE 16. Multi-cycle Example

For the above example, an ATPG tool may generate an atspeed test for this path that would launch at FF1 and capture on FF2 using two consecutive high speed test clocks. Since this path was never intended to meet high timing constraints, such a test may fail and thereby cause rejection of a properly working chip.

Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

None

Atspeed_06

Ensure that all paths crossing asynchronous clock domains are blocked.

When to Use

Use this rule to detect the path between the two terminals, which are launched and captured by asynchronous atspeed clocks.

Description

The Atspeed_06 rule highlights the path where an atspeed clock domain crossing occurs. The rule checks for domain crossing between scannable flip-flops and abstract ports.

The Atspeed_06 rule generates a CSV report for each violation.

Prerequisites

Async_07, Clock_11, Scan_08 of dft for scannability.

Rule Exceptions

The Atspeed_06 rule does not report a violation, if an atspeed clock domain crossing path is specified as a false path.

Default Weight

10

Language

Verilog , VHDL

Method

```
For each scan flip-flop and abstract_port
do fan-out traversal
If it hits a scan flip-flop in another domain give a violation
End for
```

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- [atspeed_clock_frequency](#) (optional): Use this constraint to specify frequencies associated with a testclock.
- [clock](#) (optional): Use this constraint to declare clock pins declared as testclocks.
- [clock_shaper](#) (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- [pll](#) (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [false_path](#) (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.
- [abstract_port](#) (optional): Use this constraint to specify clock-domains for input or output ports of black boxes. The SpyGlass DFT ADV Product supports only following options of the `abstract_port` constraint:
 - module
 - ports
 - clock

Operating Mode

[Capture](#)

Messages and Suggested Fix

The following violation messages are displayed for the `Atspeed_06` rule:

Message 1

[WARNING] <path_count> domain crossings from clock domain <domain_name1> to all other clock domains are not blocked under capture atspeed mode

Arguments

To view the arguments list, click [Arguments](#).

Potential Issues

A violation is reported due to one of the following reasons:

- Path is not blocked
- Clock of either source or destination may be incorrect

Consequences of Not Fixing

Domain crossing can result in potential timing violations. This can prohibit the use of test compression techniques using MISRs in signature analysis.

How to Debug and Fix

View the Modular Schematic of the violation message. The Modular Schematic highlights the terminals, which are in clock domain cross over path and are not blocked in capture mode. Click a terminal to view the Incremental Schematic. Probe the fan-in/fan-out cone of the terminal to see how the path is originating/terminating at flip-flops or abstract_port (Say A and B). Start probing the clock cone of A and B to see how it is unrelated. To check why the terminal is not blocked, overlay Info_testmode (auxiliary violation mode) under capture atspeed mode. This also shows how capture atspeed conditions are propagating to other inputs of the gate whose terminal is under investigation.

To fix the violation, see Example Code and/or Schematic

Message 2

[INFO] <path_count> domain crossings from clock domain <source-domain> to all other clock domains are blocked (by blocked-path/nonX-nets) under capture atspeed mode.

Arguments

- Path count of the domain crossings, <path_count>
- Domain name of source flip-flop, <domain_name1>

Potential Issues

Since this is an informational message, there are no corresponding potential issues.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of not fixing this violation message.

How To Debug and Fix

No debug or fix is required as this is an informational message.

Example Code and/or Schematic

Example 1

Consider the following figure:

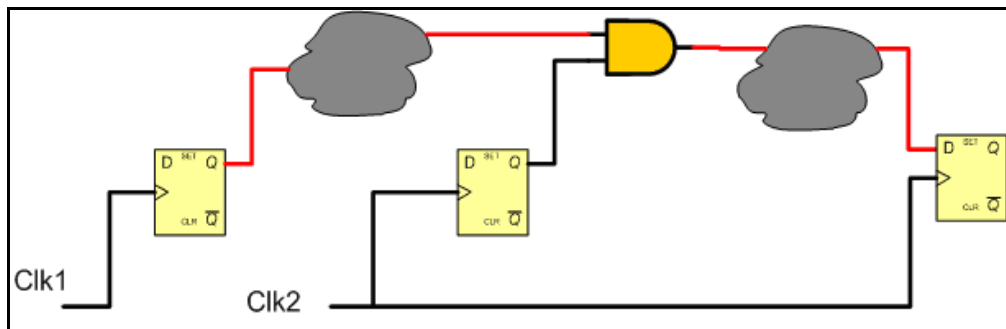


FIGURE 17. Domain Crossing Example

The above figure illustrates a domain crossing. In this case, if the top input to the AND gate is blocked, then testing of clock domain `clk2` can proceed without setup concerns in domain `clk1`.

Example 2

Consider the following RTL example:

```

module test(d1, d2, fck1, fck2, q);
input d1, d2;
input fck1, fck2;
output q;
reg ff1, ff2;
always @(posedge fck1)
    ff1 <= d1;
always @(posedge fck2)
    ff2 <= ff1;
assign q = ff2;

```

```

endmodule
// file: Atpspeed_06.sgdc
current_design test
clock -name "fck1" -atspeed -testclock -domain A
clock -name "fck2" -atspeed -testclock -domain B

```

If the same domain membership is given to both the clocks then no violation is reported for the Atpspeed_06 rule as show below:

```

clock -name "fck1" -atspeed -testclock -domain A
clock -name "fck2" -atspeed -testclock -domain A

```

The RTL example shown above reports an Atpspeed_06 violation.

Example 3

Consider the following RTL example:

```

module box1(in, out);
input in;
output out;
endmodule

module top(u1, clk1, clk2, v1);
input u1;
input clk1, clk2;
output v1;

box1 myinst(u1, w1);
reg ff1;
always @ (posedge clk2)
begin
ff1 = w1;
end
assign v1 = ff1;
endmodule

//file : Atpspeed_06.sgdc
current_design "top"

```

```
clock -name clk1 -atspeed -testclock
clock -name clk2 -atspeed -testclock
abstract_port -module box1 -ports out -clock clk2
```

The above RTL example demonstrates a case of `abstract_port`.

Default Severity Label

Warning / Info

Rule Group

Atspeed

Reports and Related Files

The `Atspeed_06` rule generates a CSV report for each violation.

The following table describes the columns listed in the CSV report and the related description for each column:

TABLE 2 Atspeed_06 CSV Report

Column	Description
Type	Severity of the rule. Can have one of the following values: <ul style="list-style-type: none"> Warning Info
Source Domain	Name of source clock domain.
Start Point	Name of source terminal.
Destination Domain	Name of destination clock domain.
Destination Point	Name of destination terminal, which can either be a flip-flop or an <code>abstract_port</code>
Start Type	Source terminal type, which can either be a flip-flop or an <code>abstract_port</code> .
Destination Type	Destination terminal type, which can either be a flip-flop or an <code>abstract_port</code> .

Atspeed_07

Use a separate signal, such as `scan_enable`, for clock gating each domain in the capture at-speed mode

When to Use

Use this rule to detect pair of at-speed testclocks that have the same gating conditions, specified through constraints.

Description

The Atpspeed_07 rule reports violation for a pair of at-speed clocks that have the same gating conditions.

Also, the Atpspeed_07 rule generates the [atspeed_07_rpt](#) report, which lists all at-speed test clocks that are gated by the same set of logic.

Default Weight

10

Language

Verilog, VHDL

Method

For each testclock

Traverse the combinational unblocked fan-out cone for the gating signals for this testclock

Tag each node reached as reached by this testclock

If any tagged node is hit by gating traversals from more than one testclock then issue a violation

End for

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- [atspeed_clock_frequency](#) (optional): Use this constraint to specify frequencies associated with a testclock.
- [clock](#) (optional): Use this constraint to declare clock pins declared as testclocks.

- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clockgating* (mandatory): Use this constraint to specify gating conditions for test clocks.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the Atspeed_07 rule.

[WARNING] The gating conditions for testclock '<net-name1>' and testclock '<net-name2>' overlap. The gating-net is '<net-name3>'

Arguments

- Name of the first testclock, <net-name1>
- Name of the second testclock, <net-name2>
- Name of the net where gating takes place, <net-name3>

Potential Issues

Gating signals are not properly separated.

Consequences of Not Fixing

It will not be possible to test the logic gating the clock through LOS (launch of shift) transition fault test

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the net, which is gating more than one testclocks. It also highlights the testclocks.

Overlay the Info_atSpeedClock (auxiliary violation mode) for the testclocks

highlighted to figure out how the net is gating the testclocks' propagation.

To fix the violation, see [Example Code and/or Schematic](#) section.

Example Code and/or Schematic

Consider the following example:

```
module top (clk1, clk2, tclk1, tclk2, sel1, sel2,
           clkGate, op1, op2, d, tm);
  output op1,op2;
  input  clk1, clk2, tclk1, tclk2, sel1, sel2, d, clkGate, tm;
  wire fclk1, fclk2;
  wire CLK1,CLK11,CLK2;
  wire q;
  assign fclk1 = sel1 ? tclk1 : clk1;
  assign fclk2 = sel2 ? tclk2 : clk2;
  assign q = ~clkGate;
  assign CLK11 = q & tm;
  assign CLK1 = CLK11 & fclk1;
  assign CLK2 = q & fclk2;
  reg op1;
  always @(posedge CLK1) op1 <= d;
  reg op2;
  always @(posedge CLK2) op2 <=d;
endmodule
```

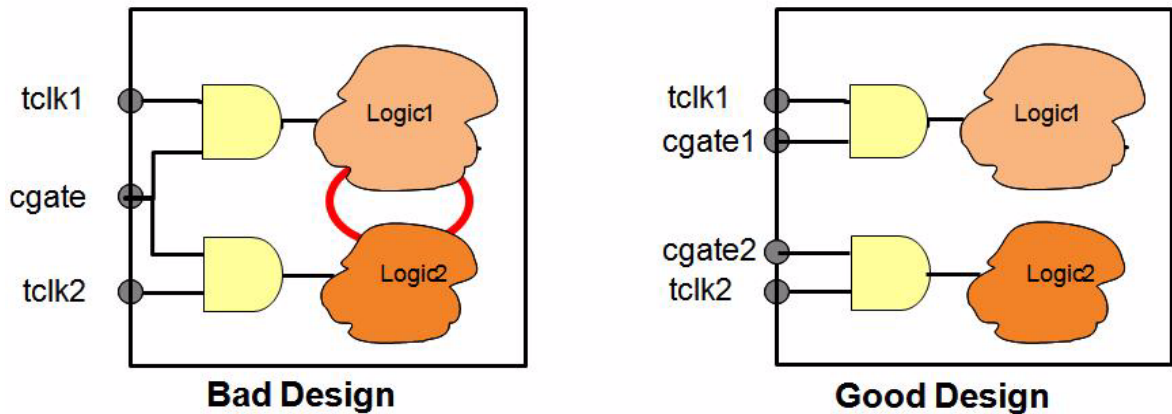
```
current_design top
clock -name tclk1 -testclock -atspeed
clock -name tclk2 -testclock -atspeed
clock -name clk1 -value rtz
clock -name clk2 -value rtz
clockgating -name tclk1 -pin top.clkGate top.q -value 0 1
clockgating -name tclk2 -pin top.q top.clkGate -value 1 0
testmode -name tm -value "1" -captureATspeed
testmode -name sel1 -value "1" -captureATspeed
testmode -name sel2 -value "1" -captureATspeed
```

The Atspeed_07 rule reports a violation for the above example as same

gating signal is used for two different clocks.

Example 2

Consider the following figure:



The above figure illustrates the clock gating for each domain in the test mode.

Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

[atspeed_07_rpt](#) Report

Atspeed_08

Clock tree in the clock generation logic should not have different logic along paths which need to be balanced

Rule Description

The `Atspeed_08` rule violates for clock generator nets (specified through `clock_root` constraint) that do not have the same logic in all fan-out cones or are not driving functional clock net.

For example, if a clock generator net is specified as:

```
clock_root -name x
```

Then, a violation is reported if signal `x` is driving gates of two different types, or if it is driving a net which is not a functional clock. If this rule is not followed, then it will be more difficult to minimize the skews.

The `Atspeed_08` rule generates the `atspeed_08_rpt` report, which lists all at-speed clocks that need to be balanced and that have different gates along the clock paths.

Constraints

- `clock` (optional): Use this constraint to declare clock pins declared as testclocks. The `Atspeed_08` rule uses the `-testclock` and `-atspeed` arguments of the clock constraint.
- `clock_root` (mandatory): Use this constraint to specify the starting node of a clock tree for checking whether clock tree is balanced.
- `clock_shaper` (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- `pll` (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- `test_mode` (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- `complex_cell` (mandatory): Use this constraint to control clock pulse propagation in a design, by enabling/disabling it under different test mode conditions.

Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture (atspeed)

Message Details

Message

Path from <net-name1> is balanced to <net-name2> but this is not a functional clock

Arguments

- Name of the source net <net-name1>
- Name of the destination net <net-name2>

Location

The place where source net is declared

Schematic highlight

Path from source net to destination net.

Rule Severity

Warning

Example

An example of design violating Atspeed_08 follows:

```
// Atspeed_08.v
module flop(d, clk, q);
input d, clk;
output q;
reg q;
always @(posedge clk)
    if (clk)
        q <= d;
endmodule
```

Atspeed Test Rules

```

module top (in1, in2, in3, data, out1, out2, out3, out4,
out5, out6);
input in1, in2, in3, data;
output out1, out2, out3, out4, out5, out6;
wire x, y, a, b, c, d, e, f, g, h, i, j;
myBB BB(in1, x, y);
assign a = x & in2;
assign b = a | in3;
assign c = ~x;
assign d = ~c;
assign e = ~d;
assign f = x & in2;
assign g = f | in2;
assign h = y & x;
assign i = ~h;
assign j = ~y;
flop f1(data, b, out1);
flop f2(data, e, out2);
flop f3(data, e, out3);
flop f4(data, g, out4);
flop f5(data, i, out5);
flop f6(data, j, out6);
endmodule

```

```
// file: Atspeed_08.sgdc
```

```

current_design top
clock -name b -testclock -atspeed
clock -name e -testclock -atspeed
clock -name i -testclock -atspeed
clock -name j -testclock -atspeed
clock_root -name x y

```

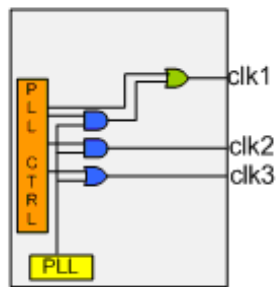
For below testcase, SGDC file would look like:

```

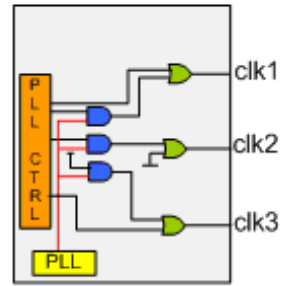
current_design top
clock -name -clk1 -testclock -atspeed
clock -name -clk2 -testclock -atspeed
clock -name -clk3 -testclock -atspeed
clock_root -name <output pin of PLL>

```

Also, the following figure shows an example of clock gating:



BAD DESIGN



GOOD DESIGN

Atspeed_09

Ensure that the data pin of scan flip-flop is fully controllable

When to Use

Use this rule to detect uncontrollable data pin of a scannable flip-flop.

Rule Description

The `Atspeed_09` rule highlights the root cause for bad controllability at the data pin of a scannable flip-flop.

Prerequisites

`Async_07`, `Clock_11`, `Scan_08` rule of the SpyGlass DFT ADV product for scannability

Rule Exceptions

None

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#)
- [dft_ignore_constant_supply_flip_flops](#)

Constraint(s)

- [atspeed_clock_frequency](#) (optional): Use this constraint to specify frequencies associated with a testclock.
- [clock](#) (optional): Use this constraint to declare clock pins declared as testclocks.
- [clock_shaper](#) (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.

- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

[WARNING] Node <node_name> (<reason>), one of the root causes of uncontrollability, impacts <num> scan flip-flop(s) <spreadsheet_path>

Arguments

- Name of node, <node_name>
- Root cause type, <reason>
- Number of scan flip-flops impacted, <num>
- Path to the spreadsheet report, which contains list of affected scan flip-flops, <spreadsheet_path>

Potential Issues

Following are some of the reasons of bad controllability, which can cause the violation:

- Combinational loop.
- Multiple driven net/floating net.
- Black box.
- Non-lockup/Non-transparent latch.
- Non-scan flip-flop.
- Constant value net.

Consequences of Not Fixing

Uncontrollable data pin of a scannable flip-flop results in low transition coverage.

How to Debug and Fix

The Atspeed_09 rule generates a spreadsheet report, which lists all the affected flip-flops.

Consider the following checks to debug the violations reported by the Atspeed_09 rule:

- If the Atspeed_09 rule flags a violation because of non-scan flip-flops, debug the violation of the Async_07 and Clock_11 rules.
- If the Atspeed_09 rule flags a violation because of black boxes, debug the violation of the InferBlackBox rule.
- If the Atspeed_09 rule flags a violation because of non-transparent latches, debug the violation of the Latch_08 rule.
- If the Atspeed_09 rule flags a violation because of undriven nets, view the Incremental Schematic of the violation message to see the undriven net.
- If the Atspeed_09 rule flags a violation because of combinational loop, debug the violation of the Topology_01 rule.
- If the Atspeed_09 rule flags a violation because of the constant value, view the Incremental Schematic of the violation message. Also, overlay the Info_testmode rule (auxiliary violation mode) under capture mode to check how the constant value has propagated.

To fix the violation, ensure that the data pin of scan flip-flop is fully controllable.

Example Code and/or Schematic

Current Unavailable

Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

None

Atspeed_10

Only valid clock sources allowed

Rule Description

The `Atspeed_10` rule detects problems when the clock pins of flip-flops are not traced backwards to a valid clock source. This rule ensures that a clean test clock be supplied to every flip-flop in the design.

NOTE: Use `Atspeed_11` rule, if you want to check whether all scan flip-flops are getting at-speed clock in `atspeed_mode`.

Rule Exception

The `Async_10` rule does not report a violation for the following types of flip-flops:

- 'no scan' (forced or inferred)
- Synthesis Redundant (SR)

Constraints

- `clock` (optional): Use this constraint to declare clock pins declared as testclocks. The `Atspeed_08` rule uses the `-testclock` and `-atspeed` arguments of the clock constraint.
- `clock_shaper` (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- `pll` (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- `test_mode` (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- `force_no_scan` (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Scanshift

Message Details

Message 1

Following violation message is reported when clock source is not driven by testclock:

Clock source '<clock-source>' is not in any clock domain.
Affects <flip-number> flip-flop(s)

Message 2

Following violation message is reported when port without testclock is found on clock path:

Top level port '<port-name>' feeding clock domain '<clock-source>' must be declared as testclock (leading to '<flip-name>')

Message 3

Atspeed_10 does topological traversal through clockshaper and it expects only one clock in terminal. Following violation message is reported when there are multiple clocks in terminal for clock shaper:

<clock-number> clock in terminal found (expected 1) for
Clockshaper cell '<inst-name>' (leading to '<flip-name>')

Message 4

Following violation message is reported when clock shaper terminal hit during tracing clock path is not a clock out terminal:

Terminal '<term-name>' found in clock path is not a clock out terminal of Clockshaper cell '<inst-name>' (leading to '<flip-name>')

Message 5

Following violation message is reported when clock in terminal of clock shaper is hanging:

Clock in terminal '<term-name>' of Clockshaper cell '<inst-name>' is hanging (leading to '<flip-name>')

Message 6

Following violation message is reported when hanging terminal is found on clock path:

Hanging terminal '<term-name>' of element '<inst-name>' found

in clock path

Message 7

Following violation message is reported when a net with no driver is found on clock path:

Undriven net '<net-name>' found on clock path (leading to '<flip-name>')

Message 8

Following violation message is reported when a net with multiple driver is found on clock path:

Multiple driver found on net '<net-name>' (leading to '<flip-name>')

Message 9

Following violation message is reported when port without testclock feeding clock shaper is found on the clock path:

Top level port '<port-name>' feeding the clock pin of a clockshaper '<inst-name>' must be declared as a testclock (leading to '<flip-name>')

Message 10

Following violation message is reported when non-x simulation value is encountered during tracing clock path:

Element '<inst-name>' is in a clock path and has a static value <sim-value> forced on its output (leading to '<flip-name>')

Message 11

Following violation message is reported when a black box is found on clock path:

Black boxes '<inst-name>' are not allowed in clock paths (leading to '<flip-name>').

Message 12

Following violation message is reported when no bypass terminal is found for bypass module:

No bypass terminal found for <term-name> terminal of bypass module '<inst-name>' (leading to '<flip-name>')

Message 13

Following violation message is reported when black box found on clock path is not properly bypassed:

Black box '<inst-name>', not properly bypassed, is not allowed in clock paths (leading to '<flop-name>')

Message 14

Following violation message is reported when hanging net is present on input terminal of bypass module found on clock path:

Found hanging net for <term-name> terminal of bypass module '<inst-name>' (leading to '<flop-name>')

Message 15

Following violation message is reported when a flip-flop or non transparent latch is found on the clock path:

The output of sequential elements '<inst-name>' must not drive clock pins in test mode (leading to '<flop-name>')

Message 16

Following violation message is reported when there is no sensitized path to output terminal of instance found on the clock path:

Element '<inst-name>' is in a clock path but prevents backwards tracing to a valid testclock input (leading to '<flop-name>')

Arguments

- Name of the port which is not declared as testclock <port-name>
- Name of the clock shaper instance <shaper-name>
- Name of instance that blocks testclock propagation for the given reason <inst-name>
- Name of the affected flip-flop <flopName>

Location

The place where the clock domain net is set

Schematic highlight

Path from flip-flop to its clock source, which is not controlled by testclock during scanshift mode.

Atspeed Test Rules

Rule Severity

Warning

Atspeed_11

Ensure that all clock sources are controlled by an at-speed clock

When to Use

Use this rule to identify path from flip-flop to its clock source, which is not controlled by atspeed clock during capture mode.

Description

The Atspeed_11 rule reports violation for clock domains that are not controlled by an atspeed clock in the test mode.

Clock logic should be designed such that each flip-flop in the design is driven by an at-speed clock in the test mode.

Default Weight

10

Language

Verilog, VHDL

Method

```
Simulate atspeed condition
Foreach atspeed clock
-Simulate pulse
-Tag all the nodes that gets the pulse (in or out of phase)
End for
Foreach clock-source
-If clock-source is not tagged report a violation
End for
```

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.

- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *no_atspeed* (optional): Use this constraint to exclude flip-flops from being used in at-speed testing, even if they qualify so.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[WARNING] Clock domain '<net-name1>' [in '<module-name>'] is not controlled by any testclock in capture-atspeed mode (<num> flipflop(s) affected) [<reason>]

Potential Issues

Violation may arise when the atspeed clock is missing from the constraint file.

Consequences of Not Fixing

Not fixing the violation results in low transition coverage as the flip-flops do not participate in atspeed launch and capture.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the flip-flop on which the atspeed clock does not reach.

Double-click the Incremental Schematic to probe the fan-in cone of failing clock source. Guess the path based on the signal name. The Signal being probed is annotated with the atspeed mode clock value ('^' for rising-edge

and 'V' for falling-edge). Absence of any value implies no testclock has reached to this point.

To fix the violation, see [Example Code and/or Schematic](#).

Message 2

[WARNING] Clock domain '<net-name1>' [in '<module-name>'] is not controlled by testclock <testclock_name> in capture-atspeed mode (<num> flipflop(s) affected). <cause_of_clk_src>

Potential Issues

Violation may arise if the atspeed mode is not properly setup and is blocking the clock propagation.

Consequences of Not Fixing

Not fixing the violation results in low transition coverage as the flip-flops do not participate in atspeed launch and capture.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the flip-flop on which the atspeed clock does not reach.

Overlay (Auxiliary violation mode) the Info_testmode rule under the capture atspeed mode. This helps in identifying the location of the blocked clock.

Double-click the Incremental Schematic to probe the fan-in cone of failing clock source. Guess the path based on the signal name. The Signal being probed is annotated with the atspeed mode clock value ('^' for rising-edge and 'V' for falling-edge). Absence of any value implies no testclock has reached to this point.

To fix the violation, see [Example Code and/or Schematic](#).

Message 3

[WARNING] Clock domain '<net-name1>' [in '<module-name>'] is not controlled by testclock <testclock_name> in capture-atspeed mode (<num> flipflop(s) affected) <cause_of_clk_src>. Other potential testclock: <potential_atspeed_test_clk>

Potential Issues

Violation may arise if the atspeed mode is not properly setup and is blocking the clock propagation.

Consequences of Not Fixing

Not fixing the violation results in low transition coverage as the flip-flops do not participate in atspeed launch and capture.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the flip-flop on which the atspeed clock does not reach.

Overlay (Auxiliary violation mode) the Info_testmode rule and Info_testclock under the capture atspeed mode.

Double-click the Incremental Schematic to probe the fan-in cone of failing clock source. Guess the path based on the signal name. The Signal being probed is annotated with the atspeed mode clock value ('^' for rising-edge and 'V' for falling-edge). Absence of any value implies no testclock has reached to this point.

To fix the violation, see [Example Code and/or Schematic](#).

Arguments

- Name of the clock domain, <net-name1>
- Name of the module, <module-name>
- Name of the testclock that is blocked from reaching clock domain, <testclock-name>. However, if no testclock reaches topologically, "any testclock" is displayed.
- Number of affected flip-flops, <num>
- The reason for non-reachability of testclock, <reason>. This argument can take one of the following values:
 - "testclock stops at <instance>", if the testclock reaches topologically, but is blocked functionally.
 - "clock source is <instance>"
- Cause of clock source not controlled by at-speed test clock, <cause_of_clk_src>
- Name of the other potential at-speed test clocks, if they exist, <potential_atspeed_test_clk> (Only one potential test clock exists for the third message.)

Example Code and/or Schematic

Consider the following example:

```
// file: Atspeed_11.v
module test(d1, d2, ck1, ck2, q1, q2);
input d1, d2;
input ck1, ck2;
output q1, q2;
reg q1, q2;
wire ck1_pll;
PLL pll_inst(.ckin(ck1), .ckout(ck1_pll));
    always @(posedge ck1_pll)
        q1 <= d1;
    always @(posedge ck2)
        q2 <= d2;
endmodule
```

```
// file: Atspeed_11.sgd
current_design test
clock -name "ck1" -domain domain3 -value rtz -testclock -
atspeed
clock -name "ck2" -domain domain4 -value rtz -testclock
pll -name PLL -clkkin ckin -clkout ckout
```

The Atspeed_11 rule reports a violation for the above message as the clock of the target flip-flop is not controlled by an 'at-speed' clock.

Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

None

Atspeed_12

A clock-pin receiving no frequency or multiple frequencies is a violation.

Rule Description

The Atpspeed_12 rule reports violation for a clock source, which receives more than one frequency or no frequency.

Rule Exception

The Async_12 rule does not report a violation for the following types of flip-flops:

- 'no scan' (forced or inferred)
- 'no_atpspeed'
- Synthesis Redundant (SR)

Constraints

- *clock* -testclock -atspeed (optional): Use this constraint to declare clock pins declared as testclocks. The Atpspeed_08 rule uses the -testclock and -atspeed arguments of the clock constraint.
- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Rule Parameters

- *dftUseOffStateOfClockInClockPropagation*
- *dsm_max_fanin_paths_for_Atpspeed_12*

Operating Mode

Capture (atspeed)

Message Details

Message 1

Clock source '`<clock-source name>`', which impacts '`<number of flip-flops>`' flip-flops, does not receive any atspeed frequency. `<Cause String>`

Arguments

Here `<cause_string>` can have one the following output values:

- No atspeed clock found in the topological fanin cone of the clock-source
- '`<no_of_atspeed clock>`' atspeed clocks (without frequency) found in the topological fanin cone of the clock-source. [Atspeed clocks: `<comma-separated list of atspeed clocks>`]
- '`<no_of_frequencies>`' atspeed clock frequencies found (but blocked) in the topological fanin cone of the clock-source. [Frequencies: `<comma-separated list of frequencies including atspeed clock name and frequency-id>`]

Message 2

Clock source '`<clock_source_name>`', which impacts '`<no_of_flip_flops>`' flip-flops, receives '`<no_of_frequencies>`' atspeed frequencies. [Frequencies: `<comma-separated frequency list (including atspeed clock name and frequency ID)>`]

Location

The first place where this clock net is used.

Schematic highlight

- The node that does not receive any frequency
- The propagation path of the clk-net, other potential frequencies and the flip-flops affected.

Rule Severity

Warning

Atspeed_13

Expected frequencies must be achieved

Rule Description

The Atspeed_13 rule checks whether or not a particular user-specified frequency can be reached at user-specified net or terminal to enable at-speed testing.

The Atspeed_13 rule checks if the frequencies expected by the user in the nodes, specified using the *expect_frequency* constraint, are achieved. The rule reports a violation, if it finds a mismatch between the actual-frequencies on the net with the user-specified expected frequency.

NOTE: *To debug the violation messages reported by the Atspeed_13 rule, it is recommended to run this rule with the *Info_atSpeedFrequency* rule, which displays the atspeed frequency propagation.*

Constraints

- *expect_frequency* (mandatory)
- *clock* (optional)
- *atspeed_clock_frequency* (optional)
- *test_mode* (mandatory)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture

Message Details

Message 1

[WARNING] [constraint_message_tag: <value>] Node <name> does not meet expected frequency list: <missing-freqs>. Actual frequency list: <actual-freqs>

Message 2

[WARNING] [constraint_message_tag: <value>] Node <name> gets

extra frequency list: <extra-freqs>. Actual frequency list: <actual-freqs>

Arguments

- Constraint tag value, <value>
- Name of the node, <name>
- Frequencies that are in the expected-list but not in the actual-list, <missing-freqs>
- Frequencies that are in the actual-list but not in the expected-list, <extra-freqs>
- Frequencies that reach the net. This list is determined by analysis and simulation. The frequency is reported in MHz, <actual-freqs>

Message 3

[INFO] <no_of_nodes> Node(s) in the design <design_name> get(s) expected frequency list <spreadsheet_path>

Arguments

- Number of nodes which got the expected frequency, <no_of_nodes>
- Design name, <design_name>
- Path name of the spreadsheet associated with the message, <spreadsheet_path>

Location

Specific expect_frequency constraint line number in the SGDC file.

How to Debug

To debug the violation messages reported by the Atspeed_13 rule, see the [Info_atSpeedFrequency](#) rule, which displays the atspeed frequency propagation.

The Atspeed_13 rule also generates a spreadsheet report, which lists net name with SGDC name, expected frequency list, and constraint message tag.

Schematic highlight

- Instance and terminal where expected frequency is not achieved
- Instance and terminal where extra frequencies are achieved.

Atspeed Test Rules

Rule Severity

Warning

Atspeed_14

Test clocks must not be used as data signals

Rule Description

The fan-out from one signal source to the clock and data pins of the same register can lead to race conditions, confusing fault simulation and ATPG.

Most ATPG tools assume simple timing models (frequently zero delay), and therefore cannot properly handle such cases. Combinational ATPG tools produce tests by assuming (during test generation) that system data pins are directly observable. If the system data has a dependency on a particular clock phase, then tests may be generated that do not detect their targeted faults. Fault coverage may decrease if the requirements for some portion of the logic contradict the operation of the clock with respect to the capture and shift modes.

Constraints

- *clock* -testclock -atspeed (optional):
- *clock_shaper* (optional):
- *pll* (optional)
- *test_mode* (optional)
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture, Capture (atspeed)

Message Details

Message

The clock signal '*<clk-name>*' is reaching to data pin of '*<num_of_inst>*' flip-flop(s) in capture mode

Arguments

- Name of the clock, <clk-name>
- Number of flip-flop instances reported, <num_of_inst>

Location

The place where the flip-flop is inferred.

Schematic highlight

Path from the clock to the data pin of the data pin of flip-flop via CGCs, if clock receives the CGC enable pin.

Example Code and/or Schematic

Example 1

The following figure illustrates the testclock, tclk, acting as a data signal to the flip-flop, FF2:

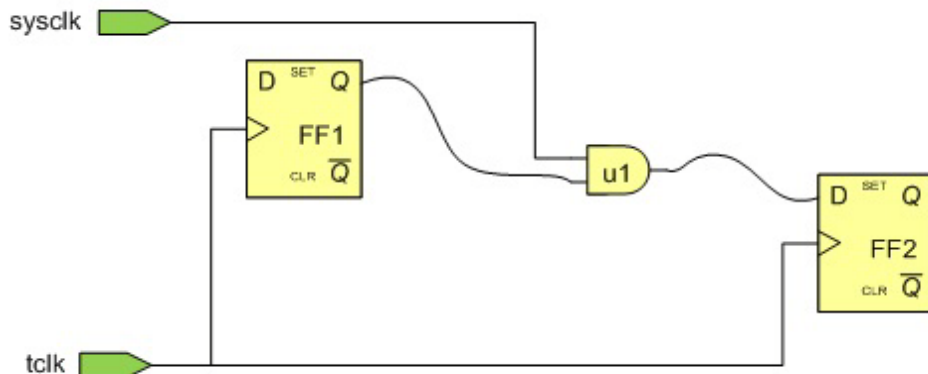


FIGURE 18. Clock as data example

In the above example, if FF1 and FF2 are scannable, then ATPG may create atspeed tests for u1 that launch from FF1 and capture in FF2. Any such tests that are incompatible with the values on the system clock, sysclk, would fail. This results in coverage lower than expected.

Example 2

Consider the following design in [Figure 44](#) where the clock signal, clk, is reaching to data pin of flip-flops, qb1_reg, qb2_reg, qb3_reg, and qb4_reg:

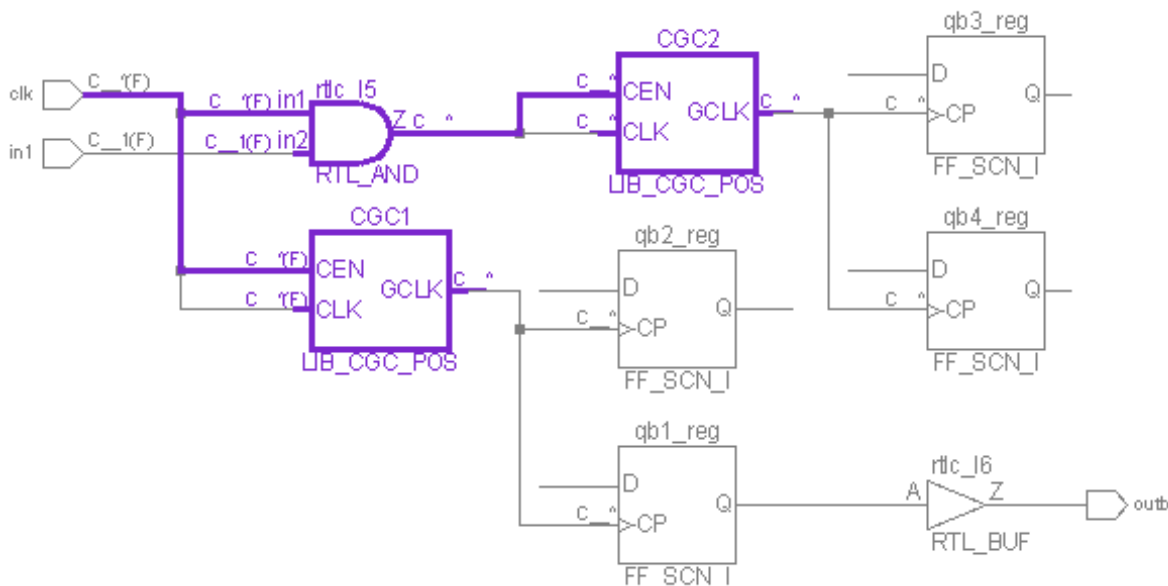


FIGURE 19. Clock Signal acting as a data pin of flip-flop

The Atspeed_14 rule reports the following violation message for the above design:

The clock signal 'ClockAsData.clk' is reaching to data pin of '4' flip-flop(s) in 'capture' mode, test.v, 6

Double-clicking the violation message displays the spreadsheet in the spreadsheet viewer as shown in [Figure 45](#):

Atspeed Test Rules

	B "Clock_Name"	C "Flop_Name"	D "EN-CGC name"	E "Mode_Name"
1	ClocksAsData.clk	ClocksAsData.qb1_reg	ClocksAsData.CGC 1	capture
2	ClocksAsData.clk	ClocksAsData.qb2_reg	ClocksAsData.CGC 1	capture
3	ClocksAsData.clk	ClocksAsData.qb3_reg	ClocksAsData.CGC 2	capture
4	ClocksAsData.clk	ClocksAsData.qb4_reg	ClocksAsData.CGC 2	capture

FIGURE 20. Spreadsheet report for the Atpspeed_14 rule

Double-click on a row in the spreadsheet to display the complete path from the clock pin to the affected flip-flop in an Incremental Schematic as shown in [Figure 46](#):

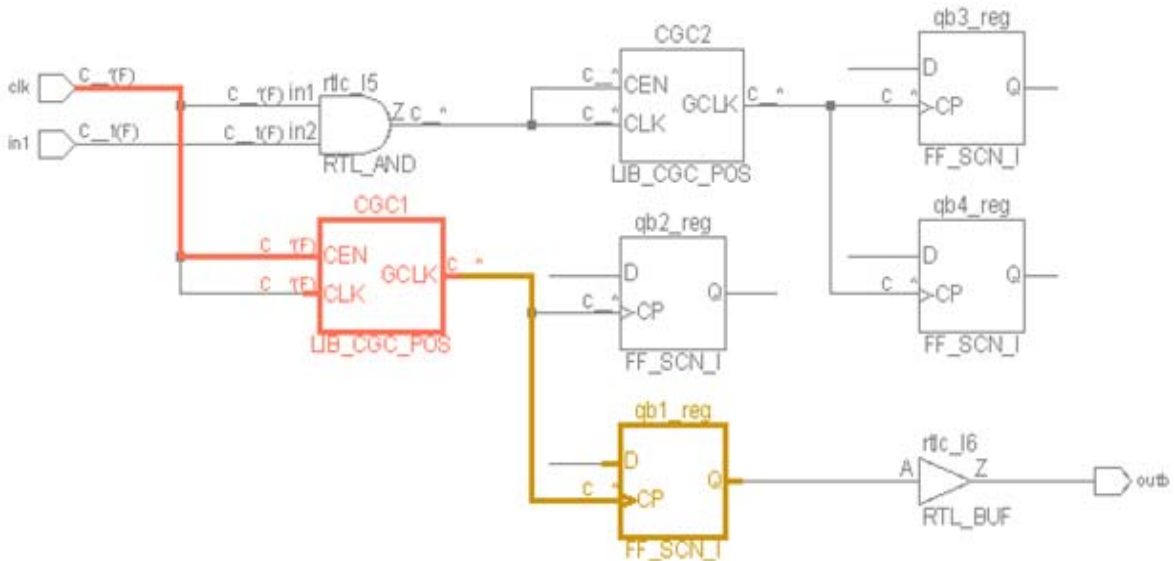


FIGURE 21. Violating Path

Rule Severity

Warning

Atspeed_15

Combinational loops are not allowed in capture mode

Rule Description

Combinational feedback may create memory and therefore defeats full-scan assumptions.

ATPG tools associate the circuit state variables with specific library cell types and therefore do not recognize the sequential nature of combinational feedback resulting in loss of coverage and increased ATE debug time. Such feedback structures may also defeat the logic BIST since circuit initialization may not be guaranteed.

Constraints

clock -testclock -atspeed (optional)

clock_shaper (optional)

pll (optional)

test_mode (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture, Capture (atspeed)

Message Details

Message

Combinational loop, having <num_of_nets> elements, detected in '<mode_name>' mode. Loop path: <loop_path>

Arguments

- Number of nets in the loop, <num_of_nets>
- Mode name as Capture or Capture (atspeed), <mode_name>
- The path of the loop, <loop_path>

Location

The place where the first instance of the loop is inferred.

Schematic highlight

Combinational path that is forming the loop.

Rule Severity

Warning

Atspeed_16

This rule has been deprecated

The functionality of this rule is now covered by the `Tristate_01` rule of the SpyGlass DFT ADV product.

Atspeed_17_capture

clockshaper cell must be enabled in capture mode

Rule Description

The Atpspeed_17_capture rule ensures that the clock_shaper cells, specified through the *clock_shaper* sgdc command, are set up to be enabled in the capture condition.

While other rules check for topological correctness, the Atpspeed_17_capture rule enables the atspeed test clock that needs to be propagated in the capture mode.

If a clock_shaper cell is not enabled in the capture mode and a clock hits its input clock pin, then it may block the clock propagation through it.

Constraints

clock -testclock -atspeed (optional)

clock_shaper (optional)

pll (optional)

test_mode (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture

Message Details

Message

The <pin-type> pin(s) '<pin-name(s)>' of clock_shaper '<shaper-name>' has incorrect enabling condition in '<mode>'. Actual '<simulation-pattern>' (Xpt : '<enable-value-list>')

Arguments

- Enable or testmode, <pin-type>
- Enable/testmodes pin name(s) <pin-name(s)>

- Hierarchical name of the clockshaper instance <shaper-name>
- Capture mode, <mode>
- String of value where each value represents the simulation value present at the corresponding pin in pin-name(s), <simulation pattern>
- List of expected simulation pattern at pins, <enable-value-list>

Location

Place where the clockshaper instance is first inferred.

Schematic highlight

- The Clock shaper instance
- Simulation value(s) at enable/testmode pin(s)
- List of expected simulation pattern at pins

Rule Severity

Warning

Atspeed_17_captureatspeed

Clockshaper cell must be enabled in captureatspeed mode

Rule Description

The Atspeed_17_captureatspeed rule ensures that the clock_shaper cells, specified through the *clock_shaper* sgdc command, are set up to be enabled in the capture-atspeed condition.

While other rules check for topological correctness, the Atspeed_17_captureatspeed rule enables the atspeed test clock that needs to be propagated in the captureatspeed mode.

If a clock_shaper cell is not enabled in the capture-atspeed mode and a clock hits its input clock pin, then it may block the clock propagation through it.

Constraints

clock -testclock -atspeed (optional)

clock_shaper (optional)

pll (optional)

test_mode (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture (atspeed)

Message Details

Message

The <pin-type> pin(s) '<pin-name(s)>' of clock_shaper '<shaper-name>' has incorrect enabling condition in '<mode>'. Actual '<simulation-pattern>' (Xpt : '<enable-value-list>')

Arguments

- Enable or testmode, <pin-type>

Atspeed Test Rules

- Enable/testmodes pin name(s) <pin-name(s)>
- Hierarchical name of the clockshaper instance <shaper-name>
- Capture-atspeed mode, <mode>
- String of value where each value represents the simulation value present at the corresponding pin in pin-name(s), <simulation pattern>
- List of expected simulation pattern at pins, <enable-value-list>

Location

Place where the clockshaper instance is first inferred.

Schematic highlight

- The Clock shaper instance
- Simulation value(s) at enable/testmode pin(s)
- List of expected simulation pattern at pins

Rule Severity

Warning

Atspeed_17_shift

clockshaper cell must be enabled in scanshift mode

Rule Description

The Atpspeed_17_shift rule ensures that the clock_shaper cells, specified through the *clock_shaper* sgdc command, are set up to be enabled in the shift condition.

While other rules check for topological correctness, the Atpspeed_17_shift rule enables the atspeed test clock that needs to be propagated in the scanshift mode.

If a clock_shaper cell is not enabled in the shift mode and a clock hits its input clock pin, then it may block the clock propagation through it.

Constraints

clock -testclock -atspeed (optional)

clock_shaper (optional)

pll (optional)

test_mode (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Shift

Message Details

Message

The <pin-type> pin(s) '<pin-name(s)>' of clock_shaper '<shaper-name>' has incorrect enabling condition in '<mode>'. Actual '<simulation-pattern>' (Xpt : '<enable-value-list>')

Arguments

- Enable or testmode, <pin-type>
- Enable/testmodes pin name(s) <pin-name(s)>

- Hierarchical name of the clockshaper instance <shaper-name>
- Shift mode, <mode>
- String of value where each value represents the simulation value present at the corresponding pin in pin-name(s), <simulation pattern>
- List of expected simulation pattern at pins, <enable-value-list>

Location

Place where the clockshaper instance is first inferred.

Schematic highlight

- The Clock shaper instance
- Simulation value(s) at enable/testmode pin(s)
- List of expected simulation pattern at pins

Rule Severity

Warning

Atspeed_19

Latches should be transparent in capture mode

Rule Description

The Atspeed_19 rule reports violation for the non-transparent latches. However, the Atspeed_19 rule doesn't report violation for the following cases:

- Transparent latches
- Shadow latches
- Scannable latches
- Latch is contained in a valid user-defined scan cell

Combinational ATPG tools may treat transparent latches, thereby simplifying test generation. You must exercise caution, in cases where latches are embedded with other sequential devices. In this case, some of the logic may become untestable.

Violations are reported on the sensitized source of the latch enable term. Sensitized source of the latch enable term is found by traversing through the various gates in its path.

For Latch enable source driving on both levels, violations are reported when any one of the latch is non-transparent/controllable-transparent.

Constraints

clock -testclock -atspeed (optional)

clock_shaper (optional)

pll (optional)

test_mode (optional)

Rule Parameters

- *dftUseOffStateOfClockInClockPropagation*: Set this parameter to off to keep all clocks, including functional and test clocks to unknown state. See the table below to view the list of other possible values.
- *dft_use_new_Atspeed_19*: Use this parameter to enable/disable the source-centric message reporting by the Atspeed_19 rule.

- [*dft_ignore_constant_latches_in_rule_checking*](#): Set the value of the parameter to off to consider latches for rule checking, when one of the following conditions is set to true:
 - D-pin has a constant value and EN-pin gets a test clock
 - Q-pin has a constant value
- [*dft_ignore_no_scan_latches_in_rule_checking*](#): The default value is off. Set the value of the parameter to on to ignore forced no_scan latches from rule checking.

Operating Mode

Capture, Capture (atspeed)

Message Details

Message 1

[INFO] Latch enable source <source_name> is controllable-transparent in <mode> (drives <number1> Latch(es) out of which <number2> Latch(es) are controllable transparent)

Arguments

- Source of the Latch enable term, <source_name>
- Mode for which the analysis is done, <mode>
- Total no of latches the latch enable source is driving, <number1>
- Number of latches which are controllable-transparent, <number2>

Potential Issues

A violation is reported when latch enable is controllable to active state but is currently not active.

Consequences of Not Fixing

If the violation is not fixed, ATPG will have to make latch transparent through vector shift-in

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Latch enable source <source_name> is non-transparent in <mode> (drives <number1> Latch(es) out of which <number2> Latch(es) are non-transparent

Arguments

- Source of the Latch enable term, <source_name>
- Mode for which the analysis is done, <mode>
- Total no of latches the latch enable source is driving, <number1>
- Number of latches which are non-transparent, <number2>

Potential Issues

A violation is reported due to a non-transparent latch in the design.

Consequences of Not Fixing

Not fixing the violations results in reduced coverage due to bad controllability and observability around the latch.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Latch enable source <source_name> is driving latch enables on both levels in <mode> (drives <number1> Latch(es) out of which <number2> Latch(es) are non-transparent/ controllable transparent)

Arguments

- Source of the Latch enable term, <source_name>
- Mode for which the analysis is done, <mode>
- Total no of latches the latch enable source is driving <number1>
- Number of latches which are non-transparent/ controllable transparent, <number2>

Potential Issues

A violation is reported when the latch enable source is driving on both the levels.

Consequences of Not Fixing

Since latch enable source is driving latches on both the levels, none of them can be made transparent simultaneously.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] <'module_name' > has <'number' > non transparent latch(es) with floating enable pin

Arguments

- Name of the Top module, <module_name>
- Number of non transparent latches, <number>

Potential Issues

A violation is reported when the latch enable pin is floating.

Consequences of Not Fixing

Not fixing the violation results in reduced coverage due to bad controllability and observability around the latch.

How to Debug and Fix

Double-click on the message to open the spreadsheet which displays all the latches along with its types.

To view the schematic, double-click on a latch in the spreadsheet and then open incremental schematic.

Example Code and/or Schematic

Example 1

Consider the following incremental schematic where the latch enable source, top.ff1, is controllable-transparent:

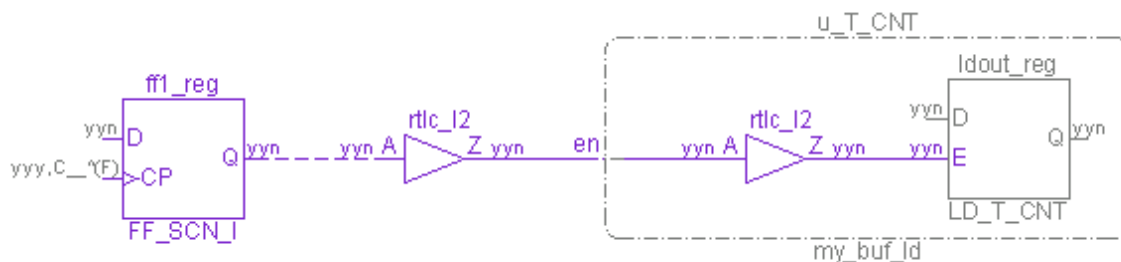


FIGURE 22. Controllable-Transparent Latches

For the above design, the Atspeed_19 rule reports the following violation message:

Latch enable source 'top.ff1' is controllable-transparent in Capture mode (drives 1 Latch(es) out of which 1 Latch(es) are controllable Transparent), test.v, 100

The above violation message is reported when latch enable is controllable to active state but is currently not active.

Example 2

Consider the following incremental schematic where the latch enable source, `top.en_T_uncnt`, is driving non-transparent latches:

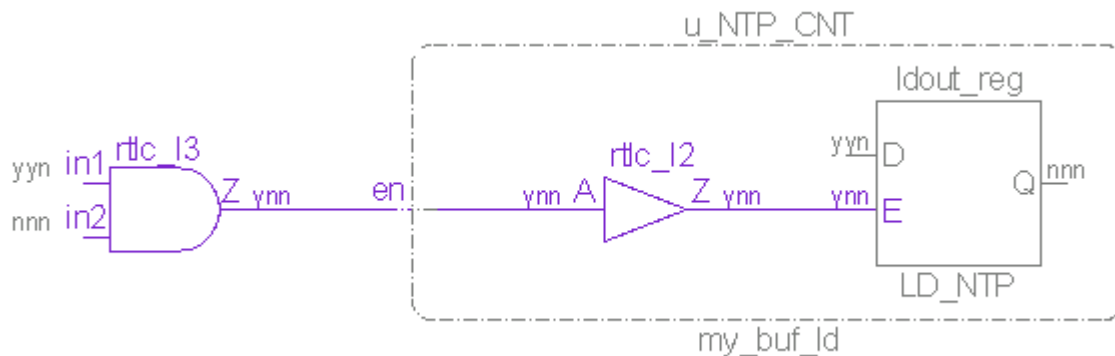


FIGURE 23. Non-Transparent Latches

For the above design, the Atspeed_19 rule reports the following violation message:

Latch enable e source 'top.en_T_uncnt' is driving non transparent latches in Capture mode (drives 1 Latch(es) out of which 1 Latch(es) are non-transparent), test.v, 106

The above violation message is reported due to the presence of a non-transparent latch in the design.

Example 3

Consider the following incremental schematic where the clock enable source, `top.clk`, is driving latch enables on both the levels:

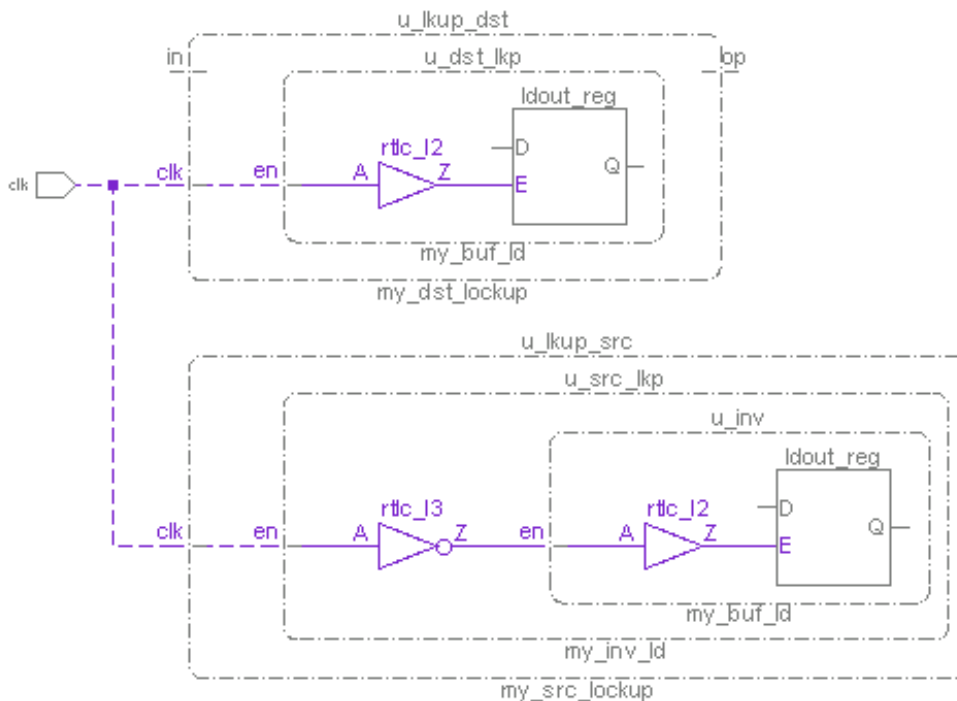


FIGURE 24. Non-Transparent/Controllable-Transparent Latches

For the above design, the Atspeed_19 rule reports the following violation

message:

Latch enable source 'top.clk' is driving latch enables on both levels in Capture mode (drives 5 Latch(es) out of which 2 Latch(es) are non-transparent/control lable transparent), test.v, 53

The above violation message is reported because the latch enable source is driving on both the levels.

Rule Severity

Warning

Atspeed_20

Asynchronous set/reset pins of all the flip-flops should be fully controllable during capture atspeed mode.

Rule Description

The logic driving the set or reset pins should be designed so that the sets and resets of all flip-flops may be fully controllable in capture atspeed. This ensures that ATPG will be able to generate vectors to detect faults on data line and set/reset lines of all the flip-flops.

Flip-flops marked as unused are ignored by this rule.

Constraints

clock -testclock -atspeed (optional)
test_mode (optional)

Rule Parameters

- *dftUseOffStateOfClockInClockPropagation*: Sets clocks, except the propagated clock, to the X (don't care) state or their off state during clock propagation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Operating Mode

Capture (atspeed)

Message Details

Message

```
source net '<Hierarchical_signal_name>' feeding '<pin-name>'
pin of '<no_of_flip_flop>' is not fully controllable in capture
atspeed mode
```

Arguments

1. Hierarchical signal name of the source
2. SET or RESET <pin-name>
3. Number of affected flip-flops <no-of_flip_flop>

4. Hierarchical name of example flip-flop <hierarchical_path_name>

Location

File / Line where source signal is set

Schematic highlight

- Flip-Flop
- Path from asynchronous (set/reset) source to the set/reset pin of the flip-flop

Reports

Double-click on the violation message to view the spreadsheet that lists the details of the flip-flops that are not fully controllable in the capture atspeed mode.

Rule Severity

Warning

Atspeed_22

No "ANDing" (Merging) of atspeed test clocks in capture atspeed mode.

Rule Description

Merging of atspeed test clocks must be treated just as other internally generated clocks. Such merging may reduce the clock pulse width so that flip-flop timing requirements are more difficult to satisfy.

Constraints

- *test_mode* (optional)
- *clock* -atspeed (mandatory)

Rule Parameters

- *dft_allow_clock_merge_at_mux_via_data*: The default value is off. Set the value of the parameter to on enable the Clock_08 and Atspeed_22 rules to ignore the violation if a clock is merging at a mux through its data-pins.
- *dftReportIfUsedAsClock*: Enables the Atspeed_22 rule to report clock merging only if the merged node is used as clock in the design, driving CP-pin of a flip-flop or the EN-pin of a latch.
- *dftAllowMergeOfSameClock*: Enables the *Atspeed_22* rule to report clock merging only if multiple user-specified test clocks are participating.
- *dftUseOffStateOfClockInClockPropagaton*: Sets clocks, except the propagated clock, to the X (don't care) state or their off state during clock propagation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Operating Mode

Capture (atspeed)

Message Details

The following violation message is displayed for the Atspeed_22 rule:

<no_of_atspeedcl ks> atspeed cl ocks <cl k-name> are mergi ng at

net '<net-name>' in <mode_name>

Arguments

- Name of the gated clock output net. (<net-name>)
- List of merging clocks (<clk-name>)
- Number of atspeed clocks (<no_of_atspeedclks>)
- Mode name which is captureatspeed for this rule, <mode_name>

Schematic Highlight

Path of merging testclocks

Example Code and/or Schematic

The following figure illustrates the ANDing of test clocks, tclk1 and tclk2:

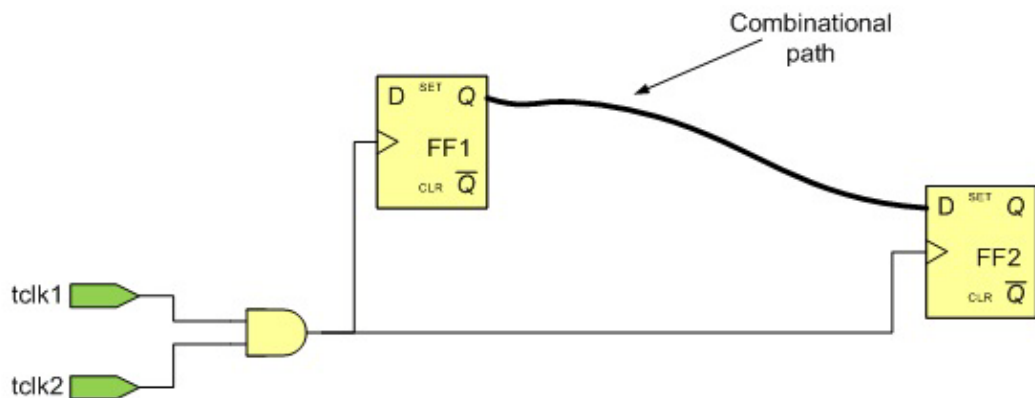


FIGURE 25. Clock ANDing example

Clock ANDing may reduce the clock pulse width to a point so that the pulse has insufficient energy to trigger either the launch flip-flop or the capture flip-flop. In either case, test outcome is compromised.

Rule Severity

Warning

Atspeed_23

The clock pin of memories must pass through a pll.

When to Use

Use this rule to ensure that the clock pin of memories must pass through a pll.

Description

The rule reports violation for all the clock pin memories that are clock-driven by other sources.

Rule Exceptions

Any memory/hard-macro defined as `no_scan` is skipped from the rule checking.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dsm_memory_clock_check](#): The default value is both. Set the value of the parameter to either `atspeed_clock` or `atspeed_clock_via_pll` based on the type of check you want to perform on the memory clock-pin, using the `Atspeed_23` rule.
- [checkPLLSourceForAllNonFFCellClockPins](#): The default value is off. Set the value of the parameter to on to perform rule checking on all clock pins of Non-FF cells that are identified from the "`clock : true`" entry in the `.lib` library, whether they are memories or not.

Constraint(s)

- [memory_type](#) (optional): Specifies the memory design unit (black box) names.

- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[WARNING] Clock source '*clk_src_name*' does not get atspeed clock *clk_name* through a PLL. Count of affected memories: *num*

Arguments

To see list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported because memory clock source does not get clock through PLL.

Consequences of Not Fixing

You can not perform atspeed testing of the memory.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the path from root clock source to memory clock source. Use Info_atSpeedClock rule to debug why clock is not coming from a PLL.

To control the frequency, the clock inputs of all memories must be driven by a pll.

Message 2

[WARNING] Clock source '*clk_src_name*' does not get atspeed clock ('*clk_name*' '*other_potential_atspeed_clocks*'). Count of affected memories: *num*

Arguments

- Name of clock source, *clk_src_name*
- Number of affected memories, *num*

Potential Issues

A violation is reported because memory clock source does not get atspeed clock.

Consequences of Not Fixing

You can not perform atspeed testing of the memory.

How to Debug and Fix

View the Incremental Schematic for the violation message. Use Atspeed_11 and Info_atSpeedClock rules to debug why clock source is not getting atspeed clock.

To control the frequency, the clock inputs of all memories must be driven by a pll.

Message 3

[WARNING] Clock source '`<clk_src_name>`' does not get atspeed clock `<clk_name>`. Count of affected memories: `<num>`

Arguments

- Name of clock source, `<clk_src_name>`
- Number of affected memories, `<num>`

Potential Issues

A violation is reported because memory clock source does not get atspeed clock.

Consequences of Not Fixing

You can not perform atspeed testing of the memory.

How to Debug and Fix

View the Incremental Schematic for the violation message. Use Atspeed_11 and Info_atSpeedClock rules to debug why clock source is not getting atspeed clock.

To control the frequency, the clock inputs of all memories must be driven by a pll.

Message 4

[INFO] `<number_of_memories>` found with hanging clock pin

Potential Issues

The violation message is generated when the memories with hanging clock pin are found in the design.

Consequences of Not Fixing

Not applicable.

How to Debug and Fix

Not applicable.

Example Code and/or Schematic**Example 1**

Consider the following scenario where atspeed clock does get to the memory clock pin but it does not go through a PLL. Also, consider that the value of the *dsm_memory_clock_check* parameter is either both or atspeed_clock_via_pll.



FIGURE 26.

The Atspeed_23 rule reports the following violation message for the above scenario:

Clock source 'top.clk_shaper' does not get atspeed clock ('top.clk') through a PLL. Count of affected memories: 28

Example 2

Consider the following scenario where there are atspeed clocks in the fanin cone of the memory clock pin but all of them are blocked. Also, consider that the value of the *dsm_memory_clock_check* parameter is either both or atspeed_clock:

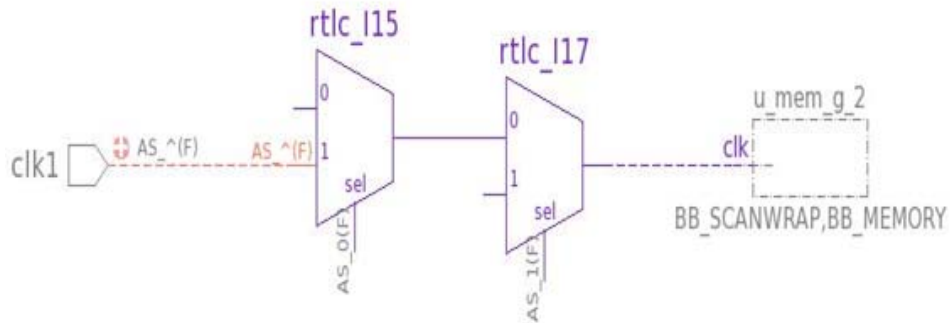


FIGURE 27.

The Atspeed_23 rule reports the following violation message for the above scenario:

Clock source 'top.l_d_op_as_clk' does not get atspeed clock ('top.clk' (Other potential atspeed clocks: 2: top.clk3 top.clk1)). Count of affected memories: 12

Example 3

Consider the following scenario where there is no atspeed clock in the fanin cone of the memory clock pin. Also, consider that the value of the Value of the [dsm_memory_clock_check](#) parameter is either both | or atspeed_clock:



FIGURE 28.

The Atspeed_23 rule reports the following violation message for the above scenario:

Clock source 'top.clk_hang' does not get atspeed clock ('No atspeed clock is found in the topological fanin cone'). Count of affected memories: 175

Default Severity Label

Warning

Rule Group

Atspeed rules

Reports and Related Files

No related reports or files.

Atspeed_24

Clock should not be shaped more than once.

When to Use

Use this rule to identify all the leaf level clock shapers whose clock pins get multiple-shaped clocks.

NOTE: *Clock dividers are treated as buffers during rule checking.*

Description

The Atpspeed_24 rule reports violation for all the leaf level clock shapers whose clock pins get multiple shaped clocks.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- *clock* (mandatory): Defines the clocks of a design.
- *clock_shaper* (mandatory): Controls clock pulse propagation in a design, by enabling/disabling it under certain conditions, or by modifying frequencies.

Operating Mode

[Capture \(atspeed\)](#)

Messages and Suggested Fix

The following violation message is displayed for the Atpspeed_24 rule:

```
[WARNING] Output terminal '<terminal_name>' of clock shaper
'<inst_hier_name>' clocks '<num>' flip-flop(s) with clock
signal [clock : '<r_clk_name>'] that is already shaped '<num1>'
```

times"

Arguments

- Terminal name of clock shaper, *<terminal_name>*
- Instance hierarchy name of clock shaper, *<inst_hier_name>*
- Number of flip-flops affected, *<num>*
- Root Clock Name, *<r_clk_name>*
- Number of time clock is shaped, *<num1>*

Potential Issues

In some design methodology, flip-flops should not get multiple shaped clocks. Therefore, a violation is reported when a leaf level clock shaper gets already shaped clock.

Consequences of Not Fixing

Not fixing the violation may affect the second capture at the flip-flops and, therefore, impacts the atspeed coverage.

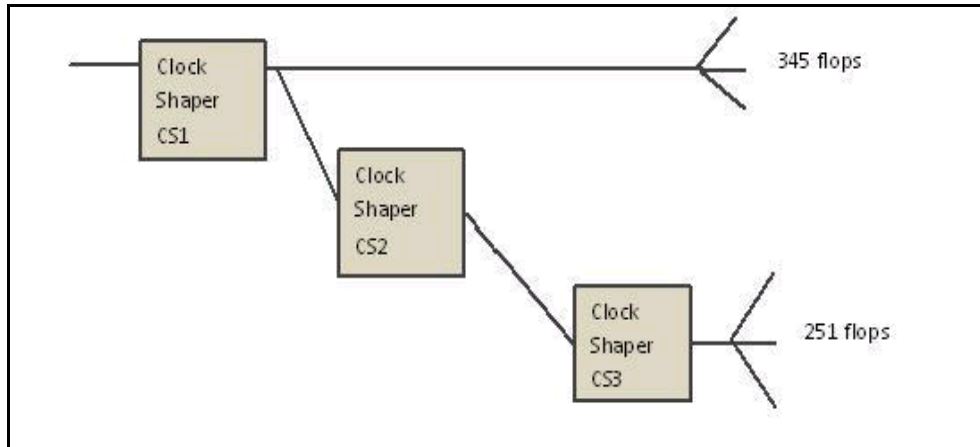
How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the path from root clock source to clock shaper that is getting multishaped clock. Use Info_atSpeedClock rule to view the clock propagation.

To fix the violation, change the clock architecture to ensure that the clock is not shaped more than once.

Example Code and/or Schematic

Consider the following figure:



In the above example, clock shaper, CS3, clocks 251 flip-flops with a clock signal that is already shaped 2 times.

However, the rule does not report any violation for CS2 as it does not clock any flip-flops. The rule also does not report any violation for CS1 as it is non-violating, that is, getting free running clock.

Default Severity Label

Warning

Rule Group

Atspeed Rules

Reports and Related Files

No related reports or files.

Atspeed_25

Reports the presence of combinational re-convergence to flip-flops asynchronous pins

When to Use

Use this rule to avoid unnecessary glitches on the set/reset tree.

Description

The Atpspeed_25 rule reports violation for reconverging combinational paths at async pins of a flip-flop.

The Atpspeed_25 rule reports reconverging combinational paths that start from a primary input port or the output pin of a flip-flop and re-converge at the asynchronous set/reset pin of any flip-flop.

If you want to filter violation messages that have re-convergence with different capture and shift values, you can overload message label to:

Topo_13_As_25_M2

Rule Exception

The *Atpspeed_25* rule ignores rule checking for the following cases:

- no scan flip-flops (forced or inferred).
- Unate re-convergence, that is, paths with no phase difference

Method

Traverse unblocked path in fan-in of set/reset pins of all flip-flops.

If any path is hit again for the fan-in traversal of the same pin

Report a violation.

Highlight the reconvergent path.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*

- *dft_ignore_unate_reconvergence*: The default value is on. Set the value of the parameter to off to check for unate re-convergence.
- *dft_report_all_paths_between_reconvergence_start_and_end*: The default value of this parameter is off. Set the value of the parameter to on to report all paths between re-convergence start and end points.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[INFO] Reconvergence report file '<report-file-name>' is generated

Arguments

Name of the report file, *<report-file-name>*

Potential Issues

The violation message appears when the text report file is generated.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[WARNING] Logic '<start_point>' reconverges at or near '<end_point>'. (Actual Reconvergence start =

```
' <rtl_start_point>' reconvergence end = ' <rt_end_point>')
[Effects ' <no_of_set_pins>' SET and ' <no_reset_pin>' RESET
pin(s)]
```

Potential Issues

The violation message appears when there is a wrong connectivity on the set/reset tree.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and glitches on the reset tree.

How to Debug and Fix

The incremental schematic highlights the re-convergent combinational path starting from the primary input/output pin of a flip-flop or a black box to the async pins of a flip-flop.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Atspeed_25 rule in the incremental schematic. To do this, select the violation for the Atspeed_25 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Atspeed_25 rule in the incremental schematic. This is useful in debugging the violation of the Atspeed_25 rule.

Additionally, double-click the violation message to open the Atspeed_25_violation_001.csv report, which lists the flip-flops whose asynchronous pins are affected by the re-convergent paths

To fix the violation, correct the connection on the set/reset tree.

Message 3

```
[WARNING] Logic ' <start_point>' (' <shift-sim-val>', ' <capture-sim-val>') reconverges at or near ' <end_point>'. (Actual
Reconvergence start = ' <rtl_start_point>' reconvergence end =
' <rt_end_point>') [Affects ' <no_of_set_pins>' SET and
' <no_reset_pin>' RESET pin(s)]
```

Potential Issues

The violation message appears when there is a wrong connectivity on the set/reset tree.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and glitches on the reset tree.

How to Debug and Fix

The incremental schematic highlights the re-convergent combinational path starting from the primary input/output pin of a flip-flop or a black box to the async pins of a flip-flop.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Atpspeed_25 rule in the incremental schematic. To do this, select the violation for the Atpspeed_25 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Atpspeed_25 rule in the incremental schematic. This is useful in debugging the violation of the Atpspeed_25 rule.

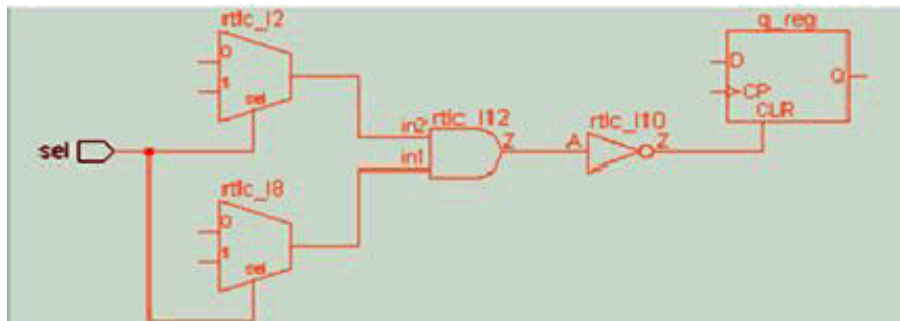
Additionally, double-click the violation message to open the Atpspeed_25_violation_001.csv report, which lists the flip-flops whose asynchronous pins are affected by the re-convergent paths

To fix the violation, correct the connection on the set/reset tree.

Example Code and/or Schematic

Example 1

Consider the following schematic:



The Atpspeed_25 rule reports a violation for the above example because the rtlc_12 and rtlc_18 multiplexers are converging.

Example 2

The following figure illustrates presence of combinational re-convergence to flip-flop's asynchronous pins:

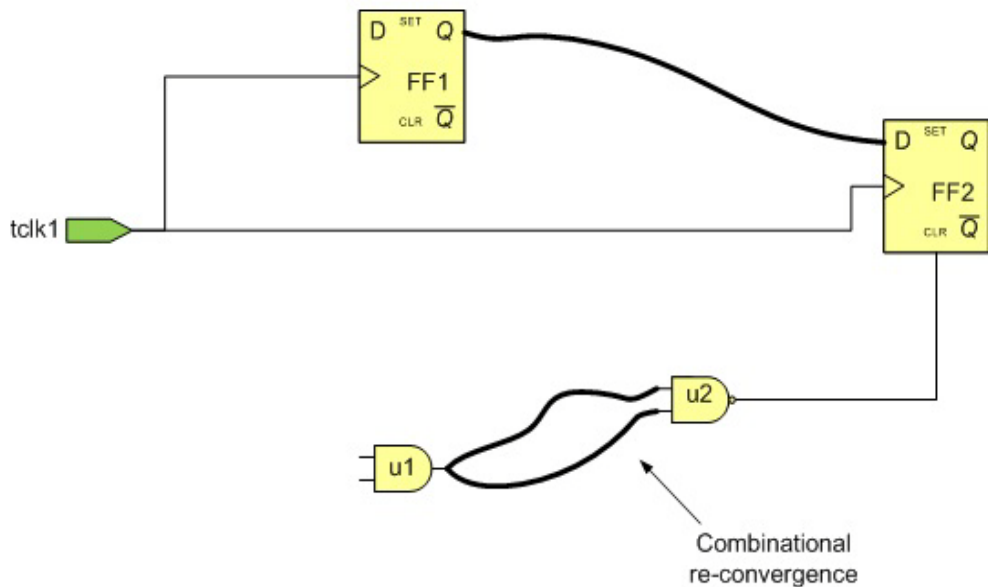


FIGURE 29. Combinational reconvergence feeding a reset pin

Both atspeed test launch and capture require that flip-flop asynchronous pins be held inactive. When combinational reconvergence structures drive the async pins, glitches may be produced by the ATPG tests. This compromise the test, and thereby, the fault coverage.

Default Severity Label

Info/Warning

Rule Group

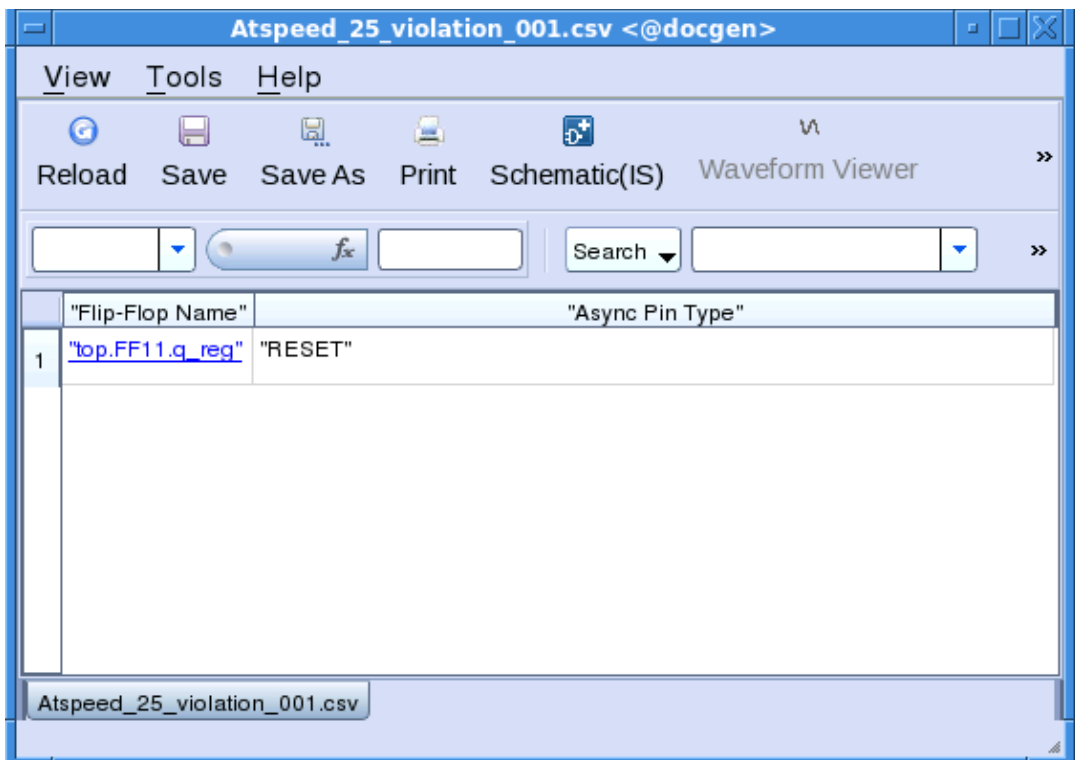
Atspeed Rules

Reports and Related Files

- *ff_async_reconvergence*: The *Atspeed_25* rule generates a text file named *ff_async_reconvergence* that lists all combinational paths that re-converge at async pins of flip-flops.
- **Atspeed_25_violation_002.csv**: The *Atspeed_25* rule generates the spreadsheet report, *Atspeed_25_violation_<xx>.csv*, which lists the flip-flops whose asynchronous pins are affected by the re-convergent paths. Here, *xxx* denotes a number, which is appended in the report name by SpyGlass, during report generation.

To view the *Atspeed_25_violation_<xx>.csv* report, double-click the [Message 2](#) generated for the *Atspeed_25* rule.

[Figure 102](#) lists illustrates the *Atspeed_25_violation_<xx>.csv* report:



	"Flip-Flop Name"	"Async Pin Type"
1	"top.FF11.q_req"	"RESET"

FIGURE 30. *Atspeed_25_violation_<xx>.csv* Report

Atspeed_26

Reports if an atspeed clock is applied on the sensitized fan-out cone of another atspeed clock

When to Use

Use this rule to ensure that an atspeed clock does not feed another atspeed clock.

Description

The Atspeed_26 rule reports a violation when an atspeed clock is applied on the sensitized combinational fan-out cone of another atspeed clock in the Capture (atspeed) mode.

The atspeed clock is specified through the `-atspeed` field of the *clock* constraint.

NOTE: *The use of the `-domain` field of the `clock` constraint neither causes nor excuses a violation of the Atspeed_26 rule.*

Method

If more than two clock constraints with `-atspeed` field are defined

 Traverse fan-in cone of all internal atspeed clocks (not on PI) in design and check if any other atspeed clock is hit during the traversal.

 If another atspeed clock is hit

 Report violation for each such occurrence.

The stop points for the traversal are sequential elements, black boxes, clock shaper cells, non-X paths and any blocked path.

Prerequisites

This rule is run when at least two `clock` constraints with the `-atspeed` field are specified in the SGDC file.

Rule Exceptions

This rule does not report a violation in the following cases:

- If the path from one atspeed clock to another traverses a divide-by sequential device.

For example, consider the following figure:

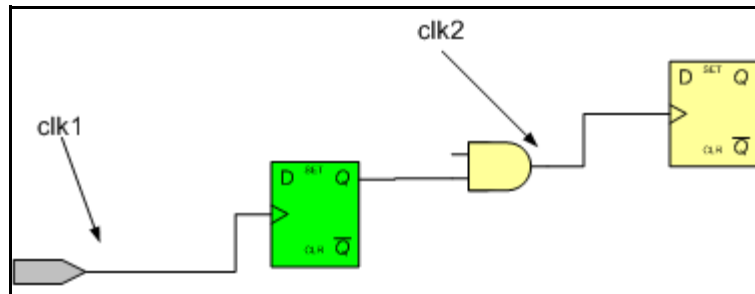


FIGURE 31. Example of a sequential traversal

In the above figure, the path from `clk1` to `clk2` traverses a divide-by sequential device. Therefore, a violation is not reported in this case.

- If the path from one atspeed clock to another traverses a clock shaper. For example, consider the following figure:

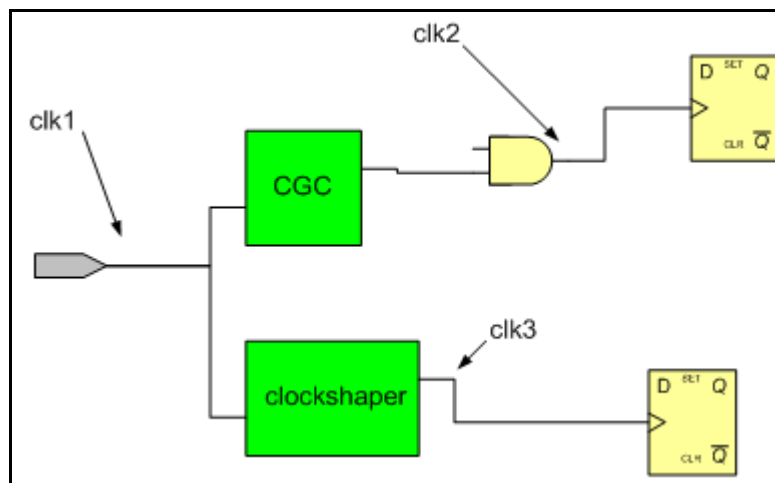


FIGURE 32. Example of a CGC and a clock shaper

In the above figure, the path from `clk1` to `clk3` traverses a clock shaper. Therefore, a violation is not reported for this path.

However, the path from `clk1` to `clk2` traverses an enabled CGC. Therefore, a violation is reported for this path.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- *clock* (mandatory): Use this constraint to define the clocks of a design. The `Atspeed_26` rule uses the `-atspeed` argument of the `clock` constraint.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

[Capture \(atspeed\)](#)

Messages and Suggested Fix

Message 1

[WARNING] `atspeed clock constraint '<overwriting-atspeedclk>' is applied on sensitized fanout of another atspeed clock '<overwritten-atspeedclk>' in '<mode-name> mode'`

Arguments

- Name of the atspeed clock that is overwriting the other atspeed clock, `<overwriting-atspeedclk>`
- Name of the atspeed clock that is being overwritten by another atspeed clock, `<overwritten-atspeedclk>`
- The mode name as Capture (atspeed), `<mode-name>`

Potential Issues

The violation is reported because an atspeed clock constraint is applied on the sensitized combinational fan-out cone of another atspeed clock constraint in the Capture (atspeed) mode.

Consequences of Not Fixing

Not fixing this violation may result in a difference between the simulation and actual/functional results.

How to Debug and Fix

View the incremental schematic of the violation message. The incremental schematic displays the path from the overwritten atspeed clock to the overwriting atspeed clock.

To fix the violation, verify that the highlighted path is correct and is not missing any testmode condition. If the path is correct, remove the constraint specifying the overwriting atspeed clock from the SGDC file.

Message 2

[WARNING] atspeed clock constraint '`<used_constraint>`' and '`<ignored_constraint>`' is applied on same net '`<net_name>`' in '`<mode_name>`'

Arguments

- Clock constraint defining atspeed_clock, `<used_constraint>`
- Clock constraint defining atspeed_clock, `<ignored_constraint>`
- Net Name, `<net_name>`
- Working mode, `<mode-name>`

Potential Issues

The violation message appears because two clock constraints are applied on the same net but only one is used.

Consequences of Not Fixing

Effect of ignored constraint will not be reflected on the design.

How to Debug and Fix

View the incremental schematic of the violation message. The incremental schematic displays the net on which constraint are applied. Two clock constraints on the same net will not work.

To fix the violation, analyze the constraints and modify accordingly.

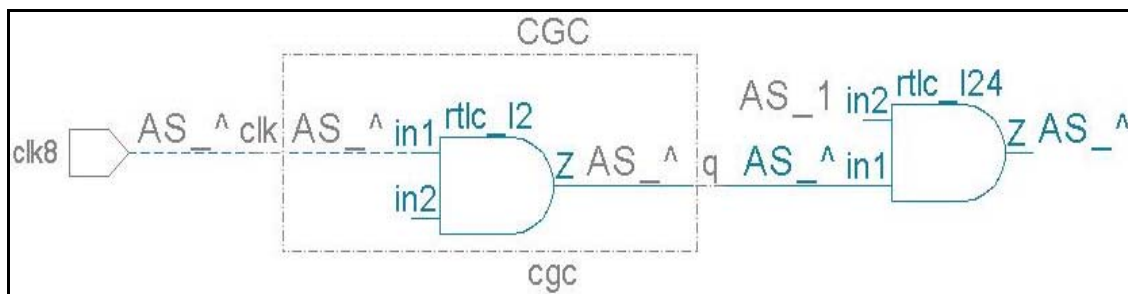
Example Code and/or Schematic

Consider the following SGDC file snippet:

```
clock -name clk8 -atspeed -freq 100
clock -name w10 -atspeed -freq 100
```

In this case, a violation is reported by the `Atspeed_26` rule because the `w10` atspeed clock constraint is applied on the sensitized combinational fan-out cone of the `clk8` atspeed clock constraint in the Capture (atspeed) mode.

The following schematic is generated for the violation message:



The above schematic shows the path from the overwritten atspeed clock (`clk8`) to the overwriting atspeed clock (`w10`). The path passes through a clock gating cell.

Default Severity Label

Warning

Rule Group

Atspeed Rules

Reports and Related Files

No related reports or files.

Atspeed_27

Reports if there is a convergence at the async pins of a flip-flop

When to Use

Use this rule to ensure that a single design node drives the async pins of a flip-flop.

Description

The Atpspeed_27 rule reports a violation if an async (set/reset) pin of a flip-flop is driven by multiple design nodes.

Rule Exception

The *Atpspeed_27* rule ignores no scan flip-flops (forced or inferred).

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftDsmConvergingPathsLimit*: The default value is 5. Set the value of the parameter to any positive integer value to specify the number of converging paths to be shown in the violation of the Atpspeed_27 rule. Other possible value is 0.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Atpspeed_27 rule uses the `-atspeed` argument of the `clock` constraint.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the Atspeed_27 rule:

```
[WARNING] ' <number>' design nodes are converging at node  
' <name>' which drives ' <no_of_set_reset_pins>' pins in  
' <mode>' [Converging nodes: <details>]
```

Arguments

- Number of converging nodes, <number>
- Node name, <name>
- Number of affected set/reset pins, <no_of_set_reset_pins>
- Operating mode of the rule, <mode>
- Details of the converging node, <details>

Potential Issues

The violation is reported because more than one design nodes are driving the async pins of the reported flip-flops.

Consequences of Not Fixing

Not fixing this violation may cause glitches at the set/reset pin, resulting in an undesired functionality of the flip-flops.

How to Debug and Fix

View the incremental schematic of the violation message. It displays the design nodes that drive the async pins of the reported flip-flop.

To fix the violation, perform the following steps:

1. Verify if all the constraints are present in the SGDC file.
2. Specify any missing constraint in the SGDC file or modify the design.
3. Check message spreadsheet to view the list of affected asynchronous pins.

Example Code and/or Schematic

Consider the following schematic of the Atspeed_27 rule:

Atspeed Test Rules

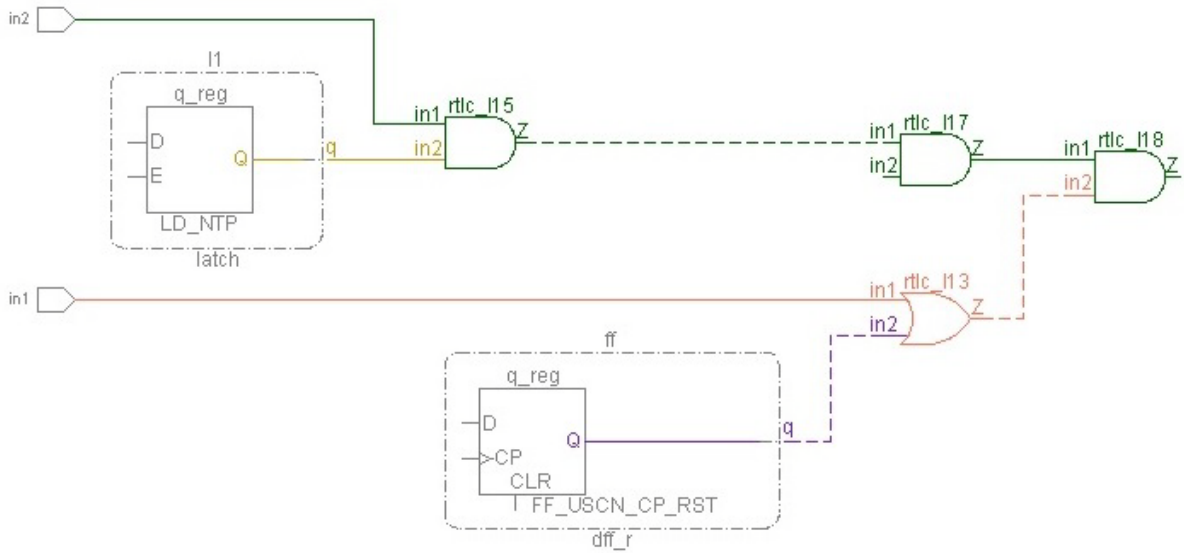
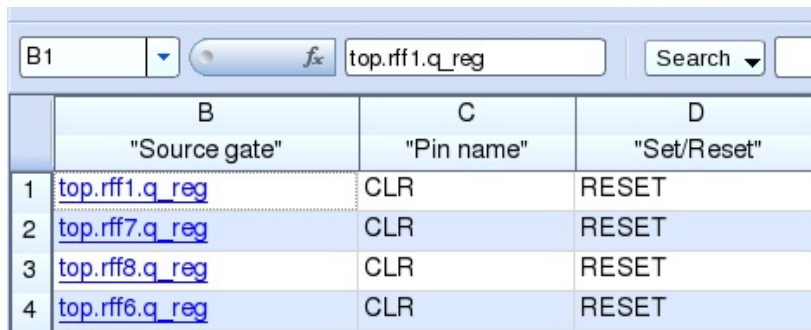


FIGURE 33. Atspeed_27 Schematic

The Atspeed_27 rule reports following violation message for the above design:

'4' design nodes are converging at node 'top.rtl_l18' which drives '0 Set and 4 reset' pins in 'Capture atspeed mode' [Converging nodes: top.ff.q_reg.Q, in1, top.i1.q_reg.Q, in2]

Double-clicking the violation message displays the following spreadsheet report:



	B	C	D
	"Source gate"	"Pin name"	"Set/Reset"
1	top.rff1.q_reg	CLR	RESET
2	top.rff7.q_reg	CLR	RESET
3	top.rff8.q_reg	CLR	RESET
4	top.rff6.q_reg	CLR	RESET

FIGURE 34. Aspeed_27 Spreadsheet report

Default Severity Label

Warning

Rule Group

Atspeed Rules

Reports and Related Files

No related reports or files.

Atspeed_29

Detects supply of bad clock during atspeed test.

When to Use

Use this rule to detect supply of bad clock during atspeed test.

Description

An atspeed clock can carry different frequencies depending on various frequency enable signals. However, enabling a specific frequency may cause inappropriate clock to be presented to certain parts of the design. The Atspeed_29 rule reports such situations.

Method

For each valid frequency of atspeed clock frequency defined by constraint, check and report for the flip-flops whose clock pin is neither tied to '0' nor getting any multiple of the clock frequency.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *atspeed_clock_frequency* (mandatory): Use this constraint to specify frequencies associated with a testclock.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The Atspeed_29 rule reports the following violation message:

[WARNING] '<count>' flip-flops are neither getting a clock frequency nor respective clock pin is tied to ground when atspeed clock frequency '<freq>' for atspeed clock '<clk_name>' is propagated in design '<du_name>'

Arguments

- Number of flip-flops that are neither getting any multiple of the atspeed clock frequency nor are tied to '0', <count>
- Atspeed clock frequency for which violation is reported, <freq>
- Name of the atspeed clock, <clk_name>
- Name of the top design, <du_name>

Potential Issues

The violation message is reported for flip-flops whose clock is not tied ground or getting specified frequency.

Consequences of Not Fixing

Not fixing this violation may cause inappropriate clock to be presented to certain parts of the design.

How to Debug and Fix

Double-click the violation message. A csv report opens up which should list all violating flip-flops. Clicking a flip-flop in the violation message displays the violating flip-flop in the schematic.

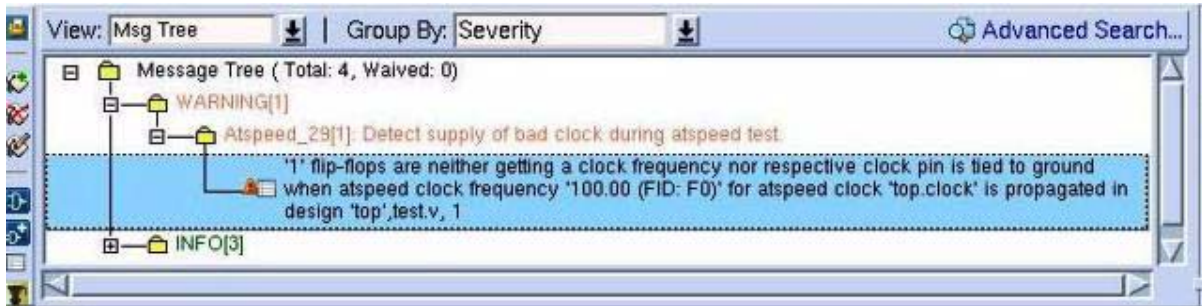
Example Code and/or Schematic

Consider the following SGDC declaration:

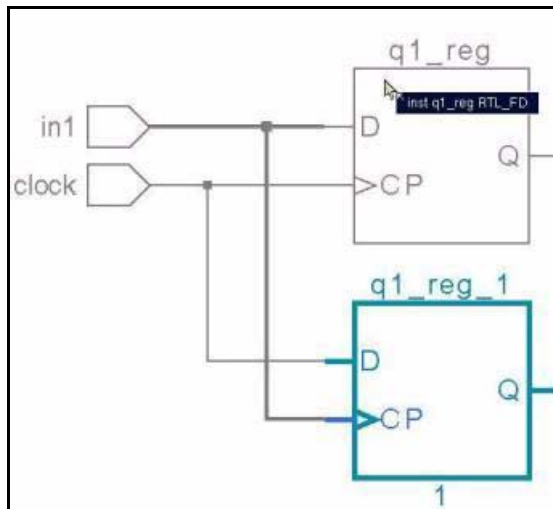
```
current_Design_top
  test_mode -name in1 -value 1
  clock -name clock -atspeed
  atspeed_clock_frequency -name clock -freqList 100
  -enables in1 -value 1
```

Atspeed Test Rules

The Atspeed_29 rule report following violation message for the above command:



When you double-click the violation message, the following Incremental Schematic is generated:



Default Severity label

Warning

Rule Group

Atspeed Rules

Reports and Related Files

No related reports or files.

Atspeed_30

Reports the presence of combinational re-convergence to flip-flop clock pin

When to Use

Use this rule to avoid glitches on clock tree.

Description

The `Atspeed_30` rule reports violation for reconverging combinational paths that start from a primary input port or the output pin of a flip-flop and reconverge at the clock pin of any flip-flop.

Method

Traverse unblocked path in fan-in of clock pin of all flip-flops.

If any path is hit again for the fan-in traversal of the same pin

Report a violation.

Highlight the reconvergent path.

Rule Exception

The `Atspeed_30` rule ignores no scan flip-flops (forced or inferred).

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dft_report_all_paths_between_reconvergence_start_and_end`: The default value of this parameter is off. Set the value of the parameter to on to report all paths between re-convergence start and end points.

Constraint(s)

- `test_mode` (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[INFO] Reconvergence report file '<report-file-name>' is generated

Arguments

Name of the report file, <report-file-name>

Potential Issues

The violation message appears when the text report file is generated.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[WARNING] [Clock/Clock Enable] Logic '<start_point>' reconverges at or near '<end_point>'. (Actual Reconvergence start = '<rtl_start_point>' reconvergence end = '<rtl_end_point>') [Effects '<no_of_clock_pins>' flip-flop(s)]

Potential Issues

The violation message appears when there is a wrong connectivity on the clock tree.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and other faults on the clock tree.

How to Debug and Fix

Atspeed Test Rules

The Incremental Schematic highlights the reconvergent combinational path starting from the primary input/output pin of a flip-flop to the clock pin of a flip-flop.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Atpspeed_30 rule in the Incremental Schematic. To do this, double-click the violation for the Atpspeed_30 rule and open the Incremental Schematic window.

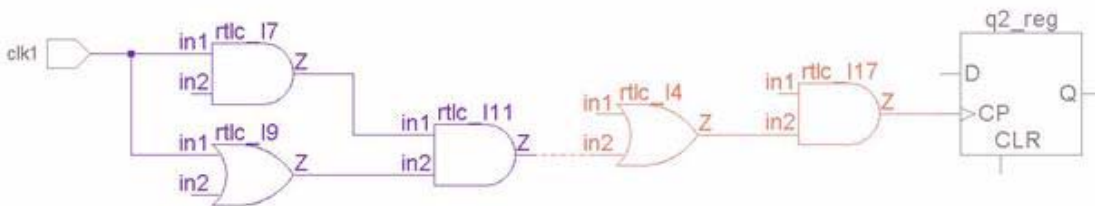
The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Atpspeed_30 rule in the incremental schematic. This is useful in debugging the violation of the Atpspeed_30 rule.

To fix the violation, correct the connection on the clock tree.

Example Code and/or Schematic

Example 1

Consider the following figure:



The Atpspeed_30 rule reports a violation for this case because signal from `clk1` re-converges at the output of the AND gate, `rtlc_I11`, which is driving clock pin of flop-flop, `q2_reg`.

Example 2

The following figure illustrates the presence of combinational re-convergence to flip-flop's clock pin:

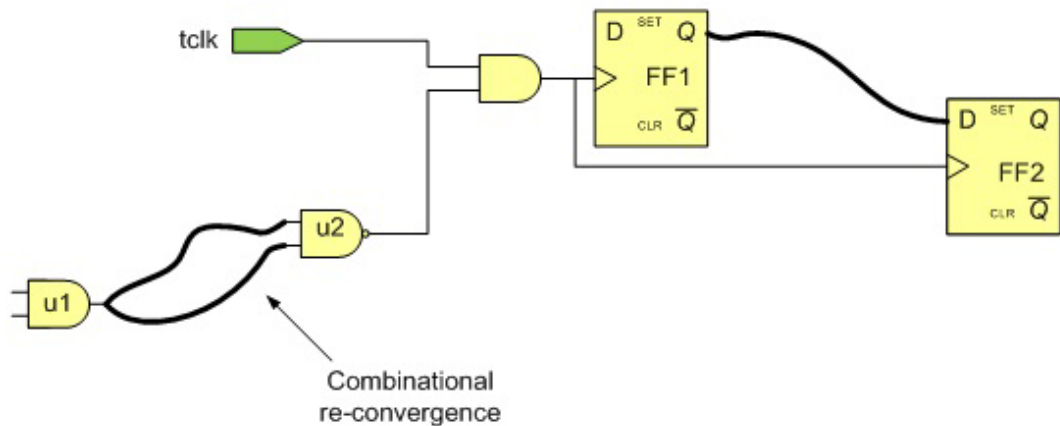


FIGURE 35. : Combinational re-convergence feeding flip-flop clock pins

Circuits with re-convergence feeding into clock pins can give rise to one of the following conditions when none is expected:

- Prevent a proper pulse for launch or capture
- Inject a pulse

Such cases can compromise coverage.

Default Severity Label

Info/Warning

Rule Group

Atspeed Rules

Reports and Related Files

ff_clock_reconvergence: The Atspeed_30 rule generates this report in the `<current-working-directory>/spyglass_reports/dft/` directory. This report lists all re-convergent combinational paths that start from a primary

Atspeed Test Rules

input port or the output pin of a flip-flop and re-converge at the clock pin of a flip-flop.

Atspeed_31

Ensure that each clock shaper clock pin is directly controlled via PLL in the capture atspeed mode

When to Use

Use this rule to check whether every clock shaper (CS) in the design is driven by a PLL clock or PLL reference clock, if PLL is not present.

Description

The Atspeed_31 rule reports violation for clock shaper in the design, which is neither driven by a PLL clock or a PLL reference clock, if PLL is not present.

During atspeed test, use functional clock for testing as launch and capture pulse should come as *atspeed*. Since functional clock comes from *On-Chip Clock Controller (OCC)* which is generally PLL, we must ensure that PLL drives all the clock sources. This check is only relevant for SOC where we have a PLL on chip. At block level this check is not relevant because PLL block may not have been instantiated yet.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- ***test_mode*** (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- ***clock_shaper*** (mandatory): Use this constraint to declare:
 - Clock shaper module to control clock pulse propagation in the design
 - PLL module using `-pll` option
 - Clock divider using `-divider` option

- *clock* (optional): Use this constraint to define the clocks of a design. The Atspeed_31 rule uses the `-pll_reference` and `-atspeed` arguments of the `clock` constraint.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[INFO] Rule checking skipped for design '<top>' as neither PLL module nor PLL reference clock found

Potential Issues

This violation is reported because neither PLL module nor PLL reference clock is found in the design.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation add either PLL module or PLL reference clock using the appropriate SGDC command.

Message 2

[WARNING] No sensitized path found from <PLL or PLL reference clock> to the clock shaper '<clock_shaper_name>' clock pin '<clk_pin_name>'. No <PLL or PLL reference clock> found in the combinational fanin cone

Potential Issues

This violation is reported because neither PLL module nor PLL reference clock is found in the combinational fanin cone of the clock shaper clock input.

Consequences of Not Fixing

Not fixing the violation may result in chip malfunction in atspeed.

How to Debug and Fix

The incremental schematic of the violation message displays the path from the clock shaper clock pin to the nearest source.

To fix the violation define a PLL module or PLL reference clock in the clock shaper fanin.

Message 3

[WARNING] No sensitized path found from <PLL or PLL reference clock> to the clock shaper '`<clock_shaper_name>`' clock pin '`<clk_pin_name>`'. <PLL or PLL reference clock> '`<pll_instance_path_name_or_pll_reference_clock_name>`' found in the combinational fanin cone which got blocked in capture atspeed mode

Message 4

[WARNING] No sensitized path found from <PLL or PLL reference clock> to the clock shaper '`<clock_shaper_name>`' clock pin '`<clk_pin_name>`'. <PLL or PLL reference clock> '`<pll_instance_path_name_or_pll_reference_clock_name>`' found in the combinational fanin cone which got blocked in capture atspeed mode. 'N' other potential <PLL / PLL reference clock>: x1, x2, x3, ..

Potential Issues

This violation is reported because all the PLL clocks in the clock shaper fanin are blocked in the capture atspeed mode.

Consequences of Not Fixing

Not fixing the violation may result in chip malfunction in atspeed.

How to Debug and Fix

Incremental schematic of the violation message displays the blocked path from a PLL clock to the clock shaper clock input in two different colors and also highlights the other PLL clocks in the fanin.

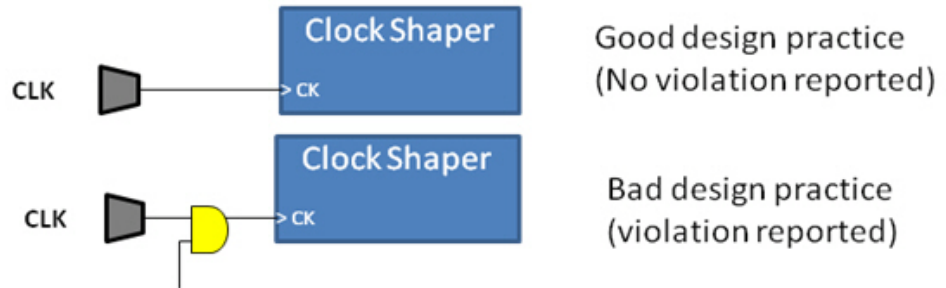
To fix the violation, ensure that a sensitized path from a PLL clock to the clock shaper clock pin is available in capture atspeed mode using `test_mode SGDC` command.

Example Code and/or Schematic

Example 1

In the following figure, no PLL module is present and PLL reference clock is considered:

SGDC: clock -name CLK -atspeed _pll_reference



Example 2

In the following figure, PLL module is present:



Example 3

In the following figure, clock divider is used as a buffer while rule checking:



Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

None

Atspeed_32

Reports the presence of cascaded reset re-convergence to asynchronous pins of flip-flop

When to Use

Use this rule to detect cascaded reset re-convergent signals that can impact at-speed transition coverage.

Rule Exception

The *Atspeed_32* rule ignores rule checking for unate re-convergence, that is, paths with no phase difference.

Description

The *Atspeed_32* rule reports violation for re-converging asynchronous paths at asynchronous pins of a flip-flop.

The rule reports re-converging asynchronous paths that start from a primary input port or the output pin of a flip-flop and re-converge before reaching the asynchronous set/reset pin of any flip-flop.

An asynchronous path is a path from the inputs a flip-flop through an asynchronous set/reset pin to its Q-output.

The following figure shows the example of a re-converging asynchronous path:

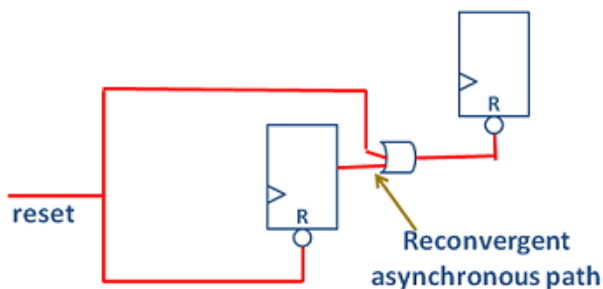


FIGURE 36. Re-convergent asynchronous path

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dft_ignore_unate_reconvergence*: The default value is on. Set the value of the parameter to off to check for unate re-convergence.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[WARNING] Logic '<start_point>' ('<shift-sim-val>', '<capture-sim-val>') reconverges at or near '<end_point>'. (Actual Reconvergence start = '<rtl_start_point>' reconvergence end = '<rtl_end_point>') [Effects '<no_of_set_pins>' SET and '<no_reset_pin>' RESET pin(s)]

Potential Issues

See [Potential Issues](#).

Consequences of Not Fixing

See [Consequences of Not Fixing](#)

How to Debug and Fix

See [How to Debug and Fix](#)

Message 2

[WARNING] Logic '<start_point>' reconverges at or near '<end_point>'. (Actual Reconvergence start = '<rtl_start_point>' reconvergence end = '<rtl_end_point>')

[Effects '<no_of_set_pins>' SET and '<no_reset_pin>' RESET pin(s)]

Potential Issues

The violation message appears when there is a wrong connectivity, that is re-convergent paths, on the set/reset tree.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and glitches on the reset tree.

How to Debug and Fix

The incremental schematic highlights the re-convergent asynchronous path starting from the primary input/output pin of a flip-flop or a black box to the async pins of a flip-flop.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Atpspeed_32 rule in the incremental schematic. To do this, double-click the violation for the Atpspeed_32 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Atpspeed_32 rule in the Incremental Schematic. This helps in debugging the violation of the Atpspeed_32 rule. To fix the violation, correct the connection on the set/reset tree.

Example Code and/or Schematic

Example 1

Consider the following schematic, which illustrates the presence of re-convergent path in the reset signal:

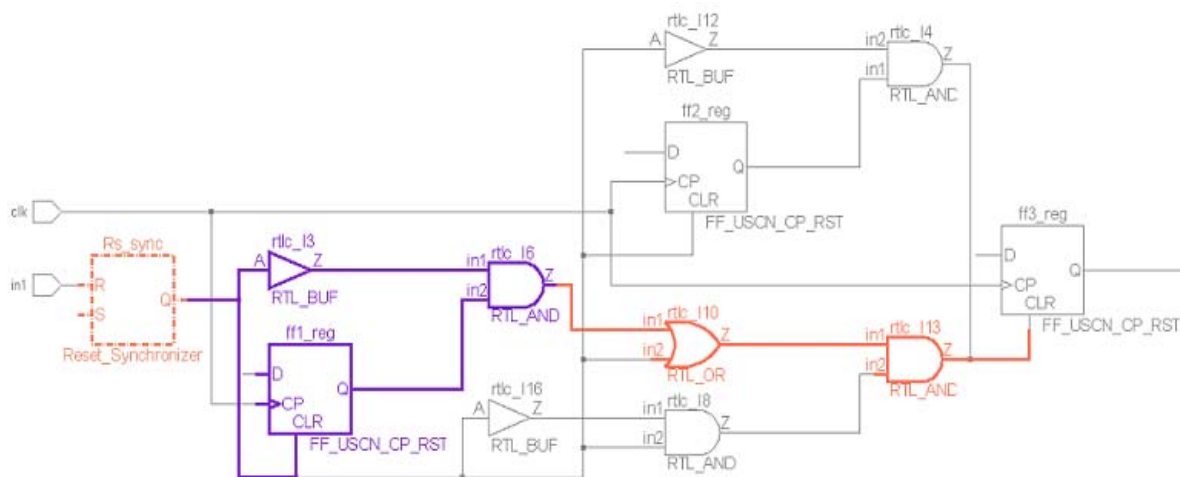


FIGURE 37. Re-convergent path example

In [Figure 37](#), reset signal to CLR pin of the flip-flop, `ff3_reg`, which started from the reset synchronizer, `Rs_sync`, has re-converged at the RTL buffer, `rtl_c_i16`. Therefore, the `Atspeed_32` rule reports a violation message for this case.

Example 2

Consider the following schematic where there is no re-convergent path for both `DFTRSTDISABLE` and `DFTRSTDISABLE1` signals:

Atspeed Test Rules

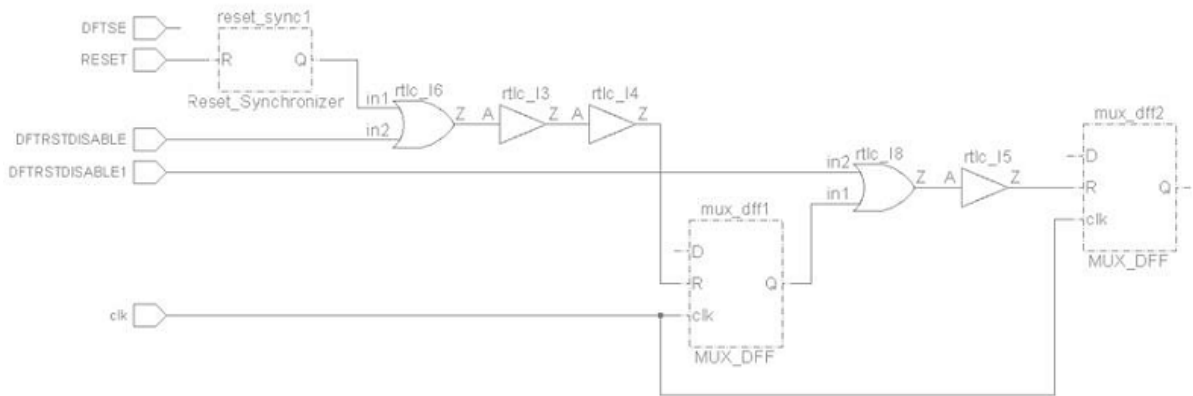


FIGURE 38. No re-convergent path in the design

The design illustrated in [Figure 38](#) represents a good design example as no re-convergent path is there for the reset signal.

Default Severity Label

Warning

Rule Group

Atspeed

Reports and Related Files

None

Atspeed_33

All latch inputs should be driven by controllable sources in the capture-atspeed mode

When to Use

Use this rule when you do not want to treat the latches in the design as X-source for APTG pattern generation, which otherwise may increase the pattern count and aborted faults.

Description

The Atspeed_33 rule reports violation when any of the following conditions are true:

- A set/reset pin is not controllable during capture-atspeed mode
- Enable pin is not driven by any atspeed clock
- Data pin is not driven by controllable sources

A latch data pin is driven by a controllable source, if the sensitized path to the following are found:

- scannable flip-flop
- controllable primary input
- scan wrapped black box

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's -testclock and -atspeed arguments for this rule.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

[WARNING] Latch '<name>' will be treated as x-source by ATPG (<reason>) in <mode >

Arguments

- Name of the latch, <name>
- Reason for treating the latch as x-source, <reason>
- Operating mode name, <mode>

Potential Issues

If latch is not driven by a controllable source (scan flip-flop or scannable black box or controllable primary port) and other inputs are not controllable in capture-atspeed mode, then ATPG may treat it as X source.

Consequences of Not Fixing

Number of patterns required by ATPG will be higher and may result in aborted faults.

How to Debug and Fix

Double-click the violation message to view the Incremental Schematic displaying the violating latch. Analyze the latch accordingly.

Example Code and/or Schematic

Consider the following violation message generated by the Atspeed_33 rule:

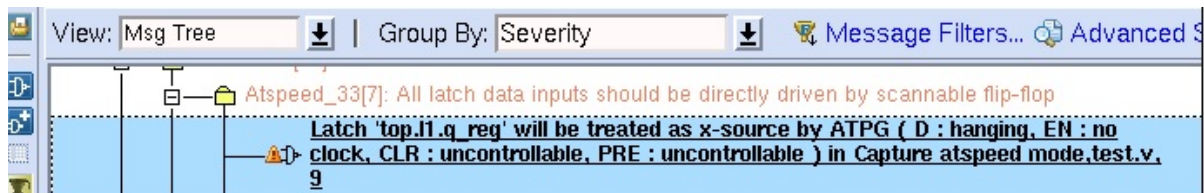


FIGURE 39. Atspeed_33 violation message

The above message reports that the latch, top.l1.q_reg, is treated as x-

source by ATPG because of the following reasons:

- D-pin is hanging
- No clock is specified in the Enable pin
- Clear (CLR) is not disabled
- Preset (PRESET) is not disabled

Double-clicking the message displays the following incremental schematic:

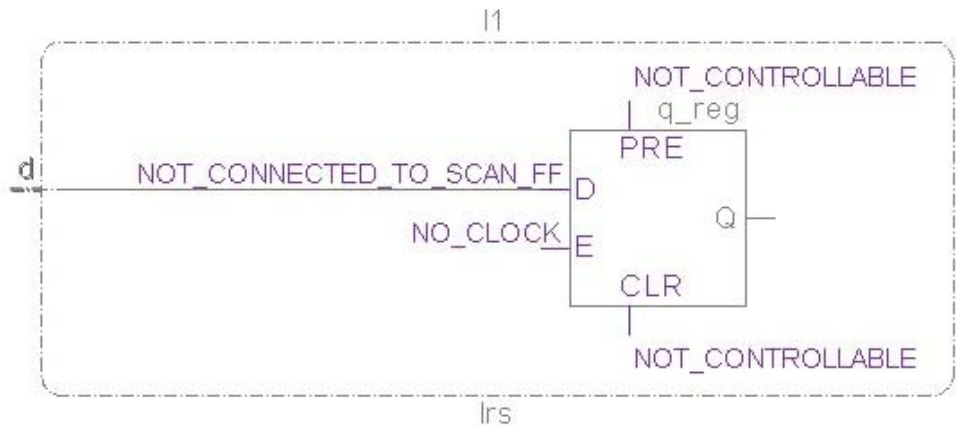


FIGURE 40. Atspeed_33 Incremental Schematic

Default Severity Label

Warning

Rule Group

Atspeed Rules

Reports and Related Files

No related reports or files.

Atspeed_34

Reports flip-flops having feedback from q-terminal to d-terminal

When to Use

Use this rule to view flip-flops with its feedback loop from q-terminal to d-terminal.

Description

The *Atspeed_34* rule reports flip-flops having feedback from q-terminal to d-terminal.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dft_state_holding_ff_identification_effort_level*: The default value of the parameter is 4. Set the value of the parameter to any natural number to change the effort level for identifying the state-holding flip-flops.

Constraint(s)

None

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

[WARNING] Flip-flop '`<flip-flop>`' has a 'Q' to 'D' feedback which makes '`<fault(s)>`' fault(s) undetectable

Arguments

- Name of the flip-flop having 'Q' to 'D' feedback, `<flip-flop>`
- Type of the fault(s) which is(are) undetectable, `<fault(s)>`

Potential Issues

A violation message is reported when a feedback loop is present from q-terminal to d-terminal of a flip-flop.

Consequences of Not Fixing

Feedback loops from flip-flop out to flip-flop data makes transition faults undetectable, resulting in lower transition coverage.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the corresponding flip-flop having 'Q' to 'D' feedback, feedback path including all logic elements and nets. To fix the violation, see Example Code and/or Schematic.

Example Code and/or Schematic

Consider the following schematic:

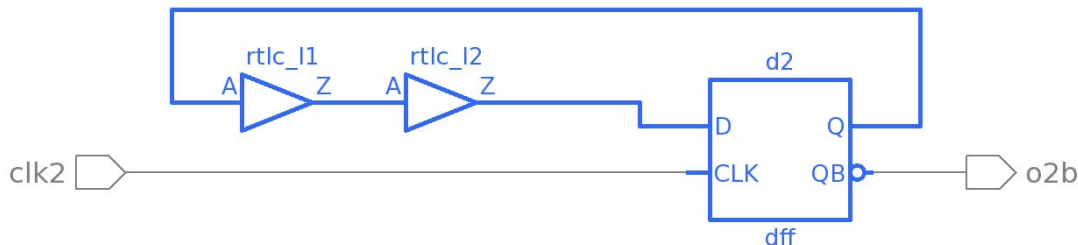


FIGURE 41. Schematic for Atspeed_34

In the above example, the 'Q' terminal of the flip-flop 'top.d2' is connected to its 'D' terminal through 'top.rtlc_I1' and 'top.rtlc_I2', which ultimately creates a 'Q' to 'D' feedback. Due to this connection both 't01' and 't10' faults are undetectable in the 'D' terminal of the particular flip-flop. Therefore Atspeed_34 rule reports the following violation message:

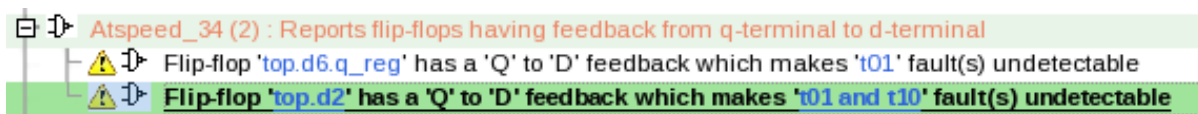


FIGURE 42. Violation Messages for Atspeed_34

Atspeed Test Rules

The above violation message states that the 'top.d2' has a 'Q' to 'D' feedback so that 't01' and 't10' faults are undetectable.

Default Severity Label

Warning

Rule Group

Atspeed Rules

Reports and Related Files

No related reports or files.

BIST Rules

Overview

The BIST rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>BIST_01</i>	Flip-flops that have more than the specified number of flip-flops and black boxes in their fan-in cones
<i>BIST_02</i>	Identifies the gates with a large number of inputs
<i>BIST_03</i>	Flip-flops that remain in unknown state after initialization
<i>BIST_04</i>	Primary output ports or data pin of scannable flip-flops, which have unknown values in their fan-in cones
<i>BIST_05</i>	Primary output ports or data pin of scannable flip-flops, which have an unblocked TIE-X cell output in their fan-in cone

BIST_01

Restrict input cone width for BIST

When to Use

Use this rule to identify flip-flops that have fan-in cone width more than the specified value.

Description

The BIST_01 rule reports violation for flip-flops that have more than the specified number of flip-flops and black boxes in their fan-in cones. The default value for the width is 150.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode capture conditions. The data pin fan-in cone for each flip-flop is walked through unblocked paths. Stop the walk at sequential devices or at nodes with non-x simulation values. All transparent latches are considered as unblocked and the walk continues through the data pin only. Black boxes are considered as sequential. Any flip-flop with a fan-in cone wider than a user-supplied number is flagged.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *flipInFaninCount*: The default value is 150. Set the value of the parameter to any natural number to set the maximum limit on the total number of flip-flops and black boxes allowed in the data pin fan-in cone of any flip-flop as checked by the *BIST_01* rule.
- *showPath*: The default value is off. Set the value of the parameter to on to enable the BIST_01 rule to display complete path on the incremental schematic.
- *noBlackBoxReporting*: The default value is off. Set the value of the parameter to on to suppress the rule messages for nets made uncontrollable or unobservable by black boxes.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Fanin cone to flip-flop '<flip-flop-name>' has <count> (>\$flipInFaninCount) sources, <m> flip-flop(s) and <n> blockbox(es)

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to debug and Fix

For more information on debugging and fixing the violation, click [How to debug and Fix](#).

Message 2

[WARNING] Fanin cone to flip-flop '<flip-flop-name>' has <m>

(>\$fl opl nFani nCount) fl i p-fl op(s)

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to debug and Fix

For more information on debugging and fixing the violation, click [How to debug and Fix](#).

Message 3

[WARNI NG] Fani n cone to fl i p-fl op ' <fl i p-fl op-name>' has <n>
(>\$fl opl nFani nCount) bl ackbox(es)

Arguments

- Name of the flip-flop with wide fan-in cone. (<flip-flop-name>)
- Total number of flip-flops and black boxes in the fan-in cone. <count>
- Number of flip-flops in fan-in cone. <m>
- Number of black boxes in fan-in cone. <n>

Potential Issues

A violation is reported if there exist flip-flops with large fan-in cones.

Consequences of Not Fixing

The number of random patterns for BIST is an exponential function of the number of flip-flops feeding a particular signal. If the flip-flop count is excessive, the generated pattern set may not be exhaustive. In such a case, the fault coverage will be compromised or the ATE time will be excessive.

How to debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic displays the flip-flops that have fan-in cone width more than the specified value (default = 30)

You can also view the violations for the Info_testmode (under shift condition) and Info_path rules along with the violation of the BIST_01 rule in the Incremental Schematic window. To do this, double-click the violation for the BIST_01 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the BIST_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the BIST_01 rule.

To fix the violation, modify the logic to reduce the fan-in cone size.

NOTE: Any combination of flip-flops and black boxes that sum to more than the *flopInFaninCount* rule parameter limit will cause a message. The message wording adjusts to the three possibilities of fan-in cones driven by only flip-flops, fan-in cones driven by both flip-flops and black boxes and fan-in cones driven only by black boxes.

Example Code and/or Schematic

Example 1

Consider the following example:

```
module BistConeTooWide (i1, i2, i3, i4, i5, clk, out);
  input i1, i2, i3, i4, i5, clk;
  output out;
  reg f1, f2, f3, f4, f5, out;
  wire data;

  assign data=f1&&f2&&f3&&f4&&f5;

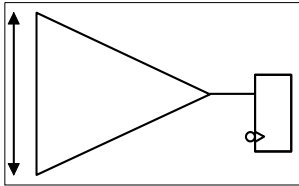
  always @(posedge clk) begin
    f1<=i1;
    f2<=i2;
    f3<=i3;
    f4<=i4;
    f5<=i5;
    out<=data;
  end
endmodule
```

The above Verilog code demonstrates a bad design scenario.

To correct the problem, insert test points.

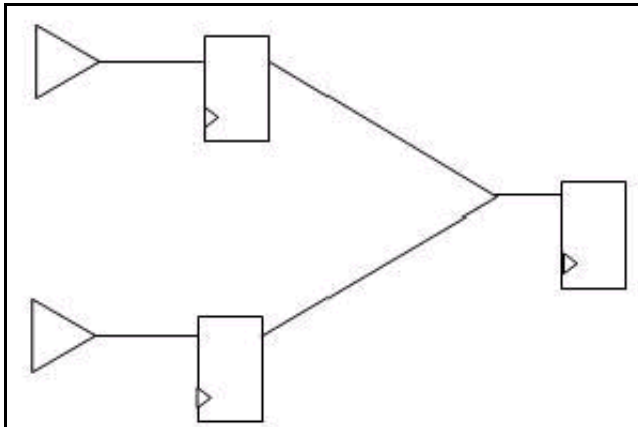
Example 2

Consider the following figure:



The above figure illustrates a bad design scenario.

Now, consider the following figure:



The above figure illustrates a converted better design.

Default Severity Label

Warning

Rule Group

BIST Rules

Reports and Related Files

[bist_ready_summary.rpt](#): This report lists the number of flip-flops for which fanin cone width is higher than the value specified by the [flopInFaninCount](#)

parameter.

BIST_02

Limit the number of gate inputs

When to Use

Use this rule to identify the gates with a large number of inputs.

Description

The BIST_02 rule reports violation for combinational gates that are driven by a gate instance with large fan-in.

Rule Exceptions

The BIST_02 rule does not report violation for the following cases:

- Flip-flop
- Latch
- Black box (including *module_bypass*)
- Memory

Method

The netlist is examined for excessively wide gates.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *gateInputCount*: The default value is 8. Set the value of the parameter to any natural number to set the maximum limit on the number of inputs of a primitive combinational logic element as checked by the BIST_02 rule.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Capture

Message Details

[WARNING] Gate driving '<comb-gate-name>' potentially has '<num>' fan-ins (><gateInputCount>)

Arguments

- Name of the combinational gate, <comb-gate-name>
- Number of fan-ins, <num>

Potential Issues

A violation is reported when there is a gate with large number of inputs.

Consequences of Not Fixing

Large gates are inherently difficult to control and hence are more likely to exhibit random pattern resistance. This can be especially important with logic BIST test methods since the probability of achieving particular output states rapidly decreases with increasing number of inputs.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the gates whose fan-in number is greater than the specified value (default = 8).

You can also view the violations for the Info_testmode (under shift condition) and Info_path rules along with the violation of the BIST_02 rule in the Incremental Schematic window. To do this, double-click the violation for the BIST_02 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the BIST_02 rule in the Incremental Schematic window. This is useful in debugging the violation for the BIST_02 rule.

To fix the violation, do not select gate with large number of inputs.

Example Code and/or Schematic

Example 1

Consider the following sample Verilog code:

```
module GateTooWide (i1, i2, i3, i4, i5, i6, i7, i8, i9, clk,
out);
  input i1, i2, i3, i4, i5, i6, i7, i8, i9, clk;
  output out;

  reg out;

  wire data;

  assign data=i1&& i2&& i3&& i4&& i5&& i6&& i7&& i8&& i9;

  always @(posedge clk)
  begin

    out<=data;
  end
endmodule
```

The BIST_02 rule reports violation for the above example as it contains a gate with large number of inputs.

Example 2

Consider the following sample VHDL code:

```
architecture arch_GateTooWide of GateTooWide is
  Signal data: std_logic;
  Begin
    Data <= i1 and i2 and i3 and i4 and i5 and i6 and i7
and i8 and i9;
    process(clk)
  begin
    If (clk'event and clk = '1' ) then
  o <= data;
  End if;
  End process;
```

```
end architecture;
```

The BIST_02 rule reports violation for the above example as it contains a gate with large number of inputs.

Default Severity Label

Warning

Rule Group

BIST Rules

Reports and Related Files

[bist_ready_summary.rpt](#): This report lists the number of gates for which fanin count is higher than the value specified by the [gateInputCount](#) parameter.

BIST_03

Checks for unknown state after initialization sequence

When to Use

Use this rule to identify non-scannable flip-flops that remain in unknown state after initialization.

Rule Description

The BIST_03 rule reports violation for the non-scannable flip-flops that remain in unknown state after initialization.

The rule ignores scannable flip-flops and check for only non-scannable flip-flops.

Rule Exceptions

The BIST_02 rule does not report violation for the scannable flip-flops.

Method

Simulate power and ground and any available “initialize_for_bist” conditions. If any flip-flop or latch has an x-value after simulation, then this rule is violated.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [test_mode](#) (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [initialize_for_bist](#) (mandatory): This constraint is merged with test_mode constraint and can be used as an argument to the test_mode constraint.

Operating Mode

intialize_for_bist

Messages and Suggested Fix

[WARNING] Flip-flop '`<flip-flop-name>`' is not reset even after the reset is applied

Arguments

Name of the non-scannable flip-flop output pin, `<flip-flop-name>`

Potential Issues

A violation is reported when a non-scannable flip-flop without set or reset is used and D-pin in feedback mode.

Consequences of Not Fixing

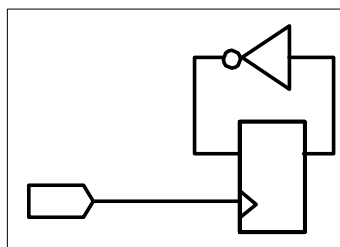
The ability to reset the circuit state is an important criterion for a number of test-related tools. Knowledge of where sequential devices remain x after application on an initialization sequence may also give clues about circuit design that can improve ATPG as well as possibly improving the system performance of the design.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the non-scannable flip-flops that remain in unknown state after initialization.

Example Code and/or Schematic

Consider the following example:



The BIST_03 rule reports violation for the above design because there is a divide by two registers that can not be initialized.

Default Severity Label

Warning

Rule Group

BIST Rules

Reports and Related Files

- *dft_initialized_ffs*: The BIST_03 rule also generates the `dft_initialized_ffs` report that lists the non-scannable flip-flops, which are initialized after the *initialize_for_bist* constraint is simulated.
- *bist_ready_summary.rpt*: This report lists the number of flip-flops for which the initial value is not set even after the initialization sequence is applied.

BIST_04

Ensure that fan-in cones of the scannable flip-flops do not have unknown values at observable points

When to Use

Use this rule to identify primary output ports or data pins of scannable flip-flops, which have unknown values in their fan-in cones.

Description

The BIST_04 rule reports violation for primary outputs or inout ports and data pins of scannable flip-flops that have unknown values in their fan-in cones.

The BIST_04 rule flags if any one of the following is found in the fan-in cone of a primary output or D-pin of a scan flip-flop:

- Non-scan flip-flop
- Floating net
- Primary input/inout port
- Combinational loop
- Black box instance
- Non-transparent latch
- Scan Flip Flops in different clock domain

Rule Exception

The BIST_04 rule ignores the primary input/output port, if it is specified with the *force_ta* constraint.

Method

Simulate capture mode.

Walk the unblocked fan-in cones on all primary output/inout ports and on D-pins for all scannable flip-flops (excluding flip-flops marked as “noscan”). Stop the walk at nodes with non-x simulation values.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftReportOnlyBBForBIST*: Default value is on. Set the value of the parameter to off to ignore all cases, except for the black box driving an observable point.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_detect_shadow_latches*: The default value is on. Set the value of the parameter to off to treat all the latches normally.
- *dft_treat_primary_inputs_as_x_source*: The default value is off. Set the value of the parameter to on to report violations at ports.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_ta* (optional): Specifies the controllabilities and/or observabilities for ports/pins/nets.
- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Specifies the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.

Operating Mode

Capture

Message Details

Message 1

[WARNING] Observable point <name> is driven by <number> 'X' sources(s). {<category-wise-breakup>}

Arguments

- Primary output/inout port or D pin of a scannable flip-flop, <name>
- total number of X sources feeding the observable point, <number>
- Number of X sources of each category, <category-wise-breakup>

Potential Issues

A violation is reported because there are X source(s) feeding a scan capture point.

Consequences of Not Fixing

Reliable capture means that predictable values are on scan inputs just prior to the capture clock. Unknown values can cause good chips to fail manufacturing tests.

Unknown values may also decrease at-speed coverage by preventing use of some scannable flip-flops from launching transitions.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the path from observable point up to the cause of propagation of unknown value.

You can also view the violations for the Info_testmode (under capture condition) and Info_path rules along with the violation of the BIST_04 rule in the Incremental Schematic window. To do this, double-click the violation for the BIST_04 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the BIST_04 rule in the Incremental Schematic window. This is useful in debugging the violation for the BIST_04 rule.

To fix the violation, block the path from non-scan flip-flop to capture point.

Message 2

[INFO] Ignoring '<LIST>' type of X-sources(s) as specified by parameter 'dft_ignore_x_sources_for_bist'

Potential Issues

This is an informational message.

Consequences of Not Fixing

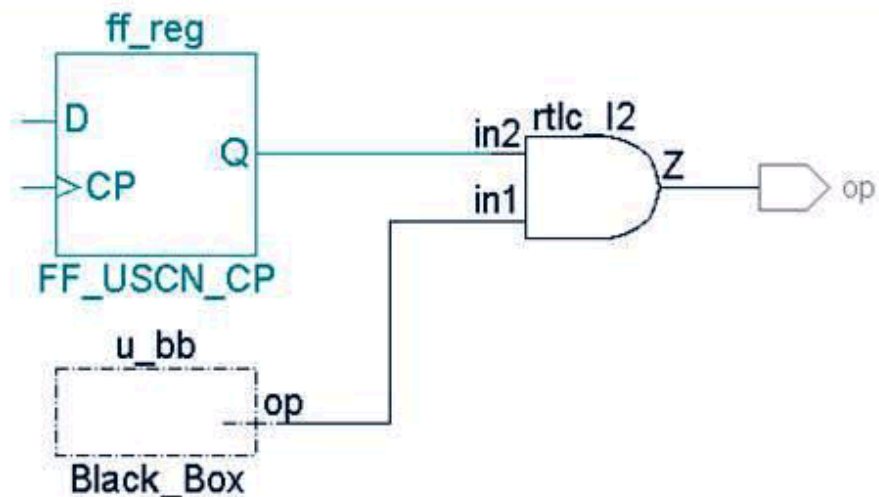
This is an informational message.

How to Debug and Fix

This is an informational message.

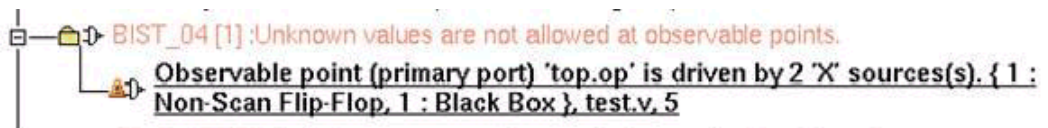
Example Code and/or Schematic

Consider the following schematic:



The above schematic shows the non-scan flip-flops and black box driving the D-pin of a scan flip-flop.

The rule reports the following violation message for the above example:



Default Severity Label

Warning

Rule Group

BIST Rules

Reports and Related Files

[bist_ready_summary.rpt](#): This report lists the number of observable points which are driven by unknown ('X') sources.

BIST_05

Ensure that in scan capture mode TIE-X cells outputs are bypassed

When to Use

Use this rule to identify primary output ports or data pin of scannable flip-flops, which have an unblocked TIE-X cell output in their fan-in cone

Description

The BIST_05 rule reports violation for TIE-X cell outputs, which are not bypassed.

Method

In scan capture mode; for each memory output, analyze the fan-out path from involved TIE-X cell output; a violation occurs if the output is not bypassed, that is, connected to a port or scan flip-flop data pin

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's - `testclock` argument for this rule.
- *tie_x* (for <tie_x_cell_list>): Specifies the X-generator design unit (black box) names.

Operating Mode

Capture

Messages and Suggested Fix

```
[WARNING] Instance '<inst-name>' drives '<flip-flop-name>' ['flip-flop']
```

Arguments

- Name of the TIE_X instance, <inst-name>
- Name of the flip-flop being driven by the TIE_X instance, <flip-flop-name>

Potential Issues

A violation is reported because the TIE_X instance is not properly bypassed.

Consequences of Not Fixing

TIE-X outputs are unpredictable, thus considered as X; bypassing them prevents the propagation of X inside the design. Rationale prevents test coverage dropping and preserve Logic BIST integration.

How to Debug and Fix

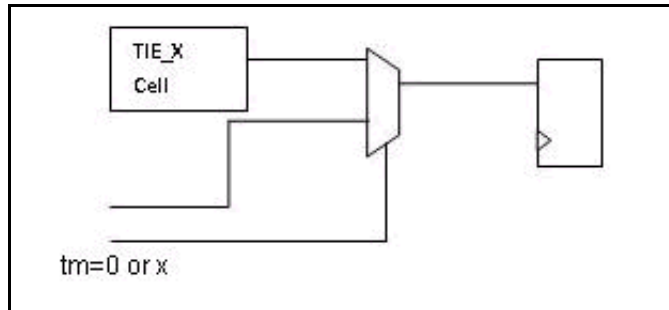
View the Incremental Schematic of the violation message. The Schematic Viewer window shows the path from output of TIE-X module to output port or sequential device.

To fix the violation, block this path.

Example Code and/or Schematic

Consider the following figure:

BIST Rules



The BIST_05 rule reports violation for the above figure because the TIE_X cell is not properly bypassed. To fix the violation, specify `tm=1`.

Default Severity Label

Warning

Rule Group

BIST

Reports and Related Files

[bist_ready_summary.rpt](#): This report lists the number of TIE_X cell outputs which are not bypassed.

Clock Rules

Overview

The Clock rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
Clock_01	Multiple clocks, flip-flops not driven by top-level clocks, and specified top-level clocks that are unused
Clock_02	Clocks that are used on both edges to trigger flip-flops in the design
Clock_03	Multiple clock domains in the design
Clock_04	Clock signals that are used as data signals to flip-flops
Clock_05	Flip-flops that are triggered by opposite clock edge but are not used as retiming flip-flops
Clock_06	Design unit instances that have more than one clocks connected
Clock_08	Generated clocks that are merged / ANDed
Clock_09	Flip-flops where the clock and data pins have common logic
Clock_10	Test clocks that drive more than one clock domains
Clock_11	All clock sources must be testclock controlled in shift mode
Clock_11_capture	All clock sources must be testclock controlled in capture mode
Clock_14	Flip-flops that are triggered at the negative edge of their clocks and are not driving primary outputs
Clock_16	Flip-flops that have data paths from flip-flops that are triggered on the opposite edge of the same clock
Clock_17	Capture clocks that are gated by flip-flops triggered by the same clock
Clock_18	Flip-flops that are triggered on the negative edge of their clocks
Clock_21	Clocks that are also setting/resetting flip-flops or latches
Clock_22	Clocks that are combinational driven by more than one test clock

Clock Rules

Rule	Description
<i>Clock_23</i>	Scan Flip-flops that are triggered on the negative edge of their clocks
<i>Clock_24</i>	Devices whose outputs are changed when a test clock is pulsed
<i>Clock_25</i>	Testclocks that drive more than the specified number of flip-flops
<i>Clock_26</i>	A testclock condition applied on the sensitized fan-out cone of another testclock condition
<i>Clock_27</i>	Clock through a negative ICG must not feed any positive edge triggered flip-flop.
<i>Clock_28</i>	Reports the presence of combinational re-convergence to flip-flop clock pin
<i>Clock_29</i>	Reports clock sources of memories/hard macros that are not testclock controlled in the shift/capture mode
<i>Clock_30</i>	Ensure that all clock pins in a memory / hard macro should get the same root level clock

Clock_01

Ensure that only one clock is specified and that clock is driven from a top-level pin

When to Use

Use this rule when methodology suggests using one clock only.

Description

The Clock_01 rule reports violation for the following:

- More than one top-level clock in the design
- Flip-flops that are not triggered by the specified top-level clock
- User-specified clock that does not reach any flip-flop

Single clock systems are preferred since they simplify scan insertion and ATPG.

NOTE: *Designs with PLL's or black box clock generators will control flip-flops that will not be controlled by any root-level system clock. They may be controlled by a testclock in testmode.*

Prerequisites

At least one system clock or at least one testclock must be specified for the Clock_01 to run.

Method

If at least one testclock is supplied, then simulate any available scan shift conditions and verify that one testclock clock controls every flip-flop.

Default Weight

10

Language

VHDL, Verilog

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation:* The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (optional): Use this constraint to declare the clock pins used as test clocks. Use this constraint's `-testclock` argument for the Clock_01 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Some flops not hit by clock '`<clk-name>`'

Potential Issues

The violation message appears, if your design has a clock signal, which is not reaching a flip-flop.

Consequences of Not Fixing

Not fixing the violation can make it difficult to control the clock and hence may reduce observability.

How to Debug and Fix

To know more about debugging the violation, click [How to Debug and Fix](#).

To fix the violation, ensure that each flip-flop is hit by a clock.

Message 2

[WARNING] Top Module has >1 top level clocks

Potential Issues

The violation message appears, if top module has multiple top-level clocks.

Consequences of Not Fixing

Multiple clock domains generally require more ATPG processing steps as well as more time in first piece debug.

How to Debug and Fix

To know more about debugging the violation, click [How to Debug and Fix](#).

To fix the violation, ensure that top module has only one top level clock.

Message 3

[WARNING] User specified top-level clock does not reach a single flop

Potential Issues

The violation appears, if the user specified clock does not reach at least a single flip-flop.

Consequences of Not Fixing

Not fixing the violation may result in reduced observability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the testclock pin and the flip-flops not driven by this testclock pin.

You can also view the violations for the Info_testmode (under capture condition) and Info_testclock rules along with the violation of the Clock_01 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_01 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Clock_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_01 rule.

To fix the violation, ensure that user specified clock reaches the specified flip-flops.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
always @(posedge clk1)
    q1 <= d1;

always @(posedge clk2)
    q2 <= d2;
```

The Clock_01 rule reports violation for the above example because more than one top-level clocks are used.

Example 2

Consider the following VHDL code:

```

Process (clk1)
  Begin
    If (clk1'event and clk1 = '1') then
      Q1<= d1;
    End if;
End process;

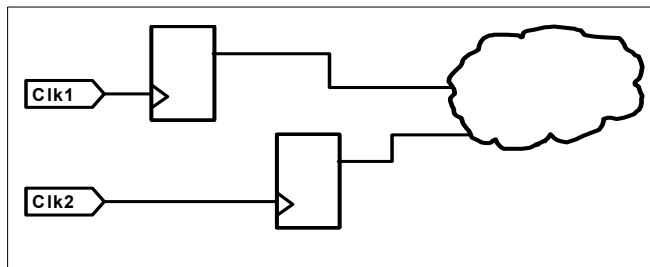
Process (clk2)
  Begin
    If (clk2'event and clk2 = '1') then
      Q2<= d2;
    End if;
End process;

```

The Clock_01 rule reports violation for the above example because more than one top-level clocks are used.

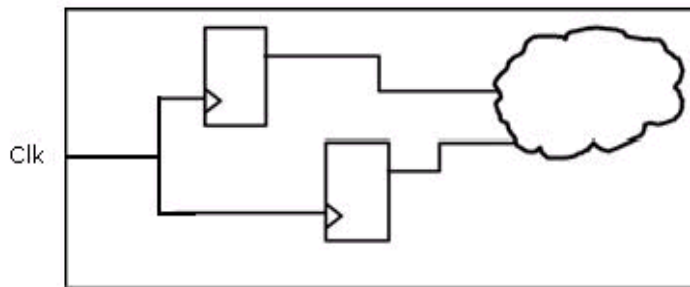
Example 3

Consider the following figure:



The Clock_01 rule reports violation for the above design because more than one top-level clocks are used.

To fix this violation, use only one top-level clock as shown in the following figure:

**Default Severity Label**

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_02

Ensure that both the edges of a clock are not switched on.

When to Use

Use this rule to avoid scan chain vector corruption using single edge and to keep the timing path unaffected.

Description

The Clock_02 rule reports violation for clocks that are used on both the edges to trigger flip-flops, or memory, or hard macros in the design.

Prerequisites

At least one system clock or at least one testclock must be specified for the Clock_02 rule to run.

Rule Exceptions

The Clock_02 rule ignores no scan flip-flops (forced or inferred), memory, and hard macros.

Method

If at least one testclock exists then simulate any available testmode scan shift conditions and report a message if both edges of any test clock are used.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_clock_end_point_types_for_Clock_02](#): Specifies the type of instances that are checked for the clock edge usage by the *Clock_02* rule. The type of instances can be either flip-flops, or all flip-flops, memories, and hard macros.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (mandatory): Use this constraint to declare the clock pins used as test clocks. Use this constraint's `-testclock` argument for the `Clock_02` rule.
- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
-
- *force_no_scan* (optional): Use this constraint to exclude black boxes, latches, and flip-flops, from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Both edges of test clock '`<clk-name>`' are used. Rising edge count is `<cnt1>` (`<dist1>`) and falling edge count is `<cnt2>` (`<dist2>`)

Arguments

- Name of the clock, (`<clk-name>`)
- Count of the rising edge for flip-flops, memories, or hard macros, `<cnt1>`
- Distribution of `<cnt1>` amongst flip-flops, hard macros, and memories, `<dist1>`
- Count of the falling edge for flip-flops, memories, or hard macros, `<cnt2>`
- Distribution of `<cnt1>` amongst flip-flops, hard macros, and memories, `<dist2>`

Potential Issues

The violation message appears if a clock uses both its edges to trigger flip-flops.

Consequences of Not Fixing

Not fixing this violation may result in scan chain vector corruption and affect the timing path.

Mixed-edge flip-flops that are not independently clocked, require special care to ensure that scanned-in setup data, necessary for a given test, is not corrupted during the capture phase for that test. If such corruption occurs, coverage may be reduced or good parts may be declared defective. Separating such devices into independently clocked chains prevents this possibility.

How to Debug and Fix

To debug, double-click the violation message. The Incremental Schematic window highlights the path from clock pin to flip-flops triggering on different edges.

You can also view the violations for the Info_testmode (under capture condition) and Info_testclock rules along with the violation of the Clock_02 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_02 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Clock_02 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_02 rule.

To fix the violation, trace either positive edge or negative edge and block the same.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
...
always @ (posedge clk)
    b <= a;

always @ (negedge clk)
    c <= b;
...
```

The above example demonstrates a bad design scenario because the clock, `clk`, is using both its edges in the design.

Example 2

Consider the following Verilog code:

```
...
always @ (posedge clk) begin
    b <= a;
    c <= b;
end
...
```

The above example demonstrates a good design scenario because the clock, `clk`, is using only its positive edge in the design.

Example 3

Consider the following VHDL code:

```
...
Process (clk)
Begin
    If (clk'event and clk = '1') then
        B<= a;
    End if;
End process;

Process (clk)
Begin
    If (clk'event and clk = '0') then
        C<= b;
    End if;
End process;
...
```

The above example demonstrates a bad design scenario because the clock, `clk`, is using both its edges in the design.

Example 4

Consider the following VHDL code:

```
...
```

```

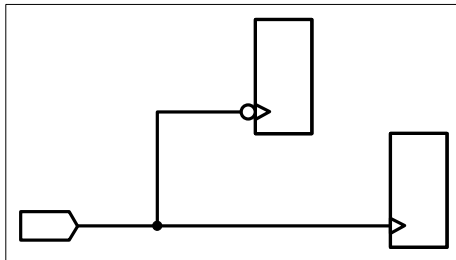
Process (clk)
  Begin
    If (clk'event and clk = '1') then
      B<= a;
      C<=b;
    End if;
  End process;
  ...

```

The above example demonstrates a good design scenario because the clock, `clk`, is using only its positive edge in the design.

Example 5

Consider the following figure: Consider the following figure:



In the above figure, the clock uses both the edges to trigger the two flip-flops.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

Clock_02_violation_<violationNumber>.csv: Lists the flip-flops along with the respective clock-edge.

Clock_03

Reports multiple clock domains.

When to Use

Use this rule when only one clock domain is required. Also, you can use this rule to avoid faults generated by crossing clock domains.

Description

The Clock_03 rule reports violation for multiple clock domains in the design.

NOTE: *If the Clock_03 rule fails, then the Clock_01 rule will also fail but the reverse is not true.*

Method

If no testmode data is supplied and at least one system clock is defined, then report a message if more than one system clock domain is found.

If testmode and at least one testclock is defined, then report a message if more than one test clock domain is found.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode. For this rule, use this constraint's `-scanshift` argument.
- [clock](#) (optional): Use this constraint to define clocks in a design. For this rule, use this constraint's `-testclock` argument.

- *balanced_clock* (optional): Use this constraint to specify points within the clock distribution system where all clocks fed from that point are to be considered in the same clock domain.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Multiple clock domains detected in design. First: '`<clk1-name>`', current: '`<clk2-name>`'

Arguments

- Name of the first clock, `<clk1-name>`
- Name of the current clock, `<clk2-name>`

Potential Issues

The violation message appears if multiple clock domains are detected in the design.

Consequences of Not Fixing

Not fixing this violation may result in faults generated by crossing clock domains.

Single clock domain systems are especially amenable for scan insertion and ATPG whereas multiple clock domain designs complicate scan test generation and require special care in skew management and may also complicate fault capture. ATPG tools create tests with results captured in one domain but the fault simulator may attempt detection of other faults in domains without skew management with respect to the domain used by the ATPG tool. This will result in optimistic coverage calculations and unreliable action on ATE. The user may prevent these problems by restricting the capture domains but this increases the operating effort and may result in longer test sequences since compaction across domains cannot be allowed.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path from first clock source to flip-flop and the path from

second clock source to flip-flop.

To fix the violation, block the unwanted clock domains.

You can also view the violations for the Info_testmode and Info_testclock rules along with the violation of the Clock_03 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_03 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Clock_03 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_03 rule.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
...
C1 = F(clk, a, b,...)
C2 = G(clk, s, t, ...)

always @ (posedge C1)
    sig1 <= dat1;

always @ (posedge C2)
    sig2 <= dat2;
...

```

The above example shows a bad design scenario because the design uses multiple clock domains.

Example 2

Consider the following Verilog code:

```
...
always @ (posedge clk) begin
    sig1 <= dat1;
    sig2 <= dat2;
end
...

```

The above example shows a good design scenario because the design uses

a single clock domain.

Example 3

Consider the following VHDL code:

```
...
Signal C1,C2 : std_logic;
C1 <= F(clk, a, b, ...);
C2 <= G(clk, s, t, ...);

Process (C1)
  Begin
    If (C1'event and C1 = '1') then
      Sig1<= dat1;
    End if;
End process;

Process (C2)
  Begin
    If (C2'event and C2 = '1') then
      Sig2<= dat2;
    End if;
End process;
...
```

The above example shows a bad design scenario because the design uses multiple clock domains.

Example 4

Consider the following VHDL code:

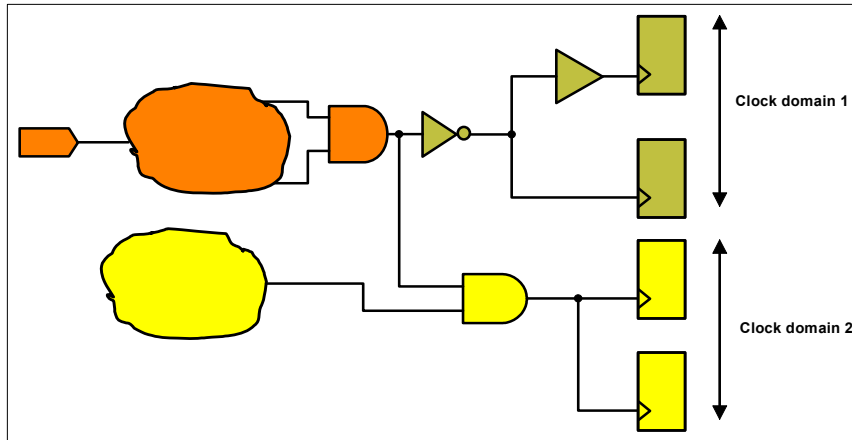
```
...
Process (clk)
  Begin
    If (clk'event and clk = '1') then
      Sig2<= dat2;
    End if;
End process;
...
```

The above example shows a good design scenario because the design uses

a single clock domain.

Example 5

Consider the following figure:



The above figure illustrates a bad design scenario because multiple clock domains are used in the design.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_04

Do not use clock signals as data signals

When to Use

Use this rule to identify the clock signals that have at least one unblocked path to a data pin of a flip-flop or clock gating cell.

Description

The Clock_04 rule reports violation for clock signals that have at least one unblocked path to a data (set/reset or enable) pin of a flip-flop or enable pin of clock gating cell.

The rule reports root level clock pin, the violating flip-flop and the path from that pin to that flip-flop data pin.

The clock path to each flip-flop is a chain of serially connected nodes that a clock edge traverses from a user specified clock pin to a flip-flop clock pin. Any other inputs to multi-input elements on such a clock path are called clock enables. Enable signals are allowed to fan-out to data pins but clock path signals are not.

Default Weight

10

Language

Verilog, VHDL

Method

If at least one testclock is defined, then simulate power, ground and testmode for capture, if any. Walk the combinational unblocked fan-out cone for each test clock. Stop the walk at nodes with non-x simulation values. If any flip-flop data pin or CGC enable pin is reached, then report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *schematicForAllBits*: The default value is off. Set the value of the parameter to on to generate highlight information for all bits of the rule-violating bus.

- *mergeN*: The default value is -1. Set the value of the parameter to any positive integer number to set a limit on the number of bit-slices that should be merged and hence reported together (assuming that they are inferred at the same line and file).
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Clock_04 rule:

[WARNING] The clock signal '<clk-name>' is reaching to data pin of '<num_of_inst>' flip-flop(s) in capture mode.

Arguments

- Name of the clock, <clk-name>
- Number of flip-flop instances reported, <num_of_inst>

Potential Issues

A violation is reported due to wrong RTL.

Consequences of Not Fixing

Fan-out from one signal source to both the clock and data pins of the same register can reduce fault coverage since ATPG tools assume that signals required for a test can be realized whereas clock signals may be under rigorous constraints. In such cases, fault coverage may be compromised.

How to Debug and Fix

Select the violation message and click the Incremental Schematic button to view the Incremental Schematic for the violation message. The Incremental Schematic displays the path from the clock to the data pin of the violating flip-flop or enable pin of a CGC.

Double-click the violation message to display the related spreadsheet report. Clicking on the spreadsheet entries displays the Incremental Schematic. The displayed Incremental Schematic shows the complete path from the clock to the data pin of the data pin of flip-flop via CGCs, if clock receives the CGC enable pin.

You can also view the violations for the Info_testmode (under capture condition) and Info_testclock rules along with the violation of the Clock_04 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_04 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Clock_04 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_04 rule.

Overlay (Auxiliary violation mode) the Info_testmode rule under capture mode. This helps in identifying how the path from the testclock to the data pin of the flip-flop is sensitized.

To fix the violation, break the data path.

Example Code and/or Schematic

Example 1

Consider the following Verilog example:

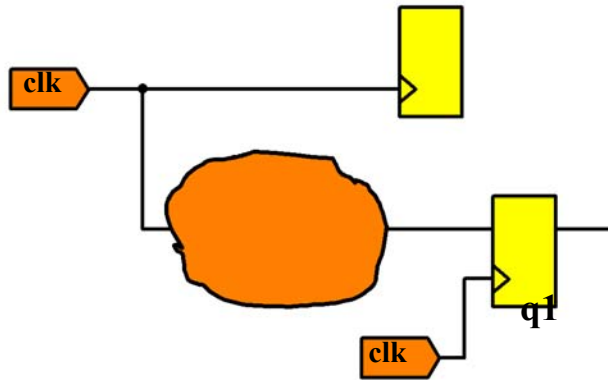
```
...
always @(posedge clk)
    q1 <= f(clk,...);
...
```

The above Verilog code demonstrates a bad design scenario.

SGDC command for Clock_04

```
clock -testclock -name clk
```

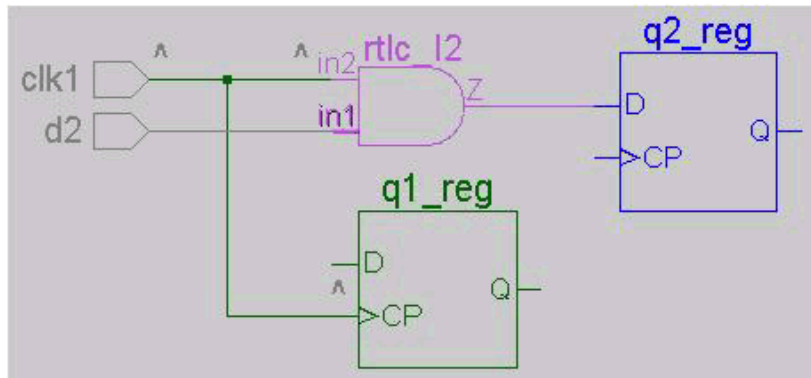
The above command is an input for the specific testcase discussed in following figure:





The above figure captures the Clock_04 scenario.

Example 2

Consider the following example:



The *Clock_04* rule reports the following violation message for the above schematic:

 **Clock_04[1]: Do not use clock signals as data signals.**
 The clock signal 'clk1' is reaching to data pin of '1' flip-flop(s) in 'capture' mode, test.v, 1

However, the *Clock_04* rule does not report a violation, if the following constraint definition is specified:

```
test_mode -name d2 -value 0 -capture
```

Example 3

Consider the following figure which illustrates a scenario when clocks are also used as data:

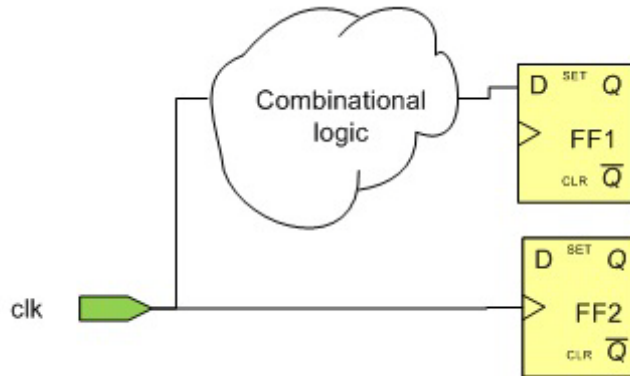


FIGURE 43. Clock as Data

For the above scenario, the ATPG tool may require a value on clk that may conflict with the sequence for transitioning from scan-in to capture. This may disrupt the coverage. Therefore, it is recommended to never use the clock as data.

Example 4

Consider the following design in [Figure 44](#) where the clock signal, clk, is reaching to data pin of flip-flops, qb1_reg, qb2_reg, qb3_reg, and qb4_reg:

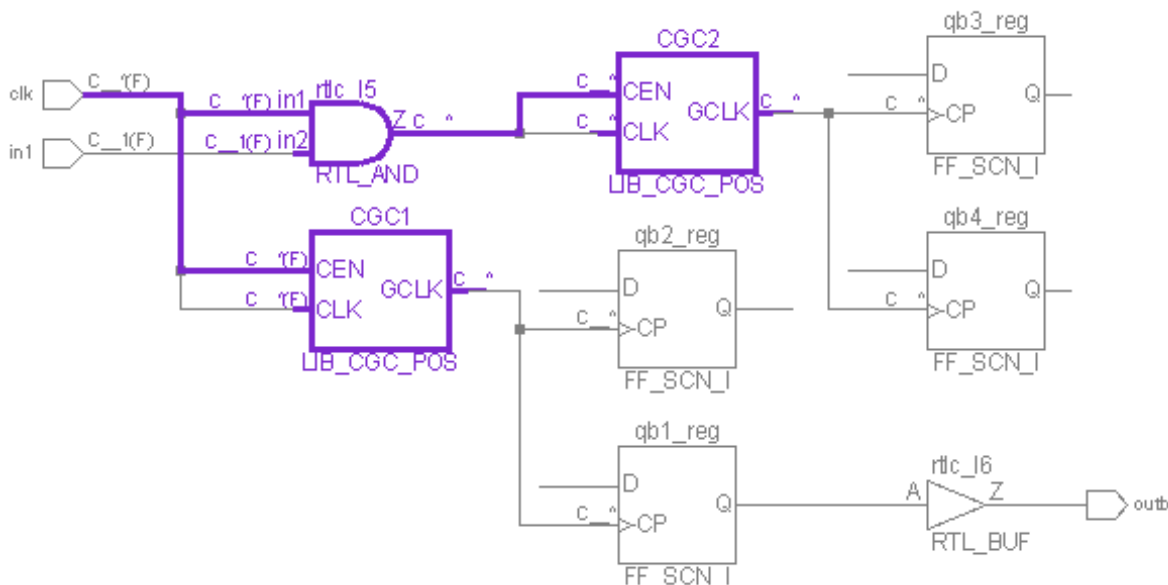


FIGURE 44. Clock Signal acting as a data pin of flip-flop

The Clock_04 rule reports the following violation message for the above design:

The clock signal 'ClockAsData.clk' is reaching to data pin of '4' flip-flop(s) in 'capture' mode, test.v, 6

Double-clicking the violation message displays the spreadsheet in the spreadsheet viewer as shown in [Figure 45](#):

Clock Rules

	B "Clock_Name"	C "Flop_Name"	D "EN-CGC name"	E "Mode_Name"
1	ClocksAsData.clk	ClocksAsData.qb1_reg	ClocksAsData.CGC 1	capture
2	ClocksAsData.clk	ClocksAsData.qb2_reg	ClocksAsData.CGC 1	capture
3	ClocksAsData.clk	ClocksAsData.qb3_reg	ClocksAsData.CGC 2	capture
4	ClocksAsData.clk	ClocksAsData.qb4_reg	ClocksAsData.CGC 2	capture

FIGURE 45. Spreadsheet report for the Clock_04 rule

Double-click on a row in the spreadsheet to display the complete path from the clock pin to the affected flip-flop in an Incremental Schematic as shown in [Figure 46](#):

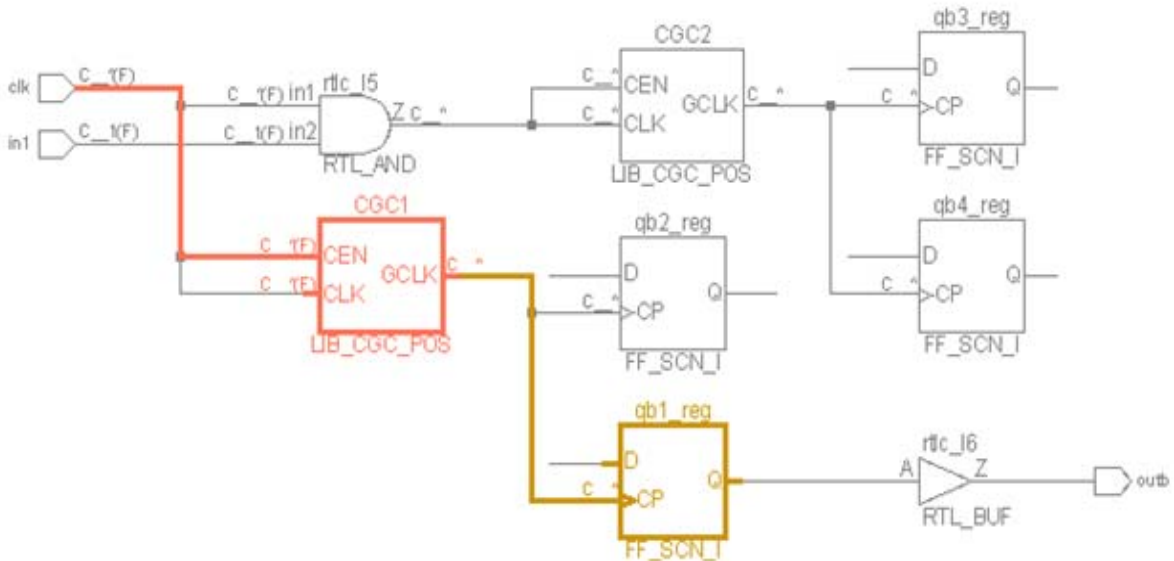


FIGURE 46. Violating Path

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_05

Ensure that all opposite edge flip-flops are used as retiming flip-flops

When to Use

Use this rule to use the same type of edge to trigger flip-flops, avoid scan chain corruption, and keep the timing-path unaffected.

Description

The Clock_05 rule reports violation for flip-flops that are triggered by opposite clock edge but are not used as retiming flip-flops.

Automatic test generation tools and testbench tools often rely on having all flip-flops change state on the same clock edge. This especially simplifies capturing test results in full scan designs since, in cases where logic paths cross clock domains, one edge may change the circuit state that captures in the opposite edge depend on.

Prerequisite

At least one system clock or at least one testclock must be specified for the Clock_05 rule to run.

Method

The rule is checked in two stages. The first stage determines which edge is the opposite edge for each specified clock by comparing the number of flip-flops that depend on each edge. The edge with the smaller number is assumed to be opposite. The second stage visits each flip-flop. If testmode information does not exist then any flip-flop marked as opposite phase with respect to a system clocks must be a retiming flip-flop.

If testmode capture information and at least one testclock are supplied then all system clock information is ignored and any flip-flop marked as opposite phase with respect to a test clock must be a retiming flip-flop.

A retiming flip-flop must not have opposite edge flip-flops in the fan-in cone that feeds its data pin or in the fan-out cone from its q-pin.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define clocks in a design. For this rule, use this constraint's `-testclock` argument.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Opposite edge flip-flop(s) '`<flip-flop-name>`', triggered by '`<clk-name>`', is(are) not retiming

Arguments

- Output netname of rule-violating flip-flop instance, `<flip-flop-name>`
- Top-level clock port that is negedge triggered, `<clk-name>`

Potential Issues

The violation message appears if an opposite edge flip-flop is triggered by a clock but not used as a retiming flip-flop.

Consequences of Not Fixing

Not fixing this violation may result in scan chain vector corruption and affect the Timing path.

How to Debug and Fix

To debug, view the Incremental Schematic of the violation message and trace to the flip-flop triggered by opposite edge.

To fix the violation, block the path.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
...
always @(posedge clk)
    q1 <= d;

always @(clk or f(q1))
    q <= d;
...
```

The above example shows a bad design scenario.

Example 2

Consider the following Verilog code:

```
...
always @(posedge clk)
    q1 <= d;

always @(!clk or f(q1))
    q <= d;
...
```

The above example shows a good design scenario.

Example 3

Consider the following VHDL code:

```
...
Process (clk)
  Begin
    If (clk'event and clk = '1') then
      q1<= d;
    End if;
  End process;

D_int <= f(q1);

Process (clk, d_int)
```

```
    Begin
      q<= d;
    End process;
    ...
```

The above example shows a bad design scenario.

Example 4

Consider the following VHDL code:

```
    ...
  Process (clk)
    Begin
      If (clk'event and clk = '1') then
        q1<= d;
      End if;
    End process;
```

```
D_int <= f(q1);
```

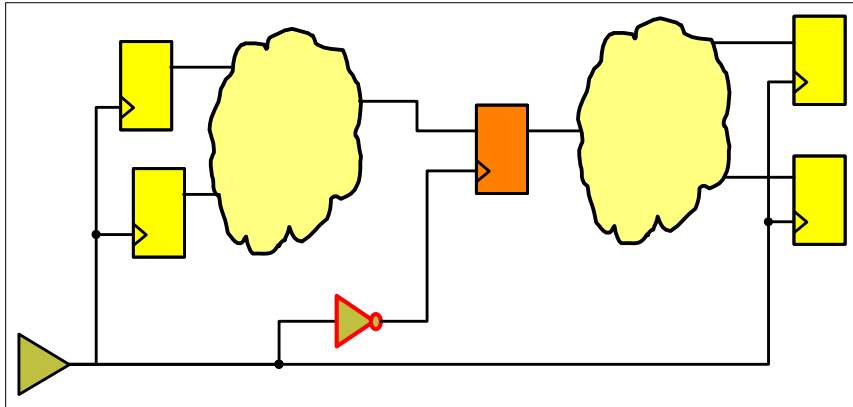
```
  Process ((not clk), d_int)
    Begin
      q<= d;
    End process;
    ...
```

The above example shows a good design scenario.

Example 5

Consider the following example:

Clock Rules



The Clock_05 rule reports violation for the above design because flip-flops in yellow are triggered by positive edge whereas the flip-flop in orange is triggered by negative edge of the flip-flop.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_06

Ensure that only one clock port is available for each block

When to Use

Use this rule to use single clock in each instance, avoid scan chain corruption, and keep the timing-path unaffected.

Description

The Clock_06 rule flags design unit instances that have more than one clocks connected.

Use the [Info_testclock](#) rule to display how testclock signals are distributed in the design.

Method

This rule is checked by walking the fan-out cones from every root-level system clock pin to determine the number of times any hierarchical block is reached. If any block is reached more than once, then a message is flagged.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[clock](#) (optional): Use this constraint to define clocks in a design. For this rule, use this constraint's `-testclock` argument.

Operating Mode

[Power Ground](#)

Messages and Suggested Fix

[WARNING] Block '<inst-name>' is hit by more than one clocks '<clk-name-list>'

Arguments

- Name of the module instance that violates this rule, <inst-name>
- List of the clocks feeding the module instance, <clk-name-list>

Potential Issues

The violation message appears if a block is connected to more than one clock.

Consequences of Not Fixing

BIST and scan operations that operate at a block or module level may require that only one clock be involved for each block to simplify scan chain routing and operation.

Therefore, not fixing this violation may result in scan chain vector corruption and affect the timing path.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the design unit that has more than one clock port and the clock ports themselves.

To fix the violation, trace back from the instance and block the unwanted clock paths.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module multiclock (clk1, clk2,...)
...
  always @(posedge clk1)
    q1 <= d1

  always @(posedge clk2)
    q2 <= d2
endmodule
```

The above example shows a bad design scenario because multiple clocks are connected in one design instance.

Example 2

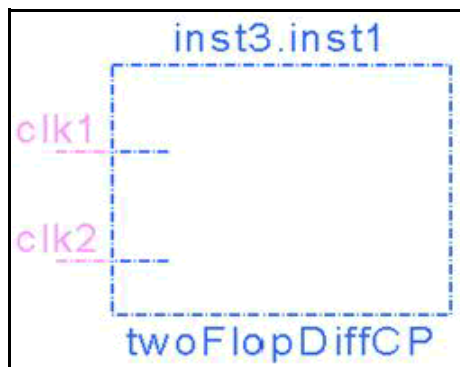
Consider the following VHDL code:

```
architecture arch_mclk of multiclock is
begin
  Process (clk1)
  Begin
    If (clk1'event and clk1 = '1') then
      Q1<= d1;
    End if;
  End process;
  Process (clk2)
  Begin
    If (clk2'event and clk2 = '1') then
      Q2<= d2;
    End if;
  End process;
End arch_mclk;
```

The above example shows a bad design scenario because multiple clocks are connected in one design instance.

Example 3

Consider the following example:



The Clock_06 rule reports violation for the above design because two

clocks, `clk1` and `clk2`, are connected to one instance.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_08

Ensure that merged testclocks are not present in the design

When to Use

Use this rule to avoid internal clock generation with unpredictable frequency and width.

Description

The Clock_08 rule flags test clocks that are ANDed or merged.

Method

If at least one testclock exists, then simulate any available testmode for capture conditions. Walk the unblocked combinational fan-out cones from all testclocks. Stop the walk at nodes with non-x simulation values and at clock sources. If any two testclocks fan-out to the same clock source, report a message.

Default Weight

10x

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftAllowMergeOfSameClock*: The default value of the parameter is `off`. Set the value of the parameter to `on` to enable the *Clock_08* rule to report for clock merging only if the merged node is coming from the different user-specified test clocks and not the same test clock.
- *dft_allow_clock_merge_at_mux_via_data*: The default value is `off`. Set the value of the parameter to `on` to enable the *Clock_08* and *Atspeed_22* rules to ignore the violation if a clock is merging at a mux through its data-pins.
- *dft_check_clock_merge_in_shift_as_well*: The default value of the parameter is `off`. Set the value of the parameter to `on` to enable the *Clock_08* rule to report for clock merging in both the shift mode and capture mode.

- *dftReportIfUsedAsClock*: The default value of the parameter is on. Set the value of the parameter to off to enable the *Clock_08* rule to report violation for clock merging even if the merged node is not used as clock in the design, that is, driving the CP pin of a flip-flop or the EN pin of a latch.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define clocks in a design. For this rule, use this constraint's `-testclock` argument.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] <no_of_testclk> testclocks <clk-name> are merging at net '<node-name>' in mode <mode_name>

Arguments

- Name of the gated clock output net or instance name in case of lib cell, <node-name>
- List of merging clocks, <clk-name>
- Number of test clocks, <no_of_testclk>

Potential Issues

A violation message appears if your design has ANDed or merged test clocks.

Consequences of Not Fixing

The Clock ANDing can potentially reduce coverage because clocks are

pulsed when required for either scan operations or especially for capture. Usually, when a pulse is not required, the clock source is maintained at its off state, which is 0 for a rtz clock and 1 for a rto clock. If rtz clocks are ANDed and one clock is pulsed, the AND output is blocked because the other clock will remain at 0, which maintains the AND output at zero.

Clock ANDing can also cause test issues because of clock pulse width reduction. The problem occurs when the resulting pulse width is too small to cause downstream flip-flops to react properly. As a result, test performance may pass faulty chips or fail working chips.

Not fixing this violation may result in internal clock generation with unpredictable frequency and width

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the paths of the merging test clocks.

You can also view the violations for the Info_testmode (under capture condition) and Info_path rules along with the violation of the Clock_08 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_08 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Clock_08 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_08 rule.

To fix the violation, block the merging of clocks.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module AndedClocks (clk, e1, e2, din, dout);
    input clk, e1, e2, din;
    output dout;
    reg dout;
    wire clk1, clk2, clk12;

    assign clk1 = clk & e1;
    assign clk2 = clk & e2;
    assign clk12 = clk1 & clk2;
```

```

    always @ (posedge clk12)
        dout <= din;
endmodule

```

The above example shows a bad design scenario because the design has merged testclocks, clk1 and clk2.

Example 2

Consider the following VHDL code:

```

Architecture arch_AndedClocks of AndedClocks is
    Signal clk1, clk2, clk12 : std_logic;
Begin
    Clk1 <= clk and e1;
    Clk2 <= clk and e2;
    Clk12 <= clk1 and clk2;

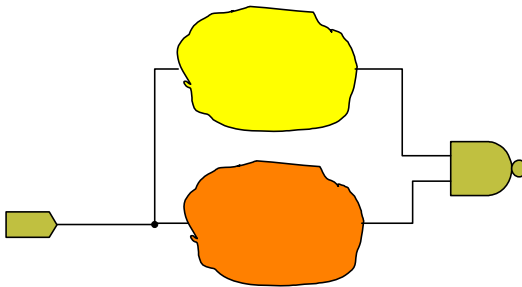
    Process (clk12)
    Begin
        If (clk12'event and clk12 = '1') then
            dout<= din;
        End if;
    End process;
End arch_AndedClocks;

```

The above example shows a bad design scenario because the design has merged testclocks, clk1 and clk2.

Example 3

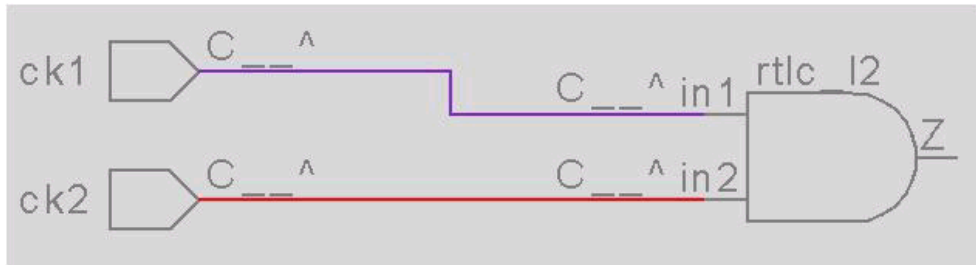
Consider the following figure:



The Clock_08 rule reports violation in the above case because merged testclocks are present in the design.

Example 4

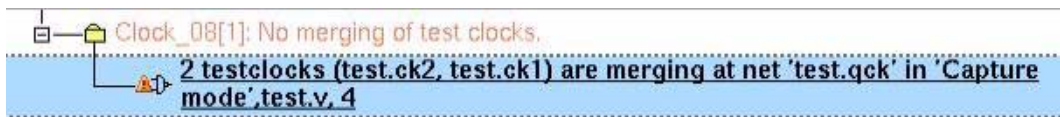
Consider the following schematic:



Also, consider the following schematic definition for the above schematic:

```
clock -name ck1 -testclock
clock -name ck2 -testclock
```

The *Clock_08* rule reports the following violation message for the above design:



Example 5

Consider the following figure, which illustrates the merging of two clocks:

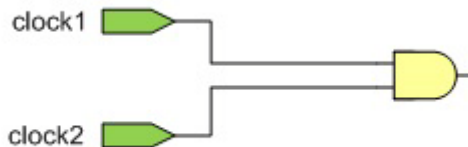


FIGURE 47. Clock ANDing

The Clock ANDing can potentially reduce coverage because clocks are pulsed when required for either scan operations or especially for capture.

Clock ANDing can also cause test issues because of clock pulse width reduction as shown in the following figure:

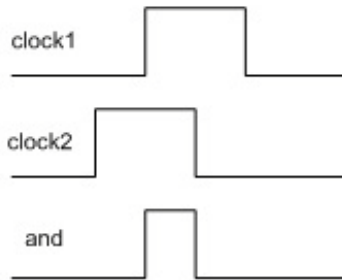


FIGURE 48. Result of ANDing of two clocks

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_09

Ensure that the clock and data pins do not have common logic

When to Use

Use this rule to identify flip-flops where the clock and data pins have common logic.

Description

The Clock_09 rule reports violation for flip-flops where the clock and data pins have common logic.

The rule reports signal and paths from this signal to the clock and data port of the same flip-flop.

While other rules check for clock as data and data as clock, the Clock_09 rule is concerned with data as clock or clock as data to the same flip-flop.

Rule Exceptions

Overlap from the D-pin of one flip-flop and the clock pin of some other flip-flop is allowed by this rule.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode capture conditions. For each flip-flop, walk the unblocked combinational clock pin fan-in cone and the data pin fan-in cone back to root level pins. There should not be any signals in common.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.

Messages and Suggested Fix

The following violation message is displayed for the Clock_09 rule:

[WARNING] Flip-flop '`<flip-flop-name>`' has logic in common with data and clock

Arguments

Name of the flip-flop, `<flip-flop-name>`

Potential Issues

A violation is reported when either *test_mode* to block the path is not present or design connectivity is not correct.

Consequences of Not Fixing

Fan-out from one signal to both clock and data of a register can lead to race conditions, confusing fault simulation and therefore the ATPG algorithm. Other rules check for clock as data and data as clock.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic displays the path from the data pin of the violating flip-flop up to the point that is driving its clock pin. Probe the fan-in cone of the data pin of the flip-flop until the path converges to the highlighted point.

You can also view the violations for the *Info_testmode* (under capture condition) and *Info_path* rules along with the violation of the *Clock_09* rule in the Incremental Schematic window. To do this, double-click the violation for the *Clock_09* rule and open the Incremental Schematic window.

The violation messages for the *Info_testmode* and *Info_path* rules overlap the violation message for the *Clock_09* rule in the Incremental Schematic

window. This is useful in debugging the violation for the Clock_09 rule.

To fix the violation, either use test_mode to block the path or design may have to be modified

Example Code and/or Schematic

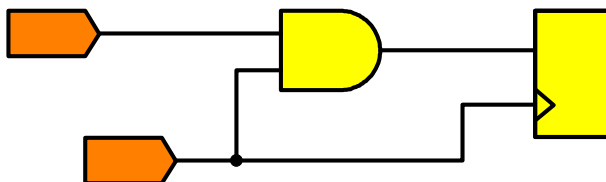
Verilog

Consider the following example:

```
...  
always @(posedge f(clk,data))  
    q1 <= data;  
...
```

The above Verilog code demonstrates a bad design scenario.

Consider the following figure:



The above figure illustrates a good design scenario.

Default Severity Label

Warning

Rule Group

Clock Rules

Reports and Related Files

No related reports or files.

Clock_10

Ensure that each clock domain has a dedicated test clock

When to Use

Use this rule to ensure that each functional clock domain has a dedicated test clock. You can also use this rule to avoid faults generated by crossing different clock domains.

Description

The Clock_10 rule reports test clocks that drive more than one clock domains.

Clock skew management with respect to the system clocks should ensure proper system operation. If one test clock is dedicated to each system clock domain, then test clocking problems and the need for additional skew management with respect to test clocks is reduced or eliminated.

Prerequisite

The Clock_10 rule ignores those clock sources that only trigger flip-flops marked as 'no scan' flip-flops (forced and inferred).

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftCPMechanism*: The default value is sequential. Set the value of the parameter to parallel to specify the clocking methodology (all clocks simulated parallel or clocks simulated one-by-one separately) to be used by the Clock_10, Clock_11, and Info_testclock rules.
- *dftIgnoreConstantOrUnusedFlipFlops*: The default value is off. Set the value of the parameter to on to ignore the violations reported by the Clock_10 rule when any of the following conditions is true:
 - A constant value is specified to the D pin of a flip-flop.
 - A constant value is specified to the Q pin of a flip-flop.
 - The value at the Q pin of a flip-flop is not captured at a scan point.
- *dft_ignore_constant_supply_flip_flops*: The default value is off. Set the value of the parameter to on to ignore Synthesis Redundant (SR) flip-flops from rule checking.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Clock_10 rule uses this constraint's `-testclock` argument.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *balanced_clock (optional)*: Use this constraint to specify points within the clock distribution system where all clocks fed from that point are considered in the same clock domain.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Clock source '<clk-src>' of functional clock domain '<func-domain-name>' does not get a testclock in 'Capture mode'

Arguments

- Clock source, which is the first element found in the fan-in cone of a flip-flop clock input pin that has more than one input, <clk-src>
- Functional clock domain name, <func-domain-name>

Potential Issues

The violation message appears if the test clock does not reach clock source in the capture mode.

Consequences of Not Fixing

Not fixing this violation may result in reduced control over the design.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window shows the clock source not getting clock.

To fix the violation modify constraints and/or design so that the testclock reaches the clock source.

Message 2

[WARNING] Clock sources of functional clock domain '<func-domain-name>' are part of different clock domains in Capture mode (<list>)

Arguments

- Functional clock domain name, <func-domain-name>
- List of capture clock domain name and clock source in the following format:
(capture-domain-name:clk-src, ...)
- Capture clock domain name, <capture-domain-name>
- Clock source, which is the first element found in the fan-in cone of a flip-flop clock input pin that has more than one input, <clk-src>

Potential Issues

The violation appears, if clock sources, which are part of the same clock domain in the functional mode are a part of different clock domains in the capture mode.

Consequences of Not Fixing

Not fixing the violation may result in faults generated by crossing different clock domains.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window shows path from sample flops of different capture domains to their test clock.

To fix the violation, modify constraints and/or design so that flops which were in the same clock domain in the functional mode are also in the same clock domain in the capture mode.

Message 3

Clock sources of different functional clock domains (<list>)

are part of same clock domain <capture-domain-name> in Capture mode

Arguments

- List of functional clock domain name and clock source in the following format:
(func-domain-name:clk-src, ...)
- Functional clock domain name, <func-domain-name>
- Clock source, which is the first element found in the fan-in cone of a flip-flop clock input pin that has more than one input, <clk-src>
- Capture clock domain name, <capture-domain-name>

Potential Issues

The violation appears, if clock sources part of different clock domains in functional mode are part of same clock domain in capture mode.

Consequences of Not Fixing

Not fixing the violation may result in faults generated by crossing different clock domains.

How to Debug and Fix

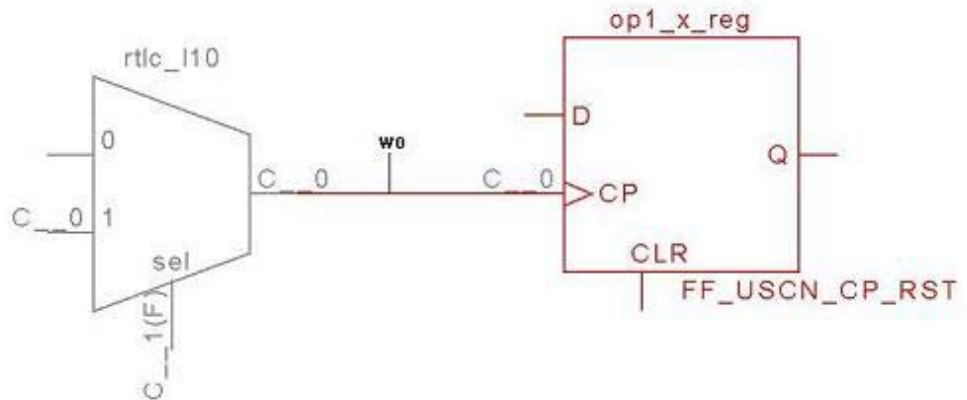
Double-click the violation message. The Incremental Schematic window shows path from sample flops of different functional domains to their testclock.

To fix the violation, modify constraints and/or design so that flip-flops which were in different clock domain in the functional mode are not in same clock domain in the capture mode.

Example Code and/or Schematic Code

Example 1

Consider the following figure illustrating clock sources that are in functional clock domain but do not get a test clock in the capture mode:

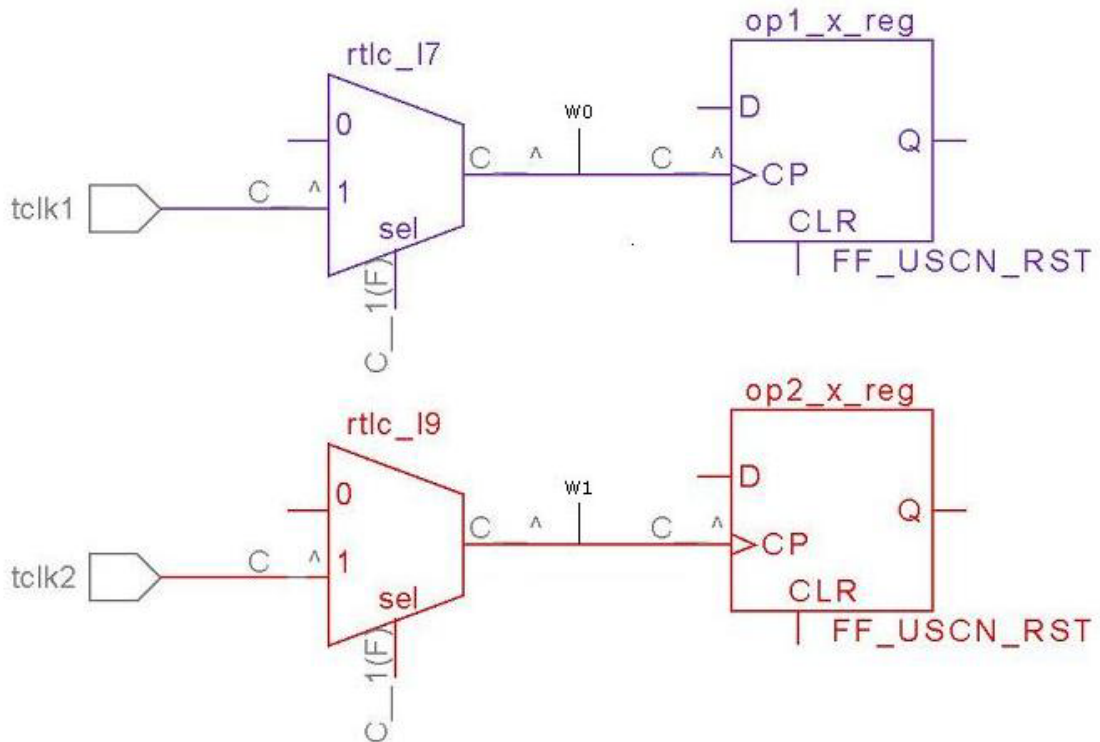


For the above case, the Clock_10 rule reports the following violation message:

Clock source 'test.w0' of functional clock domain 'test.clk' does not get a testclock in 'Capture mode'

Example 2

Consider the following figure illustrating the clock sources that are part of the functional clock domain but are in different clock domains in the capture mode:

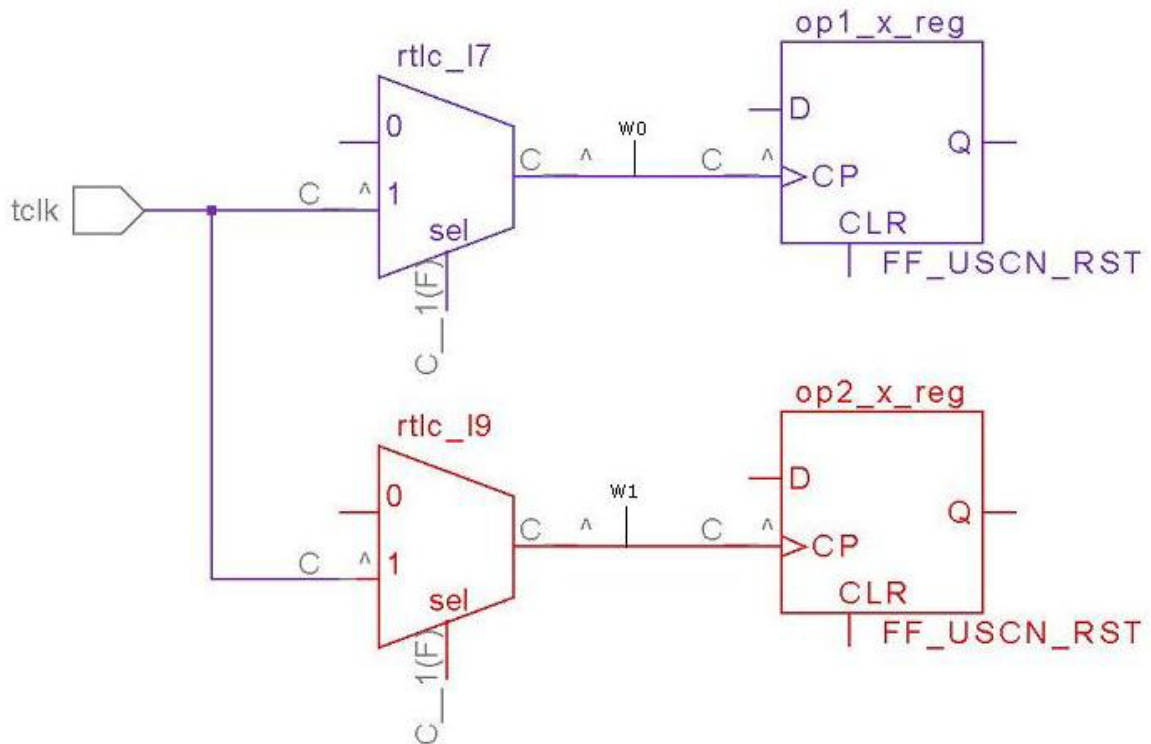


For the above case, the Clock_10 rule reports the following violation message:

Clock sources of functional clock domain ' test.clk' are part of different clock domains in Capture mode (test.tclk2: test.w1, test.tclk1: test.w0)

Example 3

Consider the following figure illustrating the clock sources, that are part of different functional clock domain but are part of same clock domain in the capture mode:

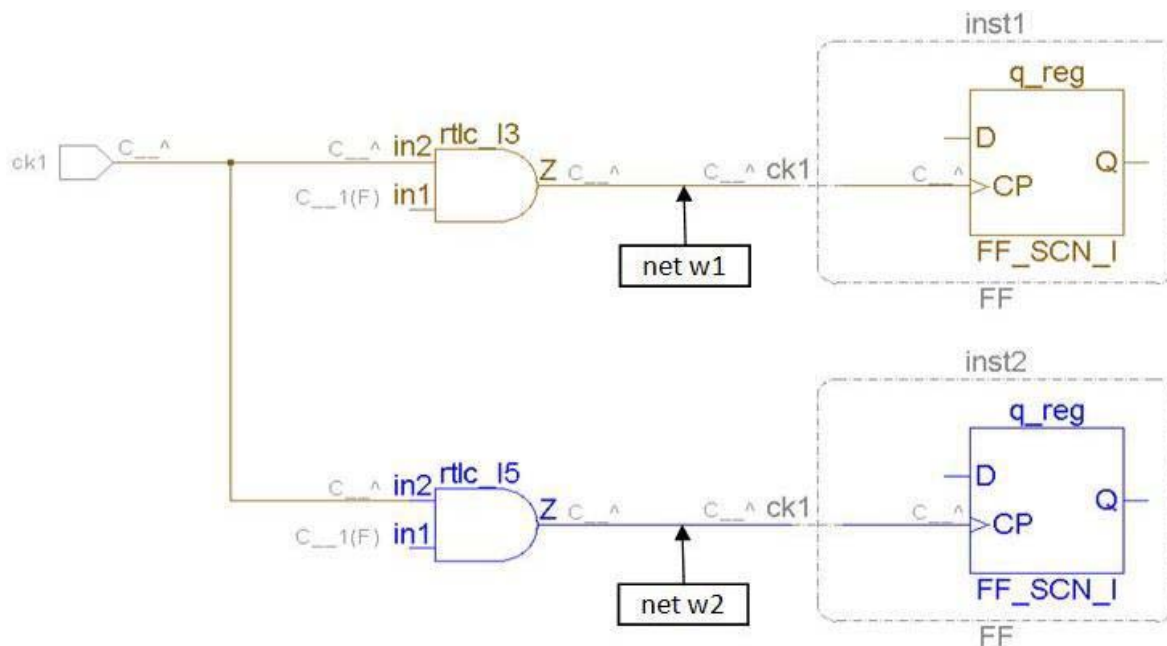


For the above case, the Clock_10 rule reports the following violation message:

Clock sources of different functional clock domains '(test.clk2: test.w1, test.clk1: test.w0)' are part of same clock domain 'test.tclk' in Capture mode

Example 4

Consider the following figure:



In the above design, no clock was reaching flip-flops in the functional mode. Therefore, the flip-flops, `inst1.q_reg` and `inst2.q_reg` are inferred to be in two different functional domains. However, in the capture mode these flip-flops are in same domain. Therefore, the Clock_10 rule reports the following message violation in this case:

```
Clock sources of different functional clock domains '(
inferred_domain_w2_1: test.w2, inferred_domain_w1_2: test.w1 )'
are part of same clock domain 'ck1' in Capture mode.
```

To make flip-flops part of the same functional domain, they should get the clock from same domain in the functional mode. Alternatively, you can use the [balanced_clock](#) constraint as shown below:

```
balanced_clock -name ck1
```

Default Severity Label

Warning

Clock Rules

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_11

Ensure that all clock sources are testclock controlled in shift mode.

When to Use

Use this rule to detect internally generated clocks that are not testclock controlled in the shift mode.

Description

The Clock_11 rule reports violation for internally generated clocks that are not testclock-controlled in the shift mode.

Mux-scan design requires that scan flip-flops be test clock controlled during scan shift. The Clock_11 rule verifies that each clock source (see glossary for definition) receives a pulse from a test clock pin in scan shift.

Rule Exceptions

When the *Clock_11* rule is run in conjunction with the *Scan_08* rule, a tag can be added in the violation messages by setting the value of the [dftTagAsync07Clock11Scan08](#) rule parameter to on. This enables the correlation of the cause of unscannability.

Also, the flip-flops marked as noscan flip-flops using the [no_scan](#) constraint or inferred as noscan flip-flops are not counted. In case, all flip-flops triggered by an internally generated clock are marked as noscan flip-flops, the clock is not considered as a rule-violating clock and no violation message is reported.

Default Weight

10

Language

Verilog, VHDL

Method

Find all clock sources.

If testmode information is not supplied, report any clock source that is not a root level pin. If testmode information is available, simulate testmode for scan shift. Then simultaneously simulate a pulse on every test clock. Verify that a pulse appears on clock source. If any flip-flop does not get a pulse, then report a message. For each domain that does not get a test clock pulse, walk the path from the test clock that should drive this clock source. This walk stops at the element that has a test clock

pulse in its input but does not have a pulse on its output. This element is called the blocking gate.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftAutoFix*: The default value of the parameter is as follows:


```
{+RULES[nothing]+TMPOR[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
"{+RULES[<ruleelist>]+TMPOR[tmport]+TCLKPORT[tclkport]}"
```
- *dftCPMechanism*: The default value is sequential. Set the value of the parameter to parallel to specify the clocking methodology used by the respective rules.
- *dftTagAsync07Clock11Scan08*: The default value is off. Set the value of the parameter to on to add a unique tag for every asynchronous source (set, reset, or clock), which is visible in the rule violation.
- *dft_ignore_constant_supply_flip_flops*: The default value is off. Set the value of the parameter to on to ignore Synthesis Redundant (SR) flip-flops from rule checking.
- *dftIgnoreConstantOrUnusedFlipFlops*: The default value is off. Set the value of the parameter to on to ignore the violations reported by the Clock_11 rule when any of the following conditions is true:
 - A constant value is specified to the D pin of a flip-flop.
 - A constant value is specified to the Q pin of a flip-flop.
 - The value at the Q pin of a flip-flop is not captured at a scan point.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *clock_pin* (optional): Use this constraint to specify black box pins that are assumed as clock pins.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *dont_touch* (optional): Use this constraint to specify the modules/nets that are not considered for AutoFix.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] <tag> Clock domain '*<clk-name>*' [in '*<du-name>*'] is not controlled by testclock '*<tclk-name>*' (*<num1>* flipflop(s) and *<num2>* user-specified instance(s) affected) [Test clock stops at '*<gate-type1>*']

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported due to missing testclock definition or incorrect connectivity.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] <tag> Clock domain '<clk-name>' [in '<du-name>'] is not controlled by testclock '<tclk-name>' (<num1> flipflop(s) affected)[Test clock stops at '<gate-type1>']

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported when either the test_modes are not properly specified or design connectivity is not correct

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] <tag> Clock domain '<clk-name>' [in '<du-name>'] is not controlled by testclock '<tclk-name>' in testmode (<num1> flipflop(s) affected) <cause_of_clk_src>. Other potential testclock: <potential_test_clk>

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported when either the test_modes are not properly specified or design connectivity is not correct

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] <tag> Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by testclock '`<tclk-name>`' (<num2> user-specified instance(s) affected) [Test clock stops at '`<gate-type1>`']

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported when either the test_modes are not properly specified or design connectivity is not correct

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 5

[WARNING] <tag> Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by any testclock in testmode (<num1> flipflop(s) and <num2> user-specified instance(s) affected) [Clock source is '`<gate-type2>`']

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported when either the test clock or test_modes are not properly specified or design connectivity is not correct.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the flip-flop on which the clock does not reach. Ensure that the complete display mode is set.

Since the violation message provides information on identification of potential clocks, overlay (auxiliary violation mode) the Info_testmode rule and Info_testclock under shift_mode.

Double-click the Incremental Schematic to probe the fan-in code of the failing clock source. Identify the path on the basis of the signal name. The probed signal is annotated with the shift_mode clock value, which is ^ for rising-edge and V for falling-edge. Absence of any value implies that no test_clock has reached this point.

To fix the violation, see Example Code and/or Schematic section.

Message 6

[WARNING] <tag> Clock domain '<clk-name>' [in '<du-name>'] is not controlled by any testclock in testmode (<num1> flipflop(s) affected) [Clock source is '<gate-type2>']

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported when either the test clock or test_modes are not properly specified or design connectivity is not correct.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#)

Message 7

[WARNING] <tag> Clock domain '<clk-name>' [in '<du-name>'] is not controlled by any testclock in testmode (<num2> user-specified instance(s) affected) [Clock source is '<gate-type2>']

Potential Issues

A violation is reported when either the test clock or test_modes are not properly specified or design connectivity is not correct.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#)

Message 8

[WARNING] <tag> Clock domain '<clk-name>' [in '<du-name>'] is not controlled by any testclock in testmode (<num1> flipflop(s) affected) [<reason>]

Potential Issues

A violation is reported when either the test clock or test_modes are not properly specified or design connectivity is not correct.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#)

Message 9

[WARNING] <tag> Clock domain '<clk-name>' [in '<du-name>'] is not controlled by testclock '<tclk-name>' in testmode (<num1> flipflop(s) affected). <cause_of_clk_src>.

Potential Issues

A violation is reported when either the test clock or test_modes are not properly specified or design connectivity is not correct.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#)

Message 10

[WARNING] <tag> Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by testclock '`<tclk-name>`' in testmode (`<num1>` flipflop(s) affected) `<cause_of_clk_src>`. Other potential testclocks: `<potential_test_clk>`

Arguments

- Asynchronous source tag in the form DFT_CLOCK_id <tag>
- Name of the testclock. `<tclk-name>`
- Name of the clock domain. `<clk-name>`
- Name of the design unit. `<du-name>`
- Number of flip-flops. `<num1>`
- Number of user-specified instances. `<num2>`
- Gate type — And gate, Multiplexor, Or gate. `<gate-type1>`
- Gate type — Hanging Net, Port, Tied to ground, Tied to power, Stuck at ground, and Stuck at power. `<gate-type2>`
- The reason for non-reachability of testclock `<reason>`. Reason can be of two types: if the testclock reaches topologically, but is blocked functionally, then the reason will be "testclock stops at `<instance>`" else it will be "clock source is `<instance>`"
- Cause of clock source not controlled by testclock `<cause_of_clk_src>`
- Name of the other potential testclocks, if they exist `<potential_test_clk>` (Only one potential test clock exists for the third message.)

Potential Issues

A violation is reported when either the test_modes are not properly specified or design connectivity is not correct.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage as the flip-flops are not put on scan chain.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the flip-flop on which the clock does not reach.

Ensure that the complete display mode is set.

Since the violation message provides information on the stopped clock propagation, overlay (auxiliary violation mode) the Info_testmode rule under shift_mode. This helps in detecting the location of the blocked clock.

View the Incremental Schematic to probe the fan-in code of the failing clock source. Identify the path on the basis of the signal name. The probed signal is annotated with the shift_mode clock value, which is ^ for rising-edge and V for falling-edge. Absence of any value implies that no test_clock has reached this point.

To locate clock sources that are not receiving testclock in shift mode, select a Clock_11 rule message in the Message Window of the Atrenta Console.

You can also double-click on the violation message to view the spreadsheet that lists the details of the flip-flops that are not testclock-controlled in the shift mode.

To fix the violation, refer to the Example Code and/or Schematic section.

NOTE: *If VCC/VSS is the source net, 'Tied to power' or 'Tied to ground' is displayed as the gate type. However, if the source net is not VCC/VSS but has constant value due to VCC/VSS, 'Stuck at power' or 'Stuck at ground' is displayed.*

Message 11

[INFO] Suggested clock pins for autofix

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic

Verilog

Consider the following example:

```
module IntGenClock (clk, reset, laststage);
```

```

input clk, reset;
output laststage;
reg q1, q2, q3;

always @ (posedge clk or negedge reset)
  if (reset == 1'b0)
    q1 <= 0;
  else q1 <= !q1;

always @ (posedge q1 or negedge reset)
  if (reset == 1'b0)
    q2 <= 0;
  else q2 <= !q2;

always @ (posedge q2 or negedge reset)
  if (reset == 1'b0)
    q3 <= 0;
  else q3 <= !q3;

assign laststage = q3;
endmodule

```

The above Verilog code demonstrates a bad design scenario. To fix the problem, refer to the schematic.

VHDL

Consider the following example:

```

...
Process (clk, reset)
Begin
  If (reset = '0') then
    q1 <= '0';
  elsif (clk'event and clk = '1') then
    q1 <= not q1;
  end process;

Process (q1, reset)
Begin

```

```

If (reset = '0') then
  q2 <= '0';
elsif (q1'event and q1 = '1') then
  q2 <= not q2;
end process;

```

```

Process (q2, reset)
Begin
  If (reset = '0') then
    q3 <= '0';
  elsif (q2'event and q2 = '1') then
    q3 <= not q3;
  end process;

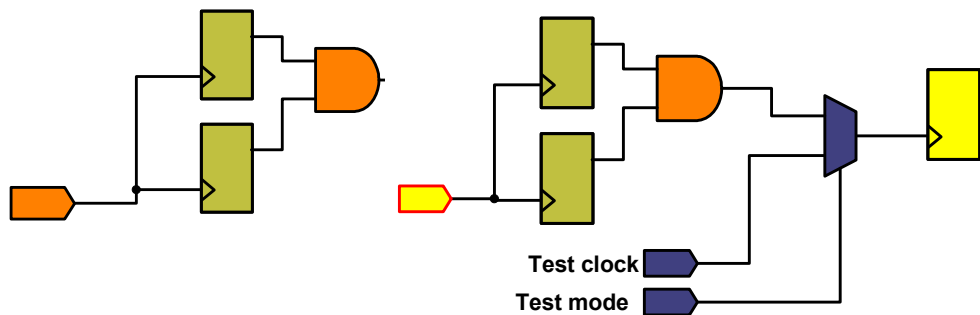
```

```

laststage <= q3;
...

```

The above VHDL code demonstrates a bad design scenario. To fix the problem, refer to the schematic.



The above figure illustrates a good design example.

Default Severity Label

Warning

Rule Group

Clock Rules

Clock Rules

Reports and Related Files

No related reports or files.

Clock_11_capture

Ensure that all clock sources are testclock controlled in capture mode

When to Use

Use this rule to cover all the flip-flops during capture mode.

Description

The Clock_11_capture rule reports internally generated clocks that are not testclock-controlled in the capture mode.

Mux-scan design requires that scan flip-flops be test clock controlled during capture. The Clock_11_capture rule verifies that each clock source (see glossary for definition) receives a pulse from a test clock pin in capture.

NOTE: *If VCC/VSS is the source net, 'Tied to power' or 'Tied to ground' is displayed as the gate type. However, if the source net is not VCC/VSS but has constant value due to VCC/VSS, 'Stuck at power' or 'Stuck at ground' is displayed.*

Rule Exceptions

The flip-flops marked as noscan flip-flops using the *no_scan* constraint or are inferred as noscan flip-flops are not counted. In case, all flip-flops triggered by an internally generated clock are marked as noscan flip-flops, the clock is not considered as a rule-violating clock and no message is generated.

Method

Find all clock sources.

If testmode information is not supplied, report any clock source that is not a root level pin. If testmode information is available, simulate testmode for capture. Then simultaneously simulate a pulse on every test clock. Verify that a pulse appears on clock source. If any flip-flop does not get a pulse, then report a message. For each domain that does not get a test clock pulse, walk the path from the test clock that should drive this clock source. This walk stops at the element that has a test clock pulse in its input but does not have a pulse on its output. This element is called the blocking gate.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*

- *dftAutoFix*: The default value of the parameter is as follows:

```
"{+RULES[nothing]+TMPORT[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}"
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
"{+RULES[<rulelist>]+TMPORT[tmport]+TCLKPORT[tclkport]}"
```

- *dftCPMechanism*: The default value is sequential. Set the value of the parameter to parallel to specify the clocking methodology used by the respective rules.
- *dftCaptureCriteria*: The default value is simulation. Set the value of the parameter to controllability to select the criteria based on which the flip-flops are marked capture-ready and latches are treated transparent in capture conditions.
- *dftTagAsync07Clock11Scan08*: The default value is off. Set the value of the parameter to on to add a unique tag for every asynchronous source (set, reset, or clock), which is visible in the rule violation.
- *dftInferredDriverControl*: The default value is optimistic. Set the value of the parameter to pessimistic. Therefore, the presence of a single non-scan source (black box, hanging net, noscan) in the unblocked fan-in cone will make the node uncontrollable.
- *dft_ignore_constant_supply_flip_flops*: The default value is off. Set the value of the parameter to on to ignore Synthesis Redundant (SR) flip-flops from rule checking.
- *dftIgnoreConstantOrUnusedFlipFlops*: The default value is off. Set the value of the parameter to on to ignore the violations reported by the Clock_11_capture rule when any of the following conditions is true:
 - A constant value is specified to the D pin of a flip-flop.

- ❑ A constant value is specified to the Q pin of a flip-flop.
- ❑ The value at the Q pin of a flip-flop is not captured at a scan point.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_maximum_number_of_rows_with_schematic*: Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design. The *Clock_11_capture* rule uses -testclock argument of the *clock* constraint.
- *clock_pin* (optional): Use this constraint to specify black box pins that are assumed as clock pins.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] <tag>Clock domain '*<clk-name>*' [in '*<du-name>*'] is not controlled by testclock '*<tclk-name>*' (<num1> flipflop(s) and <num2> user-specified instance(s) affected)[Test clock stops at '*<gate-type1>*']

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain is not controlled by a test clock and some flip-flops and user specified instances are affected in the

design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] <tag>Clock domain '[<clk-name>](#)' [in '[<du-name>](#)'] is not controlled by testclock '[<tclk-name>](#)' (<num1> flipflop(s) affected)[Test clock stops at '[<gate-type1>](#)']

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain is not controlled by a test clock and some flip-flops are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] <tag>Clock domain '[<clk-name>](#)' [in '[<du-name>](#)'] is not controlled by testclock '[<tclk-name>](#)' (<num2> user-specified instance(s) affected)[Test clock stops at '[<gate-type1>](#)']

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain is not controlled by a test clock and some user specified instances are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] <tag>Clock domain '<clk-name>' [in '<du-name>'] is not controlled by any testclock in testmode (<num1> flipflop(s) and <num2> user-specified instance(s) affected) [Clock source is '<gate-type2>']

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain of <gate-type2> is not controlled by a test clock and some user specified instances and flip-flops are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 5

[WARNING] <tag>Clock domain '<clk-name>' [in '<du-name>'] is not controlled by any testclock in testmode (<num1> flipflop(s) affected) [Clock source is '<gate-type2>']

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain of <gate-type2> is not controlled by a test clock and some flip-flops are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 6

[WARNING] <tag>Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by any testclock in testmode (`<num2>` user-specified instance(s) affected) [Clock source is '`<gate-type2>`']

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain of `<gate-type2>` is not controlled by a test clock and some user-specified instances are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 7

[WARNING] <tag> Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by any testclock in testmode (`<num1>` flipflop(s) affected) [`<reason>`]

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain is not controlled by any test clock and some flip-flops are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 8

[WARNING] <tag> Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by testclock '`<tclk-name>`' in testmode (<num1> flipflop(s) affected). `<cause_of_clk_src>`.

Arguments

To see the list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if a clock domain is not controlled by some test clock and some flip-flops are affected in the design.

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 9

[WARNING] <tag> Clock domain '`<clk-name>`' [in '`<du-name>`'] is not controlled by testclock '`<tclk-name>`' in testmode (<num1> flipflop(s) affected) `<cause_of_clk_src>`. Other potential testclock: `<potential_test_clk>`

Arguments

- Asynchronous source tag in the form DFT_ASYNC_CLR_id, DFT_ASYNC_SET_id <tag>
- Name of the testclock. `<tclk-name>`
- Name of the clock domain. `<clk-name>`
- Name of the design unit. `<du-name>`
- Number of flip-flops. `<num1>`

- Number of user-specified instances. <num2>
- Gate type — And gate, Multiplexor, Or gate. <gate-type1>
- Gate type — Hanging Net, Port, Tied to ground, Tied to power, Stuck at ground, and Stuck at power. <gate-type2>
- The reason for non-reachability of testclock <reason>. Reason can be of two types: if the testclock reaches topologically, but is blocked functionally, then the reason will be "testclock stops at <instance>" else it will be "clock source is <instance>"
- Cause of clock source not controlled by testclock <cause_of_clk_src>
- Name of the other potential testclocks, if they exist <potential_test_clk> (Only one potential test clock exists for the third message.)

Potential Issues

The violation message appears, if a clock domain is not controlled by a test clock and some user specified instances are affected in the design.

Consequences of Not Fixing

Not fixing this violation may result in reduced observability and coverage.

How to Debug and Fix

Select the violation message and open the Incremental Schematic window. The Incremental Schematic window highlights the path from the flip-flop to its clock source that is not controlled by the testclock during capture mode.

To locate clock sources that are not receiving testclock in capture mode, select a Clock_11_capture rule message in the Message Window of the Atranta Console.

When the Clock_11_capture rule is run in conjunction with the [Scan_08](#) rule, a tag can be added in the violation messages by setting the value of the [dftTagAsync07Clock11Scan08](#) rule parameter to on. This enables the correlation of the cause of unscannability.

To view the violations under capture condition for the Info_testmode and Info_testmode rules along with the violation of the Clock_11_capture rule in the Incremental Schematic window, select the violation for the Clock_11_capture rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Clock_11_capture rule in the

Incremental Schematic window. This is useful in debugging the violation for the Clock_11_capture rule.

You can also double-click on the violation message to view the spreadsheet that lists the details of the flip-flops that are not testclock-controlled in the capture mode.

To fix the violation message, assign test clocks to affected flip-flops.

Message 10

[INFO] Suggested clock pins for autofix

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

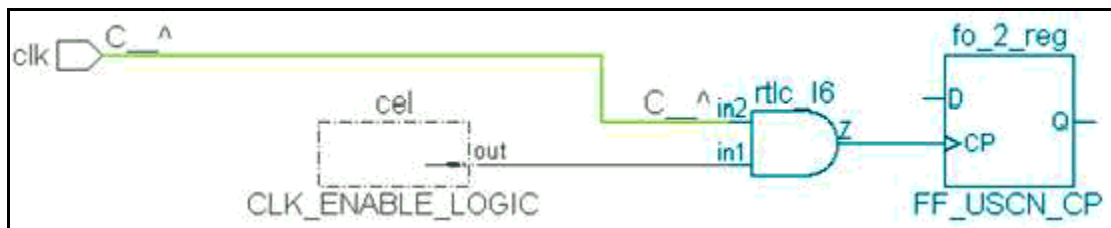
This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic

Consider the following figure:



In the above figure, the Clock_11_capture rule reports a violation because CLK_ENABLE_LOGIC is not controlled by any testclock.

Default Severity Label

Warning

Clock Rules

Rule Group

Clock Rules

Reports and Related Files

No related reports or files.

Clock_13

This rule has been deprecated.

The functionality of the Clock_13 rule is covered by the Clock_sync01 rule of the SpyGlass CDC product.

Clock_14

Ensure that the negedge registers are only driving primary outputs and no other flip-flop

When to Use

Use this rule to identify flip-flops that are triggered at the negative edge of their clocks and are driving some other flip-flop.

Description

The Clock_14 rule reports violation for flip-flops that are triggered at the negative edge of their clocks and are driving some other flip-flop.

Method

If no testmode information exists, then, find clock phase from each user-specified system clock to each flip-flop. A negative edge flip-flop is a flip-flop with an odd number of phase inversions in the path from the root-level clock to the clock pin of this flip-flop. Walk the flip-flop q pin fan-out cone of each negative edge flip-flop. If any sequential device is hit, report a message.

If testmode for capture and at least one testclock exist, find clock phase from each test clock to each flip-flop. Walk the flip-flop q pin combinational fan-out cone of each negative-edge testclock flip-flop. Stop the walk at any node with non-x simulation values. If any sequential device is hit, report a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

NOTE: *At least one testclock must be specified for the Clock_14 rule to run.*

- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Flip-flop '`<flip-flop-name>`', driving a register, is triggered at negative edge of clock '`<clk-name>`'

Arguments

- Name of the flip-flop, `<flip-flop-name>`
- Top-level clock port that is negedge triggered, `<clk-name>`

Potential Issues

A violation is reported because the negedge flip-flop is driving the scan point, except PO.

Consequences of Not Fixing

During capture, vector may get corrupted before this flip-flop captures data.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the path from the negative edge triggered flip-flop to the flip-flop driven by this flip-flop.

You can also view the violations for the Info_testmode (under capture condition) and Info_path rules along with the violation of the Clock_14 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_14 rule and open the Incremental Schematic window.

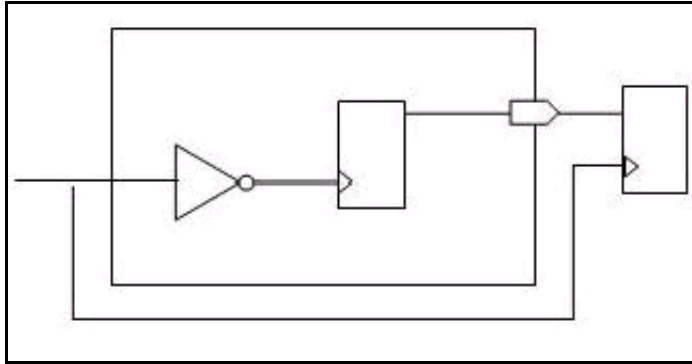
The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Clock_14 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_14 rule.

To fix the violation, use positive edge or posedge triggered flip-flop.

Negative registers are allowed only if they feed root-level output pins.

Example Code and/or Schematic

Consider the following figure:



The Clock_14 rule reports violation for above example because the flip-flop is triggered at the negative edge of its clocks and is not driving primary output.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files

Clock_16

Ensure that the flip-flops capturing on the falling (rising) edge must not have any data paths from flip-flops capturing on the rising (falling) edge of the same rtz (rto) clock source

When to Use

Use this rule to identify clock source driving both rising and falling edge devices.

Description

The Clock_16 rule reports violation for flip-flops that have data paths from flip-flops that are triggered on the opposite edge of the same clock.

For information on rtz and rto clocks, see [Test clocks](#) section.

Default Weight

10

Language

Verilog, VHDL

Method

If testmode for capture and at least one testclock exists:

Find all test clock domains. For each test clock, use rto/rtz information to define first/second edge. If this test clock reaches flip-flops on one edge only, then go to the next test clock since no message is possible. If this test clock does feed flip-flops on both edges, then for each flip-flop using the first edge, walk its q-pin fan-out cone stopping at sequential or primary output ports. If a flip-flop that lies in the domain of this test clock and uses its second edge is hit on its input pin, report a message.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed:

```
[WARNING] Flip '<flip-flop2-name>' has data path from flip-flop
'<flip-flop1-name>' which captures on the other edge of same
clock ('<clk-name>')
```

Arguments

- Name of the q-pin of the flip-flop where such path starts. <flip-flop1-name>
- Name of the d-pin of the flip-flop where such path ends. <flip-flop2-name>
- Name of the common clock <clk-name>

Potential Issues

A violation is reported due to wrong design connectivity or wrong edge usage.

Consequences of Not Fixing

Mixed-edge flip-flops that are not independently clocked, require special care to ensure that scanned in setup data, necessary for a given test, is not corrupted during the capture phase for that test. If such corruption occurs, coverage may be reduced or good parts may be declared defective. Separating such devices into independently clocked chains prevents this possibility.

How to Debug and Fix

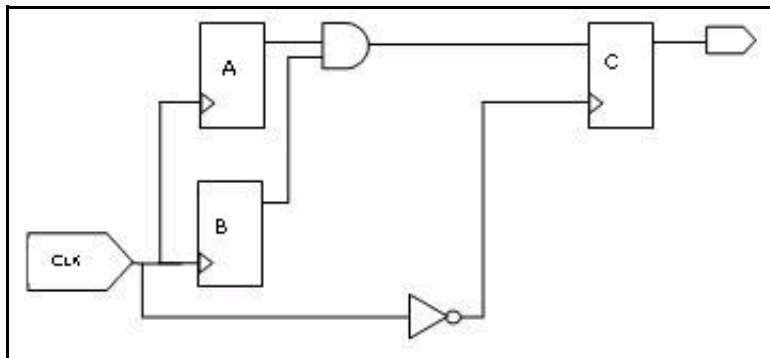
View the Incremental Schematic of the violation message. The Incremental Schematic displays the path between two flip-flops. Here, one flip-flop passes data to other flip-flop and both are triggered by same clock in different phase. Probe the fan-in cone of the clock pins of the concerned flip-flops to see how same clock is driving both the flip-flops in different phases.

To fix the violation, either put both the flip-flops on the same edge or reverse the order.

Example Code and/or Schematic

Example 1

Consider the following example:

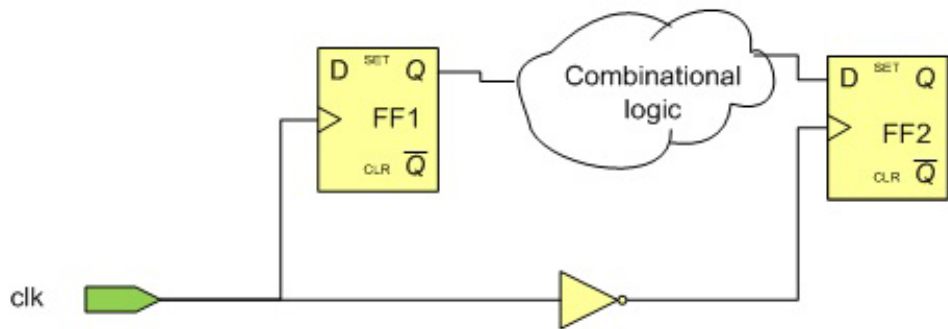


In the above example, the Clock_16 rule reports a violation for flip-flop, C, because it has data path from flip-flops, A and B, that are triggered on the opposite edge of the same clock.

Example 2

The following figure illustrates the scan flip-flops:

Clock Rules



In the above figure, FF1 changes state before FF2 because of clock inversion. This can affect FF1's scanin state and can also affect coverage as well as pass/fail status.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_17

Ensure that the capture clocks are not gated by flip-flops capturing on the same clock

When to Use

Use this rule to identify the capture clocks that are gated by flip-flops triggered by the same clock.

Description

The Clock_17 rule reports violation for capture clocks that are gated by flip-flops triggered by the same clock.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power, ground and testmode for capture.

For each testclock, mark all devices on paths from this testclock to all clock sources clocked by this testclock. For each such marked combinational device, walk the combinational fan-in cone from any non-clock path input to this device. Stop the walk for any node with a non-x simulation value. If any flip-flop hit in this walk is clocked by this testclock, report a message.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftShowSourceCentricViolation](#): The default value is none. Specify a rule list as an input to the parameter to check whether a rule should report instance specific violation or source specific violation.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- [dft_maximum_number_of_rows_with_schematic](#): Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

[Capture](#)

Messages and Suggested Fix

Message 1

The following message is displayed when the [dftShowSourceCentricViolation](#) rule parameter is set to off.

[WARNING] Path from pin '<pin-name>' to clk source '<clk-name>' is gated through flip-flop '<flip-flop-name>'

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

This message is displayed when the [dftShowSourceCentricViolation](#) rule

parameter is set to on.

[WARNING] Path from pin <pin-name> to clock source <clk-name> is gated through <num> flip-flop(s)

Arguments

- The clock path device that is gated. <clk-name>
- The output net of a flip-flop gating this source. <flip-flop-name>
- The testclock pin that clocks the flip-flop and also drives this device. <pin-name>
- The number flip-flop gating this source. <num>

Potential Issues

A violation is reported due to wrong design connectivity.

Consequences of Not Fixing

Circuits flagged by the Clock_17 rule may cause capture to operate incorrectly. If capture clocks are gated by flip-flops in the domain of a capture clock, then the enable values, required to propagate the capture pulse to all flip-flops in the domain of this capture clock, may be corrupted by capture act unless additional and possibly excessive burdens are placed on skew management.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic displays the path from clock pin of the violating flip-flop to the clock source through the gating flip-flop. Start probing the clock pin of the gating flip-flop to check how same clock is driving the gating flip-flop as well as a violating flip-flop.

You can also view the violations for the Info_testmode (under capture condition) and Info_path rules along with the violation of the Clock_17 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_17 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Clock_17 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_17 rule.

To fix the violation, ensure that the capture clocks are not gated by flip-flops capturing on the same clock.

Example Code and/or Schematic

Example 1

The following figure illustrates capture clock that changes the state of the gating device, FF1, before it reaches the gating device, u1:

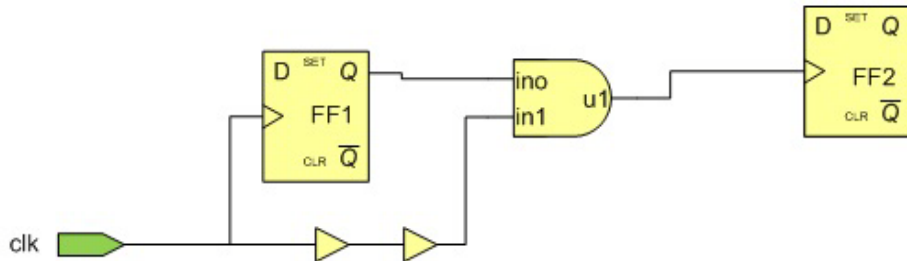
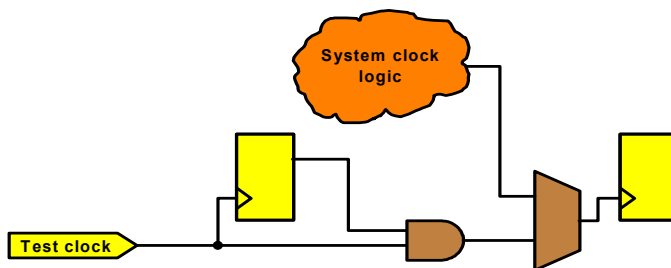


FIGURE 49. Clock Gating Example

In the above example, ATPG sets that the scanin state of FF1 as 1 to enable the clock path through u1. However, if FF1 changes to 0 sufficiently before the pulse on u1.in1 can transition through u1, then capture at FF2 does not take place. Therefore, the tests fail to operate as designed.

Example 2

Consider the following figure:



The above figure illustrates a good design example.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_18

Ensure that the flip-flops are not triggered on the negative edge of their clocks.

When to Use

Use this rule to identify flip-flops that are triggered on the negative edge of their clocks.

Description

The Clock_18 rule flags flip-flops that are triggered on the negative edge of their clocks with respect to their testclocks.

Method

If testmode information is not supplied, report any system clock pin that clocks any flip-flop on the negative edge.

If testmode for capture and at least one testclock are supplied, simulate any available testmode conditions. Report any testclock pin that clocks any flip-flop on the negative edge.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

NOTE: *At least one testclock must be specified for the Clock_18 rule to run.*

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Clock '`<clk-name>`' is triggering `<num>` flipflop(s) (e.g.: `<flop-name>`) at its negative edge

Arguments

- Name of clock pin, `<clk-name>`
- Number of flip-flops triggered on the falling edge of specified clock, `<num>`
- One of the flip-flops triggered on the falling edge of specified clock, `<flop-name>`

Potential Issues

A violation is reported because negative edge of a clock is used for capturing data.

Consequences of Not Fixing

Use of both positive edge and negative edge clocks lengthens the test program (since edges must be considered instead of pulses) and can decrease time margins.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the path from the clock pin to the flip-flop, which is triggered at its negative edge.

You can also view the violations for the Info_testmode (under capture condition) and Info_testclock rules along with the violation of the Clock_18 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_18 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Clock_18 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_18 rule.

To fix the violation, correct the phase.

Example Code and/or Schematic

Example 1

Consider the following sample Verilog code:

```
...
reg q1, q2;

always @ (posedge clock)
    q1 <= data1;

always @ (negedge clock)
    q2 <= data2;
...
```

The above example demonstrates a bad Verilog code because negative edge of a clock is used for capturing data.

Example 2

Consider the following sample Verilog code:

```
...
reg q1, q2;

always @ (posedge clock)
    q1 <= data1;

always @ (posedge clock)
    q2 <= data2;
...
```

The above example demonstrates a good Verilog example because positive edge of a clock is used for capturing data.

Example 3

Consider the following sample VHDL code:

```
...
Process (clock)
Begin
    If (clock'event and clock = '1') then
        Q1 <= data1;
```

```
    End if;
End process;

Process (clock)
Begin
    If (clock'event and clock = '0') then
        Q2 <= data2;
    End if;
End process;
...
```

The above example demonstrates a bad VHDL code because negative edge of a clock is used for capturing data.

Example 4

Consider the following example:

```
...
Process (clock)
Begin
    If (clock'event and clock = '1') then
        Q1 <= data1;
    End if;
End process;

Process (clock)
Begin
    If (clock'event and clock = '1') then
        Q2 <= data2;
    End if;
End process;
...
```

The above example demonstrates a good VHDL code because positive edge of a clock is used for capturing data.

Default Severity Label

Warning

Clock Rules

Rule Group

Clock

Reports and Related Files

No related files or reports.

Clock_21

Ensure that clocks do not drive flip-flop set or reset pins

When to Use

Use this rule to identify clocks that drive flip-flop's set or reset pins.

Description

The Clock_21 rule reports violation for clocks that are also setting/resetting flip-flops or latches.

Prerequisites

At least one testclock must be specified for the Clock_21 rule to run.

Default Weight

10

Language

Verilog, VHDL

Method

If testmode information is not supplied, walk the fan-out cone from all system clocks. If any flip-flop set or reset pins are reached, report a message.

If testmode for capture and at least one testclock are supplied, walk the unblocked paths in the combinational fan-out cone from all testclocks. Stop the walk on any node with a non-x simulation value. If any flip-flop set or reset pins are reached, report a message.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (mandatory): Use this constraint to define the clocks of a design.

NOTE: *The Clock_21 requires you to specify at least one testclock.*

- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Clock_21 rule:

[WARNING] Clock '*<clk_name>*' drives '*<count>*' set/reset pin(s) of flip-flop(s)

Arguments

- Name of the clock, *<clk-name>*
- Set/Reset pin count, *<count>*

Potential Issues

A violation is reported due to either wrong design connection or if the path is not blocked.

Consequences of Not Fixing

Overlapping clock logic and asynchronous logic gives rise to race conditions and should be avoided.

How to Debug and Fix

Double-click the violation message to view the spreadsheet, which lists the clock names and the asynchronous pin names.

Click on an asynchronous pin name and view the Incremental Schematic for the same. The Incremental Schematic displays the path from the clock to the asynchronous pin of the flip-flop.

You can also view the violations for the Info_testclock (under capture condition) and Info_path rules along with the violation of the Clock_21 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_21 rule and open the Incremental Schematic window.

The violation messages for the Info_testclock and Info_path rules overlap the violation message for the Clock_21 rule in the Incremental Schematic

window. This is useful in debugging the violation for the Clock_21 rule. Overlay (Auxiliary violation mode) the Info_testmode rule under capture_mode. This helps in identifying how the violating path is sensitized.

To fix the violation, correct the connectivity or bypass the reset path.

Example Code and/or Schematic

Consider the following figure illustrating clocks that feed flip-flop set or clear pins:

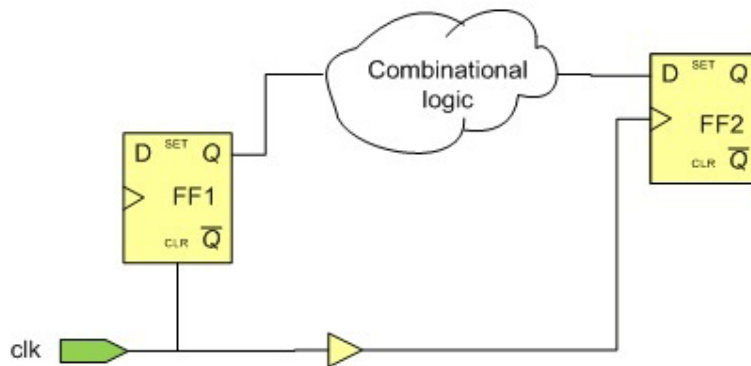


FIGURE 50. : Clock feeding async pin

In the above example, the clocks feeding asynchronous pins can corrupt the scanin state or possibly prevent a flip-flop from being scanned. In either case, coverage is compromised. Therefore, you should ensure that clock in the design should not drive flip-flop set or clear pins.

Default Severity Label

Warning

Rule Group

Clock

Related Reports or Files

The Clock_21 rule generates the following spreadsheet report:

	B Clock Name	C Async Pin Name
1	clk1	test.sff2.RD
2	clk1	test.sff3.RD

FIGURE 51. Clock_21 Spreadsheet Report

The Clock_21_<xx>.csv report lists the clock name and the respective asynchronous pin name, which the clock is driving.

Clock_22

Ensure that each clock source should be combinationaly driven by one testclock only.

When to Use

Use this rule to get clear control on every root level testclock.

Description

The Clock_22 rule reports violation for clocks that are combinationaly driven by more than one test clock.

Each clock source should have a fan-in path from only one testclock pin.

NOTE: *The Clock_22 rule has been deprecated and will be removed in a future release.*

Method

Simulate power, ground and capture conditions.

Walk the unblocked combinational fan-in cone from each clock source. Stop the walk at any node with non-x simulation values.

If two or more testclocks are hit or no testclocks are hit, report the clock source and the testclocks that hit this source.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (optional): Use this constraint to define the clocks of a design.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Clock source '`<clk-name>`' is multiply combinationally driven by test-clocks [`<tclknamelist>`]

Arguments

- Name of the violating clock-source, `<clk-name>`
- Names of the root-level test-clock ports combinationally driving the specified clock, `<tclknamelist>`

Potential Issues

The violation message appears if a clock is driven by more than one test clock.

Consequences of Not Fixing

Not fixing the violation may result in reduced control over every root level clock.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the clock source of a flip-flop that is driven by the combination of multiple clocks.

To fix the violation, allow only one test clock to drive the flip-flop and block the unwanted clocks.

Message 2

[WARNING] Clock source '`<clk-name>`' is not combinationally driven by any test-clock

Arguments

Name of the violating clock-source, `<clk-name>`

Potential Issues

This violation message appears if no testclock reaches the source clock.

Consequences of Not Fixing

Not fixing this violation may result in reduced controllability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the clock source of a flip-flop that is driven by the any test clock.

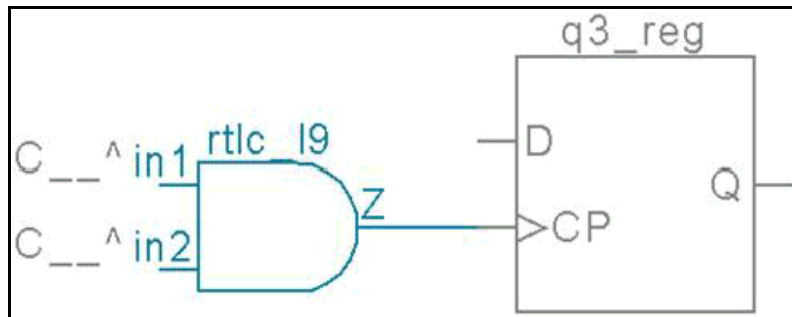
You can also view the violations for the Info_testmode (under capture condition) and Info_path rules along with the violation of the Clock_22 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_22 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Clock_22 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_22 rule.

To fix the violation, assign a test clock to the source clock.

Example Code and/or Schematic

Consider the following figure:



In the above figure, the Clock_22 rule reports a violation because the clock pin, CP, of q3_reg is driven by two test clocks.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock Rules

Clock_23

Ensure that the scan flip-flops must only trigger on the positive edge

When to Use

Use this rule to identify scan flip-flops that are triggered on the negative edge of their clocks.

Description

The Clock_23 rule reports violation for scan flip-flops that are triggered on the negative edge of their clocks.

Method

Simulate testmode for capture and power, ground conditions.

If any flip-flop, not marked as “noscan” is triggered at the negative edge of any user specified testclock then report a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftClockEdgeType*: The default value is none. Set the value of the parameter to `posedge` or `negedge` to specify the type of flip-flops to be highlighted in the Atrenta Console when *Clock_23* violation message is probed.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they so qualify.

Messages and Suggested Fix

[WARNING] Clock '`<clk-name>`' is triggering `<num>` scan flip-flop(s) (e.g.: '`<flip-flop-name>`') at its negative edge

Arguments

- Name of clock pin, `<clk-name>`
- Number of scan flip-flops triggered on the falling edge of specified clock, `<num>`
- One of the scan flip-flops triggered on the falling edge of specified clock, `<flip-flop-name>`

Potential Issues

A violation is reported when the negative edge of a clock is being used in the design.

Consequences of Not Fixing

If a design have both positive edge and negative edge triggered flip-flops, then the following consequences are possible:

1. The fault coverage could be negatively impacted as the test pattern space is restricted due to the fact that the negative-edge triggered flip-flop and the previous positive-edge triggered flip-flop will always have the same value at the end of scan shift.
2. Having a mix of negative- and positive-edge triggered flip-flops is in general not a good practice as it would introduce "half cycle" paths.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the path from the clock pin to the scan flip-flop that is triggered at its negative edge.

You can also view the violation for the `Info_testclock` rule along with the violation of the `Clock_23` rule in the Incremental Schematic window. To do this, double-click the violation for the `Clock_23` rule and open the Incremental Schematic window.

The violation message for the Info_testclock rule overlaps the violation message for the Clock_23 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_23 rule.

To fix the violation, use positive edge triggered flip-flop.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_24

Ensure that the test clocks do not change shift mode signals.

When to Use

Use this rule to keep the testmode logic constant in the sequential design

Description

The Clock_24 rule reports violation for devices whose outputs are set by the application of testmode for scanshift, are changed when a test clock is pulsed.

The testmode logic must not be changed by pulses on the testclocks.

The Clock_24 rule reports the first device, with a non-x value in testmode, whose output changes state when the testclock is pulsed.

Method

If all testmode for scan shift conditions are combinational (that is specified by a single value) then skip this check.

Simulate power, ground and all available testmode for scan shift conditions. For each testclock, simulate a pulse with all other testclocks at their "return to state." If any testclock causes a signal with a non-x value resulting from testmode changes its state, report a message.

Default Weight

1

Language

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design. The Clock_24 rule uses the `-testclock` argument of the `clock` constraint.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Test clock '`<clk-name>`' has changed state of instance '`<inst-name>`' (`<state1>=><state2>`)

Arguments

- Name of the test clock pin, `<clk-name>`
- Name of the device, `<inst-name>`
- Original or changed state, `<state1>`, `<state2>`

Potential Issues

The violation message appears if a test clock when pulsed changes the output of a device.

Consequences of Not Fixing

Not fixing this violation may result in changing the value of the testmode logic in the sequential design which can affect the design.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the testclock pin and the instances whose output net shift mode simulation value changes when the testclock is pulsed.

To view the violations under capture condition for the Info_testmode and Info_testclock rules along with the violation of the Clock_24 rule in the Incremental Schematic window, double-click the violation for the Clock_24 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules

overlap the violation message for the Clock_24 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_24 rule.

To fix the violation, back annotate the text to indicate the value change such as 1=>x near the reported device.

Example Code and/or Schematic

Example 1

Consider the following sample test.sgdc file:

```
current_design test
  clock -name clk -testclock
  test_mode -name ff1 -value 1 -scanshift
  test_mode -name d -value 0
  test_mode -name clk -value 010x
```

Now, consider the following RTL:

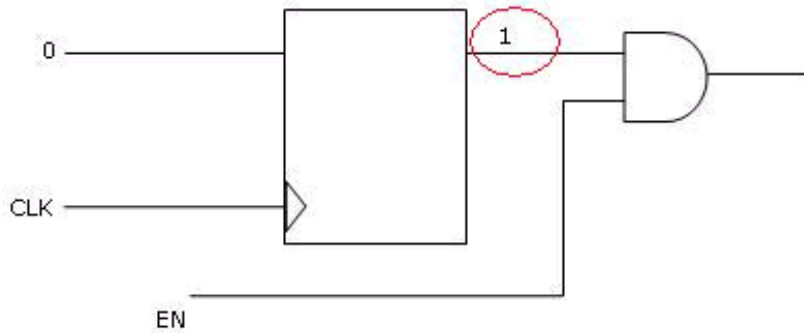
```
test.v
module test(d, en, clk);
input d, en, clk;

reg ff1;
wire w;

always @(posedge clk)
  ff1 <= d;

assign w = ff1 & en;
endmodule
```

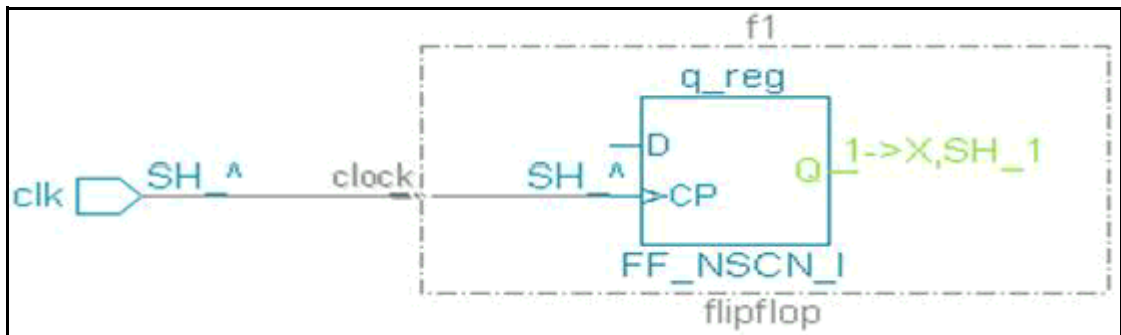
Following is the schematic pertaining to the above RTL:



The Clock_24 rule reports a violation for the test clock, `test.clk`, it has changed state of instance to `test.ff1_reg'` (`1=>0`).

Example 2

Consider the following figure:



In the above figure, the Clock_24 rule reports violation for flip-flop, `f1`, as its output changes when the test clock, `clk`, is pulsed.

Default Severity Label

Warning

Clock Rules

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_25

Ensure that the test clock must not drive flip-flops greater than that specified in `-fflimit` or `-fflimit_percentage` field(s) of the clock constraint.

When to Use

Use this rule to keep the testmode logic constant in the sequential design.

Description

The Clock_25 rule reports violation for testclocks that drive more than the specified number or percentage of flip-flops.

Method

For each testclock

- if `fflimit/fflimit_percentage` is specified

 - Bring circuit in shift mode

 - simulate clock pulse at testclock

 - Count the number of flip-flops which receive the clock pulse

 - if this number exceeds `fflimit/fflimit_percentage`, violate

 - Repeat same for capture mode.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (mandatory): By default, the Clock_25 rule allows a testclock to drive any number of flip-flops. Use the `-fflimit` or

-fflimit_percentage arguments of the clock constraint specifying the test clock to specify a limit.

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] Test clock '<clk-name>' has -fflimit as '<limit>' but drives '<num>' flip-flops in <mode-name> mode

Arguments

- Name of the testclock, <clk-name>
- The limit specified for the testclock, <limit>
- The actual number of flip-flops driven by the testclock, <num>
- Mode string (shift or capture), <mode-name>

Potential Issues

For information on the cause of the violation message, click [Potential Issues](#).

Consequences of Not Fixing

For information on the consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For information on how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Test clock '<clk-name>' has -fflimit_percentage as '<upper_limit_percentage> (count: <upper_limit_count>)' but drives '<actual_percentage> (count: <actual_count>)' flip-flops in <mode-name> mode

Arguments

- Test clock name, *<tclk-name>*
- Upper limit of flip-flop percentage that should be driven by the test clock, *<upper_limit_percentage>*
- Upper limit of flip-flop count (derived from percentage) that should be driven by the test clock, *<upper_limit_count>*
- Actual flip-flop percentage driven by the test clock, *<actual_percentage>*
- Actual flip-flop count driven by the test clock, *<actual_count>*
- Mode string (shift or capture), *<mode-name>*

Potential Issues

For information on the cause of the violation message, click [Potential Issues](#).

Consequences of Not Fixing

For information on the consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For information on how to debug and fix the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Test clock '*<tclk-name>*' has `-fflimit` as '*<limit>*' and `-fflimit_percentage` as '*<upper_limit_percentage>*' (count: *<upper_limit_count>*)' but drives '*<actual_percentage>*' (*<actual_count>*)' flip-flops in *<mode-name>* mode

Arguments

- Test clock name, *<tclk-name>*
- Upper limit of flip-flop count that should be driven by the test clock, *<limit>*
- Upper limit of flip-flop percentage that should be driven by the test clock, *<upper_limit_percentage>*
- Upper limit of flip-flop count (derived from percentage) that should be driven by the test clock, *<upper_limit_count>*

- Actual flip-flop percentage driven by the test clock, `<actual_percentage>`
- Actual flip-flop count driven by the test clock, `<actual_count>`
- Mode string (shift or capture), `<mode-name>`

Potential Issues

The violation is reported when the testclock drives more than specified number of flip-flops.

Consequences of Not Fixing

Not fixing this violation may result in reduced strength of clock pulse and skews might appear on pulse which can lower the functionality.

How to Debug and Fix

View the Incremental Schematic and block the unwanted clock paths.

To view the violation for the Info_testclock rule along with the violation of the Clock_25 rule in the Incremental Schematic window, perform the following steps, double-click the violation for the Clock_25 rule and open the Incremental Schematic window.

The violation message for the Info_testclock rule overlaps the violation message for the Clock_25 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_25 rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_26

Reports if a test clock is applied on the sensitized fan-out cone of another test clock

When to Use

Use this rule to ensure that a test clock does not feed another test clock.

Description

The Clock_26 rule reports a violation when a test clock is applied on the sensitized combinational fan-out cone of another test clock in the Shift or Capture mode.

The test clock is specified through the `-testclock` field of the `clock` constraint.

NOTE: *The use of the `-domain` field of the `clock` constraint does not affect a violation of the Clock_26 rule.*

Method

If more than two clock constraints with `-testclock` field are defined

Traverse fan-in cone of all internal testclocks (not on PI) in design and check if any other testclock is hit during the traversal.

If another testclock is hit

Report violation for each such occurrence.

The stop points for the traversal are sequential elements, black boxes, clock shaper cells, non-X paths and any blocked path.

Prerequisites

This rule is run when at least two `clock` constraints with the `-testclock` field are specified in the SGDC file.

Rule Exceptions

This rule does not report a violation in the following cases:

- If the path from one test clock to another traverses a divide-by sequential device.

For example, consider the following figure:

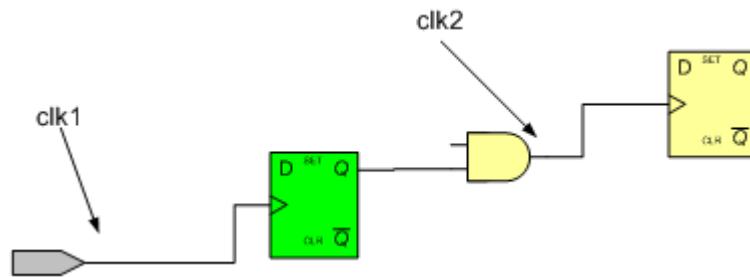


FIGURE 52. Example of a sequential traversal

In the above figure, the path from `clk1` to `clk2` traverses a divide-by-sequential device. Therefore, a violation is not reported in this case.

- If the path from one test clock to another traverses a clock shaper.

For example, consider the following figure:

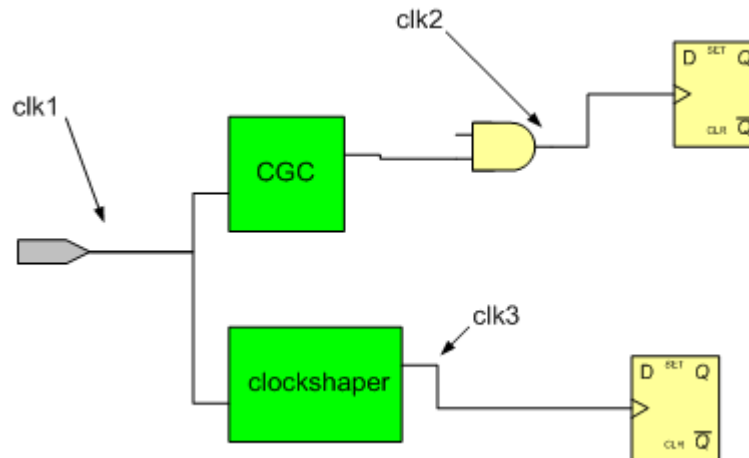


FIGURE 53. Example of a CGC and a clock shaper

In the above figure, the path from `clk1` to `clk3` traverses a clock shaper. Therefore, a violation is not reported for this path.

However, the path from `clk1` to `clk2` traverses an enabled CGC.

Therefore, a violation is reported for this path.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (mandatory): Use this constraint to define the clocks of a design. The Clock_26 rule uses the `-testclock` argument of the `clock` constraint.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

[Scanshift](#), [Capture](#)

Messages and Suggested Fix

[WARNING] testclock constraint '`<overwritten-tclk>`' is applied on sensitized fanout of test clock '`<overwriting-tclk>`' in '`<mode-name>` mode'

Arguments

- Name of the test clock that is being overwritten by another test clock, `<overwritten-tclk>`
- Name of the test clock that is overwriting the other test clock, `<overwriting-tclk>`
- The mode name as Shift or Capture, `<mode-name>`

Potential Issues

The violation is reported because a test clock constraint is applied on the sensitized combinational fan-out cone of another test clock constraint in the Shift or Capture mode.

Consequences of Not Fixing

Not fixing this violation may result in a difference between the simulation and actual/functional results.

How to Debug and Fix

View the incremental schematic of the violation message. The incremental schematic displays the path from the overwritten test clock to the overwriting test clock.

To fix the violation, verify that the highlighted path is correct and is not missing any testmode condition. If the path is correct, remove the constraint specifying the overwriting test clock from the SGDC file.

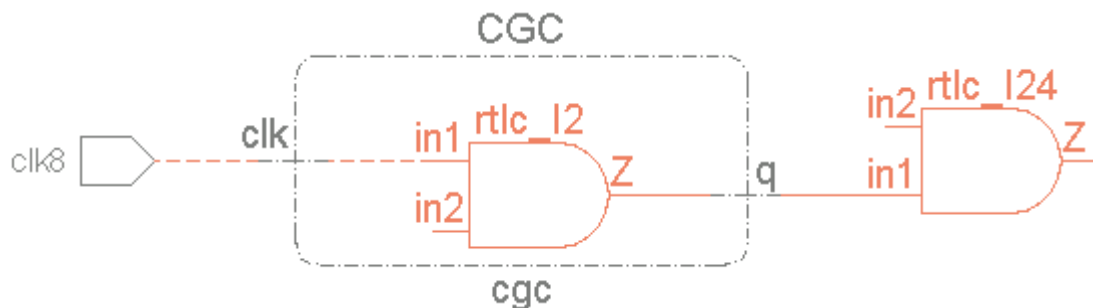
Example Code and/or Schematic

Consider the following SGDC file snippet:

```
clock -name clk8 -testclock -shift -capture
clock -name w10 -testclock -shift -capture
```

In this case, a violation is reported by the Clock_26 rule because the w10 test clock constraint is applied on the sensitized combinational fan-out cone of the clk8 test clock constraint in the Shift and Capture modes.

The following schematic is generated for the violation message:



The above schematic shows the path from the overwritten test clock

(clk8) to the overwriting test clock (w10). The path passes through a clock gating cell.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files.

Clock_27

Detects edge inconsistency between CGC and driven flip-flops

When to Use

This rule is used to check for edge inconsistency between clock gating cells and the flip-flops driven by it.

Description

The Clock_27 rule reports a violation when an 'rτζ' clock through negative CGC is driving flip-flops at its rising edge or an 'rτo' clock through positive CGC is driving flip-flops on its falling edge.

For information on rtz and rto clocks, see [Test clocks](#) section.

Prerequisite

This rule will run if there are clock gating cells present in the design.

Rule Exception

If a CGC and driven flip-flops are edge consistent, then the Clock_27 rule does not report any violation for that CGC.

Default Weight

1

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- *clock* (mandatory): Declares the clock pins used as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.

Operating Mode

[Scanshift](#), [Capture](#)

Messages and Suggested Fix

Message 1

'rtz' clock, through a negative clock gating cell <cgc-name>, is driving <no-of-flip-flops> flip-flop(s) at its rising edge in <mode>.

Arguments

- name of the clock gating cell, <cgc-name>
- number of flip-flops which are edge inconsistent, <no-of-flip-flops>
- shift or Capture mode, <mode >

Potential Issues

A violation is reported when the flip-flops capturing data at rising edge of the rtz clock, which is coming through a negative CGC, may fail to capture data.

Consequences of Not Fixing

Design may not function properly, as the affected flip-flops may fail to perform as desired.

How to Debug and Fix

Open the Incremental Schematic in and analyze the connectivity between gating cell and flip-flops.

To fix the violation, modify the RTL.

Message 2

'rto' clock, through a positive clock gating cell <cgc-name>, is driving <no-of-flip-flops> flip-flop(s) at its falling edge in <mode>

Arguments

- name of the clock gating cell, <cgc-name>
- number of flip-flops which are edge inconsistent, <no-of-flip-flops>
- shift or Capture mode, <mode >

Potential Issues

A violation is reported when the flip-flops capturing data at falling edge of

the `rtz` clock, which is coming through a positive CGC, may fail to capture data.

Consequences of Not Fixing

Design may not function properly, as the affected flip-flops may fail to perform as desired.

How to Debug and Fix

Open the Incremental Schematic and analyze the connectivity between gating cell and flip-flops.

To fix the violation, modify the RTL.

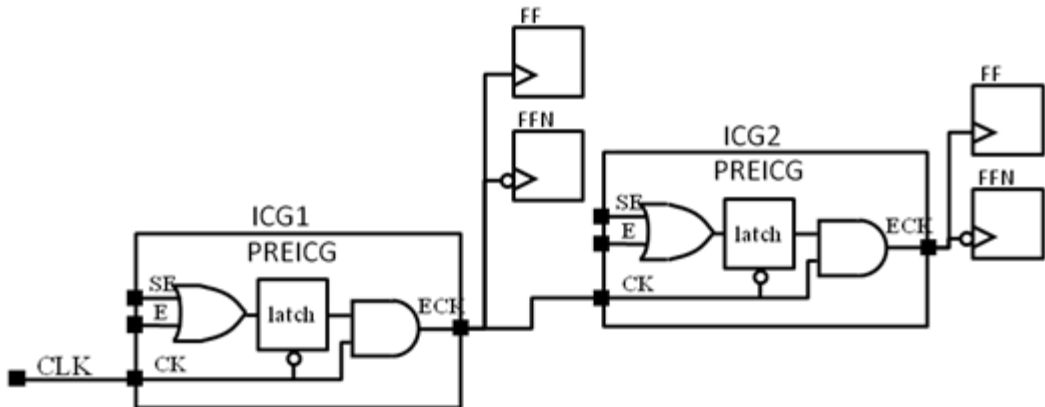
Example Code and/or Schematic

The following examples explain various scenarios under which the `Clock_27` rule reports a violation.

NOTE: *It is assumed that the examples described below use `rtz` clock.*

Example 1

Consider the circuit shown in the following figure:



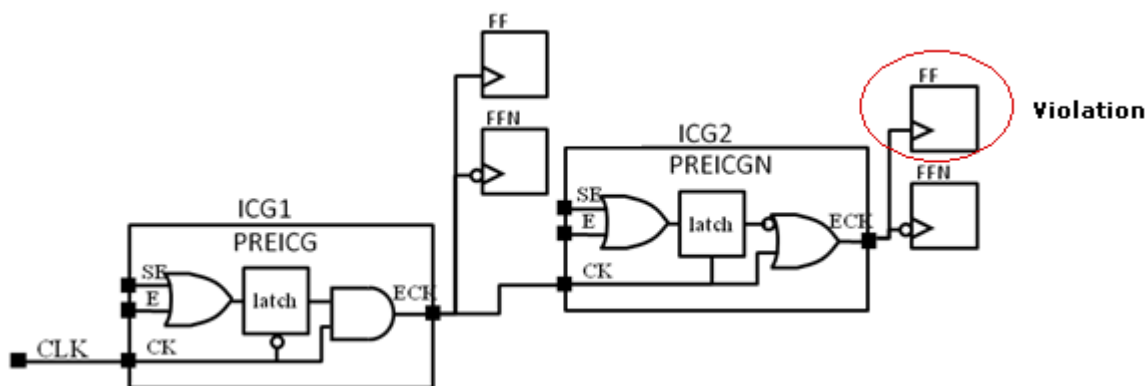
The above circuit contains two PREICG cells (ICG1 and ICG2) in series. The following table shows the simulation values when `CLK = 1` and `CLK=0`:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	0	0
1	1	1	1

In this example, as the values of ICG1.ECK and ICG2.ECK are deterministic, both can drive flip-flops triggered on either the positive or the negative edge. Therefore, the *Clock_27* rule does not report any violation in this case.

Example 2

Consider the circuit shown in the following figure:



In the above circuit, the PREICG cell drives the PREICGN cell.

This rule reports a violation in this case because of the following reasons:

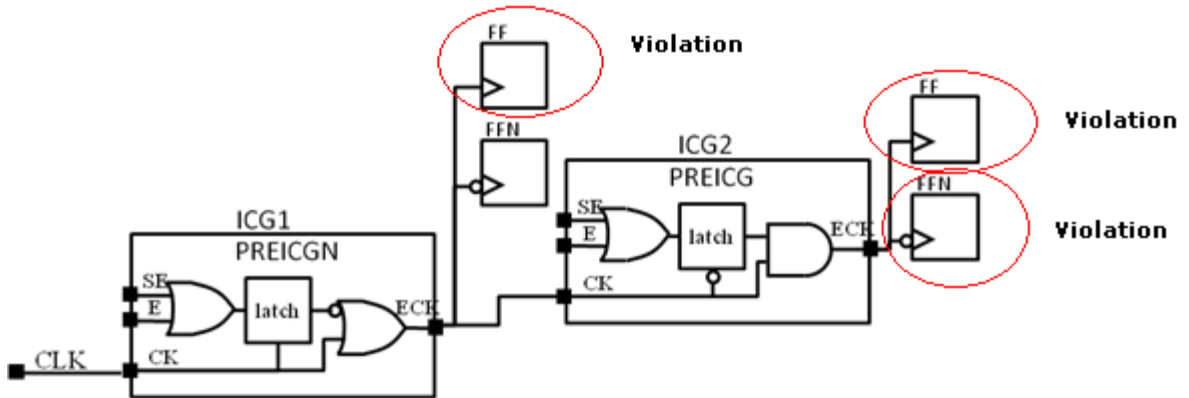
- When CLK=0 and SE=1, ICG1.ECK=0, ICG1 latch is initialized to 1, and ICG2.ECK=X as the initial state of ICG2 latch is X.
- When CLK=1 and SE=1, ICG1.ECK and ICG2.ECK is 1. Therefore, ICG1.ECK can drive both positive and negative edge flip-flops. However, ICG2.ECK can drive only negative edge flip-flops.

The following table shows the simulation values when CLK = 1 and CLK=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	0	X
1	1	1	1

Example 3

Consider the circuit shown in the following figure:



For the above example, this rule reports violation because ICG2 .ECK cannot drive any positive edge or negative edge flip-flop. The following table shows the simulation values when CLK = 1 and CLK=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	X	X
1	1	1	X

Example 4

Consider the following figure illustrating edge inconsistency between CGC and driven flip-flops:

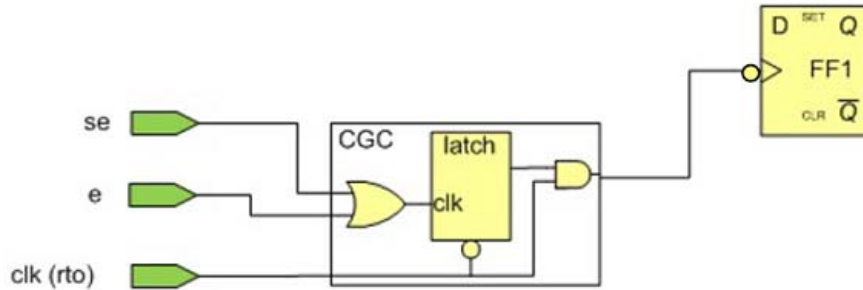


FIGURE 54. Simple CGC connection

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

No related reports or files

Clock_28

Reports the presence of combinational re-convergence to flip-flop clock pin

When to Use

Use this rule to avoid glitches on clock tree.

Description

The Clock_28 rule reports violation for re-converging combinational paths that start from a primary input port or the output pin of a flip-flop and re-converge at the clock pin of any flip-flop.

Rule Exceptions

The Clock_28 rule does not report violation for the no scan flip-flops (forced or inferred).

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_report_all_paths_between_reconvergence_start_and_end*: The default value of this parameter is off. Set the value of the parameter to on to report all paths between re-convergence start and end points.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[INFO] Reconvergence report file '<report-file-name>' is generated

Arguments

Name of the report file, *<report-file-name>*

Potential Issues

The violation message appears when the text report file is generated.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[WARNING] [Clock/Clock Enable] Logic '<start_point>' reconverges at or near '<end_point>'. (Actual Reconvergence start = '<rtl_start_point>' reconvergence end = '<rt_end_point>') [Affects '<no_of_clock_pins>' flip-flops(s)]

Potential Issues

The violation message appears if wrong connectivity on the clock tree exists.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and glitches on the clock tree.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the entire reconvergent combinational paths.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Clock_28 rule in the Incremental Schematic window. To do this, double-click the violation for the Clock_28 rule and open the Incremental Schematic window.

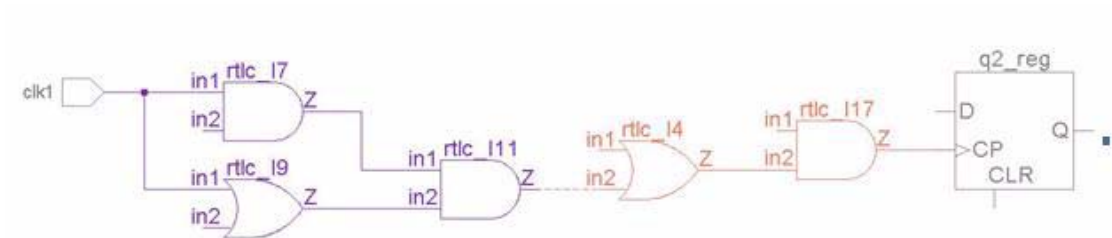
The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Clock_28 rule in the Incremental Schematic window. This is useful in debugging the violation for the Clock_28 rule.

To fix the violation, correct the connection on the clock tree.

Example Code and/or Schematic

Example 1

Consider the following figure:



The Clock_28 rule reports a violation for this case because signal from `clk1` re-converges at the output of the AND gate, `rtlc_111`, which is driving clock pin of flip-flop, `q2_reg`.

Example 2

The following figure illustrates the presence of combinational re-convergence to a flip-flop clock pin:

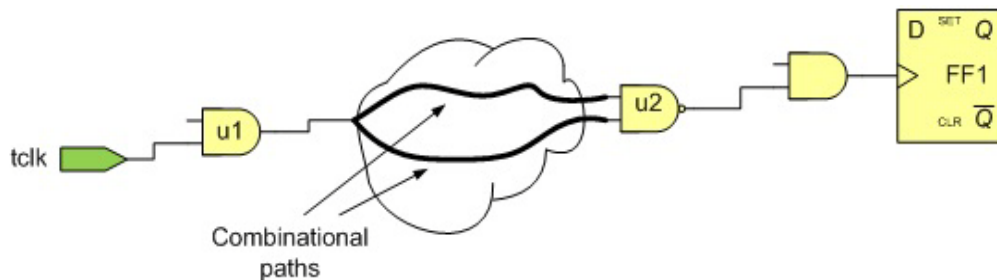


FIGURE 55. Clock pin fed by re-convergence clock

Re-convergence in a clock path can alter the clock pulse shape as well as produce glitches that can affect capture results. In either case, fault coverage can be compromised and test results changed.

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

ff_clock_reconvergence: The Clock_28 rule generates this report in the `<current-working-directory>/spyglass_reports/df/` directory. This report lists all re-convergent combinational paths that start from a primary input port or the output pin of a flip-flop and re-converge at the clock pin of a flip-flop.

Clock_29

Ensure that all clock sources of memories/hard macros are testclock controlled in shift/capture mode

When to Use

Use this rule to detect clock pins of memories and hard macros that are not testclock controlled in the shift or capture mode.

Description

The Clock_29 rule reports violation for clock pins of memories or macros that are not testclock-controlled in the shift or capture mode.

Rule Exception

The Clock_29 rule ignores no scan memories and hard macros.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_maximum_number_of_rows_with_schematic](#): Specifies the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (mandatory): Use this constraint to define the clocks of a design.
- [memory_type](#) (optional): Specifies the memory design unit (black box) names.
- [force_no_scan](#) (optional): Use this constraint to exclude black boxes, latches, and flip-flops, from being declared scannable even if they so qualify.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] Clock source <clk_src_name> does not get testclock in <mode_name>. Count of affected <memories/macros>: '<num>']

Arguments

- Name of the clock source, <clk_src_name>
- mode name, <mode_name>
- Number of affected memories/macros, <num>

Potential Issues

A violation is reported because memory/macro clock source does not get a testclock.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Clock source '<clk_src_name>' does not get testclock in '<mode_name>'. Topological clock '<clock_name>' found in the fanin cone which got blocked. [Count of affected <memories/macros>: '<num>']

Arguments

- Name of the clock source, <clk_src_name>
- mode name, <mode_name>
- Name of the topological clock present in the fanin cone of memory/macro, <clock_name>
- Number of affected memories/macros, <num>

Potential Issues

A violation is reported because memory/macro clock source does not get testclock.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

```
[WARNING]Clock source '<clk_src_name>' does not get testclock in '<mode_name>'. Topological clock '<clock_name>' found in the fanin cone which got blocked. '<count>' other potential clock(s): <clk_name_list>. [Count of affected <memories/macros>: '<num>']
```

Arguments

- Name of the clock source, *<clk_src_name>*
- mode name, *<mode_name>*
- Name of the topological clock present in the fanin cone of memory/macro, *<clock_name>*
- Count of potential clocks, *<count>*
- Name of the topological clocks present in the fanin cone which got blocked, *<clk_name_list>*
- Number of affected memories/macros, *<num>*

Potential Issues

A violation is reported because memory/macro clock source does not get testclock.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage.

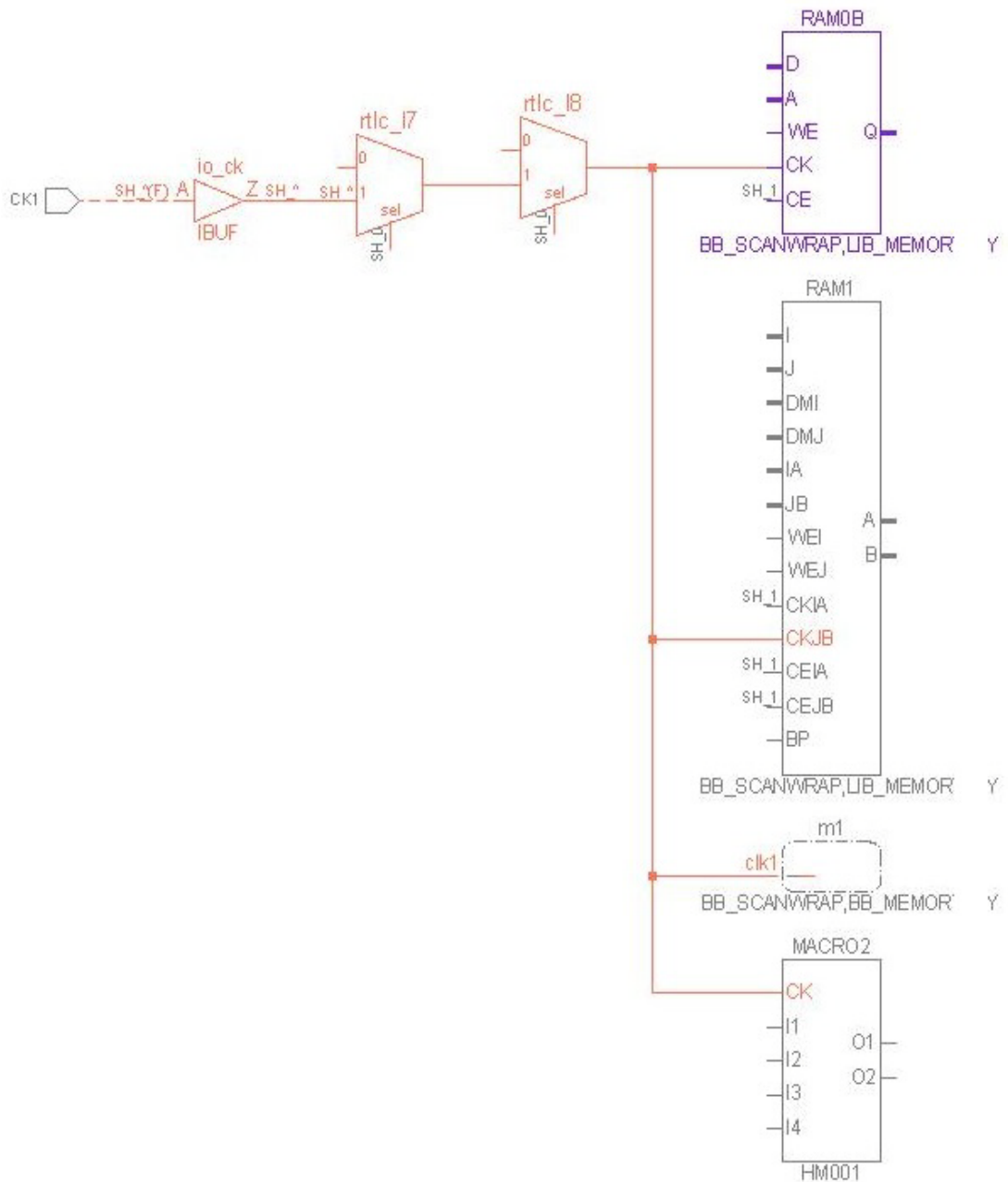
How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window shows the memory/macro on which the clock does not reach. Ensure that the complete display mode is set. The violation message also provides information on identification of potential clocks.

Example Code and/or Schematic

In the following figure, the Clock_29 rule reports violation as driving clock pin of memory is not getting testclock because value provided on other input (A) of and gate is zero:

Clock Rules



For the above design, the Clock_29 rule reports the following violation message:

```
Clock source 'TEST.and_0_o' does not get testclock in 'Shift mode'. Topological clock 'CK1' found in the fanin cone which got blocked. [Count of affected memories/macros: '4']
```

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

None

Clock_30

All clock pins in a memory / hard macro should get the same root level clock

When to Use

Use the rule to check if memory and hard macro with more than one clock pin should only operate on the same root-level clock.

Description

The Clock_30 rule reports violation, if all the clock pins of a memory / hard macro do not get the same root level clock.

Rule Exceptions

The *Clock_30* rule does not report violation for the following cases:

- If a memory or a hard-macro clock-pin does not get a test clock, it is skipped by the rule for checking.
- no scan memory and hard macro.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design. The Clock_30 rule uses the `-testclock` argument of the `clock` constraint.
- *memory_type* (optional): Specifies the memory design unit (black box) names.

- *force_no_scan* (optional): Use this constraint to exclude black boxes, latches, and flip-flops, from being declared scannable even if they so qualify.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

<no_of_ckl_pins> pins of <inst_type> '<inst_type_name>' gets <testclk_count> clocks in <mode_name>(<string>)

Arguments

- Number of clock pins on memory or hard-macro, which are getting the test clock, <no_of_ckl_pins>
- Whether an instance is a memory or a hard-macro, <inst_type>
- Name of the memory or hard-macro instance, <inst_type_name>
- Number of test clocks that are driving the memory or hard-macro, <testclk_count>
- Mode name, <mode_name>
- String containing information on root test clocks and clock pins, <string>

Potential Issues

Violation message is reported because of an incorrect clock connection or when the recommended methodology is not followed.

Consequences of Not Fixing

Not fixing the violation makes memory or the hard-marco difficult to manage.

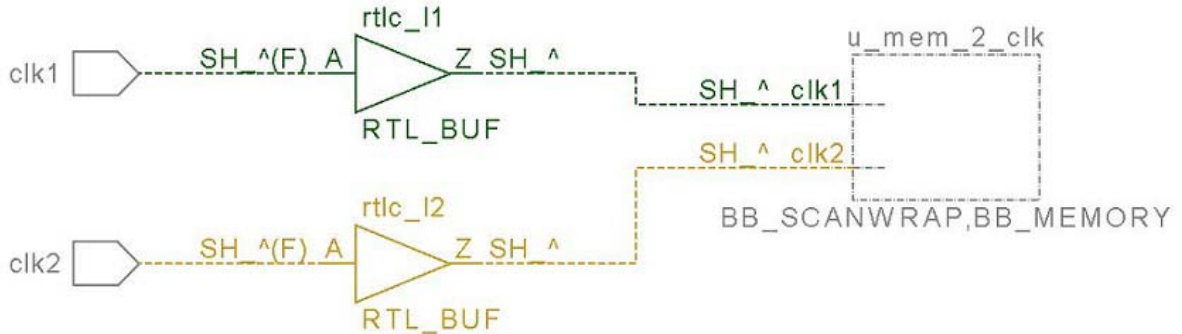
How to Debug and Fix

Double-click the violation message to open the Incremental Schematic to debug.

To fix the violation, correct the clock connection.

Example Code and/or Schematic

Consider the following figure, where the memory instance, `u_mem_2_clk`, is receiving inputs from two clocks, `clk1` and `clk2`:



The Clock_30 rule reports following violation message for the above design:

```
2 pins of memory 'top.u_mem_2_clk' gets 2 clocks in Shift
mode(clk1->top.u_mem_2_clk.clk1, clk2-
>top.u_mem_2_clk.clk2), test.v, 23
```

Default Severity Label

Warning

Rule Group

Clock

Reports and Related Files

None

Clock Gating Rules

Overview

The Clock Gating rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>CG_01_atspeed</i>	Clock gating cell enables should be controllable to on state in atspeed mode
<i>CG_01_capture</i>	Clock gating cell enables should be controllable to on state in capture mode
<i>CG_01_shift</i>	Clock gating cell enables should be enabled in shift mode
<i>CG_02_atspeed</i>	CGC enables should be controllable to off state in atspeed mode
<i>CG_02_capture</i>	CGC enables should be controllable to off state in capture mode
<i>CG_03_atspeed</i>	System enable pins on clock gating cell should be observable in atspeed mode
<i>CG_03_capture</i>	System enable pins on clock gating cell should be observable in capture mode
<i>CG_04</i>	Test enable pins of clock gating cells should be driven by the specified gatingcell_enable ports
<i>CG_05</i>	Test enable pins of clock gating cells in the same level should be driven by the same test enable ports
<i>CG_06</i>	Reports flip-flops in the fan-in cone of a CGC, if they capture a wrong value.
<i>CG_07</i>	Detects edge inconsistency between CGC and driven flip-flops
<i>CG_08</i>	The system enable pin of a CGC, which is driving the clock pin of a flip-flop, should not be driven by the same flip-flop.

Clock Gating Rules

Rule	Description
<i>CG_consistency</i>	Behavior model of clock gating cell should be consistent to constraint applied on it
<i>CG_generateReport</i>	Generate a text report with details of all clock gating cells in design

CG_01_atspeed

Ensure that the clock gating cell enables are controllable to on state in atspeed mode

When to Use

Use this rule to identify clock gating cells that can not be enabled in atspeed mode.

Description

The `CG_01_atspeed` rule checks that the clock gating cells in the design is enabled in atspeed mode, that is, at least one enable pin is controllable to its active value.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_01_atspeed** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_01_atspeed** does not report any violations for that CGC.*

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.

Constraint(s)

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

The following high-priority violation message is displayed when atspeed clock does not reach the CGC:

```
[WARNING] No clock reached at clock gating cell  
<hierarchical_path_name> in <mode-name>
```

Potential Issues

Violation may arise when either the atspeed clock specification is missing or the atspeed mode is incomplete such that the atspeed clock is not able to get to this CGC.

Consequences of Not Fixing

Not fixing the violation results in low transition coverage as the flip-flops driven by this CGC do not participate in the launch and capture of atspeed test.

How to Debug and Fix

For more information on debugging, click [How to Debug and Fix](#).

To fix the violation, fix either the atspeed clock or the atspeed mode condition.

Message 2

The following violation message is displayed when the clock reaches the CGC, but is not controllable.

[WARNING] Clock gating cell <hierarchical_path_name> is not controllable to 'on' for <mode-name>

Potential Issues

Violation may arise when the controllability of the enable pin is not good. This can happen due to one of the violations reported by the Atspeed_09 rule.

Consequences of Not Fixing

Not fixing the violation allows the CGC to block the atspeed clock propagation. This results in low transition coverage as the CGCs will not participate in launch and capture.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose enable pins are not controllable to active value under capture atspeed mode. The Incremental Schematic also displays the expected and actual controllability value under capture atspeed mode at enable pins.

As of now there is no direct mechanism to debug this violation.

To fix the violation, ensure that the clock gating cell enables are controllable to on state in atspeed mode.

Arguments

- Hierarchical path name of the clock gating cell, <hierarchical_path_name>
- Capture_atspeed, Capture, Shift mode <mode-name>. The value of this argument for the CG_01_atspeed rule is capture_atspeed.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

ClockGating

Clock Gating Rules

Reports and Related Files

No related reports or files.

CG_01_capture

Ensure that the clock gating cell enables are controllable to on state in capture mode

When to Use

Use this rule to identify CGCs, which ca[/'not be enabled in capture mode.

Description

The CG_01_capture rule checks that the clock gating cells in the design can be enabled in capture mode, that is, at least one enable pin is controllable to its active value.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_01_capture** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_01_capture** does not report any violations for that CGC.*

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- [clock](#) (optional): Use this constraint to declare clock pins declared as testclocks.

- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the CG_01_capture rule:

[WARNING] Clock gating cell <hierarchical_path_name> is not controllable to 'on' for <mode-name>

Arguments

- Hierarchical path name of the clock gating cell
<hierarchical_path_name>
- Capture_atspeed, Capture, Shift mode <mode-name>. The value for this parameter for the CG_01_capture rule is capture.

Potential Issues

Violation may arise when the controllability of the enable pin is not good. This can happen due to one of the violations reported by the Atspeed_09 rule.

Consequences of Not Fixing

Not fixing the violation allows the CGCs to block the capture clock propagation. This results in reduced test coverage as the flip-flops driven by such CGCs do not participate in capture.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose enable pins are not controllable to active value under capture mode. The Incremental Schematic also displays the expected and actual controllability values under capture mode at enable pins.

Overlay (Auxiliary violation mode) the Info_uncontrollable rule. This helps in finding out the actual controllability propagation.

To fix the violation, improve controllability of enable pin to its active value in atspeed.

Example Code and/or Schematic

Currently Unavailable

Clock Gating Rules

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files.

CG_01_shift

Ensure that the clock gating cells are enabled in shift mode

When to Use

Use this rule to identify clock gating cell, which is not enabled in shift mode.

Description

The `CG_01_shift` rule checks that the clock gating cells in the design are enabled in the shift mode, that is, at least one enable pin is held active.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_01_shift** rule does not report any violations for that CGC. If a CGC drives a port and the `dft_treat_primary_port_as_valid_clock_point_for_cgc_check` parameter is set to off, then the **CG_01_shift** does not report any violations for that CGC.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dft_treat_primary_port_as_valid_clock_point_for_cgc_check`: Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- `dft_enable_checks_for_all_cgc`: Default value is off. Set the value of the parameter to enable rule checking.

Constraints

- `clock` (optional): Use this constraint to declare clock pins declared as testclocks.
- `gating_cell` (optional): Use this constraint to specify the user-defined clock gating cell.

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Shift

Messages and Suggested Fix

The following violation message is displayed for CG_01_shift rule:

[WARNING] Clock gating cell <hierarchical_path_name> is not enabled in <mode-name>

Arguments

- Hierarchical path name of the clock gating cell
<hierarchical_path_name>
- Capture_atspeed, Capture, Shift mode *<mode-name>*. The value for this parameter for the CG_01_shift rule is *shift*.

Potential Issues

Violation may arise when the simulation value of enable pin is not good. This is due to one of the Atspeed_09 violation or improper test_mode constraint.

Consequences of Not Fixing

Not fixing the violation allows the CGCs to block the shift clock propagation. This results in reduced controllability and test coverage as the flip-flops driven by such CGCs do not participate in shift.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose enable pins are not set to active value under shift mode. The Incremental Schematic also displays the expected values and the actual simulation value under shift mode at enable pins.

Overlay (Auxiliary violation mode) the Info_testmode rule under shift_mode. This helps in finding out how the shift_mode simulation has propagated.

To fix the violation, ensure that active value reaches the enable pin of the

CGC in shift.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files.

CG_02_atspeed

Ensure that the CGC enables are controllable to off state in atspeed mode

When to Use

Use this rule to identify clock gating cell, which cannot be disabled in the atspeed mode. You can also identify enable pins which cannot be controlled to inactive value.

Description

The `CG_02_atspeed` rule checks that the clock gating cells in the design can be disabled in the atspeed mode, that is, all enable pins are controllable to their inactive values.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the `CG_02_atspeed` rule does not report any violations for that CGC. If a CGC drives a port and the `dft_treat_primary_port_as_valid_clock_point_for_cgc_check` parameter is set to off, then the `CG_02_atspeed` does not report any violations for that CGC.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dft_treat_primary_port_as_valid_clock_point_for_cgc_check`: Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- `dft_enable_checks_for_all_cgc`: Default value is off. Set the value of the parameter to enable rule checking.

Constraints

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the CG_02_atspeed rule:

```
[WARNING] Clock gating cell <hierarchical_path_name> is not controllable to 'off' for <mode-name>
```

Arguments

- Hierarchical path name of the clock gating cell
<hierarchical_path_name>
- Capture_atspeed or Capture mode *<mode-name>*. The value for this parameter for the CG_02_atspeed rule is capture_atspeed.

Potential Issues

Violation may arise due to one of the following reasons:

- Controllability of enable pin is not good due to one of theAtspeed_09 violations.
- One of the enable pin is permanently tied to active value for allowing clock to pass through CGC

Consequences of Not Fixing

Not fixing the violation reduces the testability of the enable logic as the logic becomes unobservable.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose enable pins are not

Clock Gating Rules

controllable to active value under capture atspeed mode. The Incremental Schematic also displays the expected and actual controllability value under capture atspeed mode at enable pins.

As of now there is no direct mechanism to debug this violation.

To fix the violation, put observation point or improve controllability.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files.

CG_02_capture

Ensure that the CGC enables are controllable to off state in capture mode

When to Use

Use this rule to identify clock gating cell, which cannot be disabled in the capture mode. The rule also helps you to identify enable pins, which cannot be controlled to inactive value.

Description

The `CG_02_capture` rule checks that the clock gating cells in the design can be disabled in the capture mode, that is, all enable pins are controllable to their inactive values.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_02_capture** rule does not report any violations for that CGC. If a CGC drives a port and the `dft_treat_primary_port_as_valid_clock_point_for_cgc_check` parameter is set to off, then the **CG_02_capture** does not report any violations for that CGC.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dft_treat_primary_port_as_valid_clock_point_for_cgc_check`: Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- `dft_enable_checks_for_all_cgc`: Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the CG_02_capture rule:

[WARNING] Clock gating cell <hierarchical_path_name> is not controllable to 'off' for <mode-name>

Arguments

- Hierarchical path name of the clock gating cell
<hierarchical_path_name>
- Capture_atspeed or Capture mode *<mode-name>*. The value for this parameter for the CG_02_capture rule is capture.

Potential Issues

Violation may arise due to one of the following reasons:

- Controllability of enable pin is not good due to one of theAtspeed_09 violations.
- One of the enable pin is permanently tied to active value for allowing clock to pass through CGC

Consequences of Not Fixing

Not fixing the violation reduces the testability of the enable logic as the logic becomes unobservable.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose enable pins are not controllable to non active value under capture mode. The Incremental Schematic also displays the expected and actual controllability value reached under capture mode at enable pins.

Overlay (Auxiliary violation mode) the Info_uncontrollable rule. This helps in finding out the actual controllability propagation.

To fix the violation message, put the observation point or improve controllability.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files.

CG_03_atspeed

Ensure that the system enable pins on clock gating cell are observable in atspeed mode

When to Use

Use this rule to identify clock gating cell, whose system enables are not observable in atspeed mode.

Description

The *CG_03_atspeed* rule checks that the system enable pins of all clock gating cells in the design are observable in the atspeed mode.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_03_atspeed** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_03_atspeed** does not report any violations for that CGC.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- [clock](#) (optional): Use this constraint to declare clock pins declared as testclocks.

- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the CG_03_atspeed rule:

```
[WARNING] System enable pin of clock gating cell  
<hierarchical_path_name> is not observable in <mode-name>
```

Arguments

- Hierarchical path name of the clock gating cell
<hierarchical_path_name>
- Capture_atspeed or Capture mode *<mode-name>*. The value for this parameter for the CG_03_atspeed rule is capture_atspeed.

Potential Issues

Violation may arise when the test enable pin is tied to active value during atspeed.

Consequences of Not Fixing

Logic driving the system enable pin is not observable. Therefore, faults on this logic are not detectable.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose system enable pin is not observable under atspeed mode. The Incremental Schematic also displays the expected and actual observability under atspeed mode at system enable pin.

As of now there is no direct mechanism to debug this violation.

To fix the violation, add observation point to the CGC and control test enable pin to the off state.

Clock Gating Rules

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files.

CG_03_capture

Ensure that the system enable pins on clock gating cell are observable in capture mode

When to Use

Use this rule to identify clock gating cell, whose system enables are not observable in capture mode.

Rule Description

The `CG_03_capture` rule checks that system enable pins of all clock gating cells in the design are observable in the capture mode.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_03_capture** rule does not report any violations for that CGC. If a CGC drives a port and the `dft_treat_primary_port_as_valid_clock_point_for_cgc_check` parameter is set to off, then the **CG_03_capture** does not report any violations for that CGC.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dft_treat_primary_port_as_valid_clock_point_for_cgc_check`: Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- `dft_enable_checks_for_all_cgc`: Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- `clock` (optional): Use this constraint to declare clock pins declared as testclocks.

- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the CG_03_capture rule:

```
[WARNING] System enable pin of clock gating cell
<hierarchical_path_name> is not observable in <mode-name>
```

Arguments

- Hierarchical path name of the clock gating cell
<hierarchical_path_name>
- Capture_atspeed or Capture mode *<mode-name>*. The value for this parameter for the CG_03_capture rule is capture.

Potential Issues

Violation may arise when the test enable pin is tied to an active value during capture.

Consequences of Not Fixing

Not fixing the violation makes the logic driving the system enable pin unobservable. Therefore, faults arising in this logic become undetectable.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the clock gating cell whose system enable pin is not observable under capture mode. The Incremental Schematic also displays the expected and actual observability value under capture mode at system enable pin.

Overlay (Auxiliary violation mode) the Info_uncontrollable and Info_unobservable rule. This helps in identify the cause of bad controllability/observability.

To fix the violation, add observation point to the CGC and control test enable pin to the off state.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files.

CG_04

Test enable pins of clock gating cells should be driven by the specified gatingcell_enable ports

Rule Description

The CG_04 rule checks that the test enable pins of all clock gating cells in the design are driven by the test enable ports specified by the gatingcell_enable constraint.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_04** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_04** does not report any violations for that CGC.*

Constraints

[clock](#) -testclock -atspeed (optional)

[gating_cell](#) (optional)

[gating_cell_enable](#) (mandatory)

[test_mode](#) (optional)

Rule Parameters

- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is off. Set the value of the parameter to on so that all clocks, except the propagated clock, are set to their off state during clock propagation.
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.

Operating Mode

[Scanshift](#)

Message Details

Message

- The following violation message appears when no *gating_cell_enable* sgdc command is found in the design:

Skipping the rule check on design '*<top_du_name>*' as corresponding '*gating_cell_enable*' sgdc command is not found

- The following violation message appears when a topological path exist between the test enable signal and the TE pin of the CGC, but a functional path under the shift mode is blocked:

Test enable term *<pin_name>* of CG cell *<CGC_name>* is not driven by test enable port *<signal_name>* (TE signal stops at *<instance_name>* [*<cause>*])

- The following violation message appears when no topological path exists between the test enable signal and the TE pin of the CGC, but a unique path exists between the input port to the TE pin of the CGC:

Test enable term *<pin_name>* of CG cell *<CGC_name>* is not driven by any test enable port (TE signal source is *<port_name>*)

- The following violation message appears when no topological and functional path exist between the TE pin of the CGC and the test signal pin.

Test enable term *<pin_name>* of CG cell *<CGC_name>* is not driven by any test enable port (TE signal stops at *<instance_name>*)

Arguments

1. Name of the top design unit, *<top_du_name>*
1. Name of the test enable pin, *<pin_name>*
2. Name of the CGC cell, *<CGC_name>*
3. Test enable signal specified by gating cell enable constraint, *<signal_name>*
4. Blocked instance nearest to TE pin of CGC, *<instance_name>*
5. Cause of blocking, *<cause>*
6. Name of the port, *<port_name>*

Location

File / Line where clock gating is instantiated in the design

Schematic highlight

Following are the schematic highlights for the CG_04 rule:

- CGC TE pin, which is not driven by the TE port.
- Path to the nearest TE port, if present, and instance where it is blocked.

Rule Severity

Warning

CG_05

Test enable pins of clock gating cells in the same level should be driven by the same test enable ports

Rule Description

The CG_05 rule checks that the test enable pins of all clock gating cells at the same level, that is, having the same parent CGC are driven by the same test enable ports, and different test enable ports from that of the parent CGC.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_05** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_05** does not report any violations for that CGC.*

Constraints

- [clock](#) -testclock -atspeed (optional)
- [gating_cell](#) (optional)
- [gating_cell_enable](#) (mandatory)
- [test_mode](#) (optional)

Rule Parameters

- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is off. Set the value of the parameter to on so that all clocks, except the propagated clock, are set to their off state during clock propagation.
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.

Operating Mode

Scanshift

Message Details

Message 1

Skipping the rule check on design '<top_du_name>' as corresponding 'gating_cell_enable' sgdc command is not found

Message 2

Test enable term '<pin-name>' of CG cell '<cell-name>' (driven by '<signal_name>') and test enable term '<pin-name>' of CG cell '<cell-name>' (driven by '<signal-name>') are at the same level but driven by different TE ports

Message 3

Test enable term '<pin-name>' of CG cell '<cell-name>' is driven by the same TE port ('<signal-name>') as the test enable term '<pin-name>' of its parent CG cell '<cell-name>'

Arguments

- Name of the test enable pin, <pin_name>
- Name of test enable signal, <signal_name>
- Name of the CG cell, <cell-name>

Location

File / Line where clock gating cell is instantiated in the design

Schematic highlight

CGC test enable pins which do / don't match.

Example

Currently Unavailable

Rule Severity

Warning

CG_06

Flip-flops in the fan-in cone of a CGC may adversely affect coverage if they capture wrong value

When to Use

Use this rule to identify flip-flops in the fan-in cone of a CGC, if they capture a wrong value.

Description

The CG_06 rule reports a violation for the flip-flops in the fan-in cone of a CGC, if they capture wrong value.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_06** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_06** does not report any violations for that CGC.*

Default Weight

10

Language

Verilog, VHDL

Method

Mark all CE of CGCs as defined with the "gating_cell -name <cell-name> -enTerm <enable-pin>" constraint. These are start points for backtracing.

Trace back from the start points and stop at flip-flops, PI and BBs.

For each scannable flip-flop reached,

If dsmReportAllCGCCEFaninFlops is On, report the flip-flop in CG_06

Else If dsmReportAllCGCCEFaninFlops is Off, report the flip-flop in CG_06 only if its testclock drives more than one clock domain (as per Clock Rule 10)

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dsmReportAllCGCCEFaninFlops](#): The default value is on. Therefore, the rule reports all flip-flops hit in the fan-in cone of the CGC Enable pins. Set

the value of the parameter to off to report only those flip-flops whose testclock domain drives multiple functional domain.

- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- [clock](#) (optional): Declares the clock pins used as testclocks.
- [gating_cell](#) (mandatory): Specifies the user-defined clock gating cell.

Messages and Suggested Fix

The following violation messages are displayed for the CG_06 rule.

Message 1

[INFO] In top-module <top>, CGC <cgc> has <count> flops in fanin of enable term <en-term>. Flops are : <flip-flop-list>

Arguments

For more information on the list of arguments, click [Arguments](#).

Potential Issues

A violation is reported because of unknown values captured at the CGC enable pin.

Consequences of Not Fixing

Flip-flops for which the test clock drives more than one clock domain (as per Clock Rule 10) are at higher risk of capturing unknown values due to timing violations in test mode. If such flip-flops are in the fan-in cone of the Clock Enable of a Clock Gating Cell, then the unknown value can propagate to all the flip-flops in the fan-out of the clock gating cell, potentially affecting transition fault coverage in a more substantial way than flip-flops that are not in the fan-in of such a clock enable.

How to Debug and Fix

To debug the violation, for each flip-flop in the report, highlight the path between the flip-flop and the clock enable pin of CGCs in its fan-out.

To fix the violation, remove the flip-flops that are in the fan-in cone of the

CGC.

Message 2

[INFO] In top-module <top>, CGC '%1' has <count> flops in fanin of en term <en-term>. Atspeed clock of flop drives multiple functional domains. Flops are : <flip-flop-list>

Arguments

- Root module, <top>
- Number of flip-flops, <count>
- CGC enable term, <en-term>
- Name of the Clock Gating Cell, <CGC>
- List of all flip-flop names, <flip-flop-list>

Potential Issues

A violation is reported because of unknown values captured at the CGC enable pin.

Consequences of Not Fixing

Flip-flops for which the test clock drives more than one clock domain (as per Clock_10 rule violation) are at higher risk of capturing unknown values due to timing violations in test mode. If such flip-flops are in the fan-in cone of the Clock Enable of a Clock Gating Cell, then the unknown value can propagate to all the flip-flops in the fan-out of the clock gating cell, potentially affecting transition fault coverage in a more substantial way than flip-flops that are not in the fan-in of such a clock enable.

How to Debug and Fix

To debug the violation, for each flip-flop in the report, highlight the path between the flip-flop and the clock enable pin of CGCs in its fan-out.

To fix the violation, remove the flip-flops that are in the fan-in cone of the CGC.

Example Code and/or Schematic

Currently Unavailable

Schematic highlight

- CGC and the enable pin
- All flip-flops in the fan-in cone of the enable pin

- The path (nets only) between the flip-flops and the CGC enable pin

Default Severity Label

Info

Rule Group

Info

Reports and Related Files

The CGC(s) and flip-flop(s) are also reported in a CSV file - dft_dsm/
CG_06/CG_06_<top-module>.csv

CG_07

Detects edge inconsistency between CGC and driven flip-flops

When to Use

This rule is used to check for edge inconsistency between clock gating cells and the flip-flops driven by it.

NOTE: *If the flip-flops or latches driven by the CGC are in no scan, then the **CG_07** rule does not report any violations for that CGC. If a CGC drives a port and the [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#) parameter is set to off, then the **CG_07** does not report any violations for that CGC.*

Description

The CG_07 rule reports a violation when an `rtz` clock through negative CGC is driving flip-flops at its rising edge or an `rto` clock through positive CGC is driving flip-flops on its falling edge.

For information on `rtz` and `rto` clocks, refer to *Test clocks* section in *SpyGlass DFT ADV Rules Reference Guide*.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

Prerequisite

This rule will run if there are clock gating cells present in the design.

Rule Exception

If a CGC and driven flip-flops are edge consistent, then the CG_07 rule does not report any violation for that CGC.

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CGC.

- *dft_enable_checks_for_all_cgc*: Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- *clock* (mandatory): Declares the clock pins used as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.

Operating Mode

Scanshift, Capture (atspeed)

Messages and Suggested Fix

Message 1

'rtz' clock, through a negative clock gating cell <cgc-name>, is driving <no-of-flip-flops> flip-flop(s) at its rising edge in <mode>.

Arguments

- Name of the clock gating cell, <cgc-name>
- Number of flip-flops which are edge inconsistent, <no-of-flip-flops>
- Shift or Capture mode, <mode >

Potential Issues

A violation is reported when the flip-flops capturing data at rising edge of the rtz clock, which is coming through a negative CGC, may fail to capture data.

Consequences of Not Fixing

Design may not function properly, as the affected flip-flops may fail to perform as desired.

How to Debug and Fix

Open the Incremental Schematic in and analyze the connectivity between gating cell and flip-flops.

To fix the violation, modify the RTL.

Message 2

'rto' clock, through a positive clock gating cell <cgc-name>, is driving <no-of-flip-flops> flip-flop(s) at its falling edge in <mode>

Arguments

- Name of the clock gating cell, <cgc-name>
- Number of flip-flops which are edge inconsistent, <no-of-flip-flops>
- Shift or Capture mode, <mode >

Potential Issues

A violation is reported when the flip-flops capturing data at falling edge of the rto clock, which is coming through a positive CGC, may fail to capture data.

Consequences of Not Fixing

Design may not function properly, as the affected flip-flops may fail to perform as desired.

How to Debug and Fix

Open the Incremental Schematic and analyze the connectivity between gating cell and flip-flops.

To fix the violation, modify the RTL.

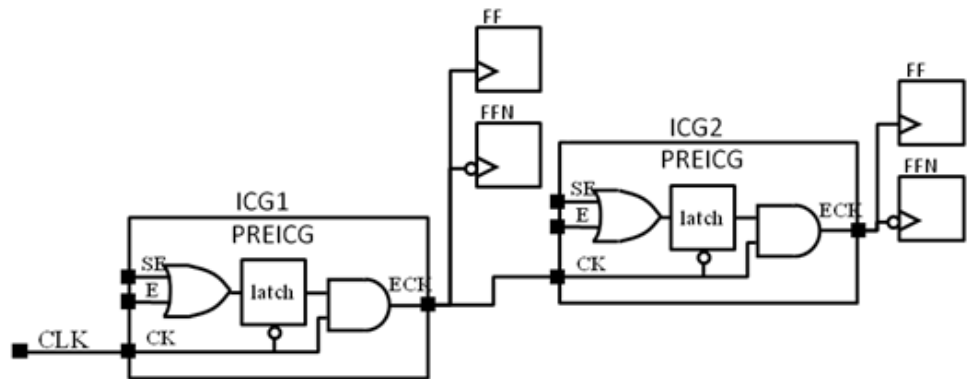
Example Code and/or Schematic

The following examples explain various scenarios under which the CG_07 rule reports a violation.

It is assumed that the examples described below use rtz clock.

Example 1

Consider the circuit shown in the following figure:



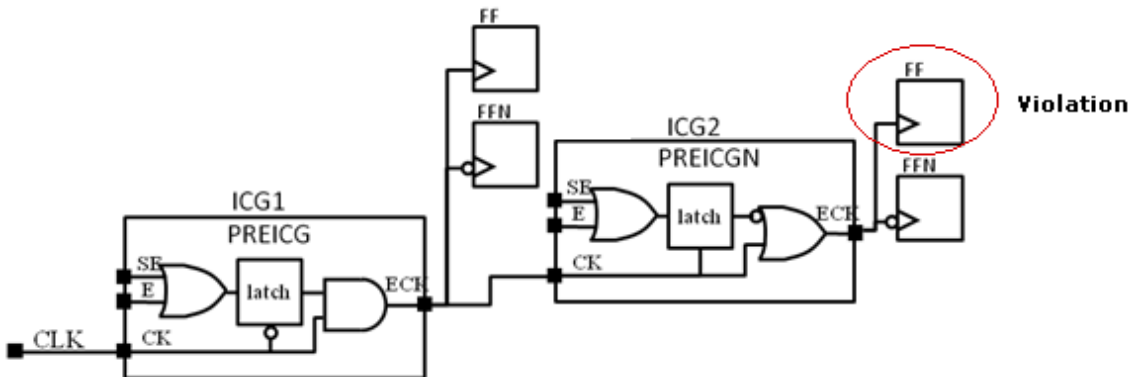
The above circuit contains two PREICG cells (ICG1 and ICG2) in series. The following table shows the simulation values when CLK = 1 and CLK=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	0	0
1	1	1	1

In this example, as the values of ICG1.ECK and ICG2.ECK are deterministic, both can drive flip-flops triggered on either the positive or the negative edge. Therefore, the *CG_07* rule does not report any violation in this case.

Example 2

Consider the circuit shown in the following figure:



In the above circuit, the PREICG cell drives the PREICGN cell.

This rule reports a violation in this case because of the following reasons:

- When $CLK=0$ and $SE=1$, $ICG1.ECK=0$, $ICG1$ latch is initialized to 1, and $ICG2.ECK=X$ as the initial state of $ICG2$ latch is X.
- When $CLK=1$ and $SE=1$, $ICG1.ECK$ and $ICG2.ECK$ is 1. Therefore, $ICG1.ECK$ can drive both positive and negative edge flip-flops. However, $ICG2.ECK$ can drive only negative edge flip-flops.

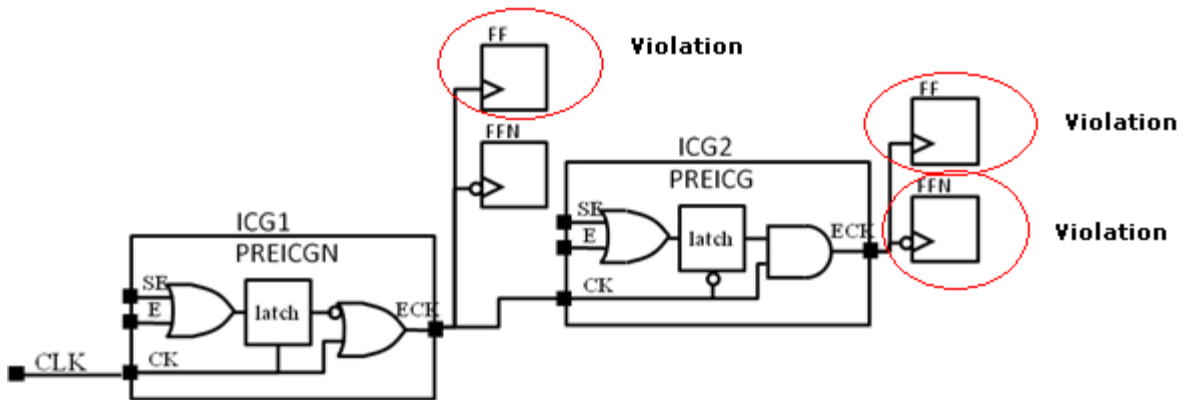
The following table shows the simulation values when $CLK = 1$ and $CLK=0$:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	0	X
1	1	1	1

Clock Gating Rules

Example 3

Consider the circuit shown in the following figure:



For the above example, this rule reports violation because ICG2 .ECK cannot drive any positive edge or negative edge flip-flop. The following table shows the simulation values when CLK = 1 and CLK=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	X	X
1	1	1	X

Default Severity Label

Error

Rule Group

ClockGating

Reports and Related Files

No related reports or files

CG_08

System enable pin of a CGC, which is driving the clock pin of a flip-flop, should not be driven by the same flip-flop.

When to Use

Use this rule to ensure that the system enable pin of a clock gating cell (CGC), which is driving the clock pin of a flip-flop, should not be driven by the same flip-flop.

If the value of the *dft_report_cgc_centric_deadlock* parameter is set to on, then a violation message is reported if no override is available from other controllable sources to break the self gating configuration.

Description

The system enable pin of a clock gating cell (CGC), which is driving the clock pin of a flip-flop, should not be driven by the same flip-flop as this may block the clock propagation.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dft_agnostic_cgc_flipflop_deadlock_check*: Default value is off. Set the value of the parameter to on to ignore the DFT related phrase during the CG_08 rule checking.
- *dft_report_cgc_centric_deadlock*: Default value is off. Set the value of the parameter to on to make the CGC_08 checks CGC centric and more accurate.

Constraint(s)

- *clock* (mandatory): Declares the clock pins used as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.

Operating Mode

Capture, Capture (atspeed)

Messages and

Suggested Fix

Message 1

[WARNING] Flip-flop '<flip-flop>' is driving system enable pin '<se_pin>' of CGC '<cgc_name>' which is driving the clock pin of the same flip-flop <no_cascaded_cgc> in '<operating_mode>'

Arguments

- Hierarchical name of the flip-flop, <hier_flip_flop>
- Hierarchical name of the clock gating cell (CGC), <cgc_name>
- Name of the system enable pin of the CGC, <se_pin>
- Number of cascaded CGCs, in case the number is greater than 0; Empty string otherwise, <no_cascaded_cgc>
- Operating mode name, <operating_mode>

Potential Issues

The violation message is displayed when a flip-flop is driving a system enable pin of CGC, which is driving the clock pin of the same flip-flop.

Consequences of Not Fixing

The specific structure may create a deadlock that may result when a clock gating cell A is disabled by a flip-flop that is clocked by CGC A, resulting in the cell being stuck in the disabled state.

How to Debug and Fix

To debug the violation, double-click the violation message to open the Incremental Schematic.

To fix the violation, do one of the following:

- Do not control the enable pin of a CGC through the flip-flop which is driven by the same CGC.
- Use the XOR logic between FF-D and FF-Q to determine the CGC-EN.

Message 2

[WARNING] Clock gating cell '<cgc_name>' has '<number_of_scan_cells>' scan cells in self gating configuration in <mode><spreadsheet_path>

Potential Issues

The violation message is displayed when a CGC is driving a clock pin of

scan cells, which in turn, are driving the enable pin of the CGC.

Consequences of Not Fixing

The specific structure may create a deadlock that may result when a clock gating cell A is disabled by a flip-flop that is clocked by CGC A, resulting in the cell being stuck in the disabled state.

How to Debug and Fix

To debug the violation, double-click the violation message to open the Incremental Schematic.

To fix the violation, do one of the following:

- Do not control the enable pin of a CGC through the flip-flop, which is driven by the same CGC.
- Use the XOR logic between FF-D and FF-Q to determine the CGC-EN.

Example Code and/or Schematic

Example 1

Consider the following example where a flip-flop, `top.u_ff_11.\ff_reg[1]`, is driving the system enable pin, `E`, of the CGC, `top.u_u_cgc_p`, which is driving the clock pin of the same flip-flop:

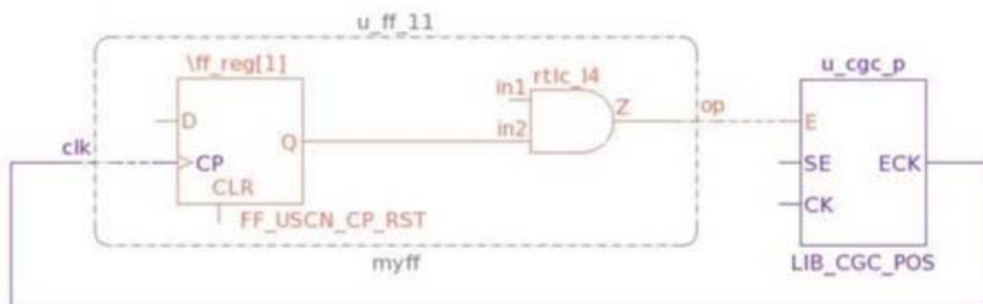


FIGURE 56. Simple Case

Clock Gating Rules

Example 2

Consider the following example where a flip-flop, `top.u_ff.\ff_reg[1]`, is driving the system enable pin, `en`, of the CGC, `top.u_u_cgc`, which is driving the clock pin of the same flip-flop:

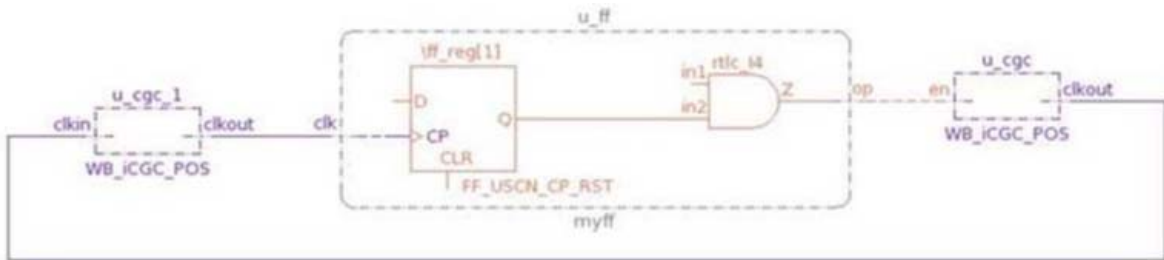


FIGURE 57. Cascaded CGCs are Present

Default Severity Label

Warning

Rule Group

ClockGating

Reports and Related Files

No related reports or files

CG_consistency

Behavior model of clock gating cell should be consistent with the constraint applied on it

Rule Description

The CG_consistency rule checks for inconsistency between the behavioral model of a clock gating cell (CGC) and the `gating_cell` constraint applied on it.

NOTE: *The CG_consistency rule will be deprecated in a future release.*

This rule reports a mismatch between the RTL and the SGDC constraint in the following cases:

- If a clock (test clock / atspeed clock) is present at the clk-out pin of a CGC but its enable pin is not yet active
- If the clk-in pin does not yet have a clock

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

Constraints

`clock` -testclock -atspeed (optional)

`gating_cell` (optional)

`test_mode` (optional)

Rule Parameters

- `dftUseOffStateOfClockInClockPropagation`
- `dft_treat_primary_port_as_valid_clock_point_for_cgc_check`
- `dft_enable_checks_for_all_cgc`

Operating Mode

Scanshift, Capture

Message Details

Message

Behavioral model of clock gating cell

<hierarchical_path_name> does not match with its constrained value in *<mode-name>*

Arguments

1. Hierarchical path name of the clock gating cell
<hierarchical_path_name>
2. Capture_atspeed, Capture, or Shift mode *<mode-name>*

Location

File / Line where clock gating cell is instantiated in the design

Schematic highlight

CGC and its violated pins.

Rule Severity

Info

CG_generateReport

Generate a text report with details of all clock gating cells in design

When to Use

Use this rule to generate a text report that lists the clock gating cells in the design.

Description

The CG_generateReport generates a text report, namely [dft_dsm_clock_gating_cell](#) report, which lists the CCG's in the design along with the following information:

- Cell type
- Number of flip-flops connected
- Enable pins
- Clock Source
- CG rules violations, if any

For more information on identifying CCGs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

Prerequisites

Run all Clock Gating (CG) rules for better results.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_treat_primary_port_as_valid_clock_point_for_cgc_check](#): Default value is off. Set the value of the parameter to on to consider ports as valid end point in order to report violations for a CCG.
- [dft_enable_checks_for_all_cgc](#): Default value is off. Set the value of the parameter to enable rule checking.

Constraint(s)

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, Capture, Capture (atspeed)

Messages and Suggested Fix

The following violation messages are displayed for the CG_generateReport rule:

Message 1

[INFO] Clock gating report file <file-name> is generated

Arguments

Name of the clock gating report file, <file-name>

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation. However, check all clock gating rules for violation.

Consequences of Not Fixing

Since this is an informational rule, there are no consequences of not fixing the violation. However, check all clock gating rules for violation.

How to Debug and Fix

The CG_generateReport is an informative rule and requires no debug and fix.

Message 2

For design root '<du-name>', cell '<module-name>' (<inst_count> instances) may act as positive/negative gating cell. It affects '<ff_count>' flip-flops

Arguments

- Current design name, <du-name>
- Name of white box module, <module-name>
- Number of instances of module, <inst_count>
- Number of flip-flop driven by output of module, <ff-count>

Potential Issues

This is an informational message. Therefore, there are no potential issues pertaining to this violation message.

Consequences Of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window displays path to a sample flip-flop and related gating cells instances.

To fix the violation, define the [gating_cell](#) constraint for the module or apply a waiver.

Message 3

[INFO] CSV file for listing which rules are skipped for each CGC is generated <du_name>, <spreadsheet_path>

Arguments

- Name of the design unit, <du_name>
- Path of the spreadsheet associated with the message, <spreadsheet_path>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no consequences of not fixing the violation.

How to Debug and Fix

This is an informational rule. Therefore, it requires no debug or fix. However, it is recommended to check all clock gating rules for violation.

Clock Gating Rules

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

ClockGating

Reports and Related Files

[dft_dsm_clock_gating_cell](#) report

Connection Rules

The Connection rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
Conn_01	Nodes that do not achieve expected values
Conn_02	Missing net connections between specified nodes
Conn_07	Structures that are not same as the user-specified structures between the user-specified nodes
Conn_08	The path between the user-specified nodes
Conn_09	Path between user specified nodes should not exist
Conn_10	Reports nets with illegal node values
Conn_11	Ensures that the node satisfies the specified constraint message tag expression
Conn_12	Ensures that the node does not have the specified constraint message tag expression
Conn_14	Ensure that specified nets are having stable values under specified condition
Conn_15	Check required pulse pattern at specified node
Diagnose_testmode	Display instances that block the testmode propagation
Info_define_tag	Show system state for a given tag
Info_testmode	Display testmode simulation results

NOTE: You need to have test-conn license to be able to run Connection Rules.

Conn_01

Ensure that the expected node value is achieved

When to Use

Use this rule to identify the nodes that do not achieve the expected simulation value.

Description

The *Conn_01* rule generates the SpyGlass Explorer highlight data for those nodes specified with *require_value* constraints that do not achieve the specified simulation value when the specified tag condition is simulated.

Expected values at arbitrary nodes and the applied values that should cause or force the expected values are checked. This is useful for ensuring that unit-level test requirements are satisfied at the SoC level.

Consider the following *require_value* constraint:

```
require_value
  -tag <tagName> -name <nodeNames> -value <value>
```

The *Conn_01* rule generates the SpyGlass Explorer highlight data for those nodes specified with the *-name* argument that do not achieve the simulation value *<value>* when the conditions of *<tagName>* are simulated.

Prerequisites

Specify the *require_value* constraint.

Default Weight

10

Language

Verilog, VHDL

Method

For each *define_tag*, simulate power, ground and all conditions defined for this *define_tag*.

For each *require_value* with the current *define_tag*, check that the defined pin has the defined value. Otherwise, report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: Default value is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_conn_check_handle_rtl_negedge*: Default value is off. Set the value of the parameter to yes to consider the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.

Constraint(s)

- *define_tag*: Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *require_value*: Use this constraint to define a check that requires a logic value to be established on a certain node when the circuit has been simulated using the condition specified by the -tag argument.

Operating Mode

Define_tag

Messages and Suggested Fix

The following violation messages are displayed for the Conn_01 rule:

Message 1

[ERROR] [constraint_message_tag: <value>] Node <name> has value <value1>. (Required: <value2>) under tag <tag-name>

Arguments

- Constraint tag value, <value>
- Name of the node <name>
- Actual value <value1>
- Expected value <value2>
- Tag name <tag-name>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the*

-*constraint_message_tag* argument for the [require_value](#) constraint.

Potential Issues

A violation is reported due to incomplete or incorrect simulation condition or incorrect design connectivity.

Consequences of Not Fixing

Not fixing the violation may result in unexpected code behavior.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic highlights the node where the required simulation mismatches the actual simulation.

The violation message for the Soc_04 rule overlaps the violation message for the Conn_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Conn_01 rule.

To fix the violation, see *Example Code and/or Schematic* section.

Message 2

[INFO] [constraint_message_tag: <value>] Node '<node-name>' has required value(<value>) under tag '<tagName>'

Arguments

- Constraint tag value, <value>
- Name of the node, <node-name>
- Simulation value, <value>
- Tag name, <tagName>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the [require_value](#) constraint.*

Potential Issues

This is an informational rule. Therefore, it does not have any related potential issues.

Consequences of Not Fixing

This is an informational rule. Therefore, it does not have any implicit impact.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental

Schematic displays the node where the required simulation matches the actual simulation.

Message 3

[WARNING] [constraint_message_tag: <value>] illegal_value command is using option -matchNBits (<value1>) without setting parameter dftShowWaveForm to 'on' under <tag-name>. Setting -matchNBits to value '1' for the rule checking purpose

Arguments

- Constraint tag value, <value>
- Illegal value <value1>
- Tag name <tag-name>

Potential Issues

A violation is reported when the following conditions hold true:

- Value of the argument under <tag-name> condition *dftShowWaveForm* parameter is not set on
- The *require_value* constraint uses the -matchNbits <value>
- The value of the -matchNBits argument is greater than 1

The Conn_01 rule does not report this violation message if the tag is defined using the *define_tag* command.

Consequences of Not Fixing

Not fixing the violation may result in unexpected code behavior.

How to Debug and Fix

To fix the violation, set the value of the *dftShowWaveForm* parameter to on.

Example Code and/or Schematic

Consider the following example:

```
require_value -name mode2 -value 0 -matchNBits 1 -useTestmode
require_value -name mode3 -value 1 -matchNBits 1 -useTestmode
```

Connection Rules

The screenshot displays the Atrenta Console interface. The top window, titled 'Incremental Schematic : tap', shows a circuit diagram with components like `rtl_c118`, `rtl_c116`, `mode0_reg`, `RTL_MUX2`, and `RTL_FDC`. The schematic is divided into `INSTR_DEC` and `instr_dec` blocks. The bottom window, titled 'View: Message', shows a list of messages:

- ERROR: [1]**
 - `Soc_01[1]:Expected node value must be achieved.`
 - Node 'mode0' has value 0 (Required: 1) under tag '-useTestmode', tap.sgdc, 25**
- INFO: [5]**
 - `Soc_01_Info[3]:Expected node value is achieved.`
 - Node 'mode1' has required value(1) under tag '-useTestmode', tap.sgdc, 26**
 - Node 'mode2' has required value(0) under tag '-useTestmode', tap.sgdc, 27**

Default Severity Label

Error

Rule Group

SoC

Reports and Related Files

`dft_connectivity_check_summary.rpt`: Reports the number of `require_value` constraints passed and failed.

Conn_02

Ensure that the paths between user-specified nodes exist

When to Use

Use this rule to identify the disjointed pair of user-specified nodes.

Rule Description

The Conn_02 rule reports connection violations between specified pair of nodes.

The Conn_02 rule checks either conditional or unconditional paths between user-specified nodes.

NOTE: *The Conn_02 rule checks all [require_path](#) constraints — without the `-tag` argument and with the `-tag` argument (earlier checked by the now obsolete Soc_03 rule.)*

While processing the [require_path](#) constraints, Conn_02 rule checking depends on `-path_type` and simulation condition as discussed below:

Specified field of <code>require_path</code> constraint	Simulation condition
<code>-use_shift</code>	Shift
<code>-use_capture</code>	Capture
<code>-use_captureATspeed</code>	Capture (atspeed)
<code>-tag <tag_name></code>	tag_name
If none of the above specified	Power ground

- If you specify the value of the `-path_type` argument as `sensitized`, the Conn_02 rule performs the strict functional checking. This ensures that the path is properly sensitized by the specified simulation condition.

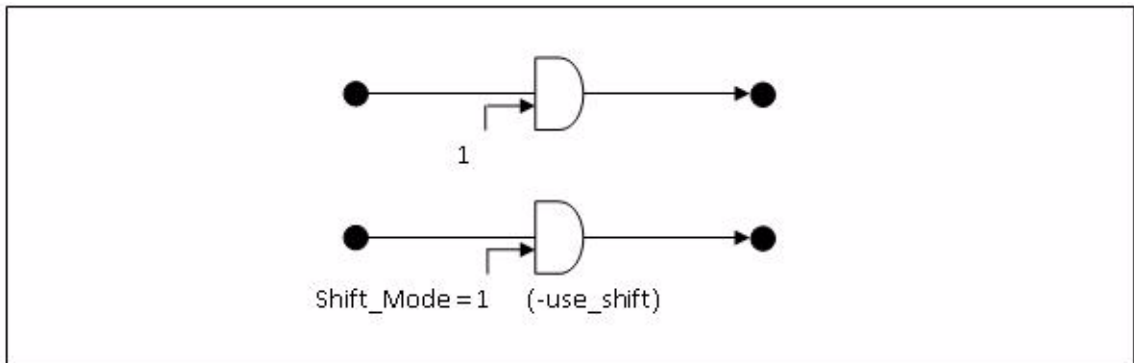
NOTE: *Multiple-drivers create blockages only when the `path_type` is sensitized.*

- If you specify the value of the `-path_type` argument as `sensitizable`, the Conn_02 rule performs the functional checking. This ensures that the path is properly sensitizable and is not blocked by the simulation condition.

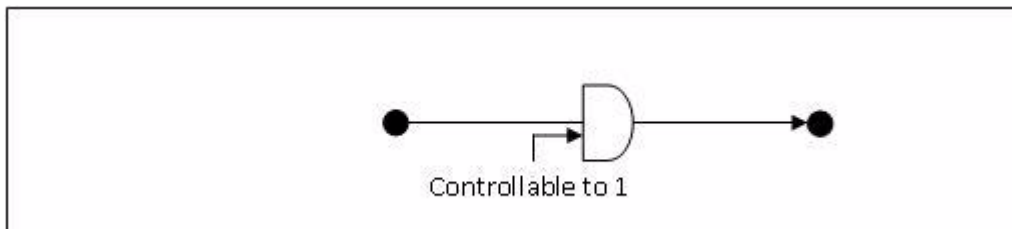
Connection Rules

- ❑ If you specify the value of the `-path_type` argument as `buffered`, the `Conn_02` rule looks for strict topological checking and checks for buffers and inverters only.
- ❑ If you do not specify the value of the `-path_type` argument, the default value of the argument, that is, `sensitizable` is used.

Consider the following example, where `Shift_Mode` is equal to 1, that is, `-use_shift` parameter is specified.



The above figure shows an example of the sensitized path. Here, the value of the `-path_type` argument is set as `sensitized`, which ensures the existing logic connectivity between the start and end points. Consider the following example, where the clock is controllable to 1:



The above figure shows an example of the sensitizable path. Here, the value of the `-path_type` argument is specified as `sensitizable`, which checks whether the target path is sensitizable during the specific simulation condition.

SoC integration often requires that connections between various units

exist. This rule allows arbitrary from-to pin to be checked.

The Conn_02 rule flags a violation if during traversal to find a path between two nodes, a contentious net is found. This path is no longer considered a correct path.

For more information, see [Impact of Different Path Types on Fanin/Fanout Cone Traversal](#).

NOTE: *If you do not specify the `-undirected` qualifier in the `require_path` constraint, the Conn_02 rule traces the path from `-from` node to `-to` node only.*

Prerequisites

Specify the `require_path` constraint.

Default Weight

10

Language

Verilog, VHDL

Method

If specified, simulate require path for <path_type> verification under specified simulation condition.

For each from-to pair of `require_path` specification:

Walk the unblocked fan-out cone under the simulation condition from the start point while maintaining the phase inversion. If the walk doesn't terminate on the specified endpoint for the required <path_type>, report a violation.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dftAllowNonXValueAtStartOfSensitizedPathInSoc_02`: This parameter is deprecated and is ignored for rule-checking.
- `dft_allow_path_from_enable_to_cgc_clkout`: The default value is off. Set the value of the parameter to on to allow a connectivity path from enable (data and test) to CGC clock-out pin.
- `dft_conn_check_allow_non_x_value_on_sensitizable_path`: The default value is on. Set the value of the parameter to off to ignore non-X (0 or 1) value on the path while performing checks on the sensitizable paths.

Connection Rules

- *dft_conn_check_allow_trace_through_async*: The default value is off. Set the value of the parameter to on to allow combinational traversal through the asynchronous pins of a flip-flop while performing the checks for require path (*require_path* constraint) or illegal path (*illegal_path* constraint).
- *dft_conn_check_allow_trace_through_clock_shaper*: The default value is off. Set the value of the parameter to on to allow combinational traversal through a clock shaper using clock_in / clock_out pins.
- *dft_conn_check_treat_endpoint_as_stoppoint*: The default value is rp_off_rsp_on. Set the value of the parameter to on to treat endpoint as stop point while performing checks for path between nodes.
- *dft_require_path_fail_limit*: The default value is 10. Set the value of the parameter to any natural number to control the number of violations reported by the *Conn_02* rule for the *require_path* constraint failure when either the *-from_one_of* or *-to_one_of* arguments of the *require_path* constraint is specified.
- *dft_conn_check_handle_rtl_negedge*: Default value is off. Set the value of the parameter to yes to consider the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.
- *dft_treat_latches_with_X_on_enable_as_combinational_for_soc_path_checks*: The default value is off. Set the value of the parameter to on to define the treatment of latches. That is, whether to consider them as combinational or sequential, where enable pin does not get either 0 or 1, when running Soc path check rules.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *require_path* (mandatory): Use this constraint to define a connectivity check for a path from a pin specified with the *-from* argument to a pin specified with the *-to* argument.
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

Scanshift, Capture, Power Ground, Define_tag

Messages and Suggested Fix

Message 1

[ERROR] [constraint_message_tag: <value>] '<path_type>' path(s) not found from '<from_node_name>' to '<to_node_count>' nodes under <simulation condition> within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[ERROR] [constraint_message_tag: <value>] '<path_type>' path(s) not found to '<to_node_name>' from '<from_node_count>' nodes under <simulation condition> within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, click Arguments.

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

NOTE: *This violation message is reported instead of Message 1, in case of '-from_one_of' field specified in 'require_path' constraint.*

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[ERROR] [constraint_message_tag: <value>] '<path_type>' path(s) not found from '<from_node_name>' to '<to_node_count>' nodes within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

NOTE: *This violation message is reported instead of Message 1, when '-path_type' is 'buffered' in 'require_path' constraint.*

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[ERROR] [constraint_message_tag: <value>] '<path_type>' path(s) not found to '<to_node_name>' from '<from_node_count>' nodes within '<sequential_depth>' sequential depth

Arguments

- Constraint tag value, <value>
- Expected path type, as specified by 'require_path' constraint, <path_type>

- Node name specified in the '-from' field of 'require_path' constraint, <from_node_name>
- Node name specified in the '-to' field of 'require_path' constraint, <to_node_name>
- No of '-to' nodes for which connectivity check failed from the specified '-from' node, <to_node_count>
- No of '-from' nodes from which connectivity check failed to the specified '-to' node, <from_node_count>
- Value specified by '-sequential_depth' field of 'require_path' constraint, <sequential_depth>
- Simulation condition specified in the 'require_path' constraint, <simulation_condition>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the [require_path](#) constraint.*

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

NOTE: *This violation message is reported instead of Message 2, when '-path_type' is 'buffered' in 'require_path' constraint.*

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist.

If the path from start point up to the instance which is blocking is displayed then analyze the text annotation date displayed to see why the path is blocked at the instance highlighted.

To fix the violation, see [Example Code and/or Schematic](#).

Message 5

[ERROR] [constraint_message_tag: <value>] ' <no_of_paths>' (<limit_breached>) ' <path_type>' path(s) found from ' <from_node_name> node under <simulation condition> within ' <sequential_depth >' sequential depth

Arguments

- Constraint tag value, <value>
- Expected path type, as specified by 'require_path' constraint, <path_type>
- Limit to the number of paths, <limit_breached>
- Node name specified in the '-from' field of 'require_path' constraint, <from_node_name>
- Simulation condition specified in the 'require_path' constraint, <simulation_condition>
- Value specified by '-sequential_depth' field of 'require_path' constraint, <sequential_depth>

Potential Issues

Specified number of paths are not present between user-specified nodes.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist.

If the path from start point up to the instance which is blocking is displayed then analyze the text annotation date displayed to see why the path is blocked at the instance highlighted.

To fix the violation, see [Example Code and/or Schematic](#).

Message 6

[WARNING] [constraint_message_tag: <value>] min_to_paths(' <min_path_value>') should have a value less than max_to_paths(' <max_path_value>'), ignoring these values

Arguments

- Constraint tag value, <value>
- Minimum number of expected successful paths, <min_path_value>
- Maximum number of expected successful paths, <max_path_value>

Potential issues

The value specified is ignored.

How to debug and fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist.

If the path from start point up to the instance which is blocking is displayed then analyze the text annotation date displayed to see why the path is blocked at the instance highlighted.

To fix the violation, review the constraint and modify to specify valid values.

Message 7

[INFO] [constraint_message_tag: <value>] ' <path_type>' path(s) found from ' <from_node_name>' to ' <to_node_count>' nodes under <simulation condition> within ' <sequential_depth >' sequential Depth

Arguments

To view the list of message arguments, see *Arguments*.

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

For more information on debugging and fixing the violation, see *How to Debug and Fix*.

Message 8

[INFO] [constraint_message_tag: <value>] '<path_type>' path(s) found to '<to_node_name>' from '<from_node_count>' nodes under <simulation condition> within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, see *Arguments*.

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

NOTE: *this violation message is reported instead of Message-1, in case of '-from_one_of' field specified in the require_path constraint.*

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

For more information on debugging and fixing the violation, see *How to Debug and Fix*.

Message 9

[Error] [constraint_message_tag: <value>] '<to_node_count>' (max_to_paths=<max_to_paths_count>)' '<path_type>' path(s) found from '<from_node_name>' to nodes under <simulation condition> within '<sequential_depth>' sequential Depth <spreadsheet_path>

Arguments

To view the list of message arguments, see *Arguments*.

Potential Issues

This violation message appears if `-max_to_paths` field is defined and valid paths count exceeds `max_to_paths` count limit.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, see *How to Debug and Fix*.

Message 10

```
[Error] [constraint_message_tag: <value>] '<to_node_count>
'(min_to_paths=<min_to_paths_count>)' <path_type>' path(s) found
from '<from_node_name>' to nodes under <simulation condition> within
'< sequential_depth >' sequential Depth <spreadsheet_path>
```

Arguments

To view the list of message arguments, see *Arguments*.

Potential Issues

This violation message appears if `-min_to_paths` field is defined and valid paths count comes out to be less than `min_to_paths` count.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, see *How to Debug and Fix*.

Message 11

```
[INFO] [constraint_message_tag: <value>] '<path_type>' path(s)
found from '<from_node_name>' to '<to_node_count>' nodes within
'<sequential_depth >' sequential depth
```

Arguments

- Constraint tag value, `<value>`
- Expected path type, as specified by 'require_path' constraint, `<path_type>`

- Node name specified in the '-from' field of 'require_path' constraint, <from_node_name>
- Node name specified in the '-to' field of 'require_path' constraint, <to_node_name>
- No of '-to' nodes for which connectivity check failed from the specified '-from' node, <to_node_count>
- No of '-from' nodes from which connectivity check failed to the specified '-to' node, <from_node_count>
- Value specified by '-sequential_depth' field of 'require_path' constraint, <sequential_depth>
- Simulation condition specified in the 'require_path' constraint, <simulation_condition>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the [require_path](#) constraint.*

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

NOTE: *This violation is reported instead of Message-1, when '-path_type' is 'buffered' in 'require_path' constraint.*

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

Example Code and/or Schematic

Consider the following example:

Case 1

```
define_tag-tag FUNC_mode-name tpin-value 1
require_path -from a -to o -tag FUNC_mode
```

Soc_02_Info[1]: Connection between user specified nodes exists.

Path from point 'a' to point 'o' is sensitizable under power ground simulation condition, test.sqdc_3

Note: in both the scenarios the select pin of the mux is 'unconstrained'

```
define_tag-tag FUNC_mode-name tpin-value 1
require_path -from a -to o -tag FUNC_mode \
-path_type sensitized
```

Soc_02[1]: Paths between user specified nodes must exist.

Path from point 'a' to point 'o' is 'not-sensitized' at node 'test.o' under power ground simulation condition, test.sqdc_3

Case 2

Schematic highlight

Start and end point would be highlighted. (No topological connection)

Path from start point till the place signal reached, would be highlighted in one color and from there to end point in different color. (Where path is blocked)

Path with the same phase in one color and invert phase in different color highlighted. This schematic is applicable only when the `-invert` or `-noinvert` argument is specified in the `require_path` constraint.

Default Severity

Label

Error

Rule Group

SoC

Reports and

Related Files

[dft_connectivity_check_summary.rpt](#): Reports the number of [require_path](#) constraints passed and failed.

Conn_07

Checks the structure between the user-specified nodes

When to Use

Use this rule when you want to use a specific structure.

Description

The Conn_07 rule checks whether the path all the -from nodes and at least one of the -from_one_of nodes to -to node have the same structure as specified in -type field of the *require_structure* constraint. The Conn_07 rule also allows functional buffers (1 input of OR/XOR gate is tied to 0, 1 input of AND is tied to 1). So, even if structure as specified in -type field acts as buffer, Conn_07 considers them as valid objects on the path.

As an example of the rule's application, you can use this rule to run a check on memories. Memory-Controllers have PASS / FAIL status signals.

A GLOBAL-PASS is AND of all individual PASS signals as specified below:

```
require_structure -from "BIST::PASS" -to G_PASS -structure and
```

A GLOBAL-FAIL is OR of all individual FAIL signals as specified below:

```
require_structure -from "BIST::FAIL" -to G_FAIL -structure or
```

Based on the above status, we may have 3 types of failures:

- Extra driver is specified in the `from-node-list`, which is not present in the combinational fanin cone of `-to` node. However, the violation message for the extra driver is not reported for nodes given through `-from_one_of` field, if at least one of them is found in the fan in cone of the `-to` node.
- Missing driver where a node is in the combinational fanin cone of `to-node` but is missing from the `from-node-list`.
- Incorrect GATE is found in the combinational cloud between `from` and `to` nodes.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [module_bypass](#) (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- [require_structure](#) (mandatory): Use this constraint to define a structure check for all paths from source pins to destination pin.

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING] [constraint_message_tag: <value>]
 [Reason: <reason_for_violation>]Path from point '<from_node>' to point '<to-node>' is not of type '<structure>'

Arguments

- Constraint tag value, <value>
- One or combination of 'missing-driver', 'xtra-driver', incorrect-driver' may come as reason for violation, <reason_for_violation>
- Name of the source node, <from_node>
- Name of the to destination node, <to_node>
- Desired structure, <structure>

NOTE: The constraint tag value is prefixed to the violation message only if you specify the `-constraint_message_tag` argument for the [require_structure](#) constraint.

Potential Issues

The violation message appears if path between user-specified nodes is not user-specified.

Consequences of Not Fixing

Not fixing this violation may result in error in evaluations in the circuit.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path between points specified in `-from` and `-to` field of the [require_structure](#) constraint.

To fix the violation, use user-specified structure only.

Message 2

[INFO] [constraint_message_tag: <value>] Path from point '<from_node>' to point '<to_node>' is of type '<structure>'

Arguments

- Constraint tag value, <value>
- Name of the source node, <from_node>
- Name of the to destination node, <to_node>
- Desired structure, <structure>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the `-constraint_message_tag` argument for the [require_structure](#) constraint.*

Potential Issues

The violation message appears when the structure between two points matches with the user-specified structure.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix information is required for this violation message.

Example Code and/or Schematic

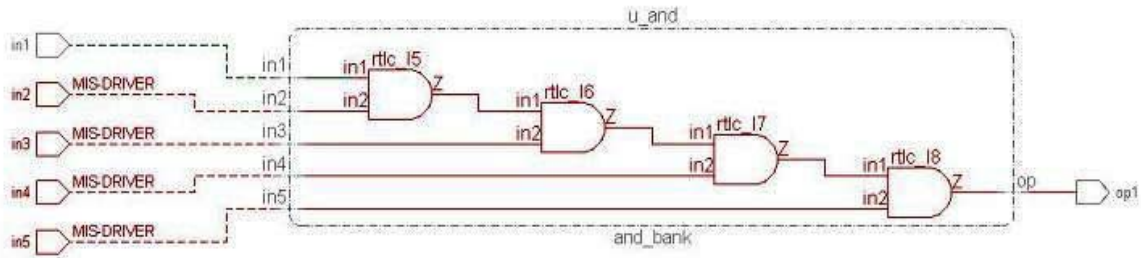
Example 1

Consider the following SGDC declaration:

Connection Rules

```
require_structure -structure and -from in1 -to op1
```

Now, consider the following figure illustrating the Conn_07 rule violation because of a missing driver:



For the above example, the Conn_07 rule reports the following violation message:

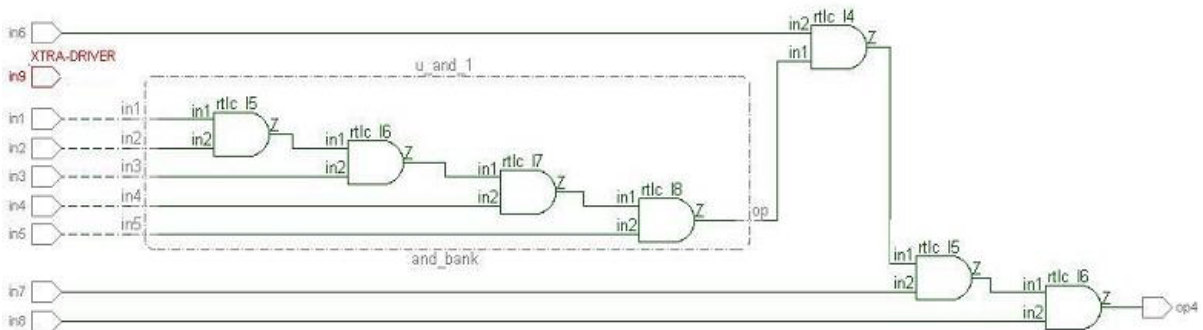
[Reason: missing-driver]Path from point 'in1' to point 'op1' is not of type 'and'

Example 2

Consider the following SGDC declaration:

```
require_structure -structure and -from in1 in2 in3 in4 in5
in6 in7 in8 in9 -to op4
```

Consider the following figure illustrating the Conn_07 rule violation because of an extra driver:



For the above example, the Conn_07 rule reports the following violation

message:

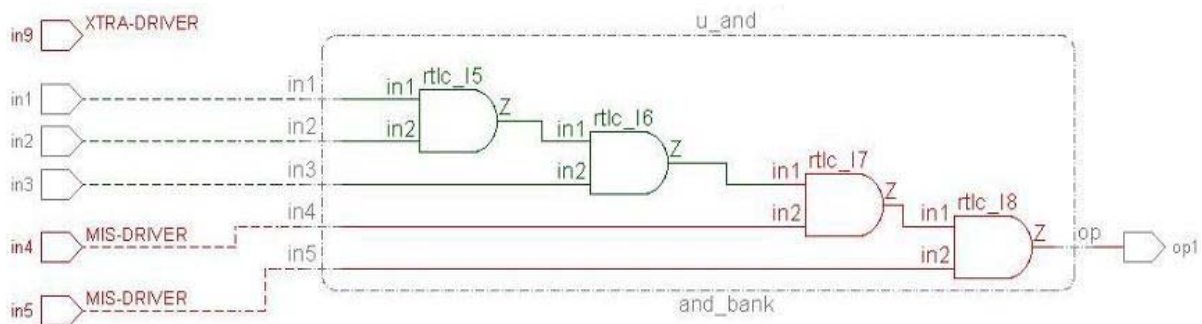
[Reason: xtra-driver]Path from point 'in9 in8 in7 in6 in5 in4 in3 in2 in1' to point 'op4' is not of type 'and'

Example 3

Consider the following SGDC declaration:

```
require_structure -structure and -from in1 in2 in3 in9 -to op1
```

Consider the following figure illustrating the Conn_07 rule violation because of extra and missing drivers:



For the above example, the Conn_07 rule reports the following violation message:

[Reason: xtra-driver, missing-driver]Path from point 'in9 in3 in2 in1' to point 'op1' is not of type 'and'

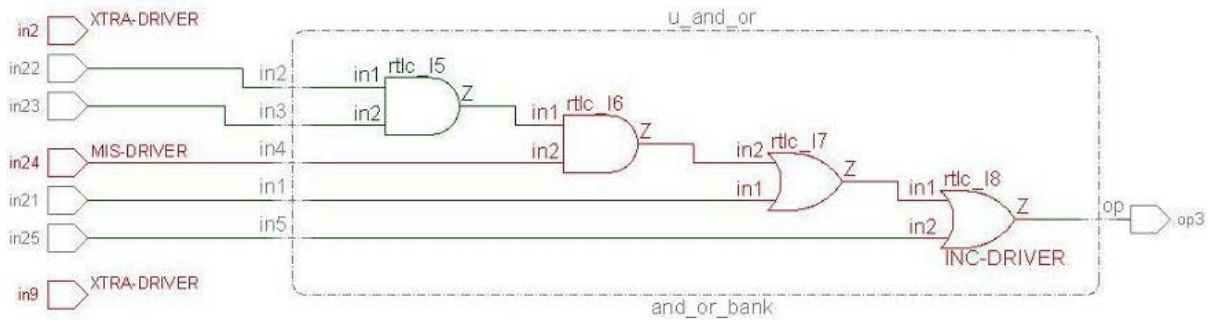
Example 4

Consider the following SGDC declaration:

```
require_structure -structure and -from in21 in22 in23 in25 -to op3
```

Consider the following figure illustrating the Conn_07 rule violation because of an extra, missing, and incorrect drivers:

Connection Rules



For the above example, the Conn_07 rule reports the following violation message:

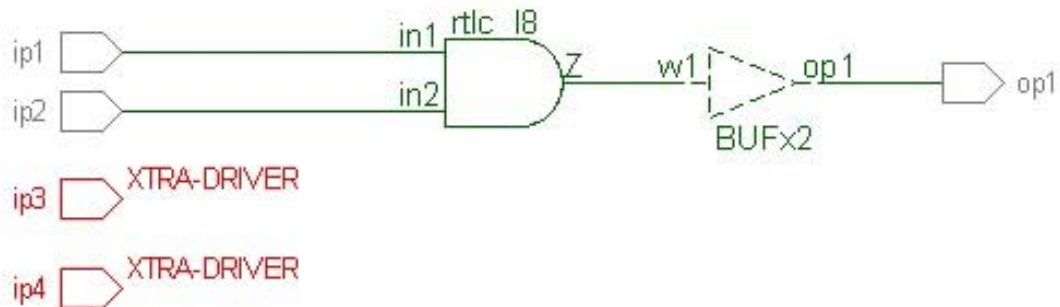
[Reason: xtra-driver, incorrect-driver, missing-driver]Path from point 'in9 in2 in25 in23 in22 in21' to point 'op3' is not of type 'and'

Example 5

Consider the following SGDC declaration:

```
require_structure -from ip1 ip2 -from_one_of ip3 ip4 -to op1
-structure and
```

Now, consider the following figure illustrating the Conn_07 rule violation because of using the -from_one_of and -structure fields of the require_structure constraint simultaneously:



For the above example, the Conn_07 rule reports the following violation message because none of the nodes specified using the -from_one_of field was found in the fanin of the -to node.

```
[Reason: xtra-driver]Path from point 'ip2 ip1, one of "ip4 ip3"'  
to point 'op1' is not of type 'and'
```

Default Severity Label

Warning

Rule Group

Soc

Reports and Related Files

[dft_connectivity_check_summary.rpt](#): Reports the number of [require_structure](#) constraints passed and failed.

Conn_08

Checks the path between the user-specified nodes

When to Use

Use this rule when you want to check the existence of a specific path.

Description

The Conn_08 rule checks if a path originating from the `-from` field to the `-to` field of the [require_strict_path](#) constraint exists in the design.

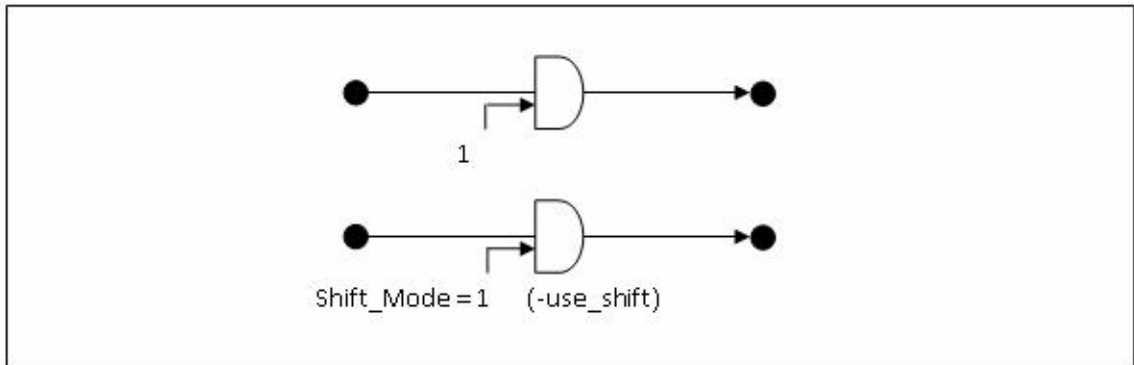
While processing the [require_strict_path](#) constraints, Conn_08 rule checking depends on `-path_type` and simulation condition as discussed below:

Specified field of <code>require_path</code> constraint	Simulation condition
<code>-use_shift</code>	Shift
<code>-use_capture</code>	Capture
<code>-use_captureATspeed</code>	Capture (atspeed)
<code>-tag <tag_name></code>	<code>tag_name</code>
If none of the above specified	Power ground

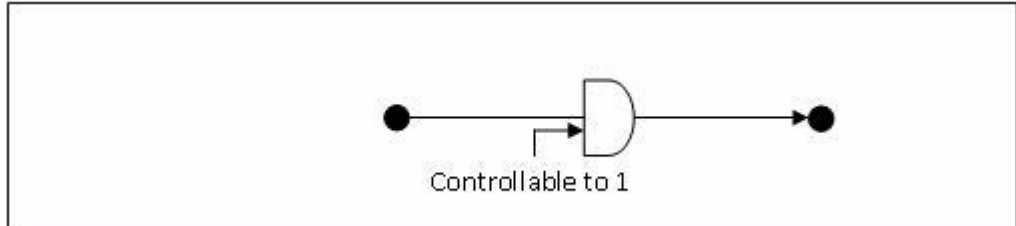
- If you specify the value of the `-path_type` argument as `sensitized`, the Conn_08 rule performs the strict functional checking. This ensures that the path is properly sensitized by the specified simulation condition.
- If you specify the value of the `-path_type` argument as `sensitizable`, the Conn_08 rule performs the functional checking. This ensures that the path is properly sensitizable and is not blocked by the simulation condition.
 - If you specify the value of the `-path_type` argument as `buffered`, the Conn_08 rule looks for strict topological checking and checks for buffers and inverters only.
 - If you do not specify the value of the `-path_type` argument, the default value of the argument, that is, `sensitizable` is used.

Consider the following example, where `Shift_Mode` is equal to 1, that

is, `-use_shift` parameter is specified.



The above figure shows an example of the sensitized path. Here, the value of the `-path_type` argument is set as `sensitized`, which ensures the existing logic connectivity between the start and end points. Consider the following example, where the clock is controllable to 1:



The above figure shows an example of the sensitizable path. Here, the value of the `-path_type` argument is specified as `sensitizable`, which checks whether the target path is sensitizable during the specific simulation condition.

SoC integration often requires that connections between various units exist. This rule allows arbitrary from-to pin to be checked.

The `Conn_08` rule flags a violation if during traversal to find a path between two nodes, a contentious net is found. This path is no longer considered a correct path.

For more information, see [Impact of Different Path Types on Fanin/Fanout Cone Traversal](#).

NOTE: If you do not specify the `-undirected` qualifier in the `require_path` constraint, the `Conn_08` rule traces the path from `-from` node to `-to` node only.

Prerequisites

Specify the `require_path` constraint.

Default Weight

10

Language

Verilog, VHDL

Method

If specified, simulate require path for `<path_type>` verification under specified simulation condition.

For each from-to pair of `require_strict_path` specification:

Walk the unblocked fan-out cone under the simulation condition from the start point while maintaining the phase inversion. If the walk doesn't terminate on the specified endpoint for the required `<path_type>` or on undesired endpoint, report a violation.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_allow_path_from_enable_to_cgc_clkout*: The default value is off. Set the value of the parameter to on to allow a connectivity path from enable (data and test) to CGC clock-out pin.
- *dft_require_path_fail_limit*: The default value is 10. Set the value of the parameter to 10. Set the value of the parameter to any natural number to controls the number of violations reported by the `Conn_08` rule for the `require_strict_path` constraint failure when either the `-from_one_of`

or `-to_one_of` arguments of the `require_strict_path` constraint is specified.

- `dft_conn_check_allow_trace_through_async`: Default value is off. Set the value of the parameter to yes to allow combinational traversal through the asynchronous pins of a flip-flop while performing the checks for require path (`require_path` constraint) or illegal path (`illegal_path` constraint).
- `dft_conn_check_allow_trace_through_clock_shaper`: The default value is off. Set the value of the parameter to on to allow combinational traversal through a clock shaper using `clock_in` / `clock_out` pins.
- `dft_conn_check_allow_non_x_value_on_sensitizable_path`: The default value is on. Set the value of the parameter to off to ignore non-X (0 or 1) value on the path while performing checks on the sensitizable paths.
- `dft_conn_check_handle_rtl_negedge`: Default value is off. Set the value of the parameter to yes to consider the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.
- `dft_conn_check_treat_endpoint_as_stoppoint`: The default value is `rp_off_rsp_on`. Set the value of the parameter to off to ignore endpoint as stop point while performing checks for path between nodes.
- `dft_conn_check_rp_rsp_unified_flow`: The default value is on. In this case, the rule performs path type dependent checking and incorporates named, positional or instance association, if specified with `require_strict_path` constraint, while performing checks.
- `dft_allow_path_from_enable_to_cgc_clkout`: The default value is off. Set the value of the parameter to on to allow a connectivity path from enable (data and test) to CGC clock-out pin
- `dft_require_path_invalid_limit`: The default value is 10. Set the value of the parameter to any natural number to control the number of invalid path violations reported by the `Conn_08` rule for the `require_strict_path` constraint failure.
- `dft_require_path_pass_limit`: The default value is -1. Set the value of the parameter to any natural number to control the number of violations reported by the `Conn_08` rule for the `require_strict_path` constraint failure when either the `-from_one_of` or `-to_one_of` arguments of the `require_strict_path` constraint is specified.

- [*dft_treat_latches_with_X_on_enable_as_combinational_for_soc_path_checks*](#): The default value is off. Set the value of the parameter to on to define the treatment of latches. That is, whether to consider them as combinational or sequential, where enable pin does not get either 0 or 1, when running Soc path check rules.

Constraint(s)

- [*module_bypass*](#) (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- [*require_strict_path*](#) (mandatory): Use this constraint to define a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument.
- [*test_mode*](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [*define_tag*](#) (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

Scanshift, Capture, Define_tag, Power Ground

Messages and Suggested Fix

The following violation messages are reported when the value of parameter [*dft_conn_check_rp_rsp_unified_flow*](#) is on:

Message 1

[ERROR] [constraint_message_tag: <value>] '<path_type>' path(s) not found from '<from_node_name>' to '<to_node_count>' nodes under <simulation condition> within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[ERROR] [constraint_message_tag: <value>] '<path_type>' path(s) not found to '<to_node_name>' from '<from_node_count>' nodes under <simulation condition> within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, click Arguments.

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

NOTE: *This violation message is reported instead of Message 1, in case of '-from_one_of' field specified in 'require_path' constraint.*

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[ERROR] [constraint_message_tag: <value>] '<no_of_paths>' (<limit_breached>) '<path_type>' path(s) found from '<from_node_name>' node under <simulation condition> within '<sequential_depth >' sequential depth

Arguments

- Constraint tag value, <value>
- Expected path type, as specified by 'require_strict_path' constraint, <path_type>

- Limit to the number of paths, <limit_breached>
- Node name specified in the '-from' field of 'require_strict_path' constraint, <from_node_name>
- Simulation condition specified in the 'require_strict_path' constraint, <simulation_condition>
- Value specified by '-sequential_depth' field of 'require_strict_path' constraint, <sequential_depth>

Potential Issues

Specified number of paths are not present between user-specified nodes.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist. If the path from start point up to the instance which is blocking is displayed then analyze the text annotation data displayed to see why the path is blocked at the instance highlighted.

To fix the violation, see Example Code and/or Schematic.

Message 4

[WARNING] [constraint_message_tag: <value>] min_to_paths(' <min_path_value>') should have a value less than max_to_paths(' <max_path_value>'), ignoring these values

Arguments

- Constraint tag value, <value>
- Minimum number of expected successful paths, <min_path_value>
- Maximum number of expected successful paths, <max_path_value>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the [require_path](#) constraint.*

Potential Issues

The value specified is ignored.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist. If the path from start point up to the instance which is blocking is displayed then analyze the text annotation date displayed to see why the path is blocked at the instance highlighted.

To fix the violation, review the constraint and modify to specify valid values.

Message 5

```
[INFO] [constraint_message_tag: <value>] '<path_type>' path(s)
found from '<from_node_name>' to '<to_node_count>' nodes under
<simulation condition> within '<sequential_depth>' sequential
depth
```

Arguments

To view the list of message arguments, see Arguments.

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

For more information on debugging and fixing the violation, see How to Debug and Fix

Message 6

[INFO] [`constraint_message_tag`: <value>] '`<path_type>`' path(s) found from '`<from_node_name>`' to '`<to_node_count>`' nodes within '`<sequential_depth >`' sequential depth

Arguments

- Constraint tag value, <value>
- Expected path type, as specified by 'require_strict_path' constraint, <path_type>
- Node name specified in the '-from' field of 'require_strict_path' constraint, <from_node_name>
- Node name specified in the '-to' field of 'require_strict_path' constraint, <to_node_name>
- No of '-to' nodes for which connectivity check failed from the specified '-from' node, <to_node_count>
- No of '-from' nodes from which connectivity check failed to the specified '-to' node, <from_node_count>
- Value specified by '-sequential_depth' field of 'require_strict_path' constraint, <sequential_depth>
- Simulation condition specified in the 'require_strict_path' constraint, <simulation_condition>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the require_strict_path constraint.*

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message. This violation is reported instead of Message-5, when '-path_type' is 'buffered' in 'require_strict_path' constraint.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation

message. The incremental schematic displays the path specified in the violation message.

Message 7

[WARNING] [constraint_message_tag: <value>] ' <path_type>' path(s) found from '<from_node_name>' to '<to_node_count>' undesired nodes under <simulation condition> within '<sequential_depth>' sequential depth

Arguments

To view the list of message arguments, click Arguments.

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click How to Debug and Fix.

Message 8

[WARNING] [constraint_message_tag: <value>] ' <path_type>' path(s) found from '<from_node_name>' to '<to_node_count>' undesired nodes within '<sequential_depth>' sequential depth

Arguments

- Constraint tag value, <value>
- Expected path type, as specified by 'require_path' constraint, <path_type>
- Node name specified in the '-from' field of 'require_path' constraint, <from_node_name>
- Node name specified in the '-to' field of 'require_path' constraint, <to_node_name>
- No of '-to' nodes for which connectivity check failed from the specified '-from' node, <to_node_count>

- No of '-from' nodes from which connectivity check failed to the specified '-to' node, <from_node_count>
- Value specified by '-sequential_depth' field of 'require_path' constraint, <sequential_depth>
- Simulation condition specified in the 'require_path' constraint, <simulation_condition>

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

NOTE: *This violation message is reported instead of Message 7, when '-path_type' is 'buffered' in 'require_strict_path' constraint.*

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, click How to Debug and Fix.

Message 9

[ERROR] [constraint_message_tag: <value>] ' <to_node_count> '(max_to_paths=<max_to_paths_count>)' <path_type>' path(s) found from '<from_node_name>' to nodes under <simulation condition> within '<sequential_depth>' sequential Depth <spreadsheet_path>

Arguments

- Constraint tag value, <value>
- Expected path type, as specified by 'require_path' constraint, <path_type>
- Node name specified in the '-from' field of 'require_path' constraint, <from_node_name>
- Node name specified in the '-to' field of 'require_path' constraint, <to_node_name>
- No of '-to' nodes for which connectivity check failed from the specified '-from' node, <to_node_count>

- No of '-from' nodes from which connectivity check failed to the specified '-to' node, <from_node_count>
- Value specified by '-sequential_depth' field of 'require_path' constraint, <sequential_depth>
- Simulation condition specified in the 'require_path' constraint, <simulation_condition>

Potential Issues

This violation message appears if -max_to_paths field is defined and valid paths count exceeds max_to_paths count limit.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, see *How to Debug and Fix*.

Message 10

[**ERROR**] [constraint_message_tag: <value>] '<to_node_count>' '(min_to_paths=<min_to_paths_count>)' '<path_type>' path(s) found from '<from_node_name>' to nodes under <simulation condition> within '< sequential_depth >' sequential Depth <spreadsheet_path>

Arguments

To view the list of message arguments, see *Arguments*.

Potential Issues

This violation message appears if -min_to_paths field is defined and valid paths count comes out to be less than min_to_paths count.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

For more information on debugging and fixing the violation, see *How to Debug and Fix*.

The following violation messages are reported when the value of parameter [dft_conn_check_rp_rsp_unified_flow](#) is off:

Message 1

[INFO] [constraint_message_tag: <value>] Found valid path(s) from '<from-node>' to '<to-nodes-count>' desired nodes <mode-name>

Arguments

To view the list of arguments, click [Arguments](#).

Potential Issues

Since this is an informational message, there are no potential issues related to this message.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this violation message.

How to Debug and Fix

No debug or fix is required as this is an informational message.

Message 2

[WARNING] [constraint_message_tag: <value>] Found path from '<from-node>' to '<to-nodes-count>' undesired node <mode-name>

Arguments

The node at which the traversal was stopped, <to-node>

To view the other arguments, click [Arguments](#).

Potential Issues

The violation is reported because a valid path to a design node, which was not specified in the -to field of the `require_strict_path` constraint, is found by SpyGlass.

Consequences of Not Fixing

Not fixing this violation may result in unexpected results.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

To fix the violation, verify the highlighted path. If the path is correct, add the `<to-node>` node in the `-to` field of the `require_strict_path` constraint in the SGDC file. However, if the path is not correct, specify any missing testmode constraint or check the design.

Message 3

[WARNING] [`constraint_message_tag: <value>`] No Valid path(s) from '`<from-node>`' to '`<to-nodes-count>`' desired nodes `<mode-name>`

Arguments

- Constraint tag value, `<value>`
- Source node from which the connectivity check is performed, `<from-node>`
- Number of `<to-nodes>` at which the traversal was stopped, `<to-nodes-count>`

Please note that this list may or may not be present in the `-to` field of the `require_strict_path` constraint.

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the `-constraint_message_tag` argument for the [require_strict_path](#) constraint.*

- The mode name, `<mode-name>`

The available mode names are specified in the following table:

Specified Field of the <code>require_strict_path</code> Constraint	Corresponding <code><mode-name></code>
<code>-use_shift</code>	Shift
<code>-use_capture</code>	Capture

Specified Field of the <code>require_strict_path</code> Constraint	Corresponding <mode-name>
-use_captureATspeed	Capture (atspeed)
-tag	Define_tag
If none of the above fields is specified	Power Ground

Potential Issues

The violation message appears if no valid path exists between the user-specified nodes.

Consequences of Not Fixing

Not fixing this violation may impact the functionality of the design.

How to Debug and Fix

To fix this violation, perform any of the following steps:

- Check the design for an error
- Check the SGDC file for a missing constraint

Message 4

[WARNING] [constraint_message_tag: <value>] '<no_of_paths>' (<limit_breached>) path(s) found from '<from_node_name>' to desired nodes in <simulation condition>

Arguments

- Constraint tag value, <value>
- Limit to the number of paths, <limit_breached>
- Node name specified in the '-from' field of 'require_strict_path' constraint, <from_node_name>
- Simulation condition specified in the 'require_strict_path' constraint, <simulation_condition>

Potential Issues

Specified number of paths are not present between user-specified nodes.

Consequences of Not Fixing

Not fixing the violation may result in unsearchable expected path. This may impact the functionality of the design.

How to Debug and Fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist.

If the path from start point up to the instance which is blocking is displayed then analyze the text annotation date displayed to see why the path is blocked at the instance highlighted.

To fix this violation, perform any of the following steps:

- Check the design for an error
- Check the SGDC file for a missing constraint

Message 5

[WARNING] [constraint_message_tag: <value>] min_to_paths(' <min_path_value>') should have a value less than max_to_paths(' <max_path_value>'), ignoring these values

Arguments

- Constraint tag value, <value>
- Minimum number of expected successful paths, <min_path_value>
- Maximum number of expected successful paths, <max_path_value>

Potential issues

The value specified is ignored.

How to debug and fix

Double-click the violation message to open the spreadsheet, which contains the list of all the violation messages. Click on a violation message in the spreadsheet to view the incremental schematic of the violation message. The incremental schematic displays the path specified in the violation message.

If endpoints are displayed start probing from either fan-out cone of from point or fan-in cone of to point to see why required path does not exist.

If the path from start point up to the instance which is blocking is displayed

then analyze the text annotation date displayed to see why the path is blocked at the instance highlighted.

To fix the violation, review the constraint and modify to specify valid values.

Example Code and/or Schematic

Consider the following SGDC file snippet:

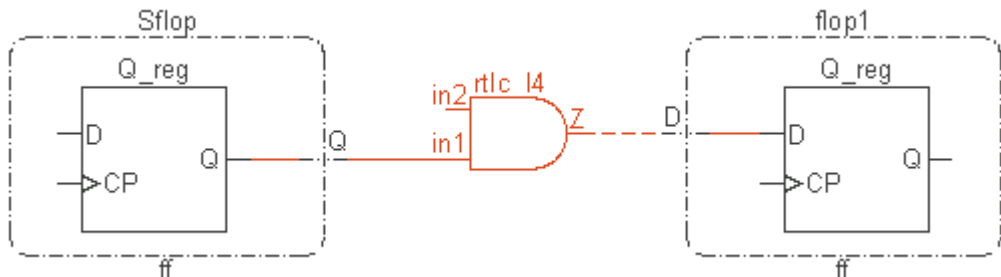
```
current_design dt
define_tag -tag mbel -name bbox.en -value 1
testmode -name bbox.en -value 0
require_strict_path -from "Sflop.*reg.Q" -to "flop2.*_reg.D"
-use_shift
```

Now, consider the following violation reported by the Conn_08 rule for the above specification of the `require_strict_path` constraint:

[WARNING] 'sensitizable' path(s) found from 'dt.Sflop.Q_reg.Q' to '1' undesired nodes under shift mode condition within '0' sequential depth

The above violation is reported because there exists a valid path from the `dt.Sflop.Q_reg.Q` node to the `dt.flop1.Q_reg.D` node, however, the `dt.flop1.Q_reg.D` node is not specified in the `-to` field of the above `require_strict_path` constraint specification.

The following schematic corresponds to the above violation message:



Default Severity Label

Info/Warning

Rule Group

SoC

Reports and Related Files

[dft_connectivity_check_summary.rpt](#): Reports the number of [require_strict_path](#) constraints passed and failed.

Conn_09

Path between user-specified nodes should not exist

When to Use

Use this rule to ensure that the path between two nodes does not exist.

Description

The Conn_09 rule reports a violation, if a path exists between from and to nodes of the *illegal_path* constraint.

The type of path to be searched is determined by the `-path_type` argument of the *illegal_path* constraint. This argument can take one of the following values: *buffered*, *sensitized*, or *sensitizable*. By default, sensitizable path is searched.

For more information, see [Impact of Different Path Types on Fanin/Fanout Cone Traversal](#).

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_treat_latches_with_X_on_enable_as_combinational_for_soc_path_checks](#): The default value is off. Set the value of the parameter to on to define the treatment of latches. That is, whether to consider them as combinational or sequential, where enable pin does not get either 0 or 1, when running Soc path check rules.
- [dft_allow_path_from_enable_to_cgc_clkout](#): The default value is off. Set the value of the parameter to on to allow a connectivity path from enable (data and test) to CGC clock-out pin.
- [dft_conn_check_allow_trace_through_async](#): The default value is off. Set the value of the parameter to on to allow combinational traversal through the asynchronous pins of a flip-flop while performing the checks

for require path (*require_path* constraint) or illegal path (*illegal_path* constraint).

- *dft_conn_check_allow_trace_through_clock_shaper*: The default value is off. Set the value of the parameter to on to allow combinational traversal through a clock shaper using clock_in / clock_out pins.
- *dft_conn_check_handle_rtl_negedge*: Default value is off. Set the value of the parameter to yes to consider the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.

Constraint(s)

- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *illegal_path* (mandatory): Use this constraint to define nodes between which path should not exist.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

Scanshift, Capture, Define_tag, Power Ground

Messages and Suggested Fix

Message 1

```
[ERROR] [constraint_message_tag: <value>]' <path_type>' path
found from '<from_node>' to undesired node '<to_node>' in
'<mode_name>'
```

Arguments

- Type of path found, <path_type>
- From node name, <from_node>
- To node name, <to_node>
- Simulation condition, <mode_name>

Potential Issues

The violation message is reported because a valid path exists from `<from_node>` to `<to_node>`, which are specified using the *illegal_path* constraint.

Consequences of Not Fixing

Not fixing this violation may impact the functionality of design.

How to Debug and Fix

View the incremental schematic of the violation message. It displays the path specified in the violation message.

To fix the violation, verify the highlighted path. If the path is correct, modify the corresponding *illegal_path* constraint. However, if the path is not correct, specify any missing *test_mode/define_tag* constraint or check the design.

Additional Information

The *Conn_09* rule also report following additional information based on the inputs specified:

- If sequential depth is specified, the message reports the current sequential depth of the reported path.
- If the *one_of* property of the *illegal_path* path constraint is used, the message reports the message count for the respective constraint.

See [Example 2](#) for more information.

Message 2

```
[ERROR] [constraint_message_tag: <value>] '<path_type>' path
found to '<to_node>' from undesired node '<from_node>' in
'<mode_name>'
```

Arguments

- Type of path found, `<path_type>`
- From node name, `<from_node>`
- To node name, `<to_node>`
- Simulation condition, `<mode_name>`

Potential Issues

The violation message is reported because a valid path exists to `<to_node>` from `<from_node>`, which are specified using the *illegal_path* constraint.

Consequences of Not Fixing

Not fixing this violation may impact the functionality of design.

How to Debug and Fix

View the incremental schematic of the violation message. It displays the path specified in the violation message.

To fix the violation, verify the highlighted path. If the path is correct, modify the corresponding *illegal_path* constraint. However, if the path is not correct, specify any missing *test_mode/define_tag* constraint or check the design.

Message 3

```
[ERROR] [constraint_message_tag: <value>]' <From | To>' node
<node name> is connected to leaf cell(s)
```

Arguments

- Constraint tag value, *<value>*
- From / To node name, *<node name>*

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the illegal_path constraint.*

Potential Issues

The violation message is reported because a from node was driving a leaf cell or a to node was driven by a leaf cell. The violation message is reported only for to or from nodes which were specified without any from or to nodes, respectively, in the *illegal_path* constraint.

Consequences of Not Fixing

Not fixing this violation may impact the functionality of design.

How to Debug and Fix

View the incremental schematic of the violation message. It displays the path specified in the violation message.

To fix the violation, verify the highlighted path. If the path is correct, modify the corresponding *illegal_path* constraint. However, if the path is not correct, specify any missing *test_mode/define_tag* constraint or check the design.

Message 4

```
[INFO] [constraint_message_tag: <value>] ' <passed_checks>' out
```

Connection Rules

of '<total_checks>' check(s) passed

Potential Issues

Not Applicable.

Consequences of Not Fixing

Not Applicable.

How to Debug and Fix

Not Applicable.

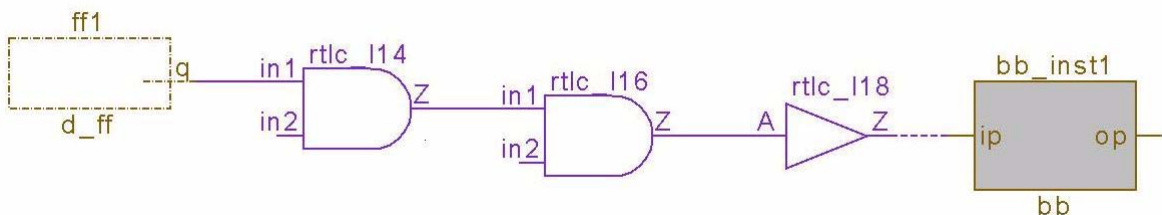
Example Code and/or Schematic

Example 1

Consider the following sample SGDC file snippet:

```
current_design test.behaviour
illegal_path -from test.ff1.q -to test.bb_inst1.ip
-use_shift
```

Now, consider the following figure:



The Conn_09 rule reports the following violation message for the above specification of the *illegal_path* constraint:

```
[ERROR] 'Sensitive' path found from 'test.ff1.q' to
undesired node 'test.bb_inst1.ip' in 'Shift mode'
```

The rule reports the above violation message because there exists a sensitizable path from the test.ff1.q node to the test.bb_inst1.ip node.

Example 2

Consider the following violation message reported by the *Conn_09* rule:

[constraint_message_tag: SEQ_2_MIN_2_FANIN] 'Sensitive' path found to 'top.u_config_register.ff2_reg.D (config_register: FLIP_FLOP_DATA)' from undesired node 'top.u_mem3.op (MEMORY_OUT) (in 2 sequential depth) (1 of 2)' in 'Power-Ground mode'

The above violation message reports the following additional information:

- Current sequential depth, when you have specified the `-sequential_depth <n>` argument for the input constraint.
- Message count (x of y) for the respective constraint, if source (FROM) or destination (TO) use the `ONE_OF` property of the [illegal_path](#) constraint.

Default Severity Label

Error

Rule Group

SoC

Reports and Related Files

[dft_connectivity_check_summary.rpt](#): Reports the number of [illegal_path](#) constraints passed and failed.

Conn_10

Reports nets with illegal node values

When to Use

Use this rule to identify the nodes that have a user-specified value under the specified simulation condition.

Description

The *Conn_10* rule generates the SpyGlass Explorer highlight data for those nodes specified with *illegal_value* constraints that contain the illegal simulation value when the specified tag condition is simulated.

Expected values at arbitrary nodes and the applied values that should cause or force the expected values are checked. This is useful for ensuring that unit-level test requirements are satisfied at the SoC level.

Consider the following *illegal_value* constraint:

```
illegal_value
  -tag <tagName> -name <nodeNames> -value <value>
```

The *Conn_10* rule generates the SpyGlass Explorer highlight data for those nodes specified with the `-name` argument get value `<value>` when the conditions specified by `<tagName>` are simulated.

Prerequisites

Specify the *illegal_value* constraint.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_conn_check_handle_rtl_negedge*: Default value is off. Set the value of the parameter to yes to consider the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.

Constraint(s)

- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *define_tag*: Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *illegal_value*: Use this constraint to check for the presence of an illegal value on a certain node when the circuit has been simulated using the condition specified by the -tag argument.

Operating Mode

Define_tag

Messages and Suggested Fix

The following violation messages are reported by the Conn_10 rule:

Message 1

[ERROR] [constraint_message_tag: <value>] Node <name> has illegal value <value1>. under tag <tag-name>

Arguments

- Constraint tag value, <value>
- Name of the node <name>
- Illegal value <value1>
- Tag name <tag-name>

NOTE: *The constraint tag value is prefixed to the violation message only if you specify the -constraint_message_tag argument for the [illegal_value](#) constraint.*

Potential Issues

A violation is reported due to incomplete or incorrect simulation condition or incorrect design connectivity.

Consequences of Not Fixing

Not fixing the violation may result in unexpected code behavior.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic highlights the node with the illegal value at the simulation.

You can also view the violation for the *Info_define_tag* rule along with the violation of the Conn_10 rule in the Incremental Schematic window. To do this, double-click the violation for the Conn_10 rule and open the Incremental Schematic window.

The violation message for the *Info_define_tag* rule overlaps the violation message for the Conn_10 rule in the Incremental Schematic window. This is useful in debugging the violation for the Conn_10 rule.

To fix the violation, see [Example Code and/or Schematic](#) section.

Message 2

[WARNING] [constraint_message_tag: <value>] illegal_value command is using option -matchNBits (<value1>) without setting parameter dftShowWaveForm to 'on' under <tag-name>. Setting -matchNBits to value '1' for the rule checking purpose

Arguments

- Constraint tag value, <value>
- Illegal value <value1>
- Tag name <tag-name>

Potential Issues

This violation is reported when the following conditions hold true:

- Value of the [dftShowWaveForm](#) parameter is not set to on
- The [illegal_value](#) constraint uses the -matchNBits <value> argument under <tag-name> condition. If you have not specified the -matchNBits argument:
 - under a tag defined using define_tag constraint, all bits in the sequence are checked
 - under use_shift, use_capture, or use_captureATspeed, the last bit value is checked. See [Example 3](#) for more information.
- The value of the -matchNBits argument is greater than 1

The Conn_10 rule does not report this violation message if the tag is

defined using the *define_tag* command.

Consequences of Not Fixing

Not fixing the violation may result in unexpected code behavior.

How to Debug and Fix

To fix the violation, set the value of the *dftShowWaveForm* parameter to on.

Message 3

[INFO] <constraint_message_tag>Node ' <node_name>' has valid value ' <current_value>' (illegal: <disallowed_value>)' under <mode>

Arguments

- Name of the constraint_message_tag, if present, <constraint_message_tag>
- Name of the design node which does not have the illegal value (check PASSED), <node_name>
- Current value, <current_value>
- Disallowed value, <disallowed_value>
- Name of the 'mode' (simulation condition), <mode>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required.

Example Code and/or Schematic**Example 1**

Consider the following sample SGDC file describing the *illegal_value* constraint description:

```
current_design top
  illegal_value -name top.op1 -value 1 -use_shift
  test_mode -name in1 -value 1
  test_mode -name in2 -value 1
```

Figure 58 describes the schematic depicting the violation displayed for the

illegal_value constraint:

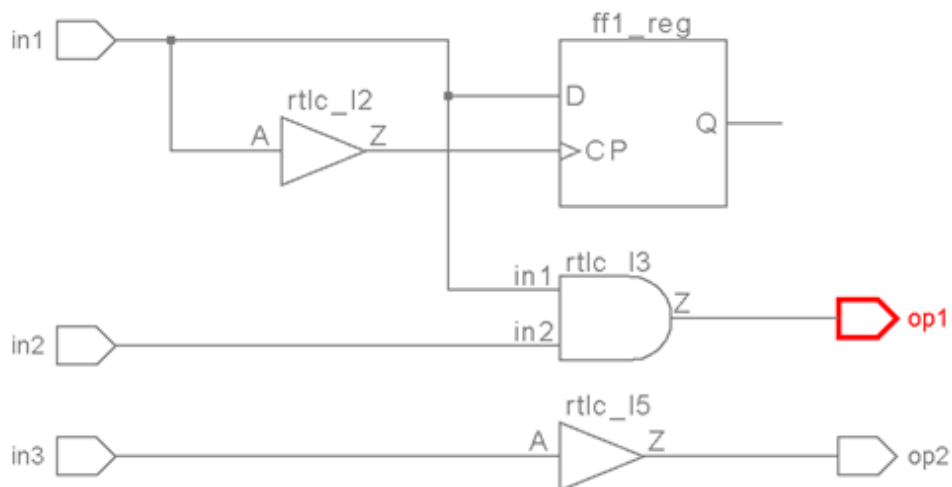


FIGURE 58. Violation for the *illegal_value* constraint

In the above design, in test mode the value at `top.op1` is 1. Since, it is defined as an illegal value in the above defined SGDC description, the *Conn_10* reports the following violation message for the above design:

```
Node 'top.op1' has illegal value '1' under tag
'use_shift', test.sgdc, 2
```

Example 2

This example illustrates the usage of the `-except` and `-except_type` arguments and the usage of the wildcard characters in the *illegal_value* constraint:

Consider the following sample SGDC file describing the *illegal_value* constraint description:

```
current_design top
testmode -name top.clk1 -value 0 -scanshift

illegal_value -type FLIP_FLOP_CLOCK -except_type
top.sub1_1.fl_2:TIED_0_SGDC -value 0 -use_shift -
constraint_message_tag m1
```

Connection Rules

```
illegal_value -name "top.sub1_1.f*.q_reg.CP" -except
top.sub1_1.f1_1.q_reg.CP -value 0 -use_shift -
constraint_message_tag m3
```

Figure 59 describes the schematic depicting the violation displayed for the *illegal_value* constraint having the *-except*, *-except_type* arguments:

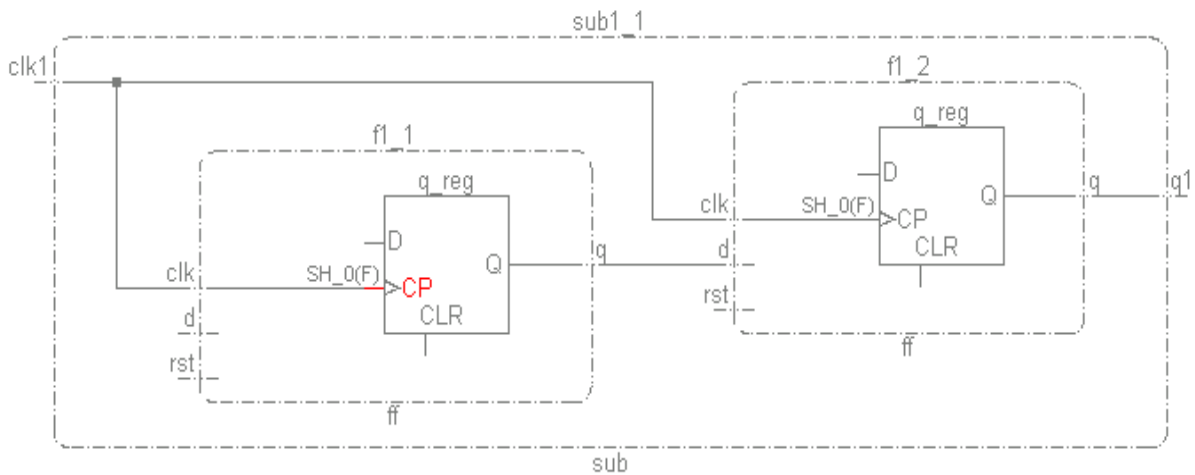


FIGURE 59. Violation for the *illegal_value* constraint

For the above design, for constraint_message_tag m1, the *Conn_10* rule reports the following violation message only for `top.sub1_1.f1_1.q_reg.CP` due to the presence of the *except_type* condition:

```
[constraint_message_tag: m1] Node 'top.sub1_1.f1_1.q_reg.CP
(FLIP_FLOP_CLOCK)' has illegal value '0' under tag '-
use_shi f', test.sgdc, 5
```

Also, for constraint_message_tag m2, the *Conn_10* rule reports the following violation message only for `top.sub1_1.f1_2.q_reg.CP` due to the presence of the *except* condition

```
[constraint_message_tag: m3] Node 'top.sub1_1.f1_2.q_reg.CP'
has illegal value '0' under tag '-use_shi f', test.sgdc, 6
```

Example 3

Consider the following SGDC description:

```
current_design top
```

```
illegal_value -name top.op1 -value 1 -tag t1
-constraint_message_tag m1
illegal_value -name top.op1 -value 1 -tag t1
-matchNBits 1 -constraint_message_tag m2
illegal_value -name top.op1 -value 1 -tag t2
-constraint_message_tag m3
illegal_value -name top.op1 -value 1 -tag t2 -matchNBits 1
-constraint_message_tag m4
illegal_value -name top.op1 -value 1 -use_shift
-constraint_message_tag m5
illegal_value -name top.op1 -value 1 -use_shift
-matchNBits 1 -constraint_message_tag m6

define_tag -tag t1 -name in1 -value 101
define_tag -tag t1 -name in2 -value 1
define_tag -tag t2 -name in1 -value 1
define_tag -tag t2 -name in2 -value 1

test_mode -name top.in1 -value 101
test_mode -name top.in2 -value 1
```

The Conn_10 rule reports the following violation message for the message tags, m2, m3, m4, m5, and m6:

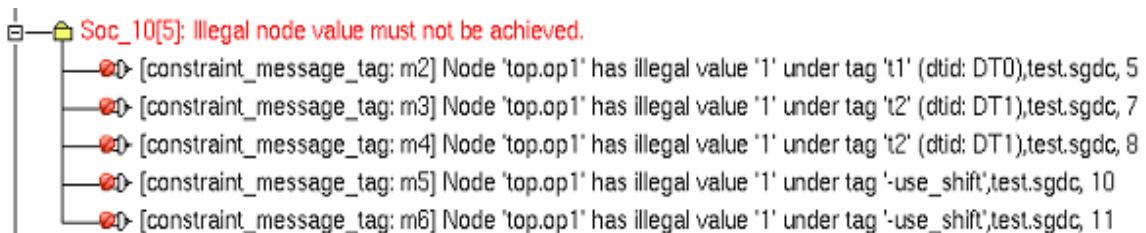


FIGURE 60. Violation messages for message tag, m5

The *Conn_10* rule does not report violation message for message tag, m1,

Connection Rules

because it is defined under tag, `t1`. This is because the value for `t1` is 101, which will be matched against `-value` field in the *illegal_value* constraint description for `m1`.

Default Severity Label

Error

Rule Group

SoC

Reports and Related Files

None

Conn_11

Node must satisfy the specified constraint message tag expression

When to use

Use this rule to perform conditional connectivity and value checks on a particular node.

Description

The Conn_11 reports violation, if the `constraint_message_tag_expression` specified using the [require_constraint_message_tag](#) is not met on the specified design node.

For information on the SGDC-based conditional connectivity check, see [Performing Conditional Connectivity Checks](#).

This rule must be run at the end of all other Soc rules so that message tagging must have happened by then.

You may also want to see the SGDC-Based conditional connectivity check.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- [module_bypass](#) (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- [require_constraint_message_tag](#) (mandatory): Use this constraint to define the `constraint_message_tag_expression`. The `constraint_message_tag_expression` must be one or a combination of `constraint_message_tags` of the following SGDC commands:
 - [require_path](#)
 - [require_value](#)
 - [require_strict_path](#)
 - [illegal_value](#)
 - [illegal_path](#)

Operating Mode

None

Messages & Suggested Fix:

Message 1

[ERROR] Node <node-name> does not have required constraint_message_tags <constraint_message_tag_expression>. Present: <constraint_message_tag>, missing: <constraint_message_tag>

Arguments

- These are nodes name present in the design, <node-name>
- Combination of constraint_message_tag_expression using operator '||' and '&&', <constraint_message_tag_expression>
- present and missing, <constraint_message_tag>

Potential Issues

The violation message appears, if at least one is missing from constraint_message_tag_expression.

Consequence of Not Fixing

Not fixing this violation may impact the functionality of the design

How To Debug and Fix

To fix this violation, perform any of the following tasks:

- Check the design for an error
- Check the SGDC file for a missing constraint

Message 2

[INFO] Node <node-name> has required <constraint_message_tag_expression>. Present: <constraint_message_tag>, missing: <constraint_message_tag>

Arguments

- These are nodes name present in the design, <node-name>
- Combination of constraint_message_tag_expression using operator '||' and '&&', <constraint_message_tag_expression>

- constraint_message_tag present and missing, <constraint_message_tag>

Potential Issues

The violation message appears, if at least one is missing from constraint_message_tag_expression.

Consequence of Not Fixing

Not fixing this violation may impact the functionality of the design

How To Debug and Fix

To fix this violation, perform any of the following steps:

- Check the design for an error
- Check the SGDC file for a missing constraint

Example Code and/or Schematic

Consider the following SGDC:

```
current_design top
```

```
clock -name clk -testclock
testmode -name rst -value 1
```

```
gating_cell -name wb_cgc -clkinTerm clkin -clkoutTerm clkout
-enTerm en -testenTerm te
```

```
require_path -from_type INPUT_PORTS -to_type
SCAN_FLIP_FLOP_DATA FLIP_FLOP_DATA LATCH_DATA MUX_SELECT -
constraint_message_tag PORT_CHECK
require_path -from "top.cgc_1.clkout" -to_type
FLIP_FLOP_CLOCK LATCH_ENABLE -constraint_message_tag
CGC_CHECK_1
require_path -from "top.cgc_2.clkout" -to_type
FLIP_FLOP_CLOCK LATCH_ENABLE -constraint_message_tag
CGC_CHECK_2
require_path -from_type LATCH_OUT -to_type FLIP_FLOP_RESET
-constraint_message_tag LATCH_CHECK -report_failure_as_info

illegal_path -from_type BLACK_BOX_OUTPUT -to_type
```


Connection Rules

```
FLIP_FLOP_DATA -constraint_message_tag BBOX_CHECK -
report_failure_as_info
```

```
require_value -name "top.l_1.out" -value 0
-constraint_message_tag LATCH_VALUE_CHECK
-report_failure_as_info
illegal_value -name "top.d_1.out" -value 1
-constraint_message_tag FLOP_VALUE_CHECK
-report_failure_as_info
```

```
require_constraint_message_tag -type LATCH
-constraint_message_tag_expression "LATCH_CHECK:PASS &&
LATCH_VALUE_CHECK:PASS"
require_constraint_message_tag -type LATCH
-constraint_message_tag_expression "LATCH_CHECK:FAIL ||
LATCH_VALUE_CHECK:FAIL"
```

For the above SGDC, the Conn_11 rule reports the following violation messages:

Message	Rule Severity	Description
Node 'top.l_1.temp_reg (LATCH)' does not have required constraint_message_tags 'LATCH_CHECK:PASS LATCH_VALUE_CHECK:PASS'. Present: 'none', Missing: 'LATCH_CHECK:PASS LATCH_VALUE_CHECK:PASS'	ERROR	In the above violation message the constraint_message_tag_expression, LATCH_CHECK:PASS LATCH_VALUE_CHECK:PASS, is not met for design node, top.l1. It means neither LATCH_CHECK = PASS nor LATCH_VALUE_CHECK = PASS constraint_message_tags are met. If anyone of them meets then this will be an info message.
Node 'top.l_1.temp_reg (LATCH)' has required constraint_message_tags 'LATCH_CHECK:FAIL LATCH_VALUE_CHECK:FAIL'. Present: 'LATCH_CHECK:FAIL', Missing: 'LATCH_VALUE_CHECK:FAIL'	INFO	In the above info message the constraint_message_tag_expression LATCH_CHECK:FAIL LATCH_VALUE_CHECK:FAIL is met for design node top.l1. It means LATCH_CHECK = FAIL constraint_message_tag is met and LATCH_VALUE_CHECK = FAIL constraint_message_tag is not met. As there is logical OR of constraint_message_tags so if anyone of them meets, info message for <i>Conn_11</i> is reported.

Default Severity Label

Info/Error

Rule Group

SoC

Conn_12

Node must not have the specified constraint message tag expression

When to Use

Use this rule to perform conditional connectivity and value checks on a particular design node.

Description

The Conn_12 rule reports violation, if the `constraint_message_tag_expression` specified using the [illegal_constraint_message_tag](#) is met on the design node. This rule must be run at the end of all other Soc rules so that message tagging must have happened by then.

For information on the SGDC-based conditional connectivity check, see [Performing Conditional Connectivity Checks](#).

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- [module_bypass](#) (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- [illegal_constraint_message_tag](#) (mandatory): Use this constraint to check whether the `constraint_message_tag_expression` matches the expression for the design nodes. If specific combination of tags are present in the design for a particular node then it reports violations. The `constraint_message_tag_expression` must be one or a combination of `constraint_message_tags` of the following SGDC commands:
 - [require_path](#)
 - [require_value](#)
 - [require_strict_path](#)
 - [illegal_value](#)
 - [illegal_path](#)

Operating Mode

None

Messages and Suggested Fix

Message 1

[ERROR] Node <node-name> has illegal constratin_message_tags <constraint_message_tag_expression>. Present: <constraint_message_tag>, missing: <constraint_message_tag>

Arguments

- These are nodes name present in the design, <node-name>
- Combination of constraint_message_tag_expression using operator '||' and '&&', <constraint_message_tag_expression>
- Constraint_message_tag present and missing, <constraint_message_tag>

Potential Issues

The violation message appears if at least one constraint_message_tag is missing from constraint_message_tag_expression

Consequence of not fixing

Not fixing this violation may impact the functionality of the design.

How To Debug And Fix

To fix this violation, perform any of the following tasks:

- Check the design for an error
- Check the SGDC file for a missing constraint

Message 2

[INFO] <constraint_message_tag>Node ' <node_name>' has valid value ' <current_value>' (illegal: <disallowed_value>) under <mode>

<constraint_message_tag>Node ' <node_name>' has valid constraint_message_tags. Disallowed: ' <disallowed_value>', Present: ' <current_value>', Missing: ' <missing_value>'

Arguments

- Name of the constraint_message_tag, if present, <constraint_message_tag>
- Name of the design node which does not have the illegal value (check PASSED), <node_name>
- Disallowed constraint_message_tag_expression, <disallowed_value>
- Constraint_message_tag which are found on the design node, <current_value>
- Constraint_message_tag which are missing on the design node, <missing_value>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required.

Example Code and/or Schematic

Consider the following SGDC:

```
current_design top
```

```
clock -name clk -testclock
testmode -name rst -value 1
```

```
gating_cell -name wb_cgc -clkInTerm clkin -clkoutTerm clkout
-enTerm en -testenTerm te
```

```
require_path -from_type INPUT_PORTS -to_type
SCAN_FLIP_FLOP_DATA FLIP_FLOP_DATA LATCH_DATA MUX_SELECT -
constraint_message_tag PORT_CHECK
```

```
require_path -from "top.cgc_1.clkout" -to_type
FLIP_FLOP_CLOCK LATCH_ENABLE -constraint_message_tag
```

```
CGC_CHECK_1
```

```
require_path -from "top.cgc_2.clkout" -to_type
FLIP_FLOP_CLOCK LATCH_ENABLE -constraint_message_tag
CGC_CHECK_2
```

```
require_path-from_typeLATCH_OUT-to_typeFLIP_FLOP_RESET
-constraint_message_tag LATCH_CHECK -report_failure_as_info
```

```
illegal_path -from_type BLACK_BOX_OUTPUT -to_type
FLIP_FLOP_DATA -constraint_message_tag BBOX_CHECK -
report_failure_as_info
```

```
require_value -name "top.l_1.out" -value 0
-constraint_message_tag LATCH_VALUE_CHECK
-report_failure_as_info
```

```
illegal_value -name "top.d_1.out" -value 1
-constraint_message_tag FLOP_VALUE_CHECK
-report_failure_as_info
```

```
illegal_constraint_message_tag -type ICG -
constraint_message_tag_expression "CGC_CHECK_1:PASS ||
CGC_CHECK_2:FAIL
```

```
illegal_constraint_message_tag -type FLIP_FLOP -
constraint_message_tag_expression "BBOX_CHECK:FAIL &&
PORT_CHECK:PASS"
```

For the above SGDC, the *Conn_12* rule reports the following error message:

```
Node 'top.cgc_2.clkout (ICG)' has illegal
constraint_message_tags 'CGC_CHECK_1:PASS || CGC_CHECK_2:FAIL'.
Present: 'CGC_CHECK_2:FAIL', Missing: 'CGC_CHECK_1:PASS'
```

In the above violation message the `constraint_message_tag CGC_CHECK_1:PASS || CGC_CHECK_2:FAIL` is met for design node `top.cgc_2`. It means in logical or operation of `CGC_CHECK_1:PASS` and `CGC_CHECK_2:FAIL`, the `constraint_message_tag CGC_CHECK_2`

Connection Rules

= FAIL is met.

Default Severity Label

Info/Error

Rule Group

SoC

Conn_14

Ensure that specified nets are having stable values under specified condition

When to Use

Use this rule to detect the sources of instability.

Description

The *Conn_14* rule reports violation for nets having unstable values under specified condition. That is, the *Conn_14* rule considers non-driven ports, black-boxes, and hanging nets as sources of instability, by default, along with scannable flip-flops / latches.

It reports violation for nodes that have unstable values, specified using the *require_stable_value* constraint.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dft_soc_unstable_value_sources*: Default value is all. Set the value of the parameter to none, blackbox, hanging_net, or port to specify unstable value sources, other than scannable flip-flops and latches, that needs to be reported by the *Conn_14* rule.
- *dft_block_unstable_value_trace_on_no_clock*: Default value is off. Set the value of the parameter to on to specify if the trace, for finding sources of unstable values, be stopped at a flip-flop / latch if it does not get a clock.
- *dft_stable_value_pessimistic_check*: Use this parameter to control the flow of stability check. The default value is accurate.
- *dft_internal_testmode_nodes_stability_check*: Default value is off. Set the value of the parameter to on to control the stability flow for internal nodes, which are constrained to a fixed value.

Constraint(s)

- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.

- *require_stable_value* (mandatory): Use this constraint to specify nodes whose value is expected to be stable.
- *force_stable_value* (optional): Use this constraint to specify nodes that will have a stable value during the scan shift and capture modes.
- *force_unstable_value* (optional): Use this constraint to specify nodes that will have an unstable value during the scan shift and capture modes.

Operating Mode

Scanshift, Capture, Power Ground, Define_tag

Messages and Suggested Fix

The following violation messages are reported when `-value` argument of the *require_stable_value* constraint is not specified:

Message 1: Unstable signal without '-value' specified

[ERROR] Node '`<node name>`' has unstable value '`<actual value>`' under tag '`<constraint_message_tag>`' due to unstable source(s) {'`<distribution>`' Scannable Flip-flop(s)}, SPREADSHEET_PATH: '`<path>`'

Arguments

- Path of the attached spreadsheet, which contains all the unstable sources impacting the specific node, `<path>`

Potential Issues

A violation message is reported when a node has unstable value under a specified condition.

Consequence of not fixing

Scan flip-flops are considered sources of corruption as their values toggle during the shift and capture mode. The impact of such corruption source propagate through multiple stages of non-scan flops in the fan-in logic cone of the target node (target for stability) until it reaches the node or it is blocked (for example, by the `test_mode` constraint).

How To Debug And Fix

Review the path from unstable sources to the design nodes specified using the *require_stable_value* constraint.

Message 2: Stable signal without '-value' specified

[INFO] Node '<node name>' has stable value '<value>' under tag '<tag name>'

Potential Issues

This is an informational message only.

Consequence of not fixing

This is an informational message only.

How To Debug And Fix

This is an informational message only.

The following violation messages are reported when `-value` argument of the `require_stable_value` constraint is specified:

Message 1: Unstable signal with '-value' specified

[ERROR] Node '<node name>' has unstable value (Initial: <actual value>, Required: <constraint value>) under tag '<constraint_message_tag>' due to unstable source(s) {'<distribution> Scannable Flip-flop(s) }, SPREADSHEET_PATH: '<path>'

Arguments

- Path of the attached spreadsheet, which contains all the unstable sources impacting the specific node, `<path>`

Potential Issues

A violation message is reported when a node has unstable value and the `require_stable_value -value` command is specified.

Consequence of not fixing

Scan flip-flops are considered sources of corruption as their values toggle during the shift and capture mode. The impact of such corruption source propagate through multiple stages of non-scan flops in the fan-in logic cone of the target node (target for stability) until it reaches the node or it is blocked (for example, by the `test_mode` constraint).

How To Debug And Fix

Review the path from unstable sources to the design nodes specified using the `require_stable_value` constraint.

Message 2: Stable signal with '-value' specified and non-conflicting case

[INFO] Node '<node name>' has stable value (Initial: <actual value>, Required: <constraint_value>) under tag '<constraint_message_tag>'

Potential Issues

This is an informational message only.

Consequence of not fixing

This is an informational message only.

How To Debug And Fix

This is an informational message only.

Message 3: Stable signal with '-value' specified and conflicting case

[ERROR] Node '<node name>' has incorrect stable value (Initial: <actual value>, Required: <constraint_value>) under tag '<constraint_message_tag>'

Potential Issues

A violation message is reported when a node has unstable value and the `require_stable_value -value` command is specified.

Consequence of not fixing

Scan flip-flops are considered sources of corruption as their values toggle during the shift and capture mode. The impact of such corruption source propagate through multiple stages of non-scan flops in the fan-in logic cone of the target node (target for stability) until it reaches the node or it is blocked (for example, by the `test_mode` constraint).

How To Debug And Fix

Review the path from unstable sources to the design nodes specified using the `require_stable_value` constraint.

Example Code and/or Schematic**Example 1**

- Consider the following example:

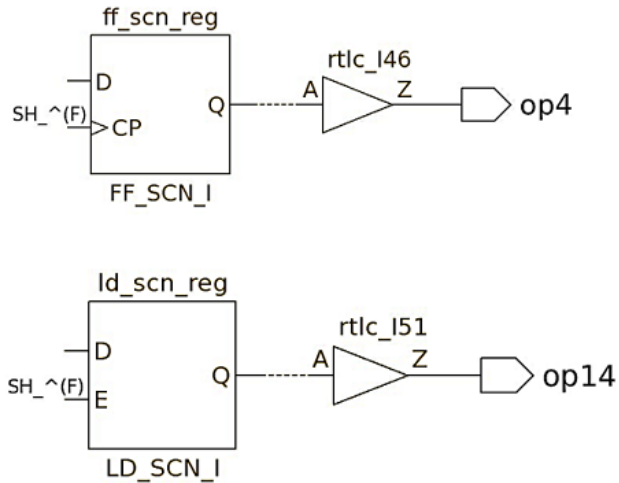


FIGURE 61. Scan Flip-Flop and Latch Driving the Output

Also, consider the following SGDC command:

```
require_stable_value -type OUTPUT_PORTS -use_shift
```

The Conn_14 rule reports violation for the above design scenario. This is because the scan flip-flop and latch are driving the output ports for which stability is needed.

Example 2

Consider the following example:

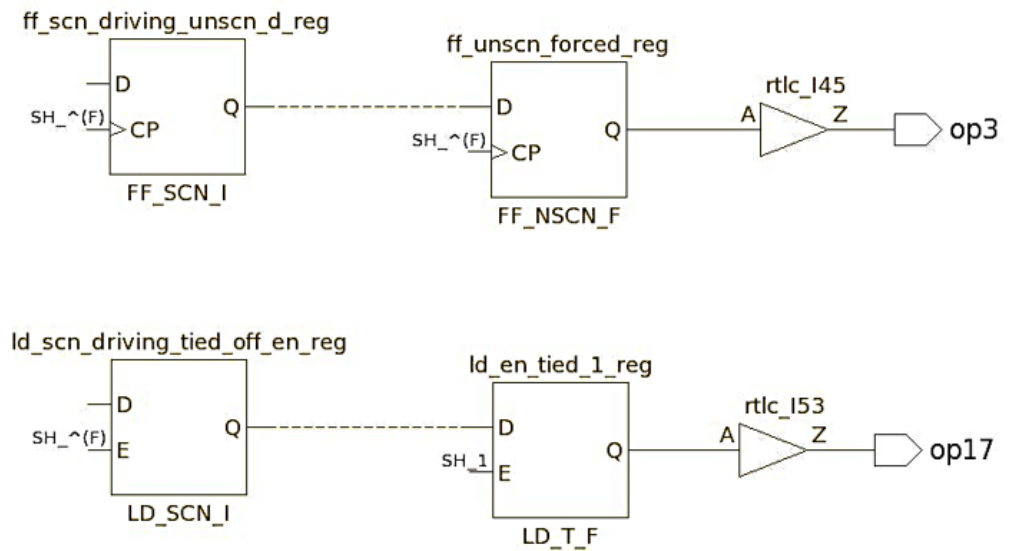


FIGURE 62. Scan Flip-Flop and Latch Driving the Output

Also, consider the following SGDC command:

```
require_stable_value -type OUTPUT_PORTS -use_shift
```

The Conn_14 rule reports violation for the above design scenario. This is because the scan flip-flop and latch is driving the output ports, for which stability is needed, via data-pin of unscannable flip-flop / transparent latch.

Example 3

Consider the following example:

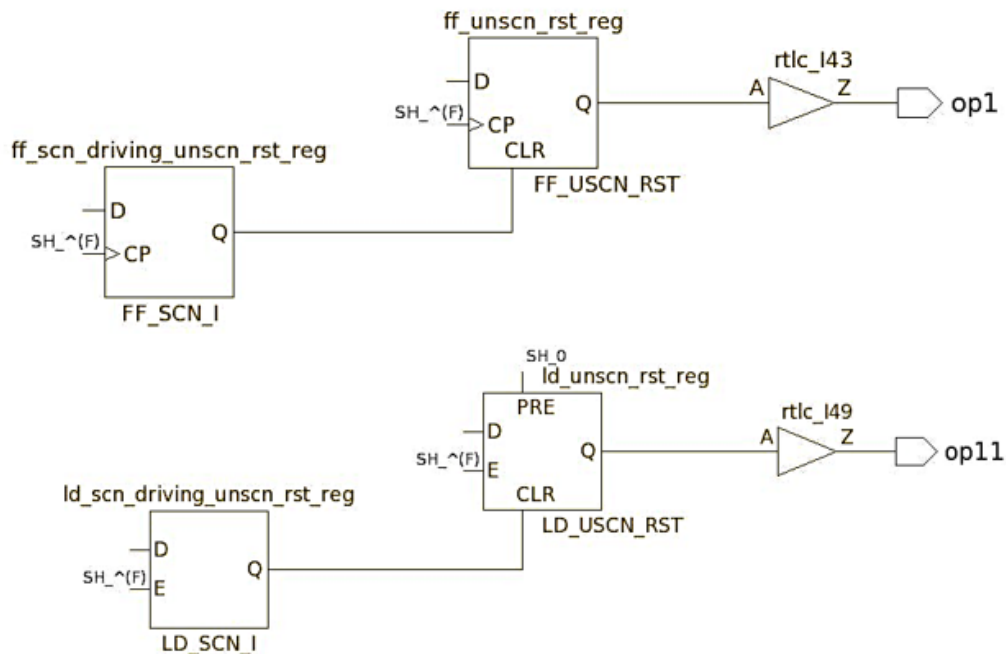


FIGURE 63. Scan Flip-Flop and Latch Driving the Output

Also, consider the following SGDC command:

```
require_stable_value -type OUTPUT_PORTS -use_shift
```

The Conn_14 rule reports violation for the above design scenario. This is because the scan flip-flop and latch is driving the output ports, for which stability is needed, via reset of unscannable flip-flop / latch.

Example 4

Consider the following example:

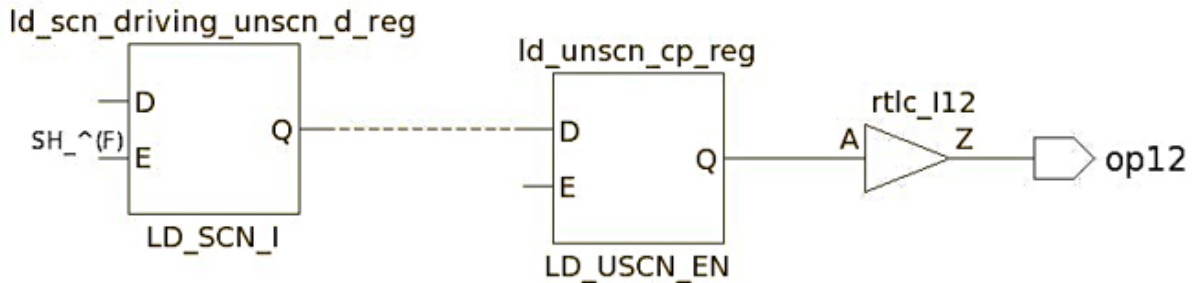


FIGURE 64. Scan Flip-Flop and Latch Driving the Output

Also, consider the following SGDC command:

```
require_stable_value -type OUTPUT_PORTS -use_shift
```

Also, assume that the value of the `dft_block_value_trace_on_no_clock` parameter is set to off.

The Conn_14 rule reports violation for the above design scenario. This is because the scannable latch is driving the output port, for which stability is needed, via data-pin of unscannable latch for which EN-pin is neither inactive nor does it get a clock.

Example 5

Consider the following example:

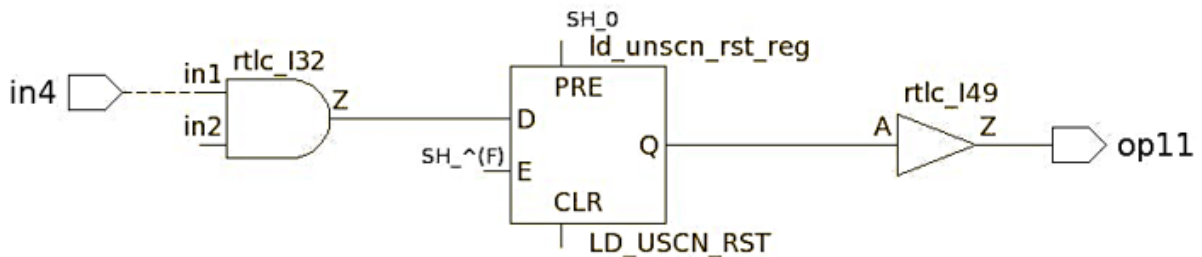


FIGURE 65. Scan Flip-Flop and Latch Driving the Output

Also, consider the following SGDC command:

```
require_stable_value -type OUTPUT_PORTS -use_shift
```

Also, assume that the value of the `dft_soc_unstable_value_sources` parameter is set to all.

The Conn_14 rule reports violation for the above design scenario. This is because the primary input port is driving the output port, for which stability is needed, via data-pin of unscannable latch.

Default Severity Label

Error

Rule Group

Connection

Conn_15

Check required pulse pattern at specified node.

Rule Description

The *Conn_15* rule reports a violation, if the simulated result pattern at a specified node does not match the expected pulse pattern.

Constraints

- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *require_pulse* (mandatory): Use this constraint to define a check that requires a pulse sequence to be established on a certain node, when the circuit is simulated using the condition specified by the *-tag* argument.
- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Rule Parameters

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*

Operating Mode

Define_tag

Message Details

Message 1

[constraint_message_tag: <value>] Node <node_name> (specified as -type) has mismatched value <actual_pulse_pattern> between <after_bit> and <before_bit> bits under tag <tag_name>. [REASON: <reason>]. Expected pulse pattern is <expected_pulse_pattern>

Message 2

[constraint_message_tag: <value>] Node <node_name> (specified as -name) has mismatched value <actual_pulse_pattern> between <after_bit> and <before_bit> bits under tag <tag_name>.

[REASON: <reason>]. Expected pulse pattern is <expected_pulse_pattern>

Message 3

[constraint_message_tag: <value>] Ignoring require_pulse constraint as neither -name nor -type field is specified.

Message 4

[constraint_message_tag: <value>] Node <node_name> (specified as -type) received <number_of_pulses> (<pulse_pattern>) under tag <tag_name>

Message 5

[constraint_message_tag: <value>] Node <node_name> (specified as -name) received <number_of_pulses> (<pulse_pattern>) under tag <tag_name>

Message 6

[constraint_message_tag: <value>] Ignoring require_pulse constraint as neither -name nor -type field is specified

Arguments

- Constraint tag value, <value>
- Name of net/heir term where pulse is expected, <node_name>
- Number of pulses obtained, <number_of_pulses>
- Pulse pattern obtained between after and before bits, <pulse_pattern>
- Tag name of the define_tag constraint under which simulation is to be done, <tag_name>

Location

The file and the line where the *require_pulse* constraint is specified

Schematic highlight

The schematic for the Conn_15 rule highlights the following:

- Specified node with obtained pulse pattern.
- Expected pulse pattern between after and before bits, specified in the *require_pulse* constraint on the net

Actual simulation value between the after and before bits resulting from the simulation of the defineTag condition specified in the `require_pulse` constraint.

Arguments

- Constraint tag value, <value>
- Name of the net or heir term where pulse is expected, <node_name>
- Pulse pattern obtained at specified node, <actual_pulse_pattern>
- After bit as specified in [require_pulse](#) constraint, <after_bit>
- Before bit as specified in `require_pulse` constraint, <before_bit>
- Tag_name of the define_tag constraint under which simulation is to be done, <tag_name>
- Short description of reason for mismatch, <reason>. The following lists the possible reasons for mismatch;
 - No pulse found, in case there are don't care bits present during the pulse
 - Pulse's low-width is too long
 - Pulse's low-width is too short
 - Pulse's high-width is too long
 - Pulse's high-width is too short
 - Mismatch in rear padding
- Expected pulse pattern as specified in the `require_pulse` constraint, <expected_pulse_pattern>

Location

The file and the line where the `require_pulse` constraint is specified.

Schematic highlight

Specified node with the obtained and the expected value.

Rule Severity

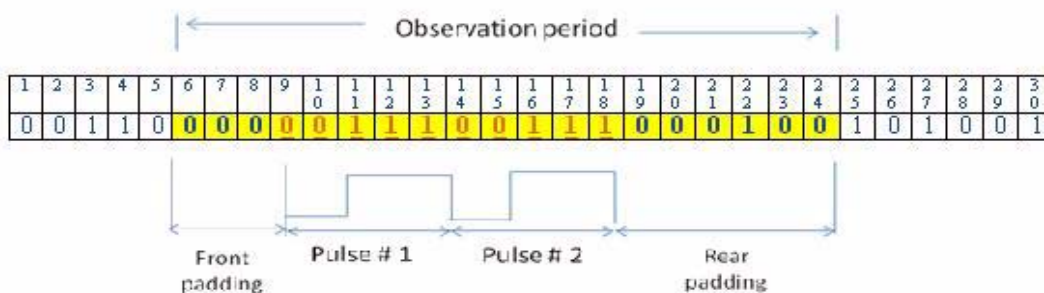
Warning

Example

The following example illustrates the padding. Consider the following definition of `require_pulse` constraint:

```
require_pulse -name xyz -tag sim_1\\two pulses required
  -pulse 2
  -after 5 -before 25 \\observation period
  -high_width 3 -low_width 2 \\pulse characteristics
```

Also, consider the following obtained pulse:



The bits in bold are in pulse observation period (-after 5, -before 25). The underline bits are the actual pulses, which are same as described by the `require_pulse` constraint (2 pulses of 3 high bit and 2 low bit).

However, the above pulse causes a violation because there is a mismatch in rear padding at bit 22. Rear padding starts after complete pulse is matched and ends at the end of observation period. So, there should not be any pulse in rear padding. If bit 22 is 0, then Conn_15 does not report any violation. Instead, the Info_Conn_15 violation message is generated stating that two pulses are obtained during this period.

In case the `require_pulse` is constraint defined with `-count 2`, `-high_width 3`, and `-low_width 2`, there are following two possible pulses to match depending on whether you start from low bits or high bits:

```
0 0 1 1 1 0 0 1 1 1
1 1 1 0 0 1 1 1 0 0
```

In the above example, during pulse observation window, the first transition is encountered at bit 11. So, bits from 6 to 10 are part of front padding. Since front padding is sufficiently large to accommodate `low_width`, we

assume pulse starts from bit 9 with low bits first and ends at bit 18.

If there are insufficient number of bits in front padding, then bit 11 is considered as pulse starting point.

NOTE: *There is no limit on length of front padding or rear padding. However, ensure that there is no transition in the rear padding.*

Also, if the design generates more pattern than expected, then there is a mismatch in rear padding. To avoid the Conn_15 violation, reduce the length of rear padding by adjusting `-before <value>`.

Diagnostic Rules

Overview

The `Diagnostic` rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>Diagnose_02</i>	Number of faults that are blocked under capture at-speed mode simulation conditions in the design
<i>Diagnose_03</i>	Number of faults that are blocked by false paths or multi-cycle paths in the design
<i>Diagnose_04</i>	Number of faults that cannot be tested in capture at-speed mode due to crossing of clock domains
<i>Diagnose_testmode</i>	Display instances that block the testmode propagation.
<i>Diagnose_testclock</i>	Display instances that block the testclock propagation.

Diagnose_02

Ensure that the at-speed paths are not blocked by testmode signals

When to Use

Use this rule to identify the faults that are blocked in capture at-speed mode.

Description

The `Diagnose_02` rule reports the number of faults that are blocked under capture at-speed mode simulation conditions in the design.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dsmUsePIForAtSpeed](#): The default value is on. Set the value of the parameter to off to disable usage of primary inputs to launch transition fault tests.
- [dsmUsePOForAtSpeed](#): The default value is on. Set the value of the parameter to off to disable usage of primary outputs to capture transition test.

Constraint(s)

- [atspeed_clock_frequency](#) (optional): Use this constraint to specify frequencies associated with a testclock.
- [clock](#) (optional): Use this constraint to declare clock pins declared as testclocks.
- [clock_shaper](#) (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- [pll](#) (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the Diagnose_02 rule:

[WARNING] <num> fault(s) in <du-name> are blocked under capture at-speed mode

Arguments

- Number of faults blocked <num>
- Name of the top design <du-name>

Potential Issues

Violation may arise due to one of the following reasons:

- Improper atspeed condition
- Missing observation points

Consequences of Not Fixing

Not fixing the violation may result in low transition coverage.

How to Debug and Fix

View the Modular Schematic of the violation message. The Modular Schematic highlights the terminals, which are blocked due to capture atspeed conditions. Click a terminal and view the Incremental Schematic.

Overlay the Info_testmode (auxiliary violation mode) rule under capture atspeed mode to check the reason for the blocked terminal.

To fix the violation, add observation points.

Example Code and/or Schematic

Currently Unavailable

Diagnostic Rules

Default Severity**Label**

Warning

Rule Group

Diagnostic

**Reports and
Related Files**

No related reports or files.

Diagnose_03

Ensure that the faults are not blocked by false paths or multi-cycle paths

When to Use

Use this rule to identify the faults blocked in false paths or multi-cycle paths.

Description

The `Diagnose_03` rule reports the number of faults that are blocked by false paths or multi-cycle paths in the design.

Prerequisites

This rule expects either a `false_path` constraint or an SDC file (supplied by `sdc_data` constraint) with `set_false_path/`
`set_multicycle_path` constraints.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dsmUsePIForAtSpeed](#): The default value is on. Set the value of the parameter to off to disable usage of primary inputs to launch transition fault tests.
- [dsmUsePOForAtSpeed](#): The default value is on. Set the value of the parameter to off to disable usage of primary outputs to capture transition test.

Constraint(s)

- [atspeed_clock_frequency](#) (optional): Use this constraint to specify frequencies associated with a testclock.

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *false_path* (mandatory): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.

Operating Mode

Capture (atspeed)

Message Details

Message

[WARNING] <num> fault(s) in <du-name> are blocked by false paths or multi-cycle paths

Arguments

- Number of faults blocked, <num>
- Name of the top design, <du-name>

Potential Issues

Violation may arise because of large number of faults on the false path.

Consequences of Not Fixing

Large number of faults on the false path reduces the transition coverage.

How to Debug and Fix

View the Modular Schematic of the violation message. The Modular Schematic highlights the terminals, which are in false path. Click a terminal and view the Incremental Schematic.

Probe the fan-in/fan-out cone of the terminal to see how the path is originating/terminating at nets specified as from/to as false path.

To fix the violation, ensure that there are lesser number of faults on the

false path.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Diagnostic

Reports and Related Files

No related reports or files.

Diagnose_04

Identifies the faults in paths crossing clock domains that cannot be tested

When to Use

Use this rule to identify the cut set for domain crossings.

Rule Description

The `Diagnose_04` rule reports the number of faults that cannot be tested in at-speed mode due to crossing of clock domains. This rule highlights a user specified number of paths between clock domains.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *false_path* (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the *Diagnose_04* rule:

[WARNING] <num> fault(s) in <du-name> cannot be at-speed tested due to clock domain crossing

Arguments

- Number of faults in domain crossing, <num>
- Name of the top design, <du-name>

Potential Issues

A violation is reported due to one of the following reasons:

- Presence of different clocks
- Path is blocked in atspeed mode

Consequences of Not Fixing

If you do not fix the violation, faults in domain-crossing logic are detected as are not tested.

How to Debug and Fix

View the Modular Schematic of the violation message. The Modular Schematic highlights the terminals, which are in clock domain cross over path. Click a terminal and view the Incremental schematic.

Probe the fan-in cone of the terminal until it hits a flip-flop, for example, A.

Probe the fan-out cone of the terminal until it hits a flip-flop, for example, B.

Also, probe the clock cone of the flip-flops, A and B, to verify the relation between the two flip-flops.

To fix the violation, see [Example Code and/or Schematic](#).

Example Code and/or Schematic

Consider the circuit shown in the figure below and assume that flip-flops

FF1 and FF2 can launch at-speed tests. Also assume that clk1 and clk2 are asynchronous.

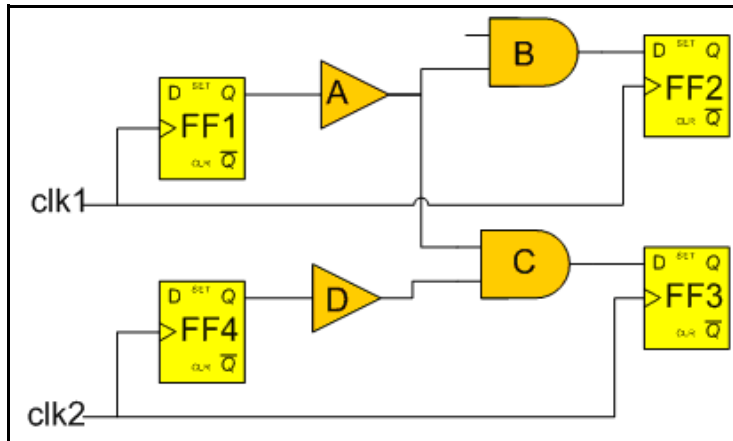


FIGURE 66. Clock Domain Crossing Example

In this case, the rule identifies the connection from A to C as crossing domains. At-speed faults in the path from A to B are fully covered but faults on the pin on C that is driven by A are not at-speed testable as this requires synchronized clock pulses in two different domains.

Since FF4 and FF3 are in the same domain, output faults in C (and the input faults on the in driven by D) are at-speed testable.

Default Severity

Label

Warning

Rule Group

Diagnostic Rules

Reports and Related Files

No related reports or files.

Information Rules

Overview

Information rules, instead of doing any DFT/ATPG specific rule check, provide some design statistics that may be useful for debug. In addition, these rules also perform some sanity check on its input arguments.

The Information rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>Coverage_audit</i>	Analyzes coverage for circuit
<i>CreateDebugSGDC</i>	Creates an SGDC file for the specified instance
<i>Diagnose_testclock</i>	Displays combinational objects that block testclock signal propagation in shift and capture modes
<i>Diagnose_testmode</i>	Displays combinational objects that block testmode signal propagation in shift and capture modes
<i>Info_atpg_conflict</i>	Reports testpoints to reduce conflicts during test pattern generation by ATPG.
<i>Info_addFault</i>	Reports all the ports and pins which are specified as add fault
<i>Info_atSpeedClock</i>	Displays at-speed test clock propagation
<i>Info_atSpeedClockSynchronization</i>	Provides info on atspeed clock pairs that will benefit most from synchronization
<i>Info_atSpeedCoverage</i>	Rule has been deprecated
<i>Info_atSpeedDomain</i>	Displays at-speed test clock domain propagation
<i>Info_atSpeedFrequency</i>	Displays atspeed frequency propagation.
<i>Info_atSpeedFrequency_EnabledConflict</i>	Report conflict of simulation value with enabling value given in constraint file.
<i>Info_blackboxDriver</i>	Reports black box drivers
<i>Info_coverage</i>	Estimates fault and test coverage

Information Rules

Rule	Description
<i>Info_dBist</i>	Display simulation values of design in DBIST mode
<i>Info_define_tag</i>	Shows system state for a given tag.
<i>Info_dft_deprecated</i>	Displays info messages for all the rules, parameters and constraints which will be deprecated future but are enabled or used in the current run
<i>Info_DftDebugData</i>	Reports dft debug data (schematic and excel sheet)
<i>Info_dftmax_configuration</i>	Enables the planning and configuration of DFTMAX Ultra in an SoC
<i>Info_enabledFlops</i>	Lists the flip-flops in the design, sorted by their respective clock domains
<i>Info_forcedScan</i>	Displays all registers and flip-flops specified as 'scan'
<i>Info_freqAssignTable</i>	Generates a frequency assignment report
<i>Info_inferredNoScan</i>	Displays inferred noscan flip-flops
<i>Info_IP_Report</i>	Generates boundary report on the IPs specified using ip_block SGDC command.
<i>Info_Latch</i>	Reports status of latches in the design
<i>Info_latchMapping</i>	Displays latches in a testclock domain and their transparency status in testmode
<i>Info_logicalRedundant</i>	Displays logically redundant faults
<i>Info_memories</i>	Generates a report on memory-instances present in a design
<i>Info_memoryforce</i>	Displays memoryforced pins
<i>Info_memorywritedisable</i>	Displays memorywritedisabled pin
<i>Info_noAtspeed</i>	Displays all registers and flip-flops specified as 'no_atspeed'.
<i>Info_noFault</i>	Reports all the ports and pins which are specified as no fault
<i>Info_noScan</i>	Displays flip-flops that have been specified with the <code>force_no_scan</code> constraint in the SpyGlass Design Constraints file(s)
<i>Info_noscanFlopsTextReport</i>	Reports scan flip-flops that have a constant outputs during the shift mode

Rule	Description
<i>Info_path</i>	Displays blocked paths in instances
<i>Info_potDetectable</i>	Display faults that are potentially detectable
<i>Info_pwrGndSim</i>	Displays simulation results of power-ground conditions
<i>Info_random_resistance</i>	Estimates random pattern fault and test coverage for design
<i>Info_schain</i>	Displays all properly stitched scan chains
<i>Info_scanwrap</i>	Report scanwrap related information
<i>Info_self_gating_logic</i>	Recognizes self-gating cells and identifies the test-points in self-gating logic.
<i>Info_soft_error_propagation</i>	Generates the soft error related metrics.
<i>Info_stilFile</i>	Generates a STIL file.
<i>Info_stil_to_sgdc</i>	Generates an SGDC file corresponding to a Stil file.
<i>Info_synthRedundant</i>	Displays synthesis-redundant terminals
<i>Info_testclock</i>	Displays testclock conditions in the design
<i>Info_testCounts</i>	This rule has been deprecated.
<i>Info_testmode</i>	Displays all signals with non-x values in testmode for both scanshift and capture
<i>Info_testmode_conflict_01</i>	Performs bottom up hierarchical migration of block level constraints
<i>Info_Top_SGDC_Report</i>	Evaluates and reports transition coverage for design
<i>Info_transitionCoverage</i>	Evaluate upper bound of transition coverage for design
<i>Info_transitionCoverage_audit</i>	Analyzes the root causes for low transition coverage
<i>Info_uncontrollable</i>	Displays nets with imperfect controllability
<i>Info_undetectedCause</i>	Reports the number of undetectable faults and displays pin controllability and pin observability back onto the structural view
<i>Info_unobservable</i>	Reports all unobservable pins
<i>Info_untestable</i>	Displays faults that can impact system performance but are blocked by the testmode logic

Information Rules

Rule	Description
<i>Info_unstable_testmode_registers</i>	which are unable to retain their respective initialized 0/1 value during scan shift operation and create unstable test mode conditions.
<i>Info_unused</i>	Displays faults that have no impact on system performance
<i>Info_x_sources</i>	Reports all the x-sources related information for the design

Coverage_audit

Analyze coverage for circuit.

When to Use

Use this rule to generate fault coverage and test coverage reports.

Description

The Coverage_audit rule reports the fault coverage and test coverage for specified types of objects when the test coverage of the design is not 100%.

The rule detects the causes and effects of low stuck-at coverage.

The rule generates the *stuck_at_coverage_audit* report that lists reasons when coverage is less than 100%. The report displays the test and fault coverage figures along with new detectable faults count.

For the purpose of coverage computation, the rule ignores the causes of the low stuck-at coverage.

Prerequisites

At least one system clock or at least one testclock must be specified for the Clock_21 rule to run.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *ATPG_credit*: The default value of the parameter is 0. Set the value of the parameter to any floating-point value between 0 and 1 to regulate the effect of untestable fault count on test-coverage percentage evaluation.
- *dftTreatControllableLatchTransparentForLoop*: The default value is off. Set the value of the parameter to on to treat controllable latches as transparent for the Coverage_audit rule.

Information Rules

- *dft_detect_shadow_latches*: The default value is on. Set the value of the parameter to off to treat all the latches normally.
- *dft_show_rule_name_in_audit*: The default value is on. Set the value of the parameter to off to disable printing the rule name for the steps in the stuck_at_coverage_audit report, for which no action is required.
- *dft_show_schematic_info_in_coverage_audit*: The default value is off. Set the value of the parameter to on to print the control (CNT), observe (OBS), and stuck-at (STUCK_AT) related information in the schematic for the Coverage_audit rule.
- *dft_generate_robustness_audit_report*: The default value is off. Set the value of the parameter to on to generate robustness audit report.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_treat_primary_inputs_as_x_source*: The default value is off. Set the value of the parameter to on to consider primary input ports as X-source and report violations for the same.
- *dft_treat_primary_outputs_as_unobservable*: The default value is off. Set the value of the parameter to on to treat the primary output ports as unobservable.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *non_pd_inputcells* (optional): Use this constraint to specify the cells that should not be present at the input stage of the power/voltage domain.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.

- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Coverage_audit rule:

```
[INFO] For Design Unit '<du-name>' [<category>], Fault-Coverage = <num1> %, Test-Coverage = <num2> %
```

The number of undetectable faults are <num>

Arguments

- Name of the design unit. <du-name>
- Category (Steps 1 to 9 as mentioned earlier. For example, Step 6: All testmode pins made controllable in capture mode). <category>
- Fault coverage percentage. <num1>
- Test coverage percentage. <num2>
- Number of undetectable faults. <num>

Potential Issues

Following are the primary reasons for coverage data not reaching 100%:

- Non-scan flip-flops
- Non-transparent latches
- Non-scanwrapped black box instances
- Tristate enables without observable testpoint
- Hanging Nets
- Testmode values in capture
- noscan flip-flops
- Uncontrollable primary inputs and unobservable primary outputs

- If test coverage of the design is not 100%, then at that point all the undetected faults are considered undetectable and test coverage reaches 100%

For each category, the fault coverage and the test coverage reported is the cumulative effect of the current category and all the previously considered categories (in the order listed above.)

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation.

How to Debug and Fix

The Coverage_audit rule is an informative rule and requires no debug. However, you can increase the coverage by performing the following steps:

- Making all flip-flops scannable (except noscan flip-flops)
- Making all latches transparent
- Making all black boxes scanwrapped
- Making all nodes which have constant value as 'x'
- Making all enable pins of tristate as observable
- Making all undriven net as controllable
- Making noscan flip-flops scannable
- Making all primary inputs controllable and primary outputs observable

No fix is required as this is an informational rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

[*stuck_at_coverage_audit*](#)

CreateDebugSGDC

Create an SGDC file for all the user-specified instances.

When to Use

Use this rule when top-down methodology is suggested.

Description

The CreateDebugSGDC rule generates a SpyGlass Design Constraints (SGDC) file for the specified instance using the *debugInst* rule parameter. For all the non-existing instances a violation is flagged.

However, for all user-specified instances, the CreateDebugSGDC rule creates an SGDC file (visible in the GUI RTL window) containing the testmode and testclock conditions as `test_mode` constraint and `clock` constraints with `-testclock` argument respectively.

NOTE: Use the *Info_IP_Report* for generating lower block `sgdc` command or report.

You can then use this SGDC file in another run when the instance's master has been made the top-module to reproduce the same conditions as before. If one or more of the instances have the same master, different current design sections are created in the SGDC file so that each instance can be debugged separately.

Method

Simulate testmode for scan shift and then for capture.

Simulate all testclocks in parallel.

Monitor values on the inputs of all user-specified instances and format in an SGDC file.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *debugInst*: The default value is a non-empty string with text 'nothing'. Set the value of the parameter to any string to identify instances for

which a SpyGlass Constraint file should be automatically generated by the *CreateDebugSGDC* file.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design. The Clock_24 rule uses the `-testclock` argument of the `clock` constraint.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] Hierarchical name of Instance '<inst_name>' must be specified in the design '<du_name>'

Arguments

- Specified instance name, <inst_name>
- Design unit name, <du_name>

Potential Issues

The violation message is reported when the hierarchical name of the instance is not specified in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation.

How to Debug and Fix

This is an informational rule. Therefore, no debugging is required.

Ensure that the file contains the correct instances.

Message 2

[WARNING] Instance <inst_name> does not exist in the design <du_name>

Arguments

- Specified instance name, <inst_name>
- Design unit name, <du_name>

Potential Issues

This message arises if a specified instance does not exist in the design.

Consequences of Not Fixing

If you do not fix this violation, it results in error and time consumption.

How to Debug and Fix

This is an informational rule. Therefore, no debugging is required.

Ensure that specified instances are correct in the design.

Message 3

[INFO] Debug SGDC file, created at <file_location> location.

Potential Issues

This is an informational rule and a debug SGDC file is created at the file location. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule and it informs about the generation of the file. Therefore, there are no direct consequences of not fixing the violation.

How to Debug and Fix

This is an informational rule. Therefore, no debugging is required.

Ensure that the file contains the correct instances.

Example Code and/or Schematic

This is an informational rule and generates an SGDC file. See [Reports and Related Files](#) section for more information on this file.

Default Severity Label

Warning

Rule Group

Information

Reports and Related Files

dft_debug_inst.sgdc

For all user-specified instances, an SGDC file named `dft_debug_inst.sgdc` is generated in the `spyglass_reports/dft/` directory. Specify the following commands to generate the `dft_debug_inst.sgdc` file:

```
current_methodology $SPYGLASS_HOME/Methodology/DFT
current_goal dft_scan_ready
set_goal_option addrules CreateDebugSGDC
set_parameter debugInst "top.dfl top.df2"
```

The following describes a sample `dft_debug_inst.sgdc` file.

```
#####
# current_design dff_normal
current_design top.df2
clock -name clk -value rtz -testclock current_design
top.dfl
test_mode -name data -value 1 -scanshift
test_mode -name data -value 1 -capture
test_mode -name data -value 1 -captureATspeed
clock -name clk -value rtz -testclock
# current_design top
#####
```

The generated SGDC file has the `test_mode` and `clock` (with `-testclock` argument) constraints for the instance with the master design unit as the `current_design`.

Diagnose_testclock

Display instances that block the testclock propagation.

When to Use

Use this rule to identify devices that block testclock signal propagation.

Description

The Diagnose_testclock rule generates the Atrenta Console highlight information for devices that block testclock signal propagation.

The Diagnose_testclock rule generates separate displays for scan shift mode and capture mode.

Testmode simulation is performed on any available testmode constraints.

If one or more `clock` constraints with `-testclock` argument are present, then the combinational instances that block the testclock signal in shift or capture mode are highlighted.

Rule Exceptions

Following are the exceptions to the Diagnose_testclock rule behavior:

- If blockage point does not drive a clock point (flip-flop, flipflop_bank, latch, memory, and black box clock pin)
- If the instance at blockage has a clock on its output through either other input or forced test clock

See [Example 2](#) for details.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftInferredDriverControl*: The default value is optimistic. Set the value of the parameter to pessimistic. Therefore, the presence of a single non-scan source (black box, hanging net, noscan) in the unblocked fan-in cone will make the node uncontrollable.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.

Operating Mode

Scanshift, Capture

Message Details

Message 1

When no blockage is found for a testclock, the Diagnose_testclock rule generates the following message with severity Info:

[INFO] Testclock '<signal-name>' propagates through all instances in '<du-name>'

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

When a blocked testclock signal <sig-name> has been found in a design unit, the Diagnose_testclock rule generates the following message with severity Warning:

[WARNING] Instances through which testclock '<sig-name>'

doesn't propagate in <mode-name> mode

Where, <mode-name> can be shift or capture.

Potential Issues

This violation is reported because one of the following is missing in the design (RTL or SGDC file):

- The clock-enabling logic
- The *test_mode* constraint specification

Consequences of Not Fixing

The clock will not be able to reach desired places, making flip-flops not ready for scan and latches non-transparent. This will result in reduced fault/test coverage.

How to Debug and Fix

View the incremental schematic for the violation message and check if valid values are specified for the clock-enable pins.

Message 3

In case, there are no clock constraints with `-testclock` argument specified, the *Diagnose_testclock* rule generates the following message with severity Info:

```
[INFO] Constraint 'clock -testclock' missing in design '<du-name>'
```

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the terminal blocking the testclock propagation under the shift/capture mode.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number

of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

You can also view the violations for the Info_testmode and Info_testclock rules along with the violation of the Diagnose_testclock rule in the Incremental Schematic window. To do this, double-click the violation message for the Diagnose_testclock rule and open the Incremental Schematic window.

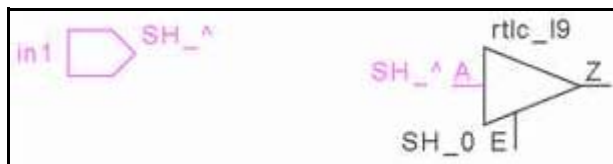
The violation messages for the Info_testmode and Info_testclock rules overlap the violation for the Diagnose_testclock rule in the Incremental Schematic window. This is useful in debugging the violation for the Diagnose_testclock rule.

Overlay (Auxiliary violation mode) the Info_testmode rule under the shift/capture mode. This helps in identifying the reason for the blocked testclock.

Example Code and/or Schematic

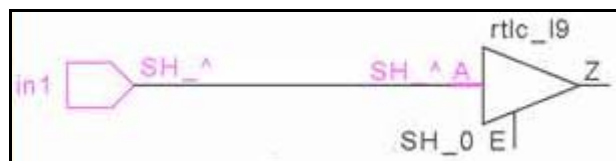
Example 1

Consider the following schematic in the shift mode:



In the above schematic, SH_0 on the enable pin, E, of the rtlc_19 tristate signifies that the E pin has value 0 in the shift mode. In addition, SH_^ on the A pin and the in1 port signifies that the clock (^) reaches at the in1 port and the A pin.

If you trace the fan-in cone of the A pin, it will lead to the user-specified test clock (via combinational logic, if present), as shown in the following schematic:

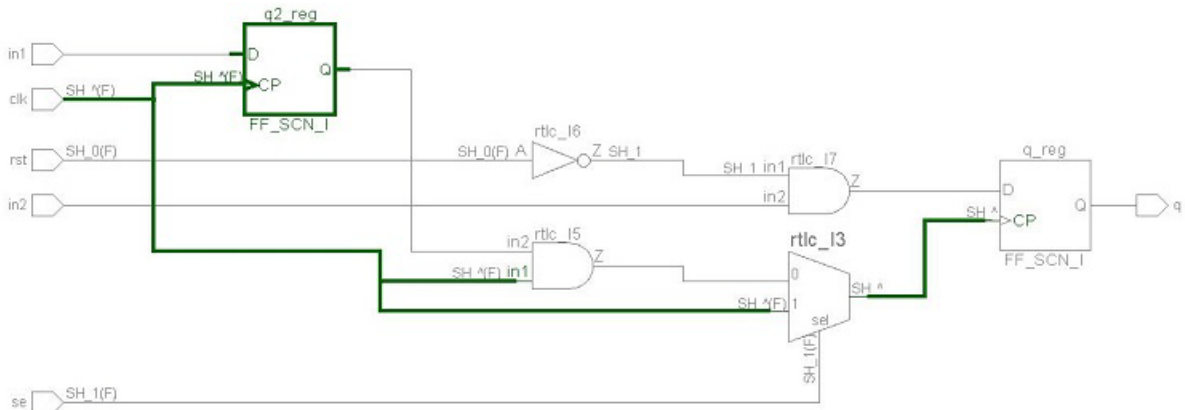


Information Rules

Similarly, the following schematic is shown in the capture mode:

**Example 2**

The following figure explains the exception to the behavior of the Diagnose_Testclock rule:



In the above schematic, you can not use the output of the AND gate, rtl_15, as a testclock as it does not drive any flip-flop clock pins. Therefore, to bypass the gated clock and ensure that the testclock always reaches the flip-flop clock pins, the MUX, rtlc_13, is added.

Default Severity Label

Info/Warning

Rule Group

Informational Rules

Reports and Related Files

No related reports and files.

Diagnose_testmode

Display instances that block the testmode propagation.

When to Use

Use this rule to identify the devices that block testmode signal propagation in shift and capture modes.

Description

The Diagnose_testmode rule generates the Atrenta Console highlight information for the devices that block testmode signal propagation.

The Diagnose_testmode rule generates separate displays for scan shift mode and capture mode.

Testmode simulation is optional. If one or more *test_mode* constraints are present, then the combinational instances and their blocking pins that block the testmode signal in shift or capture mode are highlighted.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *showPowerGroundValue*: The default value of the parameter is on. Set the value of the parameter to off to hide the simulation value of net due to power/ground.
- *dftShowWaveForm*: The default value of the parameter is off. Set the value of the parameter to on to display the waveform information for the Diagnose_testmode rule. The waveform information is displayed in the Atrenta Console's Waveform Viewer.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

When a blocked testmode signal *<sig-name>* has been found in a design unit *<du-name>* specified as *current_design* under *<mode-name>* mode and the corresponding highlight information has been generated, the *Diagnose_testmode* rule generates the following message:

```
[INFO] Instances through which testmode '<sig-name>' signal  
doesn't propagate in <mode-name> mode for design '<du-name>' is  
displayed
```

Where *<mode-name>* can be *shift* or *capture*.

Potential Issues

A violation is reported when other path is not properly sensitized with *test_mode*.

Consequences of Not Fixing

Not fixing the violation may result in improper *test_mode* propagation leading to unscannable flip-flops and reduced coverage.

How to Debug and Fix

For information on debugging, click [How to Debug and Fix](#).

Message 2

In case, there are no *testmode* constraints present for design unit *<du-name>* specified as *current_design*, the *Diagnose_testmode* rule generates the following messages depending on the mode for which the *testmode* constraints are missing:

```
[INFO] Constraint 'testmode -capture' missing in design '<du-name>'
```

Constraint 'testmode -shift' missing in design '<du-name>'

Potential Issues

A violation is reported when a test mode constraint is missing.

Consequences of Not Fixing

Not fixing the violation may result in false violation on other SpyGlass DFT ADV rules.

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental Schematic shows the terminal blocking the testmode propagation under the shift/capture mode. Overlay (Auxiliary violation mode) the Info_testmode rule under the shift/capture mode. This helps in identifying the blocked simulation value.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

No fix is required as this is an informational rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Informational Rules

Reports and Related Files

No related reports or files.

Info_atpg_conflict

Reports testpoints to reduce ATPG conflicts.

When to Use

Use this rule to reduce ATPG conflicts.

Description

The `Info_atpg_conflict` rule reports testpoints to reduce conflicts during test pattern generation by ATPG.

Parameters

- `dft_share_atpg_conflict_testpoint_at_branch`: The default value is on. Set the value of the parameter to disable sharing of testpoints at branch.
- `dft_atpg_conflict_tp_count`: The default value is 5%. Set the value of the parameter to any number or percentage to specify the upper limit on the number of test_points.

Constraints

- `test_mode` (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- `clock` (mandatory): Use this constraint to define the clocks of a design.
- `allow_test_point` (optional): Use this constraint to specify modules or instances which should be considered for suggesting test points.
- `no_test_point` (optional): Use this constraint to exclude modules or instances, which should not be considered for suggesting test points.

Operating mode

Capture

Messages and Suggested Fix

Message 1

```
[INFO] '<test_point_count>' test point(s) suggested for design  
'<design_name>', SPREADSHEET_PATH: '<spreadsheet_path>'
```

This is an informational message indicating number of testpoints found. Clicking on this message opens a spreadsheet capturing all testpoints

Message 2

[INFO] Test point(s) file '<testpoint_rpt_name>' generated

This is an informational message indicating that testpoint report has been generated.

Message 3

[INFO] Test point identification skipped as 'dft_atpg_conflict_tp_count' has been set to '0'

This is an informational message which indicates that testpoint generation was skipped because parameter 'dft_atpg_conflict_tp_count' has been set to 0

Message 4

[INFO] No test point found for design '<design_name>'

This is an informational message which indicates that no testpoint was found for design

Default Severity Label

Info/Warning

Rule Group

Informational Rules

Reports and Related Files

[atpg_conflict_testpoints.rpt](#): This report specifies the test point location along with conflict value at that point.

Info_addFault

Reports all the ports and pins which are specified as add fault.

When to Use

Use this rule to detect the number of instances and pins that are treated as add fault.

Description

This rule reports all the instances where the *add_fault* constraint has been specified. It also reports the number of terminals that are treated as add fault.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dft_generate_no_fault_and_add_fault_report*: The default value of the parameter is on. Set the value of the parameter to off to disable report generation for faults marked as add_fault.

Constraint(s)

add_fault (mandatory): Specifies faults to be considered while calculating fault/test coverage.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[INFO] '<no_of_faults>' faults are marked as 'add_fault' due to constraint 'add_fault <sgdc_commands>' in design '<du-name>'

Arguments

- Total number of faults, <no_of_faults>
- User-specified sgdc commands, <sgdc_commands>
- Name of the design unit, <du-name>

Potential Issues

The violation message is reported when the [add_fault](#) constraint is specified on the instances in a design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] '<no_of_faults>' faults are marked as 'add_fault' due to constraints on specific port and/or pin (using -fault | -net | -net_input | -net_output) in design '<du-name>'

Arguments

- Number of add_fault faults, <no_of_faults>
- Name of the design unit, <du-name>

Potential Issues

The violation message is reported when the [add_fault](#) constraint is specified on the instances in a design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 3

[INFO] Report file for add fault blocks '`<file_path_name>`' is generated

Potential Issues

The violation message is generated when a report file is generated for the add fault blocks.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 4

[INFO] 'add_fault' constraint impacts '`<no_of_faults>`' (`<percentage_of_faults>` of total `<total_number_of_faults>`) faults in design '`top_block`'

Arguments

- Number of faults marked as add_fault, `<no_of_faults>`
- Percentage of add-fault faults with respect to total number of faults, `<percentage_of_faults>`
- Total number of faults, `<total_number_of_faults>`

Potential Issues

The violation message is generated to show the impact of the [add_fault](#) constraint on a given design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 5

[INFO] Constraint 'add_fault' is not specified for design <du_name>

Potential Issues

The violation message is generated to inform that the [add_fault](#) constraint is not specified for the design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

[add_fault](#): List of all the ports and pins that are marked as add fault as per user specified [add_fault](#) sgdc commands.

Info_atSpeedClock

Displays at-speed test clock propagation

When to Use

Use this rule to display all at-speed clock nodes and their paths.

Description

The Info_atSpeedClock rule displays the propagation of each atspeed testclock and functional clock.

NOTE: Any *clock* SGDC command creates a functional clock.

Default Weight

10

Language

Verilog, VHDL

Method

For atspeed clock

Simulate each user specified atspeed clock under atspeed mode.

Display the simulation.

For functional clock

If dftFunctionalClockPropagation parameter is set to simulation

 Simulate or each user specified clock under functional mode.

 Display the simulation.

If dftFunctionalClockPropagation parameter is set to traversal

 Do unblocked traversal under functional mode.

 Display the simulation

EndFor

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.

- *clock* (mandatory): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[INFO] At-speed domain <domainName> of design <top-module> propagation is displayed This domain comprises <clockcount> clock(s), and drives <flipcount> flip-flop(s)

Potential Issues

To know more about potential issues related to this violation, click [Potential Issues](#)

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[INFO] <clocktype> clock <clockName> of design <top-module> propagation is displayed This clock drives <flipcount> flip-flop(s)

Arguments

- Name of the domain, *<domainname>*. Inferred domain is named as *inferred_i*, where *i* is a serial number for the inferred domain.
- Name of the top design, *<top-module>*
- Number of clock sources in the domain, *<clkCount>*
- Number of flip-flops in the domain, *<flopcount>*
- Clock type, *<clktype>*. This argument can assume one of the following values:
 - "Functional [simulation]"
 - "Functional [traversal]"
 - "At-speed"

Potential Issues

This is an informational rule. Therefore, there are potential issues related to this violation.

Consequences of Not Fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to Debug and Fix

The `Info_atSpeedClock` rule shows how clock `-testclock -atspeed` signals specified by user are distributed in design under the capture `atspeed` conditions.

No fix is required as this is an informational rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_atspeedClockSynchronization

Provides information on interacting atspeed clock domain pairs and non-interacting atspeed clock domain groups

When to Use

Use this rule to view the information related to the following atspeed clock domains:

- Pairs that have an impact on the fault coverage, and therefore, benefit the most from synchronization.
- Groups which do not interact. You may use this piece of information for optimizing test patterns by enabling concurrent capture.

Description

The Info_atspeedClockSynchronization rule generates information for the atspeed clock pair that has the maximum improvement in the fault coverage, and therefore, benefit the most from synchronization.

The objective is to efficiently group all the scan clocks into a few number of domains such that the maximum coverage is possible. Every clockdomain pair is evaluated for a score.

For example, for clock domain pairs, domA and domB, the score of domA-domB is equal to the sum of the flip-flops in domA reached by domB and the flip-flops in domB reached by domA. The rule prints out an ordered report of such domain pairs.

The rule also reports violation for non-interacting atspeed clock domain groups along with the affected flip-flop count.

The rule generates the Atspeed Clock Synchronization report, which lists clock domain pairs within the specified limit and atspeed clock domain groups in an ordered format.

Default Weight

10

Language

Verilog, VHDL

Method

foreach clock domain

do fan-out traversal from each flip-flop in this domain
 every flip-flop hit in another domain contributes to a score of +1 for the that pair of domains
 sort the domain pairs by their score
 report non-interacting atspeed clock domain groups sorted on affected flip-flop count

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dsmAtspeedClockSynchronizationRange*: The default value is 0. This indicates that there is no cutoff for frequency difference and all atspeed clock domains may be considered as candidates for synchronization. You can specify any positive integer value to define the maximum acceptable difference in the frequencies of two atspeed clock domains, which are considered as candidates for synchronization.

Constraint(s)

clock (mandatory): Declares the clock pins used as atspeed clocks.

Messages and Suggested Fix

Message 1

[INFO] '<count1>' interacting (within specified range: '<freq_range>') atspeed clock domain pairs, '<count2>' non-interacting-non-intersecting and '<count3>' non-interacting-intersecting atspeed clock domain groups reported in atspeedclock_synchronization.rpt for design '<top_name>'

Arguments

- Count of interacting atspeed clock domain pairs, *<count1>*
- User-specified frequency range limit to consider for synchronization, *<freq_range>*
- Count of non-interacting and non-intersecting atspeed clock domain groups, *<count2>*
- Count of non-interacting and intersecting atspeed clock domain groups, *<count 3>*
- Top module name, *<top_name>*

Potential Issues

A violation is reported when one or more of the following conditions hold true:

- Interacting atspeed clock domains which are currently not in the same domain but may be considered for synchronization
- Non-Interacting atspeed clock domains for which concurrent capture may happen

Consequences of Not Fixing

Following are the consequences of not fixing the violation:

- Reduction in transition coverage due to logic falling in clock domain crossing, which may have been made synchronous
- May increase time or pattern if concurrent capture for non-interacting domains does not happen

How to Debug and Fix

To fix the violation, synchronize the clock domain pairs.

Message 2

[INFO] Too few atspeed clock domains in design '<top>'. Rule requires at least 2 domains.

Arguments

Top module, <top>

Potential Issues

The violation message is reported when the design has less than two atspeed domains. Therefore, rule verification is skipped for the specified module.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct implications of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation.

Message 3

[INFO] No interacting (within specified range: '<freq_range>')

and no non-interacting atspeed clock domain pairs found in design '`<top>`'

Arguments

- User-specified frequency range limit to consider for synchronization, `<freq_range>`
- Top module name, `<top>`

Potential Issues

A violation is reported when all the atspeed domains of the design are interacting and their frequency difference is outside the user-specified range for considering them for synchronization.

Consequences of Not Fixing

Not fixing the violation may result in reduction in transition coverage due to logic falling in clock domain crossing, which may have been made synchronous.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation.

Message 4

[INFO] Atspeed clock synchronization report '`<rpt_name>`' is generated

Arguments

atspeed_clock_synchronization report name, `<rpt_name>`

Potential Issues

This is an informational message. Therefore, there are potential issues pertaining to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

[*atspeed_clock_synchronization*](#)

Info_atSpeedCoverage

This rule has been deprecated.

The Info_atSpeedCoverage rule will be packaged in the SpyGlass DFT ADV product.

Info_atSpeedDomain

Displays at-speed test clock domain propagation

Rule Description

The `Info_atSpeedDomain` rule displays the propagation of each at-speed testclock domain.

Method

Foreach user specified clock domain

Simulate all atspeed clock in this domain under capture at-speed mode.

Display the simulation;

EndFor

Constraints

`clock` -atspeed (mandatory) -testclock (optional)

`clock_shaper` (optional)

`pll` (optional)

`test_mode` (optional)

Rule Parameters

`dftUseOffStateOfClockInClockPropagation`

Operating Mode

`Capture (atspeed)`

Message Details

Message

At-speed domain <domain-name> of design '<du-name>' propagation is displayed. This domain comprises of <num1> clock(s), and drives <num2> of flip-flop(s).

Arguments

1. Name of the domain <domain-name>
2. Name of the top design name <du-name>
3. Number of at-speed testclocks in the domain <num1>

Information Rules

4. Number of flip-flops driven in the domain <num2>

Location

Place where top design unit name is inferred

Schematic highlight

All at-speed clock nodes and their paths

Rule Severity

Info

Info_atSpeedFrequency

Displays at-speed frequency propagation

Rule Description

The *Info_atSpeedFrequency* rule displays the propagation of each frequency. The rule also reports the total number of flip-flops hit by the frequency.

Constraints

- *atspeed_clock_frequency* (optional)
- *clock* (mandatory)
- *test_mode* (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Message Details

The following violation message is displayed for the *Info_atSpeedFrequency* rule:

Frequency <freqname> [atspeed clock <clknode>] of design <top> propagation is displayed [enabling values: <enVals>]. This frequency drives <number> flops.

Arguments

- Name of the frequency, <freqname>. You can specify a logical name or numerical value as an input to this argument.
- The net, port, or term on which the frequency is specified, <clknode>.
- Top module name, <top>.
- The enable pin values that activate the specified frequency, <enVals>.
- Number of hits by the specified frequency, <number>.

Schematic highlight

The root frequency node and all nets or ports in the frequency propagation path.

Rule Severity

Info

Info_atSpeedFrequency_EnableConflict

This rule has been deprecated. Use the [dftDsmConstraintCheck_08](#) rule as a replacement for the Info_atSpeedFrequency_EnableConflict rule.

Info_blackboxDriver

Reports black box drivers

When to Use

Use this rule to explore and debug logic around black boxes

Description

The Info_blackboxDriver rule reports violation for clock pins, set/reset pins, or tristate that is driving black boxes.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftShowSourceCentricViolation*: The default value is none. Therefore, the rule reports source-specific violation. Set the value of the parameter to space-separated list of rule name followed by the text string on or off in the following format to report instance-specific violation:


```
set_parameter dftShowSourceCentricViolation "<rule>-><off/
on> <rule>-><on/off> ..."
```
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (optional): Use this constraint to define the clocks of a design.
- *reset -async* (optional): Use this constraint to specify asynchronous set or reset pins in test mode.

Operating Mode

None

Message Details

Message 1

[INFO] No blackboxes found in design <design-name>

Arguments

The design unit name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 2

[INFO] No blackbox input is driven by clock/reset or tri state instance for design <design-name>

Arguments

The design unit name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 3

[INFO] Tri state <inst-name> is driving <terminal -name> terminal of blackbox <black-box-name>

Arguments

- Instance name, <inst-name>
- The terminal name attached to the black box, <terminal-name>
- The instance name of black box, <black-box-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 4

[INFO] Clock net <net-name> is driving <terminal-name> terminal of blackbox <black-box-name>

Arguments

- The net name connected to the terminal of black box, <net-name>
- The terminal name attached to the black box, <terminal-name>
- The instance name of black box, <black-box-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 5

[INFO] Reset net <net-name> is driving <terminal-name> terminal of blackbox <black-box-name>

Arguments

- The net name connected to the terminal of black box, <net-name>
- The terminal name attached to the black box, <terminal-name>
- The instance name of black box, <black-box-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 6

[INFO] Tri state <inst-name> is driving <num> blackbox pin(s)

Arguments

- Instance name, <inst-name>
- Number of black box pins driven by the respective net, <num>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 7

[INFO] Clock <net-name> is driving <num> blackbox pin(s)

Arguments

- The net name connected to the terminal of black box, <net-name>
- Number of black box pins driven by the respective net, <num>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 8

[INFO] Reset <net-name> is driving <num> blackbox pin(s)

Arguments

- The net name connected to the terminal of black box, <net-name>
- Number of black box pins driven by the respective net, <num>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Message 9

[INFO] No input of a true blackbox is driven by clock/reset or tri state instance for design <design-name>

Arguments

The design unit name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_coverage

Estimate fault and test coverage

When to Use

Use this rule to report an estimate of the fault and test coverage percentage.

Description

The Info_coverage rule reports an estimate of the fault and test coverage percentage, displays undetected faults back onto the structural view, and generates a report to show how the undetected faults are distributed in the design.

A fault is a pin on an instance stuck-at either 0 or 1. A potentially testable fault is a fault that is both observable and controllable to the complement of the stuck-at value as determined by testability analysis. The analysis does not use any vectors.

Total number of faults is double the number of pins on instantiated gates in the design.

The Info_coverage rule reports violation if the value of the fault coverage is less than the target coverage value specified using the [dft_target_stuck_at_fault_coverage](#) parameter.

If the denominator of test coverage is zero, then the test coverage is reported to be 100%.

When you double-click the violation message, undetected faults are displayed in the following format:

- s@0 if the pin stuck at 0 cannot be detected
- s@1 if the pin stuck at 1 cannot be detected
- s@01 if both the pin stuck at 0 and stuck at 1 cannot be detected

For more information on the Fault Coverage report, see [Viewing Results in Fault Browser](#).

Default Weight

10

Language

Verilog, VHDL

Method

Perform controllability and observability analysis taking scannability in effect.

Simulate power, ground and any available testmode conditions. Adjust controllability so that if any net is 0 because of testmode, change its 0=controllability to n and if any net is 1 because of testmode, change its 1=controllability to n.

Fault coverage calculation is estimated by the following definitions:

Faults are associated with pins on devices so no faults are on nets

Each pin has 2 faults: stuck at 0 (sa0) and stuck at 1 (sa1)

Total number of system faults is two times the number of pins in the circuit.

Total number of combATPG faults is 2 times the number of pins that are both controllable and observable in the default mode

Total number of seqATPG faults is 2 times the number of pins that are both controllable and observable in the seqATPG mode

Combinational fault coverage is $(\text{Total number of combATPG faults}) / (\text{Total number of system faults})$

Sequential fault coverage is $(\text{Total number of seqATPG faults}) / (\text{Total number of system faults})$

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [ATPG_credit](#): The default value is 0. Set the value of the parameter to any floating-point value between 0 and 1 to regulate the effect of untestable fault on test-coverage percentage evaluation by the Info_coverage rule.
- [dft_collapse_equivalent_faults](#): Default value is off. Set the value of the parameter to on to generate transition fault coverage with fault collapsing in the *stuck_at_coverage_collapsed.rpt* report.
- [dft_coverage_report_depth](#): Default value is 0. Set the value of the parameter to all or any positive integer value to specify the hierarchical depth up to which coverage summary is reported in the coverage reports.
- [faultAnnotation](#): The default value is off. Set the value of the parameter to on to generate the schematic back-annotation data for faults in addition to the text report.
- [dftDoLogicalRedundancyCheck](#): The default value is off. Set the value of the parameter to detect and process logically-redundant pins.

- *dftClockEdgeFaultAnalysis*: The default value is off. Set the value of the parameter to identify the faults that could not be detected as the capture happens at opposite edge of testclock used for these faults.
- *dftPOSDETECT_credit*: The default value is 0. Set the value of the parameter to a floating-point number between 0 and 1 to regulate the effect of potentially detectable faults on the test coverage and fault coverage evaluation.
- *dft_target_stuck_at_fault_coverage*: Use this parameter to define target stuck at fault coverage.
- *dft_target_stuck_at_test_coverage*: Use this parameter to define target stuck at test coverage.
- *dft_detect_shadow_latches*: The default value is on. Set the value of the parameter to off to treat all the latches normally.
- *dft_identify_equivalent_faults*: Default value is off. Set the value of the parameter to on to identify equivalent faults.
- *dftUUMarking*: The default value is on. Set the value of the parameter to off to determine whether unconnected sequential logic (USL) in the design should be marked as unused or not.
- *dftCaptureCriteria*: The default value is simulation. Set the value of the parameter to controllability to select the criteria based on which the flip-flops are marked capture-ready and latches are treated transparent in capture conditions.
- *dftGenerateStuckAtFaultReport*: The default value is none. Set the value of the parameter to one of the following possible values to specify the information generated in the report:
 - all
 - all_except_detected
 - detected
 - undetected
 - synthesis_redundant
 - unused
 - untestable
 - tied
 - blocked

❑ logical_redundant

- *dftGenerateTextReport*: The default value is none. Set the value of the parameter to a space-separated rule name list enclosed in double quotes to specify whether the corresponding rules generate a text report.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_use_cumulative_fault_status*: The default value is off. Set the value of the parameter to start for first run of multi-mode analysis. For subsequent runs, you can also specify `cumulative_stuck_at_fault_status.rpt` file generated in a previous run, as an input to this parameter.
- *dft_use_n_cycle_capture*: The default value is 1. Set the value of the parameter to any positive integer to specify the number of capture cycles.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *non_pd_inputcells* (optional): Use this constraint to specify the cells that should not be present at the input stage of the power/voltage domain.
- *scan_wrap* (optional): Use this constraint to specify the black box design units or instances that will be designed with scan wrappers.
- *seq_atpg* (optional): Use this constraint to specify the case analysis conditions.
- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.

- *force_ta* (mandatory): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] coverage estimate for design '<du-name>': fault_coverage = <fc-num>% and test_coverage = <tc-num>%

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation.

How to Debug and Fix

For information on how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[INFO] Stuck_at (<value>) coverage estimate for design '<du-name>' with collapsed equivalent faults: fault_coverage =<fault_coverage_percentage>% and test_coverage =<test_coverage_percentage>%

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation.

How to Debug and Fix

For information on how to debug and fix the violation, click [How to Debug and Fix](#).

Message 3

[INFO] For Design Unit <du-name>[<mode>], computed <cov-name> <cov-num>% is less than target <cov-name> <cov-num>%

Arguments

- Name of the design unit, <du-name>
- Combination or Sequential, <value>
- Mode name as NoScan-Mode, Comb-Mode, or Sequential-Mode.
- Fault coverage percentage, <fc-num>
- Test coverage percentage, <tc-num>
- Fault coverage or Test coverage name. <cov-name>
- Fault coverage or Test coverage percentage. <cov-num>

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation.

How to Debug and Fix

Perform the following steps, if the coverage is less than the expected result:

1. Check the violations reported by the Async_07 and Clock_11 rules. Fix the violations for these rules.
2. Check for unhandled black boxes. The unhandled black boxes are the ones without proper constraints: scan_wrap or module_bypass. If found, put proper constraints (scan_wrap or module_bypass) on them and re-run SpyGlass.
3. If coverage is still less than expected, check the violations reported by the TA_09 rule. Fix these violations and re-run SpyGlass.
4. If the coverage is still less than expected, check the violations reported by the TA_09 rule. Fix these violations.

Alternatively, run the Coverage_audit rule (goal name: dft_stuck_at_coverage_audit) and check the specific points affecting the coverage most and follow the recommendations of Coverage_audit report.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

No fix is required as this is an informational rule.

Message 3

[WARNING] For Design Unit '<du-name>', cumulative fault status info not found, starting fresh analysis

Arguments

Design Unit Name, <du-name>

Potential Issues

The violation message is reported when the cumulative stuck at fault status information for the design unit, <du-name>, is not present in the cumulative_stuck_at_fault_status.rpt file passed through the [dft_use_cumulative_fault_status](#) parameter.

Consequence of not fixing

Not fixing this violation causes the multi-mode analysis to start again for the design unit, <du-name>

How to Debug and Fix

No debug is required for this violation.

To fix the violation, provide the cumulative fault status data generated in the previous design run.

Message 4

[WARNING] For Design Unit '<du-name>', mismatch found with previous run cumulative fault status data '<mismatch cause>', starting fresh analysis

Arguments

- Design Unit Name, <du-name>

- Cause of mismatch in design. It shows previous and current term/port count in case of mismatch, <mismatch cause>

Potential issues

This violation message is reported when change is detected in design unit, <du-name> with respect to previous run.

Consequence of not fixing

Not fixing the violation message causes multi-mode analysis to start again for the design unit, <du-name>, ignoring the cumulative fault status information.

How to Debug and Fix

No debug is required for this violation.

To fix the violation, provide the cumulative fault status data generated in the previous design run.

Message 5

[INFO] For Design Unit '<du-name>', first stuck_at coverage result of current run is generated at '<file path>' to be used for subsequent multi-mode run, if needed

Arguments

- Design unit name, <du-name>
- Cumulative fault info file path, <file path>

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 6

[INFO] For Design Unit '<du-name>', cumulative stuck_at coverage result of previous and current run is generated at

'<file path>' to be used for subsequent multi-mode run, if needed

Arguments

- Design unit name, <du-name>
- Cumulative fault info file path, <file path>

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 7

[INFO] XML report for Info_coverage

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

- [stuck_at_faults](#)
- [stuck_at_coverage](#)
- [stuck_at_coverage_collapsed](#)

Report

The Info_coverage rule generates [stuck_at_faults](#) report that shows number of stuck at faults. Set the value of the [dftGenerateStuckAtFaultReport](#) parameter to a value other than none.

The report gives the total number of undetected faults for any instance. If that instance instantiates K modules, then there will be one line generated for each instance. If an instance also instantiates leaf-level cells then one additional line will appear that will list the sum of the undetected faults in all leaf-level cells.

The sum of the number of undetected faults on the K lines (or K+1 lines) following a module line will equal the number of faults on that module line.

The report is designed for rapid navigation to instances containing the most undetected faults. To accomplish this, each instance line representing a module instantiation at some level in the hierarchy lists the number of faults in that instantiation but not the instance name or the module name since, in large designs, these names may become long and thereby obscure the detection information. Each of the K instance lines will have a unique symbolic reference of the form [Inst_xx_xx]. No searching is necessary for the leaf-level cells so no unique symbolic reference is generated.

The process can be repeated to drill down to a module containing a large number of undetectable faults. The full name for that instance, in conjunction with the design view, may be used to display the structural view of this module and the undetected faults in that module.

Info_dBist

Display simulation values of design in DBIST mode.

When to Use

Use this rule to display simulation values of a design in the DBIST mode.

Description

The Info_dBist rule displays simulation values of design in DBIST mode.

DBIST mode simulation is optional depending upon its presence. If *dbist* constraint is specified, the corresponding simulation values of the design are displayed.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

dbist (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit into a state for DBIST mode.

Operating Mode

Dbist

Messages and Suggested Fix

Message 1

```
[INFO] 'logic bist mode' simulation value for design '<du-name>' is displayed
```

where,

<du-name> is the parent design unit name,

Potential Issues

This is an informational message. Therefore, there are no related potential issues.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing the violation.

How to Debug and Fix

This is an informational message. Therefore, no debug and fix is required.

Message 2

[INFO] Constraint 'Dbist' missing in design <du-name>

where,

<du-name> is the parent design unit name,

Potential Issues

A violation is reported because the dbist constraint is not specified in the design.

This is an informational message. Therefore, there are no consequences of not fixing the violation.

How to Debug and Fix

This is an informational message. Therefore, no debug and fix is required.

Message 3

[INFO] DBIST Mode (specified using 'dbist' sgdc command) has not been specified for design '<du-name>', hence skipping rule checking

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message is reported when rule is run without the dbist constraint. Ignore this message in case no dbist constraint is required for design.

Consequences of not fixing

Rule checking is skipped in absence of the dbist constraint.

How to Debug and Fix

To fix the violation, add necessary dbist constraints, if present.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_define_tag

Show system state for a given tag.

When to Use

Use this rule to show the system state for each condition specified by the `define_tag` constraint.

Description

The *Info_define_tag* rule shows the simulation results for each condition specified by the `define_tag` constraint.

The state of user-selected nodes, when a set of nodes are in a particular state, is checked. This is useful to verify that port requirements on one or more blocks in a design have required values when a set of nodes have particular values.

Prerequisites

Specify the *define_tag* constraint.

Default Weight

10

Language

Verilog, VHDL

Method

For each set of `define_tag` conditions
Simulate power, ground and the defined node/value pairs

Parameter(s)

- *dftShowWaveForm* (optional): The default value is off. Set the value of the parameter to on to enable the generation of waveform corresponding to the rule message in the Waveform viewer of the SpyGlass Explorer.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_show_unused_define_tag*: The default value is off. Set the value of the parameter to on to enable viewing propagation of unused define_tag through Info_define_tag/Soc_04 rules.

Constraint(s)

define_tag (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

Define_tag

Messages and Suggested Fix

Message 1

[INFO] Tag '<tag-name>' is displayed for design '<du-name>'

Arguments

To view list of arguments, click [Arguments](#).

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message

How to Debug and Fix

The *Info_define_tag* rule assists in debugging the violations reported by the *Conn_01*, and *Conn_02* rules. The *Info_define_tag* rule is an informative rule and requires no debug.

No fix is required as this is an informational rule.

Message 2

[INFO] Tag '<tag name>__as_used_in_scan_chain_tracing' is displayed for design '<du-name>'

Arguments

- Name of the tag, <tag-name>

- Name of the design unit, *<du-name>*

Potential Issues

A violation message is displayed for those *define_tag* tags that are used in *scan_chain* to define scan enable condition (-scanenable field).

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message

How to Debug and Fix

The *Info_define_tag* rule assists in debugging the violations reported by the *Conn_01* and *Conn_02* rules. The *Info_define_tag* rule is an informative rule and requires no debug.

No fix is required as this is an informational rule.

Message 3

[INFO] Display skipped for '<unused tag count>' unused tag(s): <unused tag list>. Set parameter 'dft_show_unused_define_tag' to 'on' to show unused tags

Potential Issues

"This is an info message which indicates that propagation for unused define_tags was skipped and is not shown.

Consequences of Not Fixing

This is an informational message.

How to Debug and Fix

To see their propagation values, set the *dft_show_unused_define_tag* parameter to on.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Information Rules

Rule Group

SoC Rules

Reports and Related Files

No related reports or files.

Info_dft_deprecated

Displays info messages for all the rules, parameters and constraints which will be deprecated future but are enabled/used in current run

When to Use

Use this rule to detect all the rules, parameters, and constraints, which will be deprecated in a future release but are enabled/used in the current run.

Description

This rule reports a violation if a rule, constraint, or parameter, which will be deprecated in a future SpyGlass release, is currently enabled or used.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

Message 1

[INFO] Rule 'dumpBlackBox' will be deprecated in future. Use rule Info_scanwrap instead, it generates more detailed report

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 2

[INFO] Rule 'Info_coverageAtGateLevel' will be deprecated in future. Use rule Info_coverage instead

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 3

[INFO] Rule 'TA_01' will be deprecated in future. Use rule TA_09 instead

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 4

[INFO] Rule 'TA_02' will be deprecated in future. Use rule TA_09 instead

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 5

[INFO] Rule 'TA_06' will be deprecated in future. Use rule TA_09 instead

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 6

[INFO] Rule 'CG_consistency' will be deprecated in future.

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 7

[INFO] Rule 'Power_01' will be deprecated in future.

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 8

[INFO] Rule 'Info_memorywritedisable' will be deprecated in future.

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 9

[INFO] Rule 'Clock_22' will be deprecated in future. Use rule Clock_08 instead

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 10

[INFO] Rule 'Info_atSpeedFrequency_EnableConflict' will be deprecated in future. Use rule dftDsmConstraintCheck_08 instead

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 11

[INFO] Constraint 'test_mode' will be deprecated in future.
Fields '-noFaultComb -noFaultSeq' are deprecated

Potential Issues

For more information on potential issues, see [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, see [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on how to debug and fix the violation, see [How to Debug and Fix](#).

Message 12

[INFO] Constraint 'memory_write_disable' will be deprecated in future.

Potential Issues

This is an informational message. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this message.

Information Rules

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Reports and Related Files

No related reports or files.

Info_DftDebugData

Reports dft debug data (schematic and excel sheet)

When to Use

Use this rule to generate BA data.

Description

The Info_DftDebugData rule generates debug data for the following, which you can use in debugging other SpyGlass DFT ADV rules:

- Flip-flop present in the design
- Latch present in the design
- Black box present in the design
- CGC present in the design
- Abstracted blocks present in the design

The debug data is reported in a hierarchical spreadsheet and schematic. To know more about the BA feature, see [Understanding Back Annotation](#).

NOTE: *The Info_dftDebugData rule runs automatically whenever you run the SpyGlass DFT ADV product. Set the value of the dftDebugData parameter to off to switch off this rule.*

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftDebugData*: The default value is bp_off. You can set the value of the parameter to off to disable the BA feature. Other possible values are on, cell_properties_off, and bp_and_cell_properties_off.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_maximum_number_of_rows_with_schematic*: The default value is 10. Set the value of the parameter to any non-negative integer value to specify the number of rows in the spreadsheet, which contain the schematic data.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[INFO] DFT data for design block <du-name>

Arguments

Parent design unit name, <du-name>

Potential Issues

This is an informational rule. Therefore, there are no potential design issues related to this message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation.

How to Debug and Fix

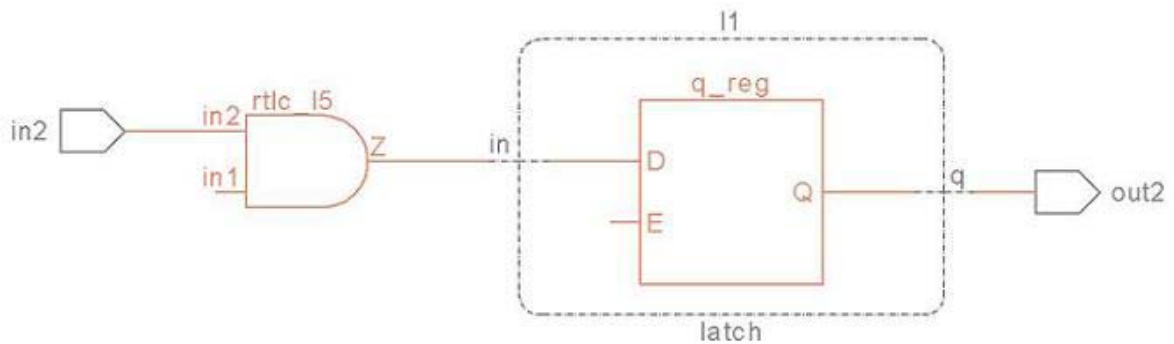
This is an informational rule. Therefore, no debug and fix is required for this rule.

Example Code and/or Schematic

Following examples illustrate the various design scenarios and the respective violation messages reported when the *dftDebugData* parameter is assigned different permissible values.

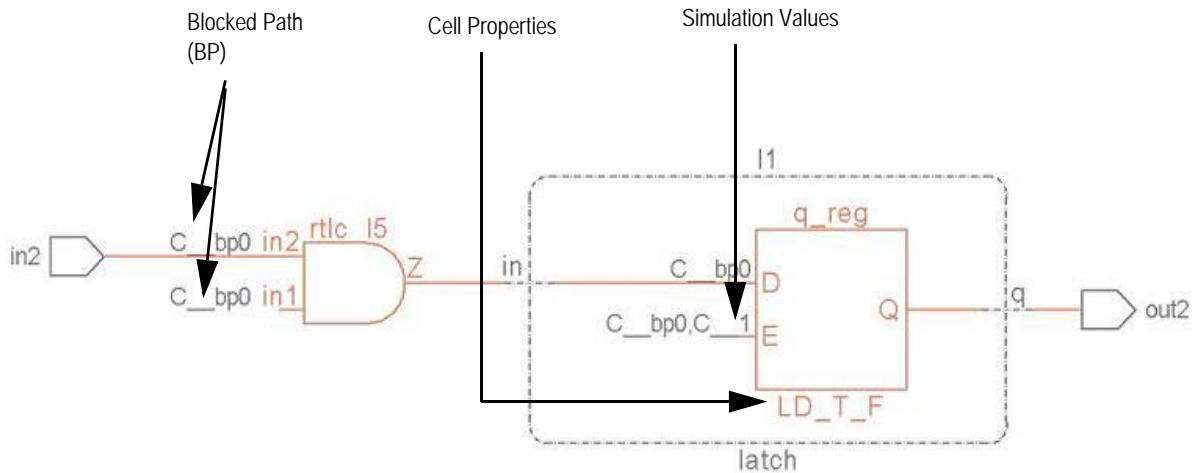
dftDebugData=off

When the value of the *dftDebugData* parameter is set to `off`, the *Info_DftDebugData* rule does not generate any back annotation data as shown in the following schematic:



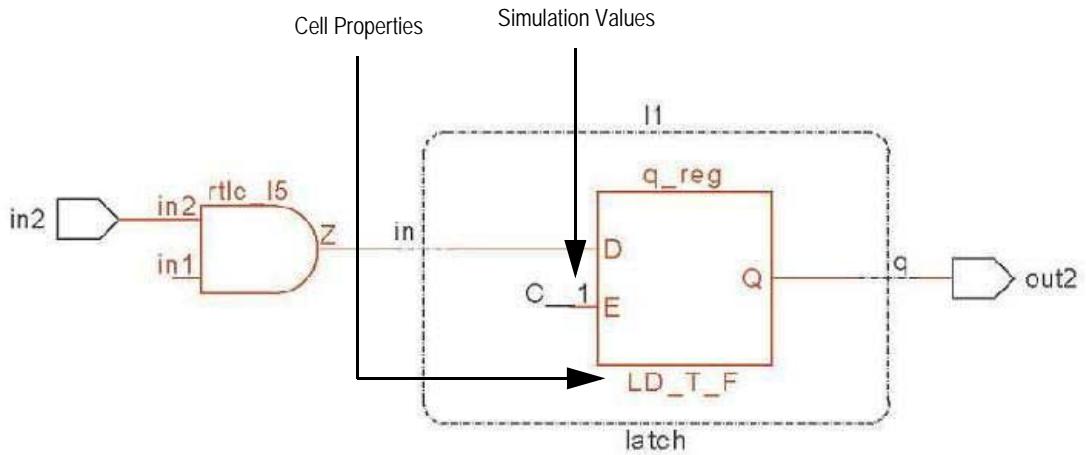
dftDebugData=on

When the value of the *dftDebugData* parameter is set to on, the *Info_DftDebugData* rule generates back annotation for blocked path, cell properties, and simulation values as shown in the following schematic:

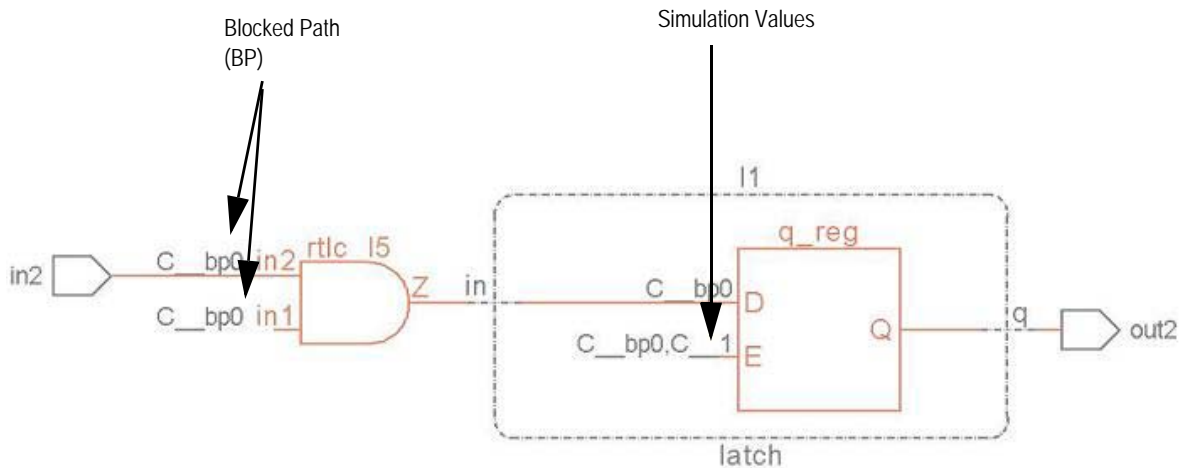


dftDebugData=bp_off

When the value of the *dftDebugData* parameter is set to `bp_off`, the *Info_DftDebugData* rule generates back annotation for only cell properties and simulation values as shown in the following schematic:

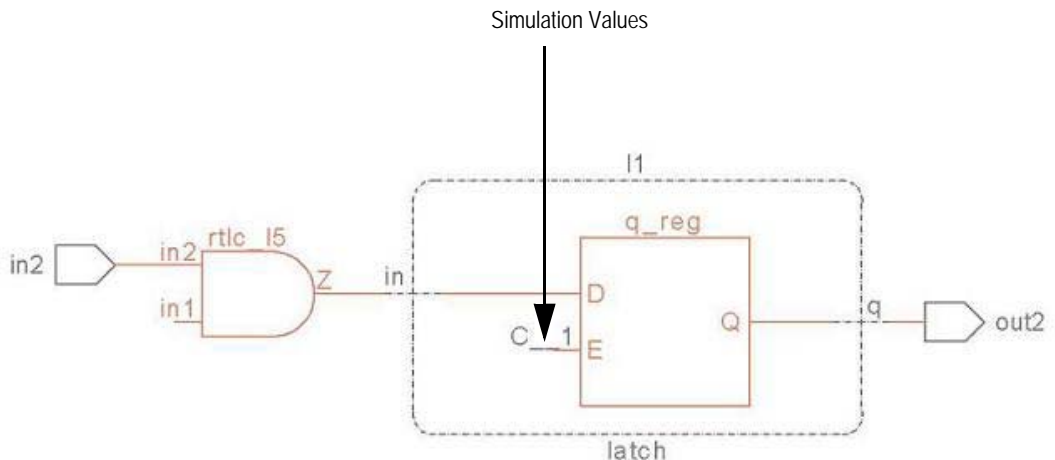
**dftDebugData=cell_properties_off**

When the value of the *dftDebugData* parameter is set to `cell_properties_off`, the *Info_DftDebugData* rule generates back annotation for only blocked path and simulation values as shown in the following schematic:



dftDebugData = bp_and_cell_properties_off

When the value of the *dftDebugData* parameter is set to *bp_and_cell_properties_off*, the *Info_DftDebugData* rule generates back annotation for only simulation values as shown in the following schematic:



Information Rules

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

The Info_DftDebugData rule generates the [dft_multibit_flipflops](#) report.

Additionally, the Info_DftDebugData rule generates the following spreadsheet (.csv) reports:

- dft_data_02.csv
- dft_data_03.csv
- dft_data_04.csv
- dft_data_05.csv

The reports are generated at the following location:

```
$wdir/spyglass_reports/dft/dft_data/<report_name>.csv
```

dft_data_02.csv

The dft_data_02.csv report contains information about the scannable and unscannable flip-flops.

Following figure shows a sample [dft_data_02.csv](#) report.

	B	C	D	E	F	G	H
	Flip-flop Name	Scannability	Shift-CP-Root	Capture-CP-Root	D-Cnt in capture	RST-Cnt in capture	Int in capture
1	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
2	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
3	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
4	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising-edge)	yy-	01_cnt(yy-)	pin ab
5	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
6	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
7	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
8	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
9	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
10	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
11	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising-ϵ)	dsp_periph_clk (rising-ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
12	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab
13	mdl_top.mdl_ip.mdl_logic.mdl_mcm.mdl_dmclog.mdl_sync_mgr0.sigout_reg	Scannable - Inferred	dsp_periph_clk (rising- ϵ)	dsp_periph_clk (rising- ϵ)	01_cnt(yy-)	01_cnt(yy-)	pin ab

The dft_data_02.csv report has the following columns:

- **TagId:** TagId is only for internal use.
- **Flip-Flop Name:** Name of the flip-flop
- **Scannability:** Represents the scannability status, which can be one of the following:
 - **Scannable Flip-Flops:** Reports the following types of flip-flops:
 - ◆ Forced scan flip-flops: The *Scan - Forced*
 - ◆ Inferred scan flip-flops: The *Scannable- Inferred*
 - **Unscannable Flip-Flops:** Reports the following types of flip-flops
 - ◆ Forced no-scan flip-flops: The *No-Scan - Forced*
 - ◆ Inferred no-scan flip-flops: The *The No-Scan - Inferred*.
 - ◆ Clock Unscannable flip-flops: The *Un-Scannable - Clock*
 - ◆ Asynchronous unscannable flip-flops: The *Un-Scannable - Async*
 - ◆ Asynchronous and clock unscannable flip-flops: The *Un-Scannable - Async and clock*

NOTE: For more information on the types of flip-flops, see [Types of Flip-Flops](#).

- **Shift-CP-Root:** Root clock name, along with the edge, which is reaching the flip-flop clock pin in the shift mode.
- **Capture-CP-Root:** Root clock name, along with the edge, which is reaching the flip-flop clock pin in the capture mode.
- **D-Cnt in capture:** Data pin controllability in capture.
- **RST-Cnt in capture:** Reset pin controllability in capture.
- **PRE-Cnt in capture:** Set pin controllability in capture.
- **Observability:** Observability of net connected to Q-pin of the flip-flop.
- **Used:** If the flip-flop is marked **unused** (using the Info_unused rule).
- **Initialized:** Data and Q-pin simulation value when initialization sequence (using the test_mode -initialize_for_bist: BIST_03) is simulated.
- **Shift-Sim:** Data and Q-pin simulation value when shift mode is simulated.
- **Capture-Sim:** Data and Q-pin simulation value when capture mode is simulated.

Information Rules

dft_data_03.csv

The *dft_data_03.csv* report contains information related to latches.

Following figure shows a sample *dft_data_03.csv* report:

	Latch Name	Shift Clock	Capture Clock	Type in Shift	Type in Capture
1	top.l1.q_reg	clk1 (rising-e...	clk1 (rising-e...	LD_NTP	LD_NTP
2	top.l2.q_reg	clk1 (falling-e...	clk1 (falling-e...	LD_T^off	LD_T^off
3	top.l3.q_reg	clk1 (rising-e...	clk1 (rising-e...	LD_NTP	LD_NTP
4	top.l4.q_reg	clk2 (rising-e...	clk2 (rising-e...	LD_NTP	LD_NTP
5	top.l5.q_reg	clk2 (rising-e...	clk2 (rising-e...	LD_NTP	LD_NTP

The *dft_data_03.csv* report has the following columns:

- **Latch Name:** Name of the latch
- **Shift Clock:** Root clock name along with the edge, which is reaching the latch enable pin in the shift mode
- **Capture Clock:** Root clock name along with the edge, which is reaching the latch enable pin in the capture mode
- **Type in Shift:** Represent the transparency and lockup status of the latches in the shift mode
- **Type in Capture:** Represent the transparency and lockup status of the latches in the capture mode

dft_data_04.csv

The *dft_data_04.csv* report contains information related to black boxes.

Following figure shows a sample *dft_data_04.csv* report:

Black-Box Name	Memory	Clock Gating Cell	Clock Shaper	Scan Wrap	Bypass
top1.U1	BB_NOT_A_MEMORY	BB_NOT_A_CGC	BB_NOT_A_SHAPER	BB_SCANWRAP	BB_NOT_MBYPASS
top1.U2	BB_NOT_A_MEMORY	BB_NOT_A_CGC	BB_NOT_A_SHAPER	BB_SCANWRAP	BB_NOT_MBYPASS
top1.U4	BB_NOT_A_MEMORY	BB_NOT_A_CGC	BB_NOT_A_SHAPER	BB_SCANWRAP	BB_NOT_MBYPASS

The *dft_data_04.csv* report has the following columns:

- **Black-Box Name:** Name of the black box
- **Memory:** Represents whether the black box is a memory or not
- **Clock Gating Cell:** Represents whether the black box is a CGC or not
- **Clock Shaper:** Represents whether the black box is a Clock Shaper or not
- **Scan Wrap:** Represents whether the black box is a scanwrap or not. You must run the Info_Scanwrap rule to view this column in the *dft_data_04.csv* report
- **Bypass:** Represents whether the black box is bypass or not using the *module_bypass* SGDC constraint

dft_data_05.csv

The *dft_data_05.csv* report contains information related to clock gating cells (CGCs).

Following figure shows a sample *dft_data_05.csv* report:

	Clock Gating Cell (CGC) Name	Type
1	top.b1	BB_CGC_POS
2	top.cg1	LIB_CGC_POS

The *dft_data_04.csv* report has the following columns:

- **Clock Gating Cell (CGC) Name:** Name of the CGC
- **Type:** Represent the type of CGC, such as, white box, black box, inferred, and so on.

Info_dftmax_configuration

Enables the planning and configuration of DFTMAX Ultra in an SoC

When to use

Use this rule to automate the planning and configuration of DFTMAX Ultra in an SoC.

Description

Implementation of DFTMAX Ultra in SoCs usually requires configuring multiple codec partitions. Configurations are defined by specifying how many top level Scan-In (SI) and Scan-out (SO) pins are assigned to each partition (SI/SO Allocation) and how many internal chains each partition will have.

The *Info_dftmax_configuration* rule automates the planning and configuration of DFTMAX Ultra in an SoC. You can specify the partitions in the design targeted for DFTMAX Ultra codecs. SpyGlass then automatically extracts all needed information from the design, such as, number of flip-flops and X-density, reports the plan and corresponding metrics. It also reports the corresponding DFTMAX Ultra commands in a separate file.

Parameter(s)

- *dftmax_si_so_pin_number*: The default value is 2. Set the value of the parameter to specify the number of top-level Scan-In and Scan-Out Pins that can be used for DFTMAX Ultra planning.
- *dftmax_is_occ_present*: The default value is off. Set the value of the parameter to on to enable the usage of on-chip clock controller (OCC).
- *dftmax_allow_asymmetric_pin_allocation*: The default value is off. Set the value of the parameter to on to enable the asymmetric SI/SO allocation.
- *dftmax_config_lookup_file*: The default value is off and SpyGlass picks the configuration estimator lookup from <SPYGLASS_HOME>/policies/dft_dsm/dftMaxUltra/dftMaxUltra_config_lookup.csv file. Set the value of the parameter to absolute path of configuration estimator lookup file (in the .csv format) for DFTMAX Ultra planning.

Constraint(s)

dftmax_partition (optional): Use this constraint to specify the partitions in

the design targeted for DFTMAX Ultra codecs to automate the planning and configuration of DFTMAX Ultra in an SoC.

Operating Mode(s)

Scanshift, Capture, Capture (atspeed)

Messages and Suggested Fix

The *Info_dftmax_configuration* rule reports the following types of violation messages:

- *Info*
- *Warning*
- *Error*

Info

The *Info_dft_max* ultra rule reports the following informational messages:

Message 1

[INFO] default_partition created with all unallocated blocks. To remove a block it should be black-boxed.

Potential Issues

The violation message is displayed when *default_partition* is not specified by using the *dftmax_partition* constraint.

Consequences of not fixing

If *default_partition* is not specified then all un-allocated blocks (all top-level instances, which are not part of any other DFTMAX Ultra codec partition) are considered for *default_partition* (default DFTMAX Ultra codec partition).

How to debug and fix

This rule generates the *dftmax_ultra_configuration* report that lists the name of blocks considered for *default_partition*.

Message 2

[INFO] Report file '<report-name>' successfully generated. Contains the DFTMAX Ultra recommended plan and configuration metrics.

Arguments

Name of a generated report, <report_name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required.

Message 3

[INFO] Report file '<report-name>' successfully generated. Contains the configuration command for recommended plan.

Arguments

Name of a generated report, <report_name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required.

Warning

The *Info_dftmax_configuration* rule reports the following warning messages:

Message 1

[WARNING] No top-level block found in design '<design-name>', ignoring DFTMAX Ultra configuration planning.

Arguments

Top-module name, <design-name>

Potential Issues

The violation message is displayed if no top-level block is present in the design.

Consequences of not fixing

In the absence of top-level blocks in the design, DFTMAX Ultra planning are ignored.

How to debug and fix

To debug the violation, check the RTL.

Message 2

[WARNING] `dftmax_partition` constraint already specified with partition name '<partition-name>', ignoring the command.

Arguments

Name of DFTMAX Ultra codec partition, <partition-name>

Potential Issues

The violation message is displayed when same partition-name is specified in more than one [dftmax_partition](#) constraint command.

Consequences of not fixing

If the [dftmax_partition](#) constraint is specified multiple times for same partition-name, then only first specification is considered for rule checking. Rest of the specifications are ignored.

How to debug and fix

to debug the violation, check the SGDC for correct specification.

Message 3

[ERROR] Ignoring non-top block '<block-name>' specified in

partition '<partition-name>'

Arguments

- Name of non-top-level block, <block-name>
- Name of DFTMAX Ultra codec partition, <partition-name>

Potential Issues

The violation message is displayed when any non-top-level block is specified in the blocklist of a *dftmax_partition* constraint command.

Consequences of not fixing

If any non-top block is specified using the `-blocklist` field of the *dftmax_partition* constraint, then the non-top-level block is ignored.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 4

[WARNING] Top-level block '<block-name>' earlier specified in partition '<partition-name1>', ignoring specification in partition '<partition-name2>'

Arguments

- Name of top-level block, <block-name>
- Name of the DFTMAX Ultra codec partition1, <partition-name1>
- Name of the DFTMAX Ultra codec partition2, <partition-name2>

Potential Issues

The violation message is displayed when same top-level block is specified in the blocklist of more than one *dftmax_partition* constraint command.

Consequences of not fixing

If same top-level block is present in multiple *dftmax_partition* constraint command, then only first specification is considered and all others are ignored.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 5

[WARNING] No top-level block to specify for partition '<default-partition-name>', ignoring the constraint registration (all unallocated blocks will be considered for default-partition).

Arguments

Name of the DFTMAX Ultra user-specified default codec partition, <default-partition-name>

Potential Issues

The violation message is displayed when no top-level block is specified for the user-specified default_partition. This may happen because of the following reasons:

- Presence of non-top-level blocks in the *dftmax_partition* constraint command.
- Ignoring of top-level blocks already present in the earlier *dftmax_partition* constraint specifications.

Consequences of not fixing

If no top-level block is present in user-specified default_partition, then all unallocated top-level blocks will be considered for creating default_partition.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 6

[ERROR] No top-level block to specify for partition '<partition-name>', ignoring the constraint registration.

Arguments

Name of DFTMAX Ultra codec partition, <partition-name>

Potential Issues

This violation message is displayed when there is no top-level block to specify for *dftmax_partition*. This may be due to specifying some non-top-level blocks in the *dftmax_partition* constraint command or due to ignoring of top-level blocks already present in the earlier *dftmax_partition* constraint

specifications.

Consequences of not fixing

If no top-level block is present in *dftmax_partition*, then the specification is ignored.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 7

[WARNING] Wrong argument <argument-name> argument value (accepted value <min-allowed-value>-<max-allowed-value>) specified for partition '<partition-name>', calculating from SpyGlass

Arguments

- Name of dftmax_partition argument (-inputs | -outputs | -chain_count), <argument-name>
- Minimum allowed value (based on entry in estimator lookup table; default is 1), <min-allowed-value>
- Maximum allowed value (based on entry in estimator lookup table; default is 32), <max-allowed-value>
- Name of DFTMAX Ultra codec partition, <partition-name>

Potential Issues

The violation message is displayed when the forced value of any of the arguments, -inputs, -outputs, -chain_count, is not within the range of maximum and minimum allowed value (calculated from entry present in configuration estimator lookup table).

Consequences of not fixing

In such a case, the forced value is ignored and calculated using the *Info_dftmax_configuration* rule analysis.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 8

[WARNING] Partition '`<partition-name>`' specified with different values of `-inputs` and `-outputs` arguments for symmetrical allocation.

Arguments

Name of DFTMAX Ultra codec partition, `<partition-name>`

Potential Issues

The violation message is displayed when the forced value for arguments `-inputs` and `-outputs` are different when the value of the [`dftmax_allow_asymmetric_pin_allocation`](#) parameter is set to off, that is, for symmetric allocation.

Consequences of not fixing

In such a case, the forced value is considered for the partition and allocation will be asymmetrical even though the symmetrical allocation is set.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 9

[WARNING] `dftmax_partition` constraint is specified with default `_partition` name, some blocks might remain unallocated.

Potential Issues

The violation message is displayed when you have specified default `_partition` using the [`dftmax_partition`](#) constraint command.

Consequences of not fixing

In such a case, default `_partition` is considered with user-specified blocklist unless all blocks are non-top-level blocks or present in earlier [`dftmax_partition`](#) constraint command.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 10

[WARNING] default_t_partition not created, either all blocks are non-top or earlier allocated in other partitions of design '`<design-name>`'

Arguments

Top-module name, `<design-name>`

Potential Issues

The violation message is displayed when default_partition can not be created due to absence of unallocated top-level blocks in the design.

Consequences of not fixing

In such a case, default_partition is not created.

How to debug and fix

To debug the violation, check the SGDC for correct specification.

Message 11

[WARNING] No sequential logic (considering scan logic) found in design '`<design-name>`', ignoring DFTMAX Ultra configuration planning.

Arguments

Top-module name, `<design-name>`

Potential Issues

The violation message is displayed when no scan logic is present in the design.

Consequences of not fixing

In such a case, no DFTMAX Ultra planning is done.

How to debug and fix

To debug the violation, check the SGDC and RTL.

Message 12

[WARNING] '`<no-of-pins>`' pin(s) will remain unallocated while allocating symmetrically. Set parameter

'dftmax_si_so_pin_number' to any even value (symmetric allocation) or set the parameter 'dftmax_allow_asymmetric_pin_allocation' to 'on' (asymmetric allocation).

Arguments

No of unallocated pins, <no-of-pins>

Potential Issues

The violation message is displayed when value of the [dftmax_allow_asymmetric_pin_allocation](#) parameter is set to off, that is, symmetric allocation, and some pin will remain unallocated.

Consequences of not fixing

In such a case, apart from unallocated number of pins, all other pins are considered for allocation among different partitions.

How to debug and fix

To debug the violation, check the value of the following parameters:

- [dftmax_allow_asymmetric_pin_allocation](#)
- [dftmax_si_so_pin_number](#)

Message 13

[WARNING] No output configuration (internal chains, compression ratio etc.) recommended for partition '<partition-name>' (SI: '<SI-count>', FFs: '<FF-count>'), as it does not meet the minimum design size requirement (FFs: '<min-design-size>').

Arguments

- Name of DFTMAX Ultra codec partition, <partition-name>
- SI count for partition, <SI-count>
- FF count for partition, <FF-count>
- Minimum design size, <min-design-size>

Potential Issues

The violation message is displayed when design size for any partition is less than the minimum design size requirement for a particular SI-count.

Consequences of not fixing

In such a case, compression ratio will be zero and instead of DFTMAX Ultra compression, regular scan should be used.

How to debug and fix

To debug the violation, check the SGDC and RTL.

Message 14

[WARNING] Codec penalty ratio is high (<penalty-ratio>%) for partition '<partition-name>' (should be < 30%). Please create more partition for better QoR.

Arguments

- Codec penalty ratio (= codec shift length/internal scan chain length*100), <penalty-ratio>
- Name of DFTMAX Ultra codec partition, <partition-name>

Potential Issues

The violation message is displayed when codec penalty ratio for any `dftmax_partition` is equal to or greater than 30%.

Consequences of not fixing

Codec penalty ratio should be always less than 30%. If not, then more partition should be created for better QoR.

How to debug and fix

To debug the violation, see the [dftmax_ultra_configuration](#) report that shows the DFTMAX Ultra configuration metrics.

To fix the violation, create more `dftmax_partition`, using the [dftmax_partition](#) constraint, for better QoR.

Error

The `Info_dftmax_configuration` rule generates the following error messages:

Message 1

[ERROR] Errors found while parsing configuration estimator lookup file, ignoring the DFTMAX Ultra planning.

Potential Issues

The violation message is displayed when an error occurs while parsing the configuration estimator lookup file. This can happen due to one of the following reasons:

- No configuration estimator lookup file (in the .csv format) is present
- Configuration estimator lookup file is empty
- Minimum value of si_count is not 1

Consequences of not fixing

If an error occurs while parsing the configuration estimator lookup file, then DFTMAX Ultra planning is ignored.

How to debug and fix

To debug this violation, check the configuration estimator lookup file.

Message 2

[ERROR] Wrong value '*<incorrect-value>*' found in configuration estimator lookup file.

Arguments

Incorrect entry in configuration estimator lookup file (.csv format),
<incorrect-value>

Potential Issues

The violation message is displayed when some non-numeral entry is found while parsing the configuration estimator lookup file.

Consequences of not fixing

If an error occurs while parsing the configuration estimator lookup file, then DFTMAX Ultra planning is ignored.

How to debug and fix

To debug the violation, check the configuration estimator lookup file.

Message 3

[ERROR] Value of parameter '*dftmax_si_so_pin_number*' should be in between <min-allowed-value>-<max-allowed-value>, ignoring the DFTMAX Ultra planning.

Arguments

Information Rules

- Minimum allowed value of *dftmax_si_so_pin_number*, <min-allowed-value>
- Maximum allowed value of *dftmax_si_so_pin_number*, <max-allowed-value>

Potential Issues

The violation message is displayed when the value of the *dftmax_si_so_pin_number* parameter is not within the limit of minimum and maximum allowed value.

When the value of the *dftmax_is_occ_present* parameter is specified as 'off' (default value):

- *dftmax_si_so_pin_number* should be $\geq 2 * (\text{no. of DFTMAX partitions})$
- allocatable top-level SI/SO pins = *dftmax_si_so_pin_number*

When the value of the *dftmax_is_occ_present* parameter is set to on, then the presence of OCC consumes an additional two pins and, therefore:

- *dftmax_si_so_pin_number* should be $\geq 2 * (\text{no. of DFTMAX partitions}) + 2$
- allocatable top-level SI/SO pins = *dftmax_si_so_pin_number* - 2

Consequences of not fixing

DFTMAX Ultra planning is ignored if the *dftmax_si_so_pin_number* parameter fails the sanity check.

How to debug and fix

To debug this violation, check SGDC and the value of parameters.

Example Code and/or Schematic

This is an informational rule and generates a text report. See the Reports and Related Files section for more information on this file.

Default Severity Label

Info

Rule Group

Information Rules

Report and Related Files

The following reports are generated by the *Info_dftmax_configuration* rule:

- [dftmax_ultra_configuration](#)
- [dftmax_ultra_configuration_command](#)

Info_enabledFlops

Reports the flip-flops in the design, sorted by their respective clock domains

When to Use

Use this rule to categorize the flip-flops in the design on the basis of their clock domains.

Description

The Info_enabledFlops rule generates a report listing the multi-bit flip-flops, memories, and macros in the design based on their clock domains.

Default Weight

1

Language

Verilog, VHDL

Method

For each flip-flop

If flip-flop is not scannable and 'dsmReportEnabledFlops=scannable'

Skip this flip-flop

Check the clock and put it in the respective category

If no clock is found

Categorize it into one of the following categories:

(a) Atspeed Clock Block

(b) Clock tied Low

(c) Clock tied high

(d) No Atspeed Clock in Fan-in

Check the async pins and if any one of them is active in atspeed mode, put it in Tied_active_async_set_or_reset category

End for (flip-flop)

Sort the flip-flops in each domain according to hierarchy.

Generate the report showing flip-flops per domain in sorted order.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

- *dsmReportEnabledFlops*: The default value is `all`. Therefore, the rule reports all the flip-flops. Set the value of the parameter to `scannable` to report only the scannable flips-flops.
- *dsm_infer_clock_domain_at_clock_shaper_boundary*: The default value is `off`. Set the value of the parameter to `on` to infer a separate clock domain at the `clock_shaper` output pin through which the `atspeed` clock is passing.

Constraint(s)

- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

[INFO] Report file '<reportName>' successfully generated. Contains '<flopCount>' flops categorized by clock domains

Arguments

- Name of the generated report, *<reportName>*
- Number of flip-flops in the domain, *<flopCount>*

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation.

Consequences of Not Fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to Debug and Fix

This is an informational rule. Therefore, no fix is needed. However, you can verify that which flip-flop is connected to which domain.

Example Code and/or Schematic

This is an informational rule and generates a text report. See the [Reports and Related Files](#) section for more information on this file.

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

[dft_dsm_enabled_flipflops](#)

Info_forcedScan

Displays all registers and flip-flops specified as 'force_scan'

When to Use

Use this rule for running a sanity check on scannable flip-flops and for design exploration and debugging.

Description

The Info_forcedScan rule generates the Atrenta Console highlight information for flip-flops specified as scannable using the *force_scan* constraint.

Rule Exceptions

The *force_no_scan* constraint overrides the *force_scan* constraint. Thus, any flip-flop specified with both the *force_no_scan* and *force_scan* constraints is assumed to be a noscan flip-flop.

Default Weight

1

Language

VHDL, Verilog

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *force_scan* (optional): Use this constraint to declare flip-flops as scannable even if they do not so qualify.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

None

Messages and Suggested Fix

Message 1

[INFO] Constraint 'force_scan' not specified for the design
<module-name>

Arguments

Name of module, <module-name>

Potential Issues

This message appears when *force_scan* constraint is not specified in your design.

Consequences of Not Fixing

If you do not fix the violation, no flip-flop is treated as forced scan.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] <count> Flip-flop in <module-name> is constrained as
'force_scan' [<percentage> of total flop count <count>]

Arguments

- Name of module, <module-name>
- Number of flip-flops, <count>
- scan constrained flip-flop count / total flip-flop count, <percentage>

Potential Issues

The violation message appears when a flip-flop is constrained as scan.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this

violation message.

Message 3

[INFO] <count> Flip-flop in '<module-name>' are(is) constrained as 'force_scan' by control signal (<control signal type>) '<control signal net name>'

Arguments

- Name of module, <module-name>
- Number of flip-flops, <count>

Potential Issues

The violation message appears when -clock_control / -set_control / -reset_control option is used for the *force_scan* constraint.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 4

The following violation message is displayed when the *force_scan* constraint is applied on a module or an instance:

[INFO] '<count>' Flip-flop is constrained as 'force_fscan' by the constraint '<constraint name>'

Arguments

Number of flip-flops <count>

Potential Issues

The violation message appears when *force_scan* constraint is applied on module or instances.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this

violation message.

Message 5

The following violation message is displayed when the *dft_scannable_latches* parameter is set to on.

```
[INFO] '<type>' specified as 'force_scan -name', in design
'<du-name>', impacts '<no_of_flip_flops>' flip-flops and
'<no_of_latches>' latches
```

Arguments

- A flip-flop or a latch, *<type>*
- Top design unit, *<du-name>*
- Number of flip-flops, *<no_of_flip_flops>*
- Number of latches, *<no_of_latches>*

Potential Issues

The violation message appears when *force_scan* constraint is applied on module or instances.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 6

The following violation message is displayed when the *dft_scannable_latches* parameter is set off.

```
[INFO] '<type>' specified as 'force_scan -name', in design
'<du-name>', does not impact any flip-flop
```

Arguments

- Name of the latch, *<type>*
- Top design unit, *<du-name>*

Potential Issues

The violation message is displayed when a flip-flop or a latch in the module is constrained as scannable and the *dft_scannable_latches* parameter is set

to off.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 7

The following violation message is displayed when the [dft_scannable_latches](#) parameter is set off.

```
[INFO] '<type>' specified as 'force_scan -name', in design  
'<du-name>', impacts <number> flip-flops
```

Arguments

- Name of the flip-flop, <type>
- Top design unit, <du-name>
- Number of affected flip-flops, <number>

Potential Issues

The violation message is displayed when a flip-flop or a latch in the module is constrained as scannable and the [dft_scannable_latches](#) parameter is set to off.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 8

```
[INFO] Generated report <report path> for 'force_scan' sgdc  
constraint.
```

Potential Issues

The violation message is displayed when the [force_scan](#) constraint is applied on some objects in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This rule generates the *forced_scan* report that lists the name of the design objects on which the *force_scan* constraint is applied.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Reports and Related Files

No related reports or files.

Info_freqAssignTable

Generates a frequency assignment report

Rule Description

The `Info_freqAssignTable` rule presents frequency assignment data of the design, as given in the constraint files. The data is displayed in a tabular form with testclock name, enable pins and frequency symbols as three columns.

The `Info_freqAssignTable` rule generates the [dft_dsm_clock_frequency](#) report, which lists the frequency assignment data of the design, as given in the constraint files. The data is displayed in a tabular form with testclock name, enable pins, and frequency symbols as three columns.

Constraints

- `clock -testclock -atspeed` (optional)
- `clock_shaper` (optional)
- `pll` (optional)
- `test_mode` (optional)

Rule Parameters

[dftUseOffStateOfClockInClockPropagation](#)

Operating Mode

[Capture \(atspeed\)](#)

Message Details

Message

Frequency assignment table <rpt-name> is generated

Arguments

1. Name of the frequency assignment report file <rpt-name>

Location

None

Information Rules

Schematic highlight

None

Rule Severity

Info

Info_inferredNoScan

Displays all flip-flops that have been inferred as no scan by SpyGlass DFT ADV.

When to Use

Use this rule to generate the information for the inferred no scan flip-flops.

Description

The Info_inferredNoScan rule generates the Atrenta Console highlight information for inferred no scan flip-flops.

A flip-flop is inferred no scan when one or more of the following conditions hold true:

- A forced value is present at the output of a flip-flop during the shift simulation
- A forced test_clock is present on the output of a flip-flop
- A flip-flop is not part of a valid scan chain in the scan-stitched design, that is, the [dftDesignState](#) parameter is set to `post_scan_stitched` and the [dft_all_scan_chains_defined](#) parameter is set to `yes`
- The output of a flip-flop is getting a non-X value due to propagation of the shift condition, the [dft_infer_sequential_propagation_as_no_scan](#) parameter is set to `on`, and the asynchronous pins of the flip-flops are not active.

NOTE: *A flip-flop bank is inferred as no scan, if any of its output pin has a forced test clock/ test mode or non-x value from sequential propagation.*

Rule Exceptions

The Info_inferredNoScan rule ignores flip-flops that are marked with [force_no_scan](#) or [force_scan](#) constraint.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftGenerateTextReport*: The default value is none. Set the value of the parameter to space-separated rule name list enclosed in double quotes to specify whether the corresponding rules generate a text report that has information of all inferred noscan flip-flops.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *force_scan* (optional): Use this constraint to declare flip-flops as scannable even if they do not so qualify.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[INFO] Flip-flop '<flip-flop>' is inferred as no_scan. Reason: <reason>

Arguments

- Name of the flip-flop. <flip-flop>
- Reason for the flip-flop being inferred as no scan. <reason>, which can be any one of the following:
 - The *clock -testclock* constraint is specified on its output.
 - The *test_mode* constraint is specified on its output.

- ❑ It's output gets a non-X ('0') *test_mode* value through sequential propagation.
- ❑ A flip-flop is not part of a valid scan chain in the scan-stitched design.

Potential Issues

The violation message is reported when any of the following conditions hold true:

- The clock -testclock constraint is specified on its output.
- The test_mode constraint is specified on its output.
- The non-X test_mode value is forced on its output through sequential propagation.
- A flip-flop is not part of a valid scan chain in the scan-stitched design.

Consequences of Not Fixing

Not fixing the violation may result in flip-flop inferred as no scan. Therefore, coverage will reduce due to bad controllability and observability

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental schematic shows the flip-flop that is being inferred as noscan.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

To fix the violation, remove the constraints.

Message 2

```
[INFO] Flip-flop '<flip-flop>' is inferred as no_scan. Reason:
<reason>
```

Potential Issues

The violation message is reported when the [dft_use_stable_simulation_conditions](#) parameter is on.

Consequences of Not Fixing

Not fixing the violation may result in flip-flop inferred as no scan. Therefore, coverage will reduce due to bad controllability and observability

How to Debug and Fix

View the Incremental Schematic for the violation message. The Incremental schematic shows the flip-flop that is being inferred as noscan.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

To fix the violation, remove the constraints.

Message 3

[INFO] No flip-flop of design '`<du-name>`' is inferred as no_scan

Arguments

Name of the design, `<du-name>`

Potential Issues

Since this is an informational message, there are no potential issues related to this message.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of not fixing this violation.

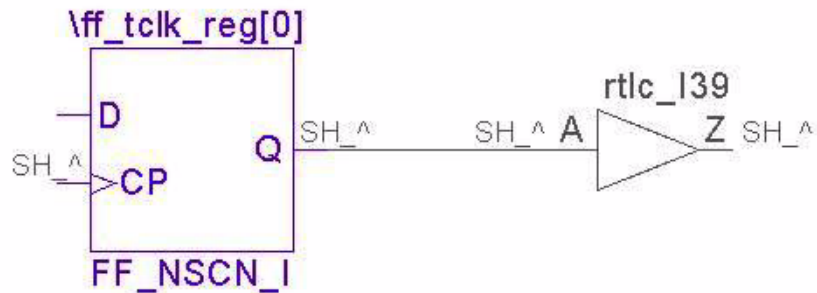
How to Debug and Fix

This is an informational message. There is no debug and fix required for this message.

Example Code and/or Schematic

Example 1

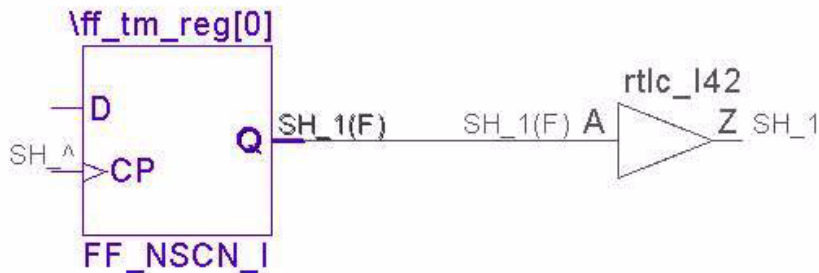
Consider the following schematic:



In the above schematic, the clock constraint is specified on the output of the `ff_tclk_reg[0]` flip-flop.

Example 2

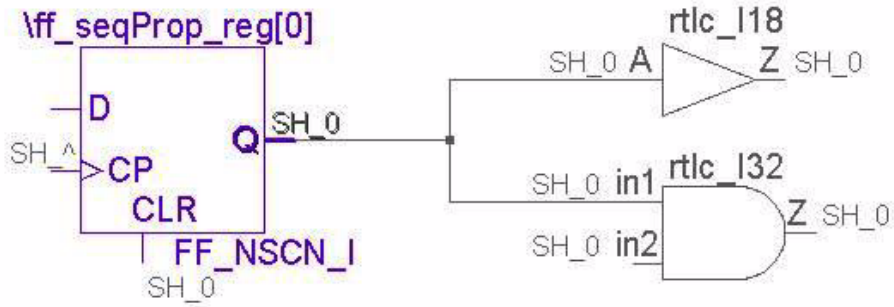
Consider the following schematic:



In the above schematic, the test_mode constraint is specified on the output of the `ff_tm_reg[0]` flip-flop.

Example 3

Consider the following schematic:



In the above schematic, a non-X testmode value is forced on the output of the `\ff_seqProp_reg[0]` flip-flop.

Default Severity Label

Info

Rule Group

Informational

Reports and Related Files

[*inferred_no_scan_ffs*](#): Lists all flip-flops that are inferred as noscan by SpyGlass DFT ADV.

Info_IP_Report

Generates boundary report on the IPs specified using `ip_block` SGDC command.

Rule Description

The Info_IP_Report rule assists in debugging the fanin/fanout cone of the boundary pin. For example, if a boundary pin does not get a required test_mode, test clock, or control / observe, you may use the schematic back annotation to debug the fanin / fanout cone of the boundary pin and root cause of the blockage points.

The Info_IP_Report rule generates boundary report, *dft_dsm_ip_report*, on the IPs specified using *ip_block* SGDC command.

The rule also generates boundary report in the SGDC format `<dft_dsm_ip_constraints.sgdc>`.

Rule Parameters

dftUseOffStateOfClockInClockPropagation: Default value is `off` and all clocks, except the propagated clock, are set to the X state during clock propagation. Set this parameter to `on` so that all clocks, except the propagated clock, are set to their off state during clock propagation.

Constraints

ip_block (mandatory): Specifies the design units for which you want to generate the boundary information.

Operating Mode

Scanshift, Capture, Capture (atspeed)

Message Details

Message 1

[INFO] Boundary report (format: SGDC) for user specified `ip_block` is generated at file `<file name>`

Message 2

[INFO] Boundary report (format: Structured Text) for user specified `ip_block` is generated at file `<file name>`

Message 3

[INFO] Boundary information <mode/clock/control/observe> for block '<block name>' (master: '<master name>') is displayed for <relevant mode name>

Message 4

[INFO] Constraint 'ip_block' is not specified for design <top module name>

Message 5

Neither -create_constraints nor -create_report modifier is specified

Schematic Highlight

Boundary Information of user specified blocks.

Rule Severity

Info

Example Code and/or Schematic

Consider the following SGDC specification:

```
current_design top

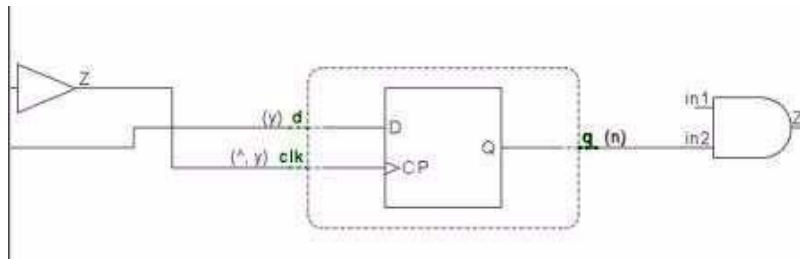
clock -name clk -testclock -atspeed

testmode -name in1 -value 0 -captureATspeed
testmode -name tm1 -value 1
force_scan -name top.U2.q
ip_block -name DEF -mode -control -observe -clock
create_report
ip_block -name BBox -mode -control -observe -clock
-create_report
```

The Info_IP_Report rule reports following informational message for the above specification:

Boundary information (mode/clock/control/observe) for block 'top.U2' (master: DEF) is displayed for capture mode, test.sgdc, 9

Double-clicking on the above message displays the following schematic:



Reports and Related Files

- [dft_dsm_ip_report](#)
- `dft_dsm_ip_constraints.sgdc`: This report may contain one or more of following information:
 - clock (shift, capture, @speed capture)
 - mode (shift, capture, @speed capture)
 - controllability of the input pins at static capture mode
 - observability of the output pins at static capture mode

A sample format of the `dft_dsm_ip_constraints.sgdc` report is as shown below:

```
# current_design CGC
current_design top1.u2
  test_mode -name in[0] -value 0 -captureATspeed
  clock -name clk -value rtz -testclock
  clock -name clk -value rtz -atspeed
  force_ta -name clk -control yyn
  force_ta -name gclk -control nnn
  force_ta -name in[0] -control yyy
  force_ta -name in[1] -control yyy
  force_ta -name in[2] -control yyy
```

Information Rules

```
force_ta -name clk -observe n  
force_ta -name gclk -observe n  
force_ta -name in[0] -observe n  
force_ta -name in[1] -observe n  
force_ta -name in[2] -observe n
```

Info_Latch

Reports status of latches in the design

When to Use

Use this rule to know the transparency/scannability status of latches in the design.

Description

The Info_Latch rule reports transparency/scannability status for all latches along with their clock source in shift and capture mode. It lists the types of latches, in the shift mode, as specified using the [dft_list_latches_of_type](#) parameter.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_list_latches_of_type](#): Default value of this parameter is off. Set the value of the parameter to one of the following values to specify the type latch, the Info_Latch rule will report:
 - all
 - transparent
 - non_transparent
 - scannable
 - non_scannable
 - shadow
 - non_shadow
 - non_transparent_non_scannable_non_shadow

Constraint(s)

None

Operating Mode

Scanshift

Messages and Suggested Fix

[INFO] 'Design '<du-name>' has '<no_of_latches>' '<latch-type>' latches '<csv_file_path>'

Potential Issues

The violation message provides information on the types of latches available in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] 'Design '<du-name>' has <number_of_latches> latches which are neither scannable, nor shadow, nor transparent '<csv_file_path>'

Potential Issues

The violation message provides information on the number of specific types of latches available in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 3

[INFO] 'Design '<du-name>' has no '<latch type>' latch

Potential Issues

The violation message reports that there is no latch of specific type available in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 4

[INFO] Latch transparency status report '<report_name>' is generated

Potential Issues

The violation message reports the name of the latch transparency report generated.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 5

[INFO] Design '<du-name>' has no latch.

Potential Issues

The violation message reports that a defined top-module does not have a latch.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

Information Rules

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 6

[INFO] Design '<du-name>' has '<number_of_latches>' latches

Potential Issues

The violation message reports the number of latches for a top module.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Informational

Reports and Related Files

[*dft_latch_enable*](#): The report lists the transparency status of latches, that is, whether the latches are transparent and are lockup latches or not.

Info_latchMapping

Reports the transparency and lockup status of all latches along with their respective test clock domain in the shift and capture mode

When to Use

Use this rule to know the transparency status of the latches.

Description

The Info_latchMapping rule reports latch transparency and lockup status along with their respective test clock domain in the shift and capture mode.

NOTE: *This rule is deprecated and will be removed in a future release. It is recommended to use [Info_Latch](#) rule as a replacement to this rule.*

For more information on retiming latches, see [Retiming or Lockup Latches](#).

For more information on transparent latches, see [Transparent](#).

Default Weight

10

Language

VHDL, Verilog

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (optional): Use this constraint to define the clocks of a design. Use this constraint's -testclock argument for the [Info_latchMapping](#) rule.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] No latch is found in the design '<top module name>'

Potential Issues

The violation message appears, if the testclock reaches one or more latches.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] Design '<top module name>' has '<number of latches>' latches

Potential Issues

The violation message appears when the Info_latchMapping rule does not run due to inadequate inputs or in absence of applicable design objects.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights all the latches present in the design with back annotation data available to show the transparency and lockup status visually.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

To fix the violation, you may review the transparency, lockup status, and the driving test clock, if any.

Message 3

[INFO] Latch transparency status report
'<path_to_dft_latch_enable_report>' is generated

Potential Issues

The violation message appears when the Info_latchMapping rule does not run due to inadequate inputs or in absence of applicable design objects.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

To fix the violation, you may review the transparency, lockup status and the driving test clock, if any.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

[dft_latch_enable](#): The report lists the transparency status of latches, that is, whether the latches are transparent and are lockup latches or not.

Info_logicalRedundant

Displays logically redundant faults

When to Use

Use this rule to identify logically redundant faults.

Description

The Info_LogicalRedundant rule displays logically redundant faults. A pin is considered logically redundant when its value does not change irrespective of the simulation values applied at its fan-in.

The Info_LogicalRedundant rule optimizes LR nodes as described in the following table:

Design Type	Mode	Considered for Logic Optimization	Considered for TI/BL Faults
RTL	Power Ground	Yes	NA
	Test Mode	No	NA
Netlist	Power Ground	NA	Yes
	Test Mode	NA	No

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftDoLogicalRedundancyCheck*: The default value is `off`. Set the value of the parameter to `on` to enable the Info_logicalRedundant rule to detect and process logically-redundant pins resulting in a different fault coverage and test coverage values for the design, if logically-redundant pins were present in the design. You can also set the value of the

`dftDoLogicalRedundancyCheck` parameter to `distinct_pg_capture` to report separate violation messages for the Power Ground and Capture modes.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

If *dftDoLogicalRedundancyCheck* parameter is set to on

Following messages are generated when the value of the *dftDoLogicalRedundancyCheck* parameter is set to on:

Message 1

[INFO] <num1> faults (over <num2> pin(s)) in '<du-name>' are logically redundant in capture mode

Arguments

- Name of the design unit, <du-name>
- Number of logically redundant faults, <num1>
- Total number of pins, <num2>

Potential Issues

A pin is logically redundant when it is fixed to a value irrespective of the simulation values applied to the circuit inputs.

Consequences of Not Fixing

Redundant faults are not ATPG testable.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic

Viewer window highlights the pins which are logically redundant.
To fix the violation, remove the logically redundant logic.

Message 2

[INFO] All faults in '<du-name>' are logically correct

Potential Issues

There are no potential issues related to this violation message.

Consequences of Not Fixing

There are no consequences of not fixing this violation message.

How to Debug and Fix

No debug and fix is required for this violation message.

Message 3

[INFO] Logically redundant data (Capture Mode) for fanout node (FN) '<node_name>' is displayed

Arguments

Name of the fanout node, <node_name>

Potential Issues

A violation is reported for the following:

- Fanout node along with reconvergent node which is either stuck to 0, stuck to 1, or is behaving like a logical buffer/inverter with respect to the fanout node.
- Faults that are marked logically redundant by these set fanout and reconvergent nodes.

Consequences of Not Fixing

Presence of logically redundant fault in the design reduces the coverage.

How to Debug and Fix

To fix the violation, re-write the RTL so that logically redundant logic is either optimized or removed.

If *dftDoLogicalRedundancyCheck* parameter is set to *distinct_pg_capture*:

Apart from the above generated messages, following additional violation messages are generated when the value of the *dftDoLogicalRedundancyCheck* parameter is set to *distinct_pg_capture*:

Message 1

[INFO] <num1> faults (over <num2> pin(s)) in '<du-name>' are logically redundant in Power-Ground mode

Arguments

- Name of the design unit, <du-name>
- Number of logically redundant faults, <num1>
- Total number of pins, <num2>

Potential Issues

A pin is logically redundant when it is fixed to a value irrespective of the simulation values applied to the circuit inputs.

Consequences of Not Fixing

Redundant faults are not ATPG testable.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the pins which are logically redundant.

To fix the violation, remove the logically redundant logic.

Message 2

[INFO] Logically redundant data (Power-Ground Mode) for fanout node (FN) '<node_name>' is displayed

Arguments

Name of the fanout node, <node_name>

Potential Issues

A violation is reported for the following:

- Fanout node along with reconvergent node which is either stuck to 0, stuck to 1, or is behaving like a logical buffer/inverter with respect to the fanout node.

Information Rules

- Faults that are marked logically redundant by these set fanout and reconvergent nodes.

Consequences of Not Fixing

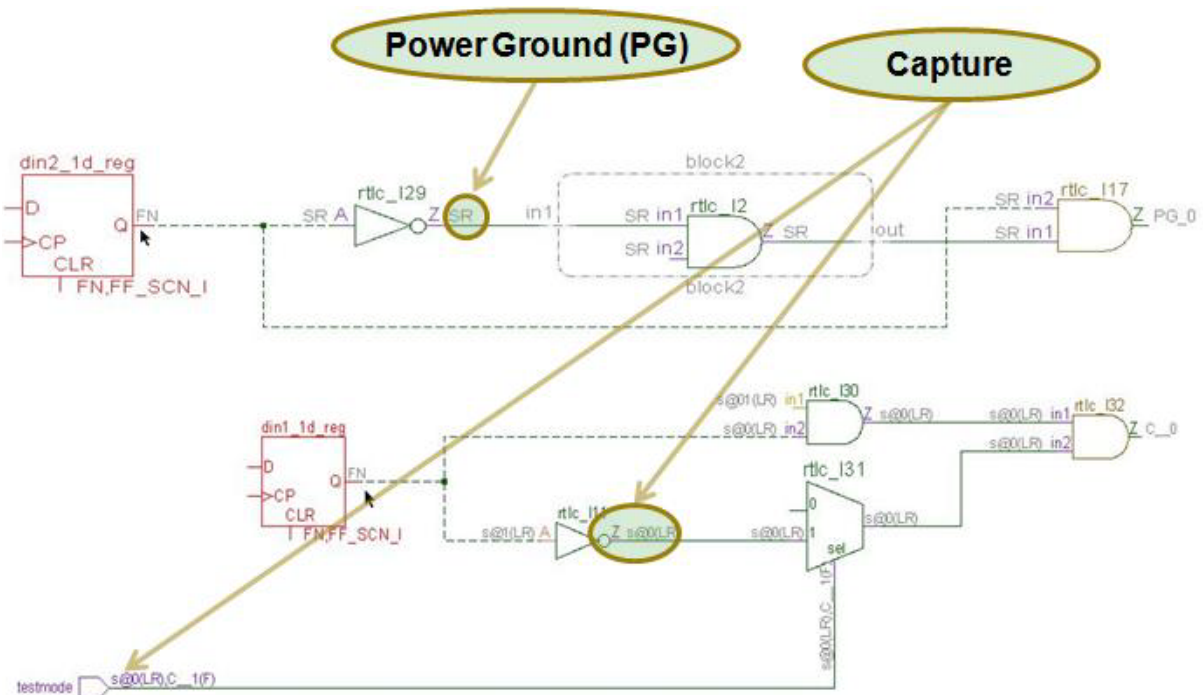
Presence of logically redundant fault in the design reduces the coverage.

How to Debug and Fix

To fix the violation, re-write the RTL so that logically redundant logic is either optimized or removed.

Example Code and/or Schematic

Consider the following figure:



If the value of the `dftDoLogicalRedundancyCheck` parameter is set to `distinct_pg_capture`, the `Info_logicalRedundant` rule reports the following violation message:

Logically redundant data (Power-Ground mode) for fanout node

(FN) 'test.din2_id' is displayed, test.v, 32

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_memories

Reports on memory instances present in a design.

When to Use

Use this rule to detect memory instances present in the design.

Description

The *Info_memories* rule reports all the memory instances for which the fastest-functional-clock reaching to a memory clock pin.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message 1

[WARNING] No functional clock reaches the clock pin '`<clock_pin_name>`' of memory instance '`<mem_instance_name>`' (module: `<mod_name>`)

Arguments

- Name of clock pin of memory instance, `<clock_pin_name>`
- Name of memory instance, `<mem_instance_name>`
- Module name, `<mod_name>`

Potential Issues

The violation message is reported if no functional clock reaches a memory-clock-pin.

Consequences of Not Fixing

Functional clock is unable to control the memory.

How to Debug and Fix

Click the violation message to display the highlighted memory instance and

the clock pin in the Incremental Schematic.

Message 2

[WARNING] Fastest frequency cannot be determined for clock pin '`<clock_pin_name>`' of memory instance '`<mem_instance_name>`' (module: `<mod_name>`). Reason: `<reason>`. `<clock/frequency_list>`"

Arguments

Name of clock pin of memory instance, `<clock_pin_name>`

- Name of memory instance, `<mem_instance_name>`
- Module name, `<mod_name>`
- Reason behind the inability to calculate fastest frequency reaching a clock pin, `<reason>`.
- Clock list or frequency list, `<clock/frequency_list>`
 - Clock List: List of the clocks with no frequency specified
 - Frequency List: List of Frequency:clock pairs. These clocks are the ones with non-numeric frequency values. Both or one of the lists are shown with the warning depending on the case.

Potential Issues

The violation is reported because of one of the following reasons:

- Missing frequency (One of the clocks reaching the clock pin has no frequency)
- Non-numeric frequency values (One of the clocks reaching the clock pin has a non numeric frequency)
- Missing frequency, non-numeric frequency values (When both the above scenarios are applicable)

Consequences of Not Fixing

The BIST planning will be incomplete.

How to Debug and Fix

Click the violation message to display the highlighted path from clock pin to the clock with empty or non-numeric frequency in the Incremental Schematic.

Message 3

[INFO] Text report containing clock-frequency information is

generated at '`<path>`'

Arguments

Path where `dft_memory_report` is generated, `<path>`

Potential Issues

The violation message is reported when the rule generates the text report.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required.

Message 4

[INFO] The memory instance count for the design `<design_name>` is 0"

Arguments

Name of the design, `<design_name>`

Potential Issues

The violation message is reported when the rule generates the text report.

Consequences of Not Fixing

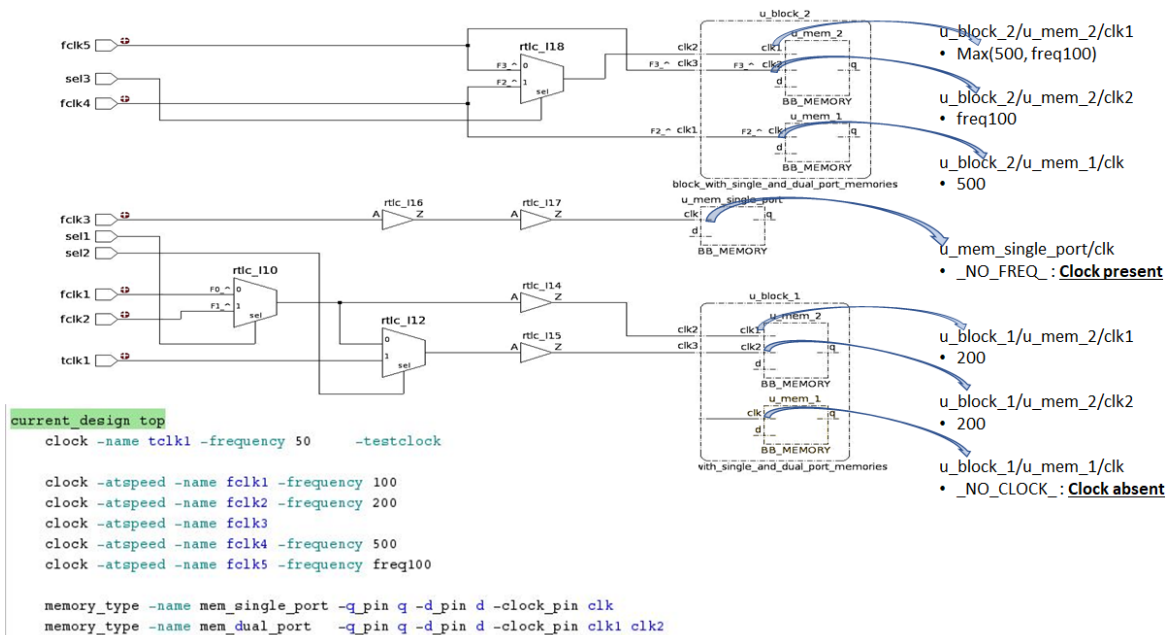
This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required.

Example Code and/or Schematic

Consider the following example:



In the above example, the rule reports violation for the following cases:

- **No clock:** u_block_1/u_mem_1/clk
- **Clock but no frequency:** u_mem_single_port/clk
- **Non-numeric frequency along with numeric:** u_block_2/u_mem_2/clk1

However, the rule does not report the violation for the following cases:

- **Only non-numeric frequency:** u_block_2/u_mem_2/clk2
- **All numeric frequencies:** u_block_1/u_mem_2/clk*

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

dft_memory_report: Reports the information on all the memory instances.

Info_memoryforce

Back annotate the memoryforce simulation results onto the structural view.

When to Use

Use this rule to run sanity check on *memory_force* constraint.

Description

The Info_memoryforce rule displays all signals with non-x values resulting from simulation of *memory_force* constraints in both shift and capture modes.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *memory_force* (optional): Currently, the memory_force constraint is merged with test_mode constraint and can be used as an argument to the test_mode constraint.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] Memoryforce pin '<pin-hier-name>' simulation value in shift mode for design '<du-name>' is displayed

Arguments

- The pin name, <pin-hier-name>
- Name of the design unit, <du-name>

Potential Issues

The violation message is reported for each -memory_force argument of the [test_mode](#) constraint in shift mode.

Consequences of Not Fixing

The memory_force constraint is ignored, if the violation message is not fixed.

How to Debug and Fix

Double-click the violation message to back-annotate all the nodes with non-x values resulting from simulation of that pin.

Message 2

[INFO] Memoryforce pin '<pin-hier-name>' simulation value in capture mode for design '<du-name>' is displayed

Arguments

- The pin name, <pin-hier-name>
- Name of the design unit, <du-name>

Potential Issues

The violation message is reported for each -memory_force argument of the [test_mode](#) constraint in capture mode.

Consequences of Not Fixing

The memory_force constraint is ignored, if the violation message is not fixed.

How to Debug and Fix

Double-click the violation message to back-annotate all the nodes with non-x values resulting from simulation of that pin.

Message 3

[INFO] Constraint 'memory_force' missing in design '<du-name>'

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message is reported, if the `-memory_force` argument of the `test_mode` constraint is not specified for any pin in a design unit.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Example Code and/or Schematic

For example, assume that the following constraints are included in a constraint file:

```
test_mode -name pinA -value 1 -scanshift
test_mode -name pinB -value 0
memorytype -name memA
memorytype -name memB
test_mode -memory_force -name pinC -value 0
test_mode -memory_force -name pinD -value 0
```

Simulation of both `testmode` and `memory_force` should cause all outputs on all instances of `memA` and `memB` to attain non-x values.

To check that `pinC` on `memA` has value 0 in `testmode`, simulate `pinA = 1` and `pinB = 0`.

To check that `pinD` on `memB` has value 0 in `testmode`, simulate `pinA = 1`.

Default Severity Label

Info

Information Rules

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_memorywritedisable

Displays memorywritedisable propagation data

When to Use

Use this rule to run sanity check on [memory_write_disable](#) constraint. You can also use this rule for design/constraint exploration and debugging.

Description

The Info_memorywritedisable rule displays all signals with non-x values resulting from simulation of [memory_write_disable](#) constraints in both shift and capture modes.

NOTE: *The Info_memorywritedisable rule will be deprecated in a future release.*

The [memory_write_disable](#) constraints define top-level pins and values on those pins that, when applied to a circuit, will drive memory write pins to inactive states. The Info_memorywritedisable rule back-annotates the memorywritedisable simulation results onto the structural view. The shift mode and the capture mode are treated as separate “messages.”

Method

Simulate power and ground and all testmode constraints for scan shift and [memory_write_disable](#) constraints. Display the incremental results after simulating the memorywritedisable.

Simulate power and ground and testmode constraints for capture and [memory_write_disable](#) constraints. Display the incremental results after simulating the memorywritedisable.

The Info_memorywritedisable rule will back-annotate the pins that go non-x due to memorywritedisable simulation (similar to the [Info_testmode](#) rule.) The Info_memorywritedisable rule highlights the 1s and 0s signals due to memorywritedisable using two additional colors so that you can <Ctrl>+double-click the Info_testmode rule message and get both results if desired.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *memory_write_disable* (mandatory): Use this constraint to specify the top-level primary ports and values that, when simulated, will disable all write enables to memories.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] Memorywritedisable pin '<pin-hier-name>' simulation value in shift mode for design '<du-name>' is displayed

Arguments

- The pin name, <pin-hier-name>
- Name of the design unit, <du-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Message 2

[INFO] Memorywrite disable pin '<pin-hier-name>' simulation value in capture mode for design '<du-name>' is displayed

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Message 3

[INFO] Constraint 'memorywrite disable' missing in design '<du-name>'

Potential Issues

The violation message is reported, if the memory_write_disable constraint is not specified for any pin in a design unit.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Example Code and/or Schematic

For example, assume the following constraints are included in a constraint file:

```
test_mode -name pinA -value 1 -scanshift
test_mode -name pinB -value 0
memorytype -name memA
memorytype -name memB
memory_write_disable -name pinC -value 0
```

Information Rules

```
memory_write_disable -name pinD -value 0
memory_write_pin -name -memname memA -writeport wr1 -value 1
memory_write_pin -name -memname memB -writeport wc -value 1
```

Simulation of both testmode and memory_write_disable should cause write pin wr1 on all instances of memA and write pin wc on all instances of memB to attain inactive values.

To check that pinC on memA has value 0 in testmode, simulate pinA = 1 and pinB = 0

To check that pinD on memB has value 0 in testmode, simulate pinA = 1

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_noAtspeed

Displays all registers and flip-flops specified as 'no_atspeed'.

Rule Description

The `Info_noAtspeed` rule displays all registers and flip-flops displayed as `no_atspeed`.

Constraints

no_atspeed (Mandatory)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Capture (atspeed)

Message Details

Message 1

After the `Info_noAtspeed` rule has generated the information, the following message is generated:

Flip '`<flip-flop-name>`' is specified as `no_atspeed`

In case, you have not specified the *no_atspeed* constraint for the design unit, the `Info_noAtspeed` rule is not run and the following input Error severity message is generated:

Constraint '`no_atspeed`' not specified for the design `<du-name>`

Arguments

- The list of flip-flops specified with the *no_atspeed* constraint `<flip-flop-list>`
- The name of the design unit `<du-name>`

Schematic highlight

Flip-flops that have been specified with the *no_atspeed* constraint in the SGDC file.

Message 2

[INFO] Generated report <report path> for 'no_atspeed' sgdc constraint.

Potential Issues

The violation message is displayed when the *no_atspeed* constraint is applied on some objects in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This rule generates the no_atspeed report that lists the name of the design objects on which the *no_atspeed* constraint is applied.

Rule Severity

Info

Info_noFault

Reports all the ports and pins which are specified as no fault.

When to Use

Use this rule to detect the number of instances and pins that are treated as no fault.

Description

This rule reports all the instances where the *no_fault* constraint has been specified. It also reports the number of terminals that are treated as no fault.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dft_generate_no_fault_and_add_fault_report*: The default value of the parameter is on. Set the value of the parameter to off to disable report generation for faults marked as *no_fault*.

Constraint(s)

no_fault (mandatory): Use this constraint to prevent faulting for a module.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[INFO] '<no_of_faults>' faults are marked as 'no_fault' due to constraint 'no_fault <sgdc_commands>' in design '<du-name>'

Arguments

- Total number of faults, <no_of_faults>
- User-specified sgdc commands, <sgdc_commands>
- Name of the design unit, <du-name>

Potential Issues

The violation message is reported when the *no_fault* constraint is specified on the instances in a design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] '<no_of_faults>' faults are marked as 'no_fault' due to constraints on specific port and/or pin (using -fault | -net | -net_input | -net_output) in design '<du-name>'

Arguments

- Number of no_fault faults, <no_of_faults>
- Name of the design unit, <du-name>

Potential Issues

The violation message is reported when the no_fault constraint is specified on the instances in a design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 3

[INFO] Report file for no fault blocks '<file_path_name>' is generated

Potential Issues

The violation message is generated when a report file is generated for the no fault blocks.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 4

[INFO] 'no_fault' constraint impacts '<no_of_faults>' (<percentage_of_faults> of total <total_number_of_faults>) faults in design 'top_block'

Arguments

- Number of faults marked as no_fault, <no_of_faults>
- Percentage of no-fault faults with respect to total number of faults, <percentage_of_faults>
- Total number of faults, <total_number_of_faults>

Potential Issues

The violation message is generated to show the impact of the no_fault constraint on a given design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 5

[INFO] Constraint 'no_fault' is not specified for design
<du_name>

Potential Issues

The violation message is generated to inform the user that the no_fault constraint is not specified for the design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

no_fault: List of all the ports and pins that are marked as "no fault" as per user specified no_fault sgdc commands.

Info_noScan

Display all registers and flip-flops specified as 'force_no_scan'

When to Use

Use this rule to run sanity check on the flip-flops specified in the SpyGlass Design Constraints file(s). You can also use this rule for design exploration and debugging.

Description

The Info_noScan rule generates the Atrenta Console highlight information for flip-flops that have been specified with the *force_no_scan* constraint in the SpyGlass Design Constraints file(s).

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

force_no_scan (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

The following violation message is displayed when the `force_no_scan` constraint is not specified in the design:

```
[INFO] Constraint 'force_no_scan' not specified for the design  
<module-name>
```

Arguments

Name of module, <module-name>

Potential Issues

This message appears if you do not specify the `force_no_scan` constraint in the design.

Consequences of Not Fixing

If this violation is not fixed, no flip-flop is considered as a forced no scan.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 2

```
[INFO] <count> Flip-flop in <module-name> is constrained as  
'force_no_scan' [<percentage> of total flop count <total-flip-  
flop-count>]
```

Arguments

- Name of module, <module-name>
- Number of flip-flops, <total-flip-flop-count>
- `force_no_scan` constraint flip-flop count / total flip-flop count, <percentage>

Potential Issues

This message appears when flip-flop in the module is constrained as `force_no_scan`.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 3

[INFO] <count> Flip-flop in '<module-name>' are(is) constrained as 'force_no_scan' by control signal (<control signal type> '<control signal net name>')

Arguments

- Name of module, <module-name>
- Number of flip-flops, <count>

Potential Issues

This message appears when you use `-clock_control /-set_control / -reset_control` option for the *force_no_scan* constraint.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 4

[INFO] '<count>' Flip-flop is constrained as 'force_no_scan' by the constraint '<constraint name>'

Arguments

- Name of module <module-name>
- Number of flip-flops <count>

Potential Issues

The violation message appears when the *force_no_scan* constraint is applied on module or instances.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 5

[INFO] '<count>' Flip-flop(s) is(are) marked as 'force_no_scan' by the constraint '<suffix>'

Arguments

- Number of flip-flops <count>
- Suffix specified to mark the flip-flops as no scan, <suffix>

Potential Issues

The violation message appears when the flip-flops are marked as no scan by the specified suffix.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 6

The following violation message is displayed when the *dft_scannable_latches* parameter is set on.

[INFO] '<type>' specified as 'force_no_scan -name', in design '<du-name>', impacts '<no_of_flip_flops>' flip-flops and '<no_of_latches>' latches

Arguments

- A flip flop or a latch, <type>
- Design unit name, <du-name>
- number of flip-flops, <no_of_flip_flops>
- number of latches, <no_of_latches>

Potential Issues

The violation message is displayed when a flip-flop or a latch in the module is constrained as no scan and the *dft_scannable_latches* parameter is set to

on.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 7

The following violation message is displayed when the [dft_scannable_latches](#) parameter is set off.

```
[INFO] '<type>' specified as 'force_no_scan -name', in design '<du-name>', does not impact any flip-flop
```

Arguments

- Name of the latch, <type>
- Design unit name, <du-name>

Potential Issues

The violation message is displayed when a flip-flop or a latch in the module is constrained as no scan and the [dft_scannable_latches](#) parameter is set to off.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 8

The following violation message is displayed when the [dft_scannable_latches](#) parameter is set off.

```
[INFO] '<type>' specified as 'force_scan -name', in design '<du-name>', impacts <number> flip-flops
```

Arguments

- Name of the flip-flop, <type>

- Top design unit, <du-name>
- Number of affected flip-flops, <number>

Potential Issues

The violation message is displayed when a flip-flop or a latch in the module is constrained as scannable and the [dft_scannable_latches](#) parameter is set to off.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 9

[INFO] Generated report <report path> for 'force_no_scan' sgdc constraint.

Potential Issues

The violation message is displayed when the [force_no_scan](#) constraint is applied on some objects in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation.

How to Debug and Fix

This rule generates the [no_scan](#) report that lists the name of the design objects on which the [force_no_scan](#) constraint is applied.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_noscanFlopsTextReport

The **Info_noscanFlopsTextReport** rule has been deprecated. Use the [Info_inferredNoScan](#) rule instead of this rule.

Info_path

Display blocked path information

When to Use

Use this rule to display all blocked paths in instances.

Rule Description

The Info_path rule displays all blocked paths in instances.

The Info_path rule considers an instance path as blocked if the instance's input pin is blocked and does not have any effect on the output pin. For example, an input to an OR gate is blocked if any other input to the same OR gate is currently assigned a logical 1 value.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power, ground and any available testmode conditions. For each instance of a gate with at least one non-x input, verify that all remaining inputs can still control the output on that gate instance. Mark each blocked input with "BP".

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_ta* (mandatory): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] Blocked path for design block '<du-name>' is displayed under capture mode

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

The test_mode is causing the path blockage.

Consequences of Not Fixing

This may result in coverage reduction.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[INFO] Blocked path for design block '<du-name>' is displayed under shift mode

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

The test_mode is causing the path blockage.

Consequences of Not Fixing

This may result in coverage reduction.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

If all paths in an instance <inst-hier-name> in a design unit are un-

blocked, the Info_path rule generates the following message:

[INFO] Design block '<inst-hier-name>' has no blocked path

Arguments

- Name of the design unit, <du-name>
- Name of the Instance's hierarchy, <inst-hier-name>

Potential Issues

Since this is an informational message, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

The Info_path rule is an informative rule, and hence, requires no debug. The violation message lists all the terminals in design that are blocked under shift/capture conditions.

If you want to see the reason for the blocked terminal, view the Modular Schematic. Select a terminal and view the corresponding Incremental Schematic.

Overlay (Auxiliary violation mode) the Info_testmode rule under shift/capture mode. This helps in identifying the reason for a blocked terminal.

The schematic of the Info_path rule highlights the following:

- Terminals that are unobservable when capture mode simulation is done.
- Terminals that are unobservable when shift mode simulation is done.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

You can also view the violations for the Info_testmode rule along with the violation of the Info_path rule in the Incremental Schematic window. To do this, double-click the violation message for the Info_path rule and open the Incremental Schematic window.

The violation messages for the Info_testmode rule overlaps the violation

Information Rules

for the `Info_path` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Info_path` rule.

Coloring Scheme

The `Info_path` rule uses the following color scheme in the schematics:

Color	Meaning of the Color
Light Red	Blocked Terminals

No fix is required as this is an informational rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports and files.

Info_potDetectable

Display faults that are potentially detectable.

When to Use

Use this rule to detect all potentially detectable faults.

Description

The *Info_potDetectable* detects all the faults that were earlier undetectable after assuming the conditions of scan flip-flops and transparent latches.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftGenerateTextReport*: The default value is none. Set the value of the parameter to either rule name or a space-separated list of rule names to generate the text report.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Clock_24 rule uses the `-testclock` argument of the `clock` constraint.
- *force_scan* (optional): Use this constraint to declare flip-flops as scannable even if they do not so qualify.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Power Ground

Messages and Suggested Fix

[INFO] <fault t-num> faults [<num1> internal and <num2> on ports] in '<module-name>' are potentially detectable. Pin Count = <term-num>

Arguments

- Total number of potentially detectable faults, <fault-num>
- Number of internal potentially detectable faults, <num1>
- Number of potentially detectable faults on ports, <num2>
- Name of top module, <module-name>
- Number of terminals in top module, <term-num>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the pins that have potentially detectable faults.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

This is an informational rule. Therefore, no fix is required.

Example Code and/or Schematic

Not Applicable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

potentially_detected_faults: This report is generated when you set the *dftGenerateTextReport* parameter. This report lists all the faults that were earlier undetectable after assuming the conditions of scan flip-flops and transparent latches.

Info_pwrGndSim

Display the power-ground simulation results.

When to Use

Use this rule for debugging and design exploration.

Description

The Info_pwrGndSim rule generates the Atrenta Console highlight information for simulation results for the power-ground connections.

This information may be superimposed with other simulation results such as [Info_testmode](#) rule whenever desired. The power and ground results are kept separate from other simulation displays to reduce screen clutter.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Messages and Suggested Fix

Message 1

[INFO] No power ground net present in the design '`<top block name>`'

Potential Issues

The violation message appears when there is no supply net in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] 'Power ground' simulation value for design '<top block name>' is displayed

Potential Issues

This message appears when supply net is present in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the power-ground simulation value on each net.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

This is an informational rule. Therefore, no fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_random_resistance

Estimates random pattern fault and test coverage for design

When to Use

Use this rule to estimate random pattern coverage for the design and to generate test points to improve random pattern coverage.

Description

The *Info_random_resistance* rule estimates design fault/test coverage for random patterns. For information on the fault coverage report, see [Viewing Results in Fault Browser](#). This rule supports multiple-threads.

NOTE: *The Info_random_resistance rule supports multiple cores. The basic license for SpyGlass DFT ADV allows you to run only 4 cores.*

SpyGlass allows 8 threads per license. Licenses are released as soon as additional threads are closed. Use the following environment variable to enable incremental release of licenses:

```
setenv SPYGLASS_INCRCHECKIN_LICENSE_SUPPORT 1
```

This rule also generates the test points to improve the random pattern coverage of the design.

NOTE: *To disable the generation of test points, set the value of the [dft_rrf_tp_count](#) parameter to 0.*

The *Info_random_resistance* rule estimates coverage using the steps illustrated in the following figure:

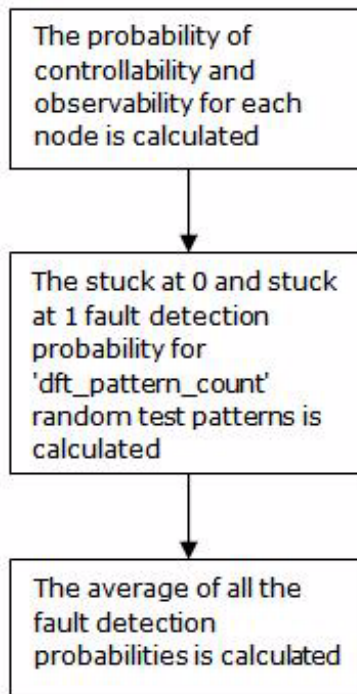


FIGURE 67. Estimating Coverage

Bucketization of Nodes

After calculating the fault detection probability, the *Info_random_resistance* rule performs bucketization for all the nodes.

The *Info_random_resistance* rule assigns a bucket ID to each node on the basis of its probability. A fault with lower detection probability has a lower bucket ID. The *Info_random_resistance* creates the following buckets:

- **Control_0 bucket, c0_<id>:** Assign bucket ID to all nodes based on their control_0 probability. Therefore, c0_00 is assigned to s@1 faults, c0_100 is assigned to easy to detect faults, and remaining nodes are assigned bucket ID from c0_01-c0_99 on basis of their control-0 probability.
- **Control_1 bucket, c1_<id>:** Assign bucket ID to all nodes based on their control_1 probability. Therefore, c1_00 is assigned to s@0 faults,

c1_100 is assigned to easy to detect faults, and remaining nodes are assigned bucket ID from c1_01-c1_99 on basis of their control-1 probability.

- **Observability bucket, o_<id>**: Assign bucket ID to all nodes based on their observe_1 probability. Therefore, o_00 is assigned to unobservable nodes, o_100 is assigned to easy to detect faults, and remaining nodes are assigned bucket ID from o_01-o_99 on basis of their observe probability.
- **Probability of detection bucket, pdt_<id>**: Assign bucket ID all nodes based on their probability of detection, where pdt_00 is assigned to s@0 or s@1 faults, pdt_100 is assigned to easy to detect faults, remaining nodes are assigned bucket ID from pdt_01-pdt_99 on basis of their probability of detection.

The *Info_random_resistance* rule does not display nodes that are not random resistant.

To view the node distribution for each bucket, see [random_pattern_node_distribution](#) report.

To identify test points to reduce random resistance of the design, see [Identifying Test Points To Reduce Random Resistance](#).

Exceptions

For more information, see [Exclusions to RRF Test Point Selection](#).

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftAutoFix*: Use this parameter to generate the modified RTL source files that have the suggested changes to fix the rule-violation of the *Info_random_resistance* rule.
- *dft_pattern_count*: Default value is 64000. Set the value of the parameter to any positive non-zero integer to specify the number of test patterns considered for coverage estimation by the *Info_random_resistance* rule.

- ***dft_coverage_report_depth***: Default value is 0. Set the value of the parameter to all or any positive integer value to specify the hierarchical depth up to which coverage summary is reported in the coverage reports.
- ***dft_rrf_display_limit***: Default value is low. Other acceptable values are high and medium. Set the value of the parameter to any of these values to specify the probability detection bucket count for faults, which need to be highlighted by the violation message generated by the *Info_random_resistance* rule.
- ***dft_target_random_pattern_fault_coverage***: Use this parameter to define target random pattern fault coverage.
- ***dft_target_random_pattern_test_coverage***: Use this parameter to define target random pattern test coverage.
- ***dft_rrf_generate_fault_report***: Default value is off. Set the value of the a parameter to on to generate the *Node List for Probability Distribution* section in the *random_pattern_node_distribution* report. This section lists probability of detection, controllability, and observability bucket IDs.
- ***dft_rrf_tp_count***: Default value is 5%. Set the value of the parameter to any positive number or percentage to specify the required number of test points to be suggested for improving the random pattern fault coverage.
- ***dft_rrf_tp_effort_level***: Default value is medium. Other acceptable values are low or high. Set the value of the parameter to any of these values to specify the effort level for finding test points for reducing random resistance of the design.
- ***dft_rrf_tp_ignore_generated_nets***: Default value is on. Set the value of this parameter to ascertain whether internal nets should be ignored while suggesting test points to reduce the random resistance of faults.
- ***dft_rrf_tp_thread_count***: Default value is 8. Set the value of this parameter to any positive integer value between 4 and 128, to specify the number of threads to be used by the RRF Test-point analysis.
- ***dft_rrf_tp_target_test_coverage***: Default value is 99.99. Set the value of the parameter to define target random pattern test coverage for RRF test point identification.
- ***dft_rrf_tp_generate_dc_report***: Default value is on. Set the value of the parameter to generate the *random_pattern_dc_testpoints* report.

- *dft_rrf_tp_cutoff_incremental_gain*: Default value is 0.00. Specify any decimal number between 1 to 100, as in input to this parameter, to specify the minimum desired gain (as percentage) for a test point to consider it in the design.
- *dft_rrf_tp_count_for_cutoff_incremental_gain*: Default value is 1. Set the value of the parameter to any positive number to specify the number of test points to be considered for cutoff cumulative incremental gain.
- *dft_rrf_tp_use_multiple_threads*: Default value is on. Set the value of the parameter to off to disable use of multiple cpu-cores in RRF test point analysis.

Constraint(s)

- *clock* (optional): Use this constraint to define the clocks of a design. The `Clock_24` rule uses the `-testclock` argument of the `clock` constraint.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *allow_test_point* (optional): Use this constraint to specify modules or instances which should be considered for suggesting test points.
- *no_test_point* (optional): Use this constraint to exclude modules or instances, which should not be considered for suggesting test points.
- *force_probability* (optional): Use this constraint to specify the probability of controllability and observability of a design node.
- *set_fully_decoded_bus* (optional): Use this constraint to specify three state buses for which if one tristate enable is active, it ensure that enables of all the other tristate devices driving it are disabled.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Modes

Capture

Messages and Suggested Fix

Message 1

[INFO] Test points report file '<report file path>' for design '<design name>' is generated

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[INFO] Random pattern stuck_at coverage estimate for design '<du-name>': fault_coverage = <num1>% and test_coverage = <num2>%

Arguments

- Name of the design, <du-name>
- Fault coverage number, <num1>
- Test coverage number, <num2>

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Refer to the schematics shown by the other informational rules that show fault detection, controllability, and observability probabilities.

Message 3

The following message appears to show information on faults:

[INFO] '<no_of_faults>' Faults with lowest '<bucket_count>'

probability of detection bucket IDs are displayed for design '<du_name>'

Arguments

- Number of faults highlighted in the schematic, <no_of_faults>
- Number of lowest non-empty buckets, <bucket_count>
- Design Name, <du_name>

Potential Issues

This is an informational message. See [random_pattern_node_distribution](#) report for more information.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Check the schematic of this violation.

In the schematic displayed by this violation, controllability and observability probabilities are shown and nodes are highlighted by a color that depends on the probability of detecting fault on that node. The **red** color denotes the worst case and the **green** color denotes the best case.

Message 4

[INFO] Suggested test points for autofix to reduce random resistance

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix. For more information, see [Identifying Test Points To Reduce Random Resistance](#).

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

To fix the violation, see [Operating Modes](#).

Message 5

[INFO] Bucket-wise fault distribution report is generated at location <path>

Potential Issues

Not Applicable

Consequences of Not Fixing

Not Applicable

How to Debug and Fix

See [random_pattern_node_distribution](#) to debug specific buckets.

Message 6

[INFO] For Design Unit '<design name>', computed '<coverage>' '<actual_coverage>' is less than target '<corresponding_coverage>' '<target_coverage>'

Arguments

- Design unit name, <design name>
- fault coverage or test coverage, <coverage>
- Actual coverage percentage, <actual_coverage>
- Actual coverage percentage, <target_coverage>

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

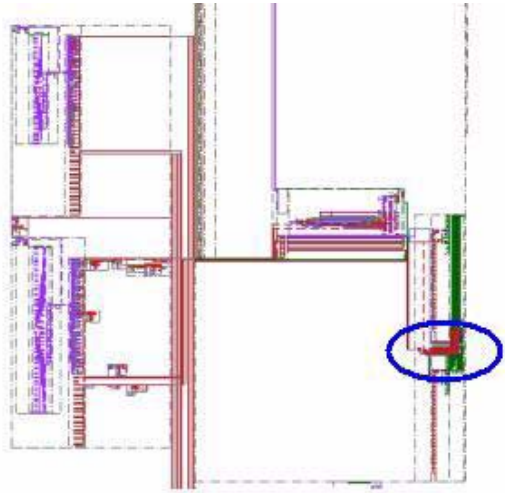
This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

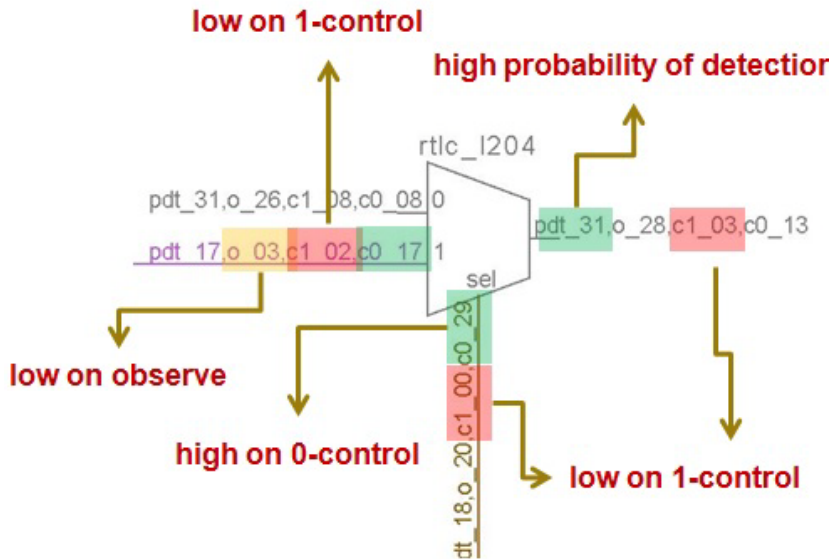
This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic**Example 1**

Consider the following schematic:



The red region denotes the area with the worst observability/controllability/ detection probabilities. You can zoom into the design schematic and the corresponding detailed schematic with probability annotation is displayed as shown below:



Information Rules

Following lists the annotations used in the above schematic:

- Observability bucket: o_<id>
- Control_0 bucket: c0_<id>
- Control_1 bucket: c1_<id>
- Detect probability bucket: pdt_<id>

Where, <id> range is from 0 (lowest) to 100 (highest)

Example 2

Figure 68 displays the fault coverage and test coverage estimates displayed in the Fault Browser:

The screenshot shows the 'Random Pattern Coverage Report' window. The table below represents the data shown in the report.

Instance Hierarchy	Module Name	Fault Coverage Estimate(%)	Test Coverage Estimate(%)	RRF Coverage Loss(%)
RefDesCore	RefDesCore	83.310	86.860	11.550
conmax	wb_conmax_to...	37.950	74.620	0.200
wb_s5m2_pci	pci_bridge32	48.020	58.980	0.910
pci_targ...	pci_target_unit	42.530	55.340	0.900
configur...	pci_conf_space	55.660	59.230	0.000
pci_res...	pci_rst_int	61.540	100.000	0.000
wishbon...	pci_wb_slave_...	65.860	66.250	0.010
pci_io...	pci_io_mux	91.460	100.000	0.000
parity_c...	pci_parity_check	99.480	100.000	0.000
output_b...	pci_cur_out_reg	100.000	100.000	0.000
input_re...	pci_in_reg	100.000	100.000	0.000
wb_s6_mc	mc_top	77.890	90.900	0.090
wb_m0m1...	or1200_top	86.330	86.410	9.980
wb_s2_usb2	usb_top	89.360	98.110	0.030
wb_s1_gpio	gpio_top	91.410	92.220	0.010
wb_s3_irda	irda_top	93.400	93.610	0.130
wb_s0_spi	simple_spi_top	96.420	96.480	0.000
wb_s4m1_eth	eth_top	97.830	97.970	0.180
wb_s7_uart	uart_top	98.820	98.840	0.010

FIGURE 68. Fault, Test, and Random Resistance Coverage

Also, the random resistance faults are displayed in the Incremental Schematic as shown in *Figure 69*:

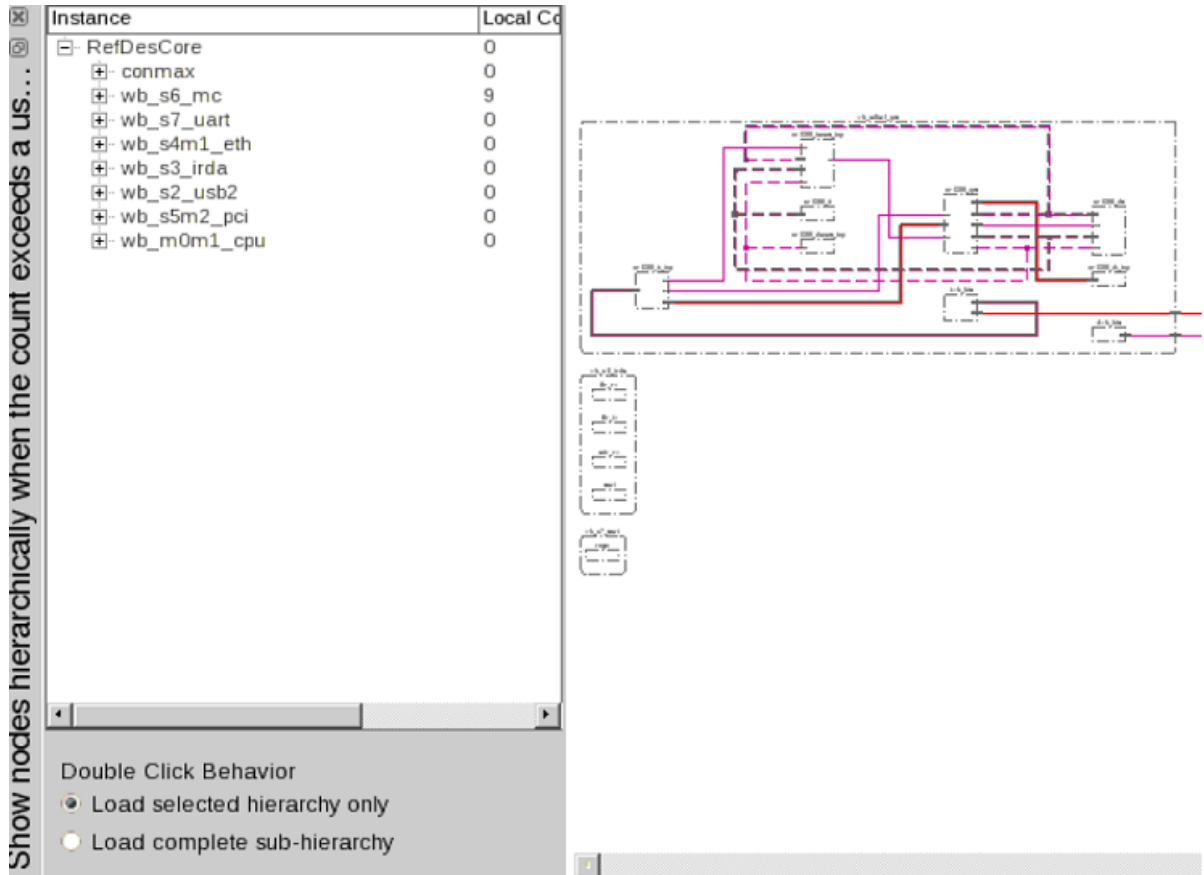


FIGURE 69. Controllability and Observability Probabilities

Information Rules

The rule also generates the *random_pattern_testpoints* report (Figure 70), which lists the test points identified for the design.

```
#####
# Parameter values :
# 'dft_rrf_tp_count'           : 10
# 'dft_rrf_tp_type'           : both
# 'dft_rrf_tp_effort_level'    : medium
# 'dft_rrf_tp_ignore_generated_nets' : on
#####

#####
# Design information :
# Top module : 'RefDesCore'
#####

current_design "RefDesCore"

force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_ic_top.orl200_ic_ram.ic_ram0.\C1117561.C30229.mem_reg[61].data_0.
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_cpu.orl200_sprs.C1_1012_n" -control_one 0.503906 -control_zero 0.508
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_cpu.orl200_rf.rf_we_2" -control_one 0.491353 -control_zero 0.508
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_ic_top.orl200_ic_ram.ic_ram0.addr_1_1" -control_one 0.302961 -con
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_cpu.orl200_rf.C1_86_n_6" -control_one 0.499997 -control_zero 0.50
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_immu_top.itlb_spr_access" -control_one 0.502953 -control_zero 0.50
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_ic_top.orl200_ic_ram.ic_ram0.addr_9_1" -control_one 0.281451 -con
force_probability -name "RefDesCore.wb_m0m1_cpu.orl200_ic_top.orl200_ic_ram.ic_ram0.addr_4_1" -control_one 0.302141 -con
```

FIGURE 70. Random Pattern Testpoints Report

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

- *random_pattern_coverage*: Lists the coverage estimate associated with a module.
- *random_pattern_testpoints*: Lists test points for improving random pattern fault coverage.
- *random_pattern_node_distribution*: Lists all the thresholds, for all three probabilities, and the node distribution for each bucket.
- *random_pattern_dc_testpoints*: Lists the test points for improving random pattern fault coverage which can be used by the Design Compiler as an input.

Info_schain

Displays all properly stitched scan chains

When to Use

Use this rule to display information for all properly stitched scan chains.

Rule Description

The Info_schain rule generates the Atrenta Console highlight information for all properly stitched scan chains.

A scan chain is assumed to be properly stitched scan chain when a valid path is detected between the scanin and scanout points of the scan chain.

The Info_schain rule dumps the following information per scan chain:

- Scan chain Id
- Scanout net name
- Scanin net name
- Clock domain information, that is, number and name
- Scan chain length

The Info_schain rule also dumps the following information about the valid scan chains in a tabular format:

- Cell order
- Scan chain Id
- Cell type
- Shift Clock
- Clock Polarity
- Hier Name
- Library Cell Name

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Info_schain rule.
- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

The following violation message is displayed for the Info_schain rule:

```
[INFO] Scan chain [schain_id: '<sc_id>'], with
'<num_scan_cell>' scan cells, detected between points
[scanin=>'<pin1-name>' scanout=>'<pin2-name>'] under condition
= '<tag-name>'
```

Arguments

- Unique identifier of the scan chain. <sc_id>
- Number of scan cells on the scan chain. <num_scan_cell>

- Name of the scanin pin. <pin1-name>
- Name of the scanout pin. <pin2-name>
- Simulation condition. <tag-name>

Potential Issues

Since this is an informational message, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

The violation message displays the scan chain identified between user specified scanin and scanout points.

The Info_scanchain rule is an informative rule and requires no debug.

No fix is required as this is an informational rule.

Message 2

[WARNING] Scan chain [scanchain_id:'<sc_id>'], with '0' scan cells, detected between points [scanin=>'<pin1-name>' scanout=>'<pin2-name>'] under condition = '<tag-name>'

Arguments

- Unique identifier of the scan chain. <sc_id>
- Name of the scanin pin. <pin1-name>
- Name of the scanout pin. <pin2-name>
- Simulation condition. <tag-name>

Potential Issues

The violation message appears if no scan cell is present between scanin and scanout of the scan_chain constraint

Consequences of Not Fixing

Required testing cannot happen using the scan chain specified by this constraint.

How to Debug and Fix

Review constraint and make sure scanin and scanout points to intended

design nodes.

When the number of nodes reported is more than number of nodes specified in the **Hierarchical view enabled for instance count greater than**: option in the **Preferences** window, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. By default, this value is 100.

Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Message 3

[INFO] Scan chain report file <file_name> is generated

Arguments

Name of the file, which is scan_chain.rpt, and is generated in the <wdir>/spyglass_reports/dft/ directory, <file_name>

Potential Issues

Not Applicable

Consequences of Not Fixing

Not Applicable

How to Debug and Fix

Not Applicable

Example Code and/or Schematic

Consider the following sample information generated by the Info_schain rule:

```

schain_id      : chain_1
scanout        : top2.sol
scanin         : top2.sil
clock domains  : 2 (top2.tclk2, top2.tclk1)
schain length  : 2

```

The Info_schain rule also generates a text file named [scan_chain](#) that has information about flip-flops that are not in any scan chain.

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

[*scan_chain*](#)

Info_scanwrap

Reports scanwrap related information

When to Use

Use this rule to get information about potentially scan wrapped objects.

Rule Description

The Info_scanwrap rule reports violation for the following cases:

- When *scan_wrap* SGDC command is not specified for a given design
- If scanwrap is applied on a non-black box design unit and hence ignored
- Impacted black box instances
- When all enable pins of black box with scan wrap are not active
- When there is difference in size of enable pin list and enable value list

For each such potential *scan_wrap* command, an additional information about controllability, observability, and fault improvement is also reported.

The Info_scanwrap rule generates a report file containing a list of all potential *scan_wrap* command that should be given for the current design.

The *Info_scanwrap* rule does not suggest *scan_wrap* for black box with all input nets observable and all output pins blocked in test_mode so that they do not drive any downward logic. Such black boxes are reported in a separate section in the *scan_wrap* report.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *scan_wrap* (optional): Specifies black box design units or instances that will be designed with scan wrappers.
- *clock* (optional): Use this constraint to define the clocks of a design. The Info_scanwrap rule uses the `-testclock` argument of the `clock` constraint.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

None

Messages and Suggested Fix

Message 1

[INFO] In design '<du-name>', scanwrap sgdc command is not supplied

Arguments

Design unit, <du-name>

Potential Issues

The violation message appears, if your design does not have *scan_wrap* command specified.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Message 2

[INFO] In design '<du-name>', scanwrap '<inst-name>' is applicable on '<bbox-inst>' instances

Arguments

- Design unit, <du-name>
- Black box instance, <bbox-inst>

Potential Issues

The violation message appears if your design has specified scan wrap and is applicable on the black box instances.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Message 3

[INFO] Scanwrap constraint file '<file-name>' is generated

Arguments

Generated file name containing potential scanwrap commands, <file-name>

Potential Issues

The violation message appears when the scan_wrap.rpt report file is generated.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there no debug or fix is required for this violation message.

Messages 4

[WARNING] In design '<du-name>', scanwrap '<inst-name>' is applied on a non-black box design unit and hence ignored

Arguments

- Design unit, <du-name>
- Scanwrap instance name, <inst-name>

Potential Issues

The violation message appears, if your design has scan wrap that is applied on non-black box design unit.

Consequences of Not Fixing

The scan_wrap constraint is ignored, if you do not fix the violation.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights all the instances that are affected.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

To fix the violation, ensure that [scan_wrap](#) constraint is applied on a black box.

Message 5

[WARNING] Scanwrap enable is not active for instance <inst-name> in <mode-name> mode

Arguments

- Black box instance name, <inst-name>
- Capture atspeed or Capture or Shift, <mode-name>

Potential issues

The violation message appears for black box instance with scan wrap whose enable pins are not active.

Consequence of not fixing

If you do not fix the violation, this black box is treated as non-scan wrapped.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window

highlights all non active enable pins.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

To fix the violation, ensure that all scan wrap enable pins get their active value.

Message 6

[WARNING] In design '<du-name>', scan_wrap '<bbox-du-name>' has mismatch in enable pin list and enable value list

Arguments

- Name of the design unit, <du-name>
- Black box design unit, <bbox-du-name>

Potential issues

The violation message appears for [scan_wrap](#) constraint with difference in size of enable pin list and enable value list.

Consequence of not fixing

If you do not fix the violation, the rule ignores the [scan_wrap](#) constraint.

How to Debug and Fix

To fix the violation, ensure that size of scan wrap enable pin list and enable value list is same.

Message 7

[INFO] [scan_wrap on '<module-name>' ('<inst-count>' instance) will improve control/observe value of '<node-count>' nodes

Arguments

- Name of the blackbox module, <module-name>
- Number of blackbox instances which are [scan_wrap](#) candidates, <inst-count>
- Gain in controllability and/or observability, <node-count>

Potential issues

This is a informational message which reports gain in controllability and

observability after applying the [scan_wrap](#) constraint.

Consequence of not fixing

Ignoring this message and not applying the [scan_wrap](#) constraint will result in lower controllability and observability.

How to Debug and Fix

Apply the [scan_wrap](#) constraint on reported module if required.

Message 8

[INFO] Specifying scan_wrap on all candidates will improve control/observe value of '<node-count>' nodes

Arguments

Gain in controllability and/or observability, <node-count>

Potential issues

This is an informational message which reports gain in controllability and observability after applying the [scan_wrap](#) constraint.

Consequence of not fixing

Ignoring this message and not applying the [scan_wrap](#) constraint will result in lower controllability and observability.

How to Debug and Fix

Apply the [scan_wrap](#) constraint on reported module if required.

Message 9

[INFO] In design '<top>', scan_wrap '<module name>' is inferred on '<N>' instance(s)

Potential issues

The violation message appears if the value of the [dft_set_scan_wrap_on_memory](#) parameter is on and the scan wrap is inferred on memory instances.

Consequence of not fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix is required for this

violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info/Warning

Rule Group

Information

Reports and Related Files

[scan_wrap Report for the Info_scanwrap Rule](#): This report contains a scanwrap list of all boxes and their related information.

Info_self_gating_logic

Recognizes self-gating cells and identifies the test-points in self-gating logic

When to Use

Use this rule to:

- Automatically recognize the power-compiler-inserted self-gating cells.
- Identify the test-points in the self-gating logic to minimize the significant impact on ATPG run-time and pattern count.

Description

The *Info_self_gating_logic* recognizes self-gating cells and identifies the test-points in self-gating logic.

NOTE: *Power compiler provides an option to insert self-gating logic that minimizes dynamic power consumption.*

The logic uses a transactor gate comparator that disables the clock if the D-input of the flip-flop has the same value as Q-output of the flip-flop. That is, when clocking the flip-flop would not result in a change in its output value.

The *Info_self_gating_logic* rule:

- Automatically identifies the self-gating logic present in the design by:
 - Reporting a metric on the proportion of flip-flops in the design that are self-gated
 - Reporting all flip-flops in the design that are self-gated
- Selects test-points for the self-gating logic to minimize the impact on ATPG run-time and pattern counts by:
 - Reporting the list of test-points for self-gating logic
 - Adding the test-points to the `random_pattern_dc_testpoints.rpt`

The test-points needed are observe points on the output of the XOR cells in the self-gating logic, as shown in the figure below:

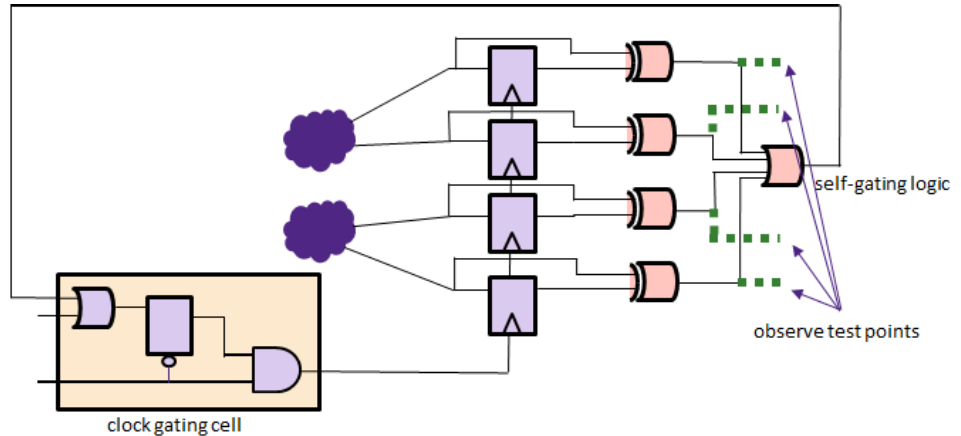


FIGURE 71. Observe Points

Parameter(s)

dft_rrf_tp_generate_dc_report

Constraint(s)

None

Operating Mode

Power Ground

Messages and Suggested Fix

Message 1

[INFO] No self-gated CGC found in design '<design-name>'

Arguments

Top module name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues

related to this violation.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required. However, you can verify whether any self-gating logic is present in the design or not.

Message 2

[INFO] <self-gated-flop-count> flip-flops out of a total of <total-ff-count> in design '<design-name>' are self-gated for a self-gating percentage of <self-gating-%age>. Please refer file '<reportName>' for self-gating logic summary report

Arguments

- Number of self-gated flops identified in the design, <self-gated-flop-count>
- Number of flops present in the design, <total-ff-count>
- Top module name, <design-name>
- Percentage of self-gated flops identified in the design, <self-gating-%age>
- Name of generated report, <reportName>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required.

Message 3

[INFO] Report file '<reportName>' successfully generated. Contains all flops categorized by self-gated CGC in design '<design-name>'

Argument(s)

- Name of the generated report, <reportName>
- Top module name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required. However, you can verify the list of flops present in each self-gating group.

Message 4

[INFO] Report file '<reportName>' successfully generated. Contains all test-points categorized by self-gated CGC in design '<design-name>'

Argument(s)

- Name of the generated report, <reportName>
- Top module name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required. However, you can verify the list of test-points present in each self-gating group.

Message 5

[INFO] In top-module '<design-name>', '<flip-count>' flip-flop(s) are impacted by self-gated CGC '<self-gated-CGC-name>'. Example flip-flop: '<flip-name>'

Argument(s)

- Top module name, <design-name>
- Number of flops impacted by self-gated CGC, <flopCount>
- Name of self-gated CGC, <self-gated-CGC-name>
- Example flop name gated by CGC, <flop-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues related to this violation.

Consequences of not fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to debug and fix

This is an informational rule. Therefore, no fix is required. However, you can verify the number of flops present in each self-gating group.

Example Code and/or Schematic

This is an informational rule and generates a text report. For more information on the reports generated, see *Reports and Related Files*.

Default Severity Label

Info

Rule Group

Information Rule

Reports and Related Files

- [dft_self_gating_logic_summary](#)
- [dft_self_gating_ff](#)
- [dft_self_gating_test_points](#)
- [random_pattern_dc_testpoints](#)

Info_selective_testpoint

Inserts selective testpoints

When to Use

You do not need to enable this rule to insert selective testpoints. This rule is enabled automatically during the second pass of the RTL testpoint insertion, if required.

Description

The Info_selective_testpoint rule performs selective testpoint insertion.

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- *rme_config* (optional):
- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *test_mode* (mandatory): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

NA

Messages and Suggested Fix

The *Info_selective_testpoint* rule reports the following violation message:

[INFO] Inserted test points to improve fault coverage

Clicking on the violation message opens the spreadsheet report. See

[Figure 19](#).

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

Not Applicable

Info_soft_error_propagation

Generates the Soft Error related metrics

Description

The *Info_soft_error_propagation* rule calculates the ISO-26262 standard compliant Single Point Fault Metric (SPFM), Latent Fault Metric (LFM) and the Probabilistic Metric for random Hardware Failures (PMHF). The rule generates a report with an ordered list of registers that can be substituted for improving the SPFM metrics. These metrics are calculated relative to user-defined Safety/Non-Safety-related signals.

You can correct the lower than expected SPFM value by analyzing the *soft_error_registers_spfm* report.

Bucketization of Nodes

The Back annotation data showing bucketed control/observe probability is added in violation messages reported by the *Info_soft_error_propagation* rule and displayed in *SoftErrorBrowser*.

The bucketization of nodes is done by dividing 0 to 1 probability range uniformly into 100 buckets. Bucket-id varies from 0 to 99 with 99 being highest probability value bucket.

The following figure shows the BA data in *SoftErrorBrowser*:

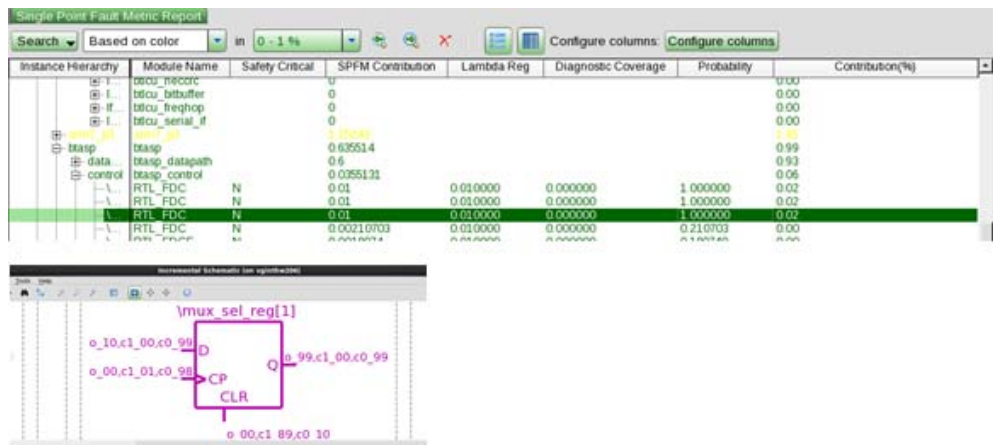


FIGURE 72. BA Data in *SoftErrorBrowser*

Also, the following figure shows sample Info_soft_error_propagation violation message and the corresponding BA data generated:

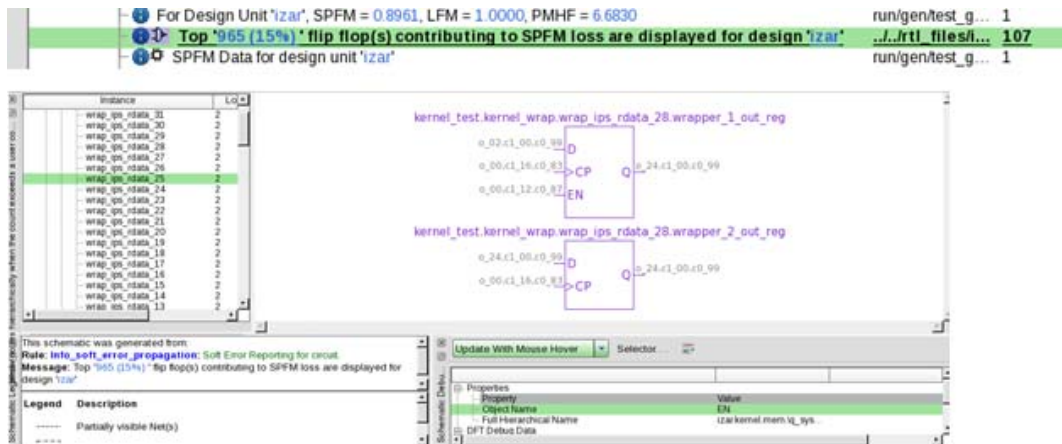


FIGURE 73. BA Data for Info_soft_error_propagation

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- **ser_control_sequential_depth**: The default value is 5. Set the value of the parameter to any non-integer value to specify the number of sequential time frames for which the forward traversal of the design is done when computing the probability of logic value controllability in the design.
- **ser_observe_sequential_depth**: The default value is 5. Set the value of the parameter to any non-integer value to specify the number of sequential time frames for which the backward traversal of the design is done when computing the probability of logic value observability in the design.
- **ser_observe_nsr_initial_value**: The default value is 0.5. Set the value of the parameter to any real value between 0 and 1 (inclusive) to specify the initial observability value assigned to non-safety-related registers

before backward traversal of the design is done to compute observability probabilities.

- *ser_blackbox_observe_probability*: The default value is 1. Set the value of the parameter to any real value between 0 and 1 to specify the observability probability value for blackbox/memory inputs.
- *ser_blackbox_control_0_probability*: The default value is 0.5. Set the value of the parameter to any real value between 0 and 1 to specify control-0 probability value for blackbox/memory outputs. A control-1 probability is calculated by subtracting control-0 probability from 1.
- *ser_propagation_difference_threshold*: The default value is 5. Set the value of the parameter to any real value between 0 and 1 to specify the minimum change in controllability or observability value evaluations for which traversal continues.
- *ser_register_report_top_contributors*: The default value is 15. Set the value of the parameter to any real value between 0 and 1 to specify the percentage of top contributor registers listed in the `soft_error_registers_spfm.rpt` report.
- *ser_ignore_safety_mechanism_register_lreg*: The default value is off. Set the value of the parameter to on to enable the `Info_soft_error_propagation` rule to ignore lreg values for safety mechanism registers.
- *ser_load_enable_n_cycle_persistence_adjustment*: The default value is off. Set the value of the parameter to on to adjust flip-flop probabilities to capture effect of retained error value due to load enable.

Constraint(s)

- *ser_data*: Specifies register properties with respect to soft errors occurrence and detection.
- *safety_mechanism_instance*: Specifies flip-flop's part of safety mechanism
- *define_ser_redundancy_cell*: Specifies the modules to be treated as a single cell with data/out/set/reset/clock/enable pins as specified
- *non_safety_related*: Specifies non-safety related modules/instances/ registers and output ports. By default, all registers are non-safety related. Explicitly specified registers and outputs non-safety overrides the default.
- *safety_critical*: Specifies safety critical modules or instances and outputs.

- *non_safety_critical*: Specifies safety critical modules or instances and outputs.

Operating Mode

Functional

Message Details

Message 1

[INFO] For Design Unit '<design-name>', SPFM = <SPFM Value>, LFM = <LFM Value>, PMHF = <PMHF Value>

Arguments

- The design unit name, <design-name>
- Single point fault metric value, <SPFM Value>
- Latent fault metric value, <LFM Value>
- Probabilistic metric for random hardware failures, <PMHF Value>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Message 2

[INFO] SPFM Data for design unit '<design-name>'

Arguments

- The design unit name, <design-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Example Code and/or Schematic

Currently Unavailable

Message 3

[INFO] Top '<flip-count> (<flip percent>)' flip flop(s) contributing to SPFM loss are displayed for design '<du_name>'

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

- [soft_error_metrics](#): Reports soft error metrics defined in ISO-26262 Standard.
- [soft_error_registers_spfm](#): Reports soft error metric contribution to ISO-26262 Standard of highest contributing registers. Lower than expected

SPFM value can be corrected by analyzing the *soft_error_registers_spfm* report.

Info_stilFile

Generates a STIL file

When to Use

Use this rule to generate a STIL file for existing constraints

Description

The Info_stilFile rule generates a Standard Test Interface Language (STIL) file by reading the RTL files and the supplied constraint files. The generated STIL file may then be used as an intermediate format for subsequent processing.

The STIL file is named as *stil_file* and is generated in the current working directory.

The following constructs need to be generated in the STIL file:

- **Signal:** Defines the name for each top-level port in the design and identifies the port type as In, Out, or InOut.
- **Signal Group:** Defines an ordered set of signals to be referenced in subsequent operations.
- **WaveformTable:** Defines the waveforms that need to be applied to each signal used in the Vector.
- **MacroDefs:** Includes the initialization sequence imposed on the `test_mode` constraint. This includes only the testmode information from the `.sgdc` file.
- **Test setup:** Creates test setup MacroDef from the `testmode` constraints without any options. The `testmode` constraints with `-scanshift`, `-capture` and `-invertInCapture` will not contribute to the test setup MacroDef.
- **Procedures:** Includes only the information from `test_mode` and `testclock` constraints. It contains the default `load_unload`, `capture` procedures and one `capture` procedure for each clock signal.
- **load_unload:** The `testmode` constraints with `-scanshift` and `-invertInCapture` conditions contribute to the `load_unload` procedure. One `load_unload` procedure is created for all the clocks.

- **capture**: The `testmode` constraints with `invertInCapture` and `capture` contribute to the capture procedure.
- **ScanStructures**: The `scan_chain` constraint specified in the SGDC file is translated to the STIL construct `ScanStructure`.

NOTE: *The `Info_stilFile` rule generates only the STIL file. This rule does not support all the STIL constructs.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- **`dftUseOffStateOfClockInClockPropagation`**: The default value of the parameter is `on`. Set the value of the parameter to `off` so that clock lines are kept at X during shift, capture, or `atspeed` mode simulation.

Constraint(s)

- **`clock`** (mandatory): Use this constraint to define the clocks of a design. The `Info_scanwrap` rule uses the `-testclock` argument of the `clock` constraint.
- **`test_mode`** (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- **`scan_chain`** (optional): Use this constraint to specify the scan chains for the `Info_stilFile` rule.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] No 'test_mode' and 'clock -testclock' constraint

provided, hence, stil file is not being generated

Potential Issues

To know more about potential issues pertaining to the violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about how to debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] No 'test_mode' constraint provided, hence, stil file is not being generated

Potential Issues

To know more about potential issues pertaining to the violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To know more about how to debug and fix the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] No 'clock -testclock' constraint provided, hence, stil file is not being generated.

Potential Issues

The violation message appears, if either clock or testmode constraints are not specified, and therefore, no STIL file is generated.

Consequences of Not Fixing

Not fixing the violation may disable the generation of the STIL file.

How to Debug and Fix

To fix the violation, ensure that all the needed constraints are provided.

Message 4

[INFO]<file-name> file has been generated

Arguments

The STIL file which is successfully generated in the current run, <file-name>

Potential Issues

There are no potential issues with respect to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation message.

How to Debug and Fix

No debug or fix information is required.

Example Code and/or Schematic

Not Applicable

Default Severity Label

Info

Rule Group

Information

Reports and Related File

stil_file.rpt: This file lists the STIL constructs for the Info_stilFile rule.

Info_stil_to_sgdc

Generates an SGDC file corresponding to a Stil file.

When to Use

Use this rule to generate an SGDC file corresponding to a Stil file.

Description

The *Info_stil_to_sgdc* rule generates an SGDC file by reading the Stil file and uses the generated SGDC file while rule checking in the same run.

The rule supports the following constructs:

Header	Signals	SignalGroups	ScanStructures
Procedures	Macrodefs	Loop statement	

NOTE: *The Info_stil_to_sgdc rule does not support all the STIL constructs.*

Assumptions

- 'stil_data' sgdc constraint should not be defined multiple times.
- Pseudo signals are not handled in current implementation and a violation message of severity, FATAL, is displayed if for pseudo signals.
- Conflicts can happen if there is already a constraint present on a signal in an SGDC file and same constraint is applied on the same signal through STIL file conversion. In those cases, constraint applied through STIL file conversion prevails.

Parameters

None

Constraints

stil_data: Use this constraint to specify the STIL file.

Messages and Suggested Fix

Message 1

[FATAL] 'stil_data' constraint has been given multiple times

Potential Issues

The violation message is reported if the stil_data constraint is specified multiple times.

Consequences of Not Fixing

If you do not fix the violation, the SpyGlass run would abort.

How to Debug and Fix

Specify stil_data constraint one time for a single run.

Message 2

[FATAL] STIL file not found in the specified path

Potential Issues

The violation message is reported, if the STIL file path specified with '-file' field of 'stil_data' constraint may be not valid.

Consequences Of Not Fixing

If you do not fix the violation, the SpyGlass run would abort

How to debug and Fix

Specify a valid path.

Message 3

[FATAL] Design names in sgdc file and stil file are different

Potential Issues

The violation message is reported when the design name for which the 'stil_data' constraint is specified in the SGDC file and the design name specified in STIL file don't match.

Consequences Of Not Fixing

If you do not fix the violation, the SpyGlass run would abort

How To Debug And Fix

Specify the 'stil_data' constraint for same design as in STIL file.

Message 4

[FATAL] Resolve Pseudo signal <signal name>

Potential Issues

While parsing STIL file, pseudo signal is encountered.

Consequences Of Not Fixing

If you do not fix the violation, the SpyGlass run would abort.

NOTE: Pseudo signals are not handled and should be avoided.

How To Debug And Fix

Avoid pseudo signals in STIL file implementation.

Message 5

[INFO] <file-name> file has been generated

Potential Issues

This is an informational message.

Consequences Of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation message.

How to debug and Fix

No debug or fix information is required.

Message 6

[INFO] SGDC report file < file path to 'dft_generated_sgdc_from_stil.sgdc' > has been generated

Potential Issues

This is an informational message.

Consequences Of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation message.

How to debug and Fix

No debug or fix information is required.

Message 7

[INFO] STIL file parsed without any errors. Check <file path to 'dft_stil_parse.log' > for more details.

Potential Issues

This is an informational message.

Consequences Of Not Fixing

This is an informational rule. Therefore, there are no direct consequences

of not fixing the violation message.

How to debug and Fix

No debug or fix information is required.

Message 8

[FATAL] STIL file parsing failed with errors. Check <file path to 'dft_stil_parse.log'> for more details.

Potential Issues

The violation message is displayed when the STIL file parsing fails.

Consequences Of Not Fixing

SpyGlass run aborts unless the violation is fixed.

How to debug and Fix

Refer to the dft_stil_parse.log file for details on parsing errors.

Example Code and/or Schematic

Consider the following SGDC file snippet:

```
current_design des_unit  
stil_data -file test.stil -mode Internal_scan
```

For a particular mode, these details are extracted and converted from a STIL file to SGDC commands as shown in the following figures.

The following figure shows the SGDC conversion for ScanStructures construct:

Information Rules

```

ScanStructures Internal_scan {
  ScanChain "1" {
    ScanLength 242;
    ScanIn "test_s11";
    ScanOut "test_s01";
    ScanEnable "test_se";
    ScanMasterClock "clk_st";
  }
  ...
}

Timing {
  WaveformTable "default_WFT_" {
    Period '100ns';
    Waveforms {
      "clk_st" {
        P {
          '0ns' D;
          '45ns' U;
          '55ns' D;
        }
      }
    }
  }
}

```

```

scan_chain -scanin "test_s11" -scanout "test_s01"

clock -name "clk_st" -testclock -value rtz -atspeed

```

FIGURE 74. ScanStructures SGDC Conversion

The following figure shows the SGDC conversion for MacroDefs construct:

```

MacroDefs Internal_scan {
  "test_setup" {
    W " default_WFT ";
    C {
      "all_inputs" = \r48 N;
      "all_outputs" = \r32 X;
    }
    V {
      "clk_st" = 0;
      "TM1" = 1;
      "TM2" = 0;
    }
    V {
    }
  }
}

"load_unload" {
  W " default_WFT ";
  C {
    "all_inputs" = 0 \r40 N 10 \r5 N;
    "all_outputs" = \r32 X;
  }
  "Internal_scan_pre_shift" : V {
    "test_se" = 1;
    "i_din[0]" = 0;
  }
  Shift {
    V {
      "_clk" = P;
      "_si" = ###;
      "_so" = ###;
    }
  }
  V {
    "test_se" = 0;
  }
}

test_mode -name "clk_st" -value 00X
test_mode -name "TM1" -value 11
test_mode -name "TM2" -value 00

test_mode -name "clk_st" -value 0X -scanshift
test_mode -name "i_din[0]" -value 0 -scanshift
test_mode -name "TM1" -value 1 -scanshift
test_mode -name "TM2" -value 0 -scanshift
test_mode -name "test_se" -value 1 -scanshift

test_mode -name "test_se" -value 0 -capture

```

FIGURE 75. MacroDefs SGDC Conversion

The following figure shows the SGDC conversion for Procedures:

Information Rules

```

Procedures Internal_scan {
  "multiclock_capture" {
    W "multiclock_capture_WFT_";
    C {
      "all_inputs" = 0 \r40 N 10 \r5 N;
      "all_outputs" = \r32 X;
    }
    F {
      "TM1" = 1;
      "TM2" = 0;
    }
    V {
      "_pi" = \r48 #;
      "_po" = \r32 #;
    }
  }
}

```

test_mode -name "TM1" -value 1 -capture
test_mode -name "TM2" -value 0 -capture

FIGURE 76. Procedures SGDC Conversion

The following is the snippet of the SGDC generated for this example:

```

current_design des_unit

scan_chain -scanin "test_si1" -scanout "test_so1"
scan_chain -scanin "test_si2" -scanout "test_so2"
scan_chain -scanin "test_si3" -scanout "test_so3"
scan_chain -scanin "test_si4" -scanout "test_so4"

clock -name "clk_st" -testclock -value rtz -atspeed

test_mode -name "clk_st" -value 00X
test_mode -name "TM1" -value 11
test_mode -name "TM2" -value 00

test_mode -name "clk_st" -value 0X -scanshift
test_mode -name "i_din[0]" -value 0 -scanshift
test_mode -name "TM1" -value 1 -scanshift
test_mode -name "TM2" -value 0 -scanshift
test_mode -name "test_se" -value 1 -scanshift

test_mode -name "TM1" -value 1 -capture
test_mode -name "TM2" -value 0 -capture

```

FIGURE 77. SGDC**Default Severity Label**

Info

Rule Group

Information

Reports and Related File

- **dft_generated_sgdc_from_stil.sgdc**: This file lists the STIL constructs for the Info_stilFile rule.
- **dft_stil_parse.log**: This file lists the parsing details, such as, procedure, macrodef names, for the syntactically correct parsed STIL file. Also, for the syntactically incorrect parsed STIL file, this file lists the syntax error details. It is generated in the following directory:

```
<current-working-directory>/spyglass_reports/dft/>
directory.
```

Info_synthRedundant

**Displays pins (for an RTL design) that are likely to be absent in an optimized netlist.
Displays faults (for a netlist design) which fall in either TIED or BLOCKED category.**

When to Use

Use this rule at the RTL stage before synthesis for finding SR pins and on a netlist design for identifying tied+blocked faults.

Description

- Pins that are tied high/low
- Pins blocked in the power-ground mode
- Unconnected combinational logic - all logic is combinational connected to nodes that do no fan-out to the primary port
- Undriven combinational logic - combinational logic that is not being driven by any primary input port

Perform the following steps to allow constant propagation through flip-flops and to mark them SR, if required conditions are met:

1. Set the value of the `dft_ignore_constant_supply_flip_flops` parameter to on.
2. Specify the following command in the project file:

```
set_option enable_const_prop_thru_seq
```

For more information on SR faults, see Example 2.

Types of Faults

The Info_synthRedundant rule considers following fault types:

- **Synthesis Redundant (SR) Fault:** Logic which may be removed during synthesis are marked as SR. SR faults may be present on a non-netlist design. Exceptionally, following are not marked as SR:
 - Ports are never marked as SR as they are retained during synthesis
 - Flip-flops, by default, are not marked as SR even if their D-pin have constant value in power-ground mode
- **Tied (TI) Fault:** Logic which is tied to power-ground after synthesis are marked as TI. Tied logic in a netlist design may have such faults. Also, ports may be marked as TI.

For more information on TI faults, see Example 3.

- **Blocked (BL) Fault:** Logic which gets blocked due to power-ground simulation after synthesis are marked as BL. Blocked (due to power-ground) logic in a netlist design may have such faults. Also, ports may be marked as BL.

For more information on BL faults, see Example 3.

NOTE: *For a netlist, no SR marking is done since synthesis has already been run. However, the Info_synthRedundant rule still operates on a netlist and marks the TI and BL faults.*

Method

Simulate power-ground conditions. Using power-ground simulation values, ascertain whether each input of an instance can influence the output. If not, then it is blocked, and marked as SR. Then identify, SR nets. These are nets, which have only SR terminals (no ports) in their fan-out. Mark all ports/terminals in their fan-in as SR. Thirdly, mark the combinational logic in the fan-out of the hanging nets leading to the primary output. Lastly, mark inputs of all SR instances (ones whose every output is SR), as SR.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [showPowerGroundValue](#): Default value is on. Therefore, the rule displays the simulation value of net due to power/ground. Set the value of the parameter to off to hide the power/ground simulation values of a net.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Power Ground

Messages and Suggested Fix

Message 1

[INFO] <num> faults in '<du-name>' are synthesis redundant

Arguments

- Number of synthesis-redundant faults, <num>
- Name of the design unit, <du-name>

Potential Issues

The violation message appears for each top-module for which the number of SR faults is reported. As both s@0 and s@1 are ignored together for a SR pin, it is obvious that the number of SR pins is half the count reported. Each SR pin is back-annotated with "SR".

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation message.

How to Debug and Fix

To know more about debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[INFO] All faults in '<du-name>' would be retained after synthesis

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message appears, if all faults in a design unit, <du-name>, are retained after synthesis.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation message.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the pins that are unobservable when power ground simulation is done.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

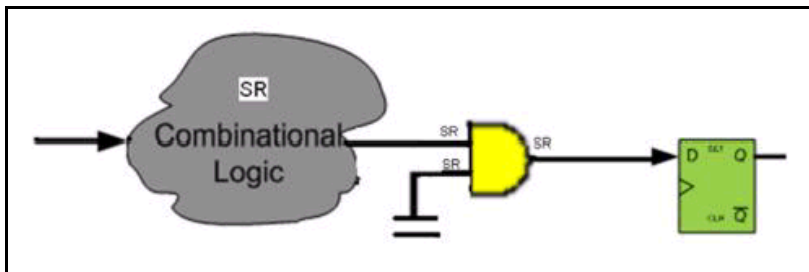
You can also view the violations for the Info_pwrGndSim rule along with the violation of the Info_synthRedundant rule in the Incremental Schematic window. To do this, double-click the violation message for the Info_synthRedundant rule and open the Incremental Schematic window.

The violation messages for the Info_pwrGndSim rule overlaps the violation for the Info_synthRedundant rule in the Incremental Schematic window. This is useful in debugging the violation for the Info_synthRedundant rule.

Example Code and/or Schematic

Example 1

Consider the following figure:

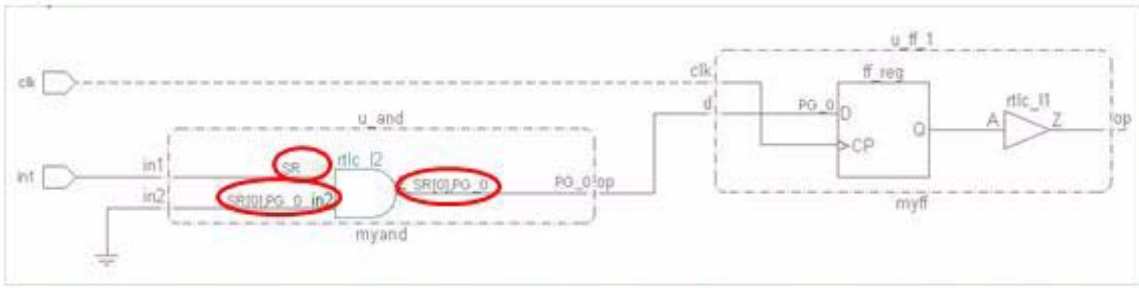


In the above example, the Info_synthRedundant rule reports a violation because as the upper pin of AND gate is tied to a high logic and the lower pin of AND gate is connected to the ground.

The Info_synthRedundant rule reports the pins that are unobservable when power ground simulation is done.

Example 2

Consider the following figure:



Info_synthRedundant[1]: Display pins that are likely to be absent in an optimized netlist.

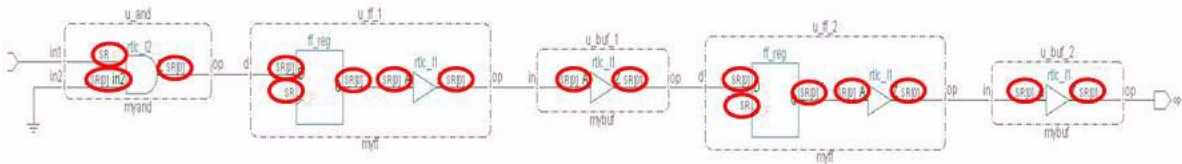
In RTL design : 6 faults in 'top' are synthesis redundant, test.v, 3

In the above schematic, three pins are marked as SR. Also, six faults are considered SR.

The following lists the various fault markings for the above schematic:

- SR: Pin is blocked in PG_MODE (Power Ground Mode)
- SR[0]: Pin has value 0 in PG_MODE (PG_0)
- SR[1]: Pin has value 1 in PG_MODE (PG_1)

Set the value of the [dft_ignore_constant_supply_flip_flops](#) parameter to on and specify the `set_option enable_const_prop_thru_seq` command in the project file to allow constant propagation through flip-flops and to mark them SR, as shown in the following figure:



Info_synthRedundant[1]: Display pins that are likely to be absent in an optimized netlist.

In RTL design : 34 faults in 'top' are synthesis redundant, test.v, 3

In the above schematic, 17 pins are marked as SR and 34 faults are considered SR.

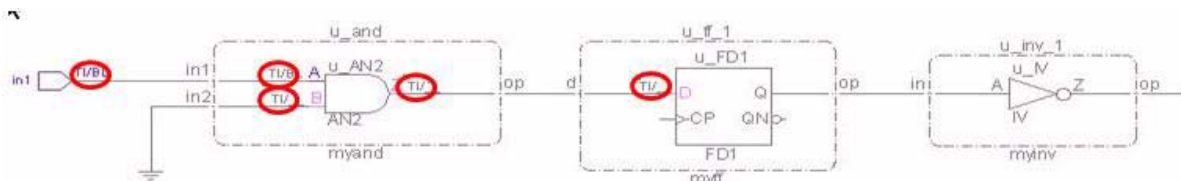
The following lists the various fault markings for the above schematic:

- SR: Pin is blocked in PG_MODE (Power Ground Mode)
- SR[0]: Pin has value 0 in PG_MODE (PG_0)
- SR[1]: Pin has value 1 in PG_MODE (PG_1)

On a Netlist Design

Example 3

Consider the following figure:



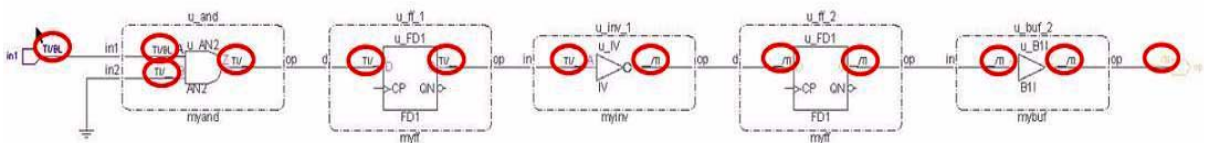
Info_synthRedundant[1]: Display pins that are likely to be absent in an optimized netlist.

In Netlist design : 7 (Tied+Blocked) faults in 'top',test.v, 3

In the above schematic, two faults are marked as BL and five faults are marked as TI.

The schematic marking xx/yy means that xx and yy can be either TI (tied) or BL (blocked). Also, __ means that fault is neither tied nor blocked.

Set the value of the *dft_ignore_constant_supply_flip_flops* parameter to on and specify the set_option enable_const_prop_thru_seq command in the project file to allow constant propagation through flip-flops and to mark them SR, if required conditions are met. The resultant schematic is shown in the following figure:



Info_synthRedundant[1]: Display pins that are likely to be absent in an optimized netlist.

In Netlist design : 15 (Tied+Blocked) faults in 'top',test.v, 3

In the above schematic, two faults are marked as BL and thirteen faults are

Information Rules

marked as TI.

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_testclock

Display test clock propagation.

When to Use

Use this rule to display test clock propagation information to diagnose scannability and capture problems.

The Info_testclock rule ignores escape names for clocks.

Description

The Info_testclock rule displays testclock propagation information for both scanshift and capture.

The Info_testclock rule generates two rule messages for each testclock so that clock propagation during scanshift and capture can be separately displayed.

Default Weight

10

Language

Verilog, VHDL

Method

scanshift:

Simulate power and ground and all testmode conditions. For each test clock, simulate a positive going pulse and display all nets with a pulse. For each displayed net, the value for that net will be back annotated at the source and at all sinks as follows:

if simulation result is a positive pulse then display " ^ " on the net

if simulation result is a negative pulse then display "V" on the net

capture:

Simulate power and ground and all testmode conditions without scanshift. For each test clock, simulate a positive going pulse and display all nets with a pulse. For each displayed net, the value for that net will be back annotated at the source and at all sinks as follows:

if simulation result is a positive pulse then display " ^ " on the net

if simulation result is a negative pulse then display "V" on the net

NOTE: *Vectors (multiple bits in the same bus) are merged into a single line on the schematic. Consequently, the display for testclock paths is suppressed for portions*

of their clock trees that are merged into buses. Each testclock path is also highlighted in color to facilitate panning and zooming.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_show_scan_clock_tree](#): The default value is 2. Set the value of the parameter to off to disable generation of the [Scan Clock Tree Report](#) and viewing results in the Clock Browser.
- [dftCPMechanism](#): The default value is sequential. Set the value of the parameter to parallel to specify the clocking methodology to be used by the Info_testclock rule.
- [dftInferredDriverControl](#): The default value is optimistic. You can set the value of the parameter to pessimistic to specify that the presence of a single non-scan source (black box, hanging net, noscan) in the unblocked fan-in cone will make the node uncontrollable.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (mandatory): Use this constraint to define the clocks of a design.

Operating Mode

[Scanshift](#), [Capture](#)

Messages and Suggested Fix

Message 1

[INFO] Testclock <clk-name>' propagation in <mode> mode. It hits <num> flip-flop(s)

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

Since this is an informational message, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

Double-click the violation message to open the [Clock Browser window](#) window and the [Scan Clock Tree Report](#) report.

The **Clock Browser** window displays flip-flop, latch, black box, hard macro, or memory as leaf node with clock polarity. The blocked nodes are displayed as leaf nodes and are marked with the keyword, BLKD. For more information on viewing results in Clock Browser, see [Scanning a Clock Tree](#).

Also, the [Scan Clock Tree Report](#) report lists the clock domains interacting with the clock, the gating signals, and the blocking nodes.

Message 2

In case, no testclock specification is found for a design unit <du-name>, the Info_testclock rule generates the following message:

```
[INFO] Constraint 'clock -testclock' missing in design  
<du-name>
```

Arguments

- Name of the testclock. (<clk-name>)
- Mode as capture or shift. <mode>
- Number of flip-flops where the specified testclock is reaching. (<num>)

Potential Issues

Since this is an informational message, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

The Info_testclock rule helps in debugging violations of the Clock_11 and

Clock_11_capture rules. It is an informative rule, and hence, requires no debug.

Use the scanshift rule message when diagnosing scannability problems (for example, the [Clock_11](#) rule) and the capture rule message when diagnosing capture problems.

Simulation value of each user-specified testclock for the circuit when the testclock is simulated under the shift and capture simulation modes.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Coloring Scheme

The `Info_testclock` rule uses the following color scheme in the schematics:

Color	Meaning of the Color
Green	Net having rising edge
Blue	Net having falling edge

No fix is required as this is an informational rule.

Message 3

When the value of the [dft_scannable_latches](#) parameter is set to on, the `Info_testclock` rule reports the following violation message:

[INFO] Testclock <clk-name>' propagation in <mode> mode. It hits <num> flip-flop(s) and <num> latches

Arguments

- Name of the testclock. (<clk-name>)
- Mode as capture or shift. <mode>
- Number of flip-flops where the specified testclock is reaching. (<num>)

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this violation.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required.

Message 4

When the value of the [dft_scannable_latches](#) parameter is set to off, the *Info_testclock* rule reports the following violation message:

```
[INFO] Testclock <clk-name>' propagation in <mode> mode. It hits <num> flip-flop(s)
```

Arguments

- Name of the testclock. (<clk-name>)
- Mode as capture or shift. <mode>
- Number of flip-flops where the specified testclock is reaching. (<num>)

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this violation.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Information Rules

Reports and Related Files

[Scan Clock Tree Report](#)

Info_testCounts

This rule has been deprecated.

Info_testmode

Display testmode simulation results

When to Use

Use this rule to display testmode simulation results for both scanshift and capture.

Description

The Info_testmode rule displays all signals with non-x values in testmode scanshift, capture, and functional mode.

NOTE: *The functional mode will be simulated on top of 'Power - Ground' mode.*

The Info_testmode rule generates one violation message each for scanshift and capture conditions.

The Info_testmode rule also allows back-referencing from the Schematic Windows to the SpyGlass Design Constraints file (displayed in the Source Window).

Prerequisites

The Info_testmode rule runs only if any power or ground connections exist or if testmode signals are defined.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode conditions. For each displayed item, the value for that net is back-annotated at the source and at all sinks as follows:

if simulation result = 1 then display "1" on the net

if simulation result = 0 then display "0" on the net

Vectors (multiple bits in the same bus) are merged into a single line on the schematic.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [showPowerGroundValue](#): The default value is on. Set the value of the parameter to off to hide the power/ground simulation values of a net.
- [dftShowForcedValues](#): The default value is on. Set the value of the parameter to off to hide the user enforced values, which are displayed with a \mathbb{F} in the bracket.
- [dftShowWaveForm](#): The default value is off. Set the value of the parameter to on to enable the generation of waveform information (displayed in the Atrenta Console's Waveform Viewer) by the Info_testmode rule.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

[Scanshift](#), [Capture](#)

Messages and Suggested Fix

Message 1

```
[INFO] 'shift mode' simulation value for design '<du-name>' is displayed
```

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

Since this is an informational message, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[INFO] 'capture mode' simulation value for design '<du-name>' is displayed

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

Since this is an informational message, there are no potential issues related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#)

Message 3

In case, no testmode specifications is found for a design unit <du-name>, the Info_testmode rule generates one of the following messages depending on the mode for which the testmode specifications are missing:

[INFO] Constraint 'test_mode -capture' missing in design <du-name>

[INFO] Constraint 'test_mode -shift' missing in design <du-name>

Arguments

Parent design unit name, <du-name>

Potential Issues

Since this is an informational message, there are no potential issues

related to this violation.

Consequences of Not Fixing

Since this is an informational message, there is no implicit impact of this message.

How to Debug and Fix

The Info_testmode rule helps in debugging the violations of the Async_07, Clock_11, and Latch_08 rules. It is informative rule, and hence, requires no debug.

Use the scanshift rule message when diagnosing scannability problems (for example, the [Clock_11](#) rule) and the capture rule message when diagnosing capture problems.

Simulation value for the circuit when shift and capture simulation modes are simulated.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Coloring Scheme

The Info_testmode rule uses the following color scheme in the schematics:

Color	Meaning of the Color
Dark Blue	Net having simulation value 1
Light Red	Net having simulation value 0

No fix is required as this is an informational rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Information Rules

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_testmode_conflict_01

Displays conflict in user specified testmode with respect to simulation result of fan-in cone.

When to Use

Use this rule to run sanity check on the `test_mode` constraint.

Description

The `Info_testmode_conflict_01` reports conflicts in user-specified testmode. If `test_mode` constraint is specified on an internal node, it may so happen that simulation result of its fan-in cone may be driving that specific internal node to a conflicting value. The `Info_testmode_conflict_01` highlights such conflicts so that the user may confirm such cases.

Rule Exceptions

The `Info_testmode_conflict_01` rule does not check for nets where conflicting `test_mode` SGDC command is specified through non-top current_design. Also, the `test_mode` constraint applied on a non-top module is ignored.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_check_test_mode_conflicts*: The default value is `allow_match_with_X`. Set the value of the parameter to `allow_exact_match` to observe the following rule behavior:

Conflict is considered, if simulation value coming from fanin-cone (internal driver) has conflicting non-X (0 or 1) value as compared to specified `test_mode` value or unknown value ('X' or 'U'). However,

conflict is not considered in case of either exact match or when the internal driver is driving the signal to a high-impedance state ('Z').

Constraint(s)

test_mode (Mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

[INFO] Net '<net-name>' has conflicting forced value ". "(Forced: <forced-value>, Computed: <computed-value>) during <mode-name> mode"

Arguments

- Mode as capture, shift, or Capture at-speed, <mode>
- Name of the net, <net-name>
- Forced value as either 0 or 1, <forced-value>
- Computed value as either 0 or 1, <computed-value>

Potential Issues

The violation message appears, if specific internal *test_mode* constraint does not match the design intent.

Consequences of Not Fixing

Not fixing the violation may result in incorrect analysis, which is based on incompatible design content.

How to Debug and Fix

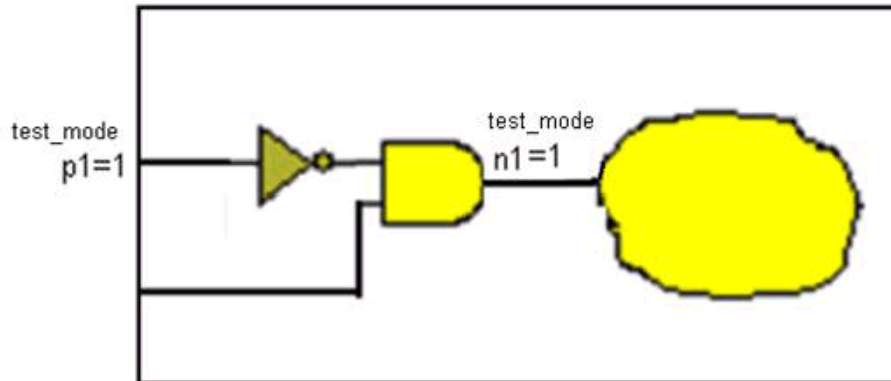
Double-click the violation message. The Incremental Schematic window highlights the gate output that is causing the conflict with simulation value on all its input pins.

To fix the violation, either change the design or change the internal *test_mode* constraint.

Example Code and/or Schematic

Example 1

Consider the following figure:



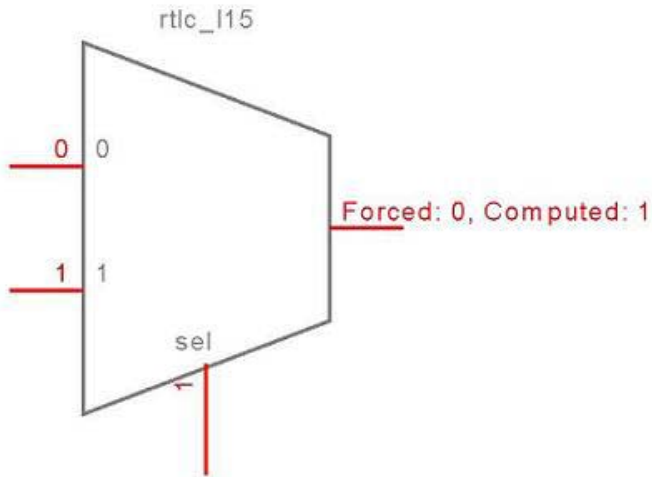
Also, consider the following testmode constraint declaration:

```
test_mode -name p1 -value 1
test_mode -name n1 -value 1
```

In the above example, the value of the user-specified test_mode constraint is specified as 1 and the value of the internal test_mode constraint is also defined as 1. When the user-specified test_mode constraint reaches to point n1, it is turned into 0, whereas the internal test_mode constraint is still 1. Therefore, the *Info_testmode_conflict_01* rule reports a violation.

Example 2

Consider the following schematic:



The *Info_testmode_conflict_01* rule reports the following violation message:

Net 'test.io_1_0' has conflicting forced value (Forced: 1, Computed: 0) during Shift mode

Example 3

Consider the following schematic:



When the value of the *dft_check_test_mode_conflicts* parameter is set to *allow_exact_match*, the *Info_testmode_conflict_01* rule reports the following violation message:

Net 'top.op' has conflicting forced value (Forced: 1, Computed: X) during Shift mode

When the value of the *dft_check_test_mode_conflicts* parameter is set to

`allow_match_with_X`, the `Info_testmode_conflict_01` rule does not report any violation message.

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

No related reports or files.

Info_Top_SGDC_Report

This rule does bottom up hierarchical migration of block level constraints

When to Use

Use this rule to generate the SGDC report for the SoC.

Rule Description

The *Info_Top_SGDC_Report* rule generates SGDC report of the top by automatically migrating block-level constraints to the top. To view the details of the report, click [Reports and Related Files](#). The Info_Top_SGDC_Report rule supports bottom up migration of *test_mode* and clock constraints.

NOTE: *While running the Info_Top_SGDC_Report rule, if you set the `dsm_gen_hiersgdc` parameter to on, SpyGlass disables the SoC abstraction flow. For details on this flow, refer to the SpyGlass SoC Methodology Guide.*

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dsm_apply_migrated_constraints*: The default value is on. When the value of the parameter is set to off, the successfully migrated constraints are not used in the migration of remaining constraints.
- *dsmLimitMigrationReportFanin*: The default value is 5. Set the value of the parameter to any positive integer value to limit the number of partially migrated fan-in reports.
- *dsm_gen_hiersgdc*: The default value is off. Set the value of the parameter to on to run the Info_Top_SGDC_Report rule.

Constraint(s)

sgdc (mandatory): Specify a block-level SGDC file to be imported. Use the

-import option.

Operating Mode

Scanshift, Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed:

[Info] Bottom up Migrated constraints file <file_path> is generated.

Potential Issues

This is an informational rule. Therefore, there are no potential issues of this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no implicit consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug or fix info is required for this rule.

Example Code and/or Schematic

Consider the following test.sgdc file:

```
current_design test

test_mode -name tm1 -value 0
test_mode -name tm2 -value 1
```

```
sgdc -import BB block.sgdc
```

To specify the block-level constraints, use the sgdc constraint.

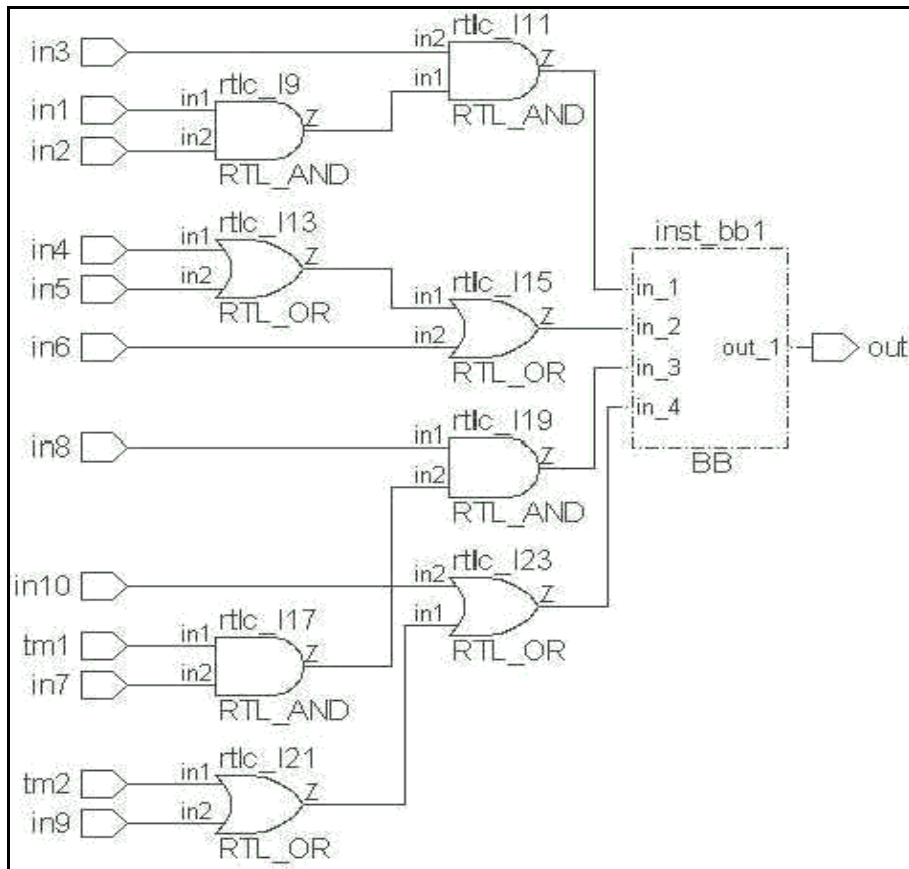
Also, consider the following block.sgdc file:

```
current_design BB
test_mode -name in_1 -value 1
test_mode -name in_2 -value 1
test_mode -name in_3 -value 0
```


Information Rules

```
test_mode -name in_4 -value 0
```

The rule generates the following schematic pertaining to the above declarations:



The above example shows that you can specify the block level constraints by sgdc constraints.

The rule also generates the following top_sgdc_constraints.sgdc report:

```
#####
# TOP MODULE 'test'

# section 1(validated during migration):

#test_mode -name tm1 -value 0 -scanshift
#validated BB block constraint
#test_mode -name test.y2 -value 0

# section 2 (completely migrated):

test_mode -name in1 -value 1
test_mode -name in2 -value 1
test_mode -name in3 -value 1
# section 3(partially migrated):

test_mode -name test.x2 -value 1
#test_mode -name in4 -value 1
#test_mode -name in5 -value 1
#test_mode -name in6 -value 1

# section 4(conflict during migration):

#test_mode -name tm2 -value 1 -scanshift
# BB block constraint 'test_mode -name 'test.z2 -value 0 '
conflict with top level constraints
```

Default Severity Label

Info

Rule Group

Info Rules

Reports and Related Files

top_sgdc_constraints.sgdc file: The report contains all the migrated constraints for top module. The following table lists the sections in the report and their respective description:

Section	Description
Section 1	Reports block-level constraints that are validated by already existing top level constraints.
Section 2	Reports block-level constraints that are migrated successfully to the top.
Section 3	Reports constraints that are partially migrated. For these backtracking from the block constraints is not conclusive but starting points are provided for review.
Section 4	Reports constraints that are conflicting by already existing top level constraints

Info_transitionCoverage

Evaluate upper bound of transition coverage for design

When to Use

Use this rule to compute coverage for design.

Description

The `Info_transitionCoverage` rule computes upper bound of transition delay fault/test coverage. For more information on the Fault Coverage report, see [Viewing Results in Fault Browser](#).

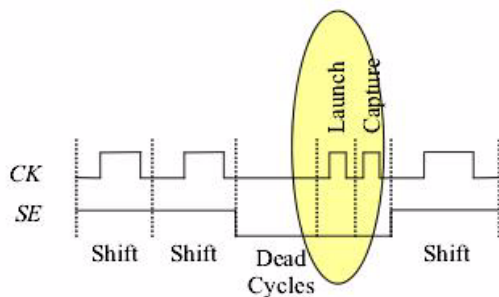
Top-down approach is used to compute transition coverage where we start from all faults marked as detected and then mark undetectable faults. Marking of the undetectable faults can be partitioned into categories and processed separately. The categories are chosen so that if faults in that category exist in the design, then coverage will decrease. This guarantees that if a category in this analysis is used, then the new coverage will decrease but the result will still be an upper bound.

You can test the transition delay using the following ways:

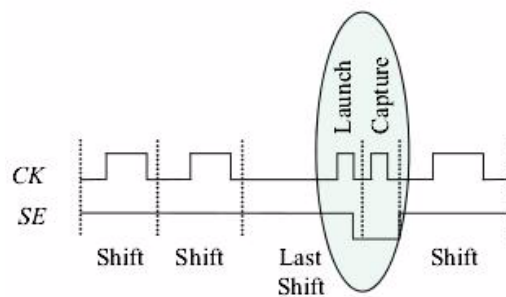
- Launch-on-Capture (LOC)
- Launch-on-Shift (LOS)

Following figure illustrates the LOC and LOS clock cycles:

Launch-on-capture



Launch-on-shift



For coverage estimation purpose, you may select the applicable approach

using the *dsm_launch_method* parameter.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dft_state_holding_ff_identification_effort_level*: The default value of the parameter is 4. Set the value of the parameter to any natural number to change the effort level for identifying the state-holding flip-flops.
- *dft_identify_equivalent_faults*: Default value is off. Set the value of the parameter to on to identify equivalent faults.
- *dft_coverage_report_depth*: Default value is 0. Set the value of the parameter to all or any positive integer value to specify the hierarchical depth up to which coverage summary is reported in the coverage reports.
- *dft_collapse_equivalent_faults*: Default value is off. Set the value of the parameter to on to generate transition fault coverage with fault collapsing in the transition_coverage_collapsed.rpt report
- *dsm_count_false_path_transition_faults*: Default value is off. Set the value of the parameter to on to consider faults on false paths while computing transition fault and test coverage. You can also set the value of the parameter to fault_coverage to consider the faults on the false paths while computing transition fault coverage only.
- *dsmUsePIForAtSpeed*: The default value is on. Set the value of the parameter to off to disable use of PIs to launch transition test.
- *dsmUsePOForAtSpeed*: The default value is on. Set the value of the parameter to off to disable use of POs to capture transition test.
- *dft_target_transition_fault_coverage*: Use this parameter to define target transition fault coverage.
- *dft_target_transition_test_coverage*: Use this parameter to define target transition test coverage.

- *dsm_launch_method*: The default value is LOC. Set the value of the parameter to LOS to define the transition test method as launch on shift.
- *dsmIgnoreControlLineFaults*: The default value is off. Set the value of the parameter to on to exclude transition faults on control signals. Therefore, the rule excludes the clock, scan enable, set, and reset pins from the transition fault space.
- *dsmGenerateTransitionFaultReport*: The default value is none. Use this parameter to generate pin based fault report for the transition faults.
- *dsm_report_domain_for_faults*: The default value is off. Set the value of the parameter to on to report launch and capture atspeed clock domains in the *transition_faults* report
- *dsm_use_cumulative_fault_status*: The default value is off. Set the value of the parameter to *start* for first run of multi-mode analysis. For subsequent runs, you can also specify *cumulative_transition_fault_status.rpt* file generated in a previous run, as an input to this parameter.

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *false_path* (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.
- *no_atspeed* (optional): Use this constraint to exclude flip-flops from being used in at-speed testing, even if they qualify so.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *no_fault* (optional): Use this constraint to prevent faulting for a module.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

Message 1

[INFO] Transition (<mode>) coverage estimate for design '<du-name>': fault_coverage = <fault_coverage>% and test_coverage = <test_coverage>%

Arguments

- Mode can be LOC or LOS, <mode>
- Name of the top design name <du-name>
- Fault coverage percentage, <fault_coverage>
- Test coverage percentage, <test_coverage>

Potential Issues

This is an informational rule. Therefore, there are potential issues related to this violation.

Consequences of Not Fixing

This is an informational rule. Therefore, there is no implicit impact of this violation.

How to Debug and Fix

If the coverage is less than the expected result, perform the following steps:

1. Fix the violations, if any, for the Async_07 and Clock_11 rules.
2. Fix the violations, if any, for the Atspeed_11 rule.
3. Check for the un-handled black boxes, that is, black boxes without proper constraints, scan_wrap or module_bypass. If there are unhandled black boxes, put proper constraints, scan_wrap or module_bypass. Re-run SpyGlass.
4. If the coverage is still less than the expected result, check for and fix the violations for the Atspeed_09 rule. Re-run SpyGlass.
5. If the coverage is still less than the expected result, check for and fix the violations for the Diagnose_04 rule.

Alternatively, run the `Info_transitionCoverage_audit` rule and see the specific points affecting the coverage. Follow the recommendations of the [transition_coverage_audit](#) report.

No fix is required as this is an informational rule.

Message 2

[WARNING] For Design Unit '`<du-name>`', cumulative transition fault status info not found, starting fresh analysis

Potential issues

The violation message is reported when cumulative transition fault status info for design unit, `<du-name>` is not present in the `cumulative_transition_fault_status.rpt` file passed through the `dsm_use_cumulative_fault_status` parameter.

Consequence of not fixing

Not fixing the violation message causes multi-mode analysis to start again for the design unit, `<du-name>`.

How to Debug and Fix

No debug is required for this violation.

To fix the violation, provide the cumulative fault status data generated in the previous design run.

Message 3

[WARNING] For Design Unit '`<du-name>`', mismatch found with previous run cumulative transition fault status data '`<mismatch cause>`', starting fresh analysis

Arguments

- Design Unit Name, `<du-name>`
- Cause of mismatch in design. It shows previous and current term/port count in case of mismatch, `<mismatch cause>`

Potential issues

The violation message is reported when change is detected in the design unit, `<du-name>` with respect to previous run.

Consequence of not fixing

Not fixing the violation message causes multi-mode analysis to start again

for the design unit, <du-name>, ignoring the cumulative fault status information.

How to Debug and Fix

No debug is required for this violation.

To fix the violation, provide the cumulative fault status data generated in the previous design run.

Message 4

[INFO] For Design Unit '<du-name>', first transition coverage result of current run is generated at '<file path>' to be used for subsequent multi-mode run, if needed

Arguments

- Design unit name, <du-name>
- Cumulative fault info file path, <file path>

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 5

[INFO] For Design Unit '<du-name>', cumulative transition coverage result of previous and current run is generated at '<file path>' to be used for subsequent multi-mode run, if needed

Arguments

- Design unit name, <du-name>
- Cumulative fault info file path, <file path>

Potential issues

This is an informational message. Therefore, there are no potential issues

related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 6

[INFO] For Design Unit '<design name>', computed '<coverage>' '<actual_coverage>' is less than target '<corresponding_coverage>' '<target_coverage>'

Arguments

- Design unit name, <design name>
- fault coverage or test coverage, <coverage>
- Actual coverage percentage <actual_coverage>
- Actual coverage percentage <target_coverage>

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 7

[INFO] For Design Unit '<du-name>', transition fault space is zero

Potential issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of not fixing

Information Rules

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to debug and fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

- [*transition_coverage*](#)
- [*transition_faults*](#)
- [*transition_coverage_clockdomain*](#)
- [*transition_coverage_collapsed*](#)

Info_transitionCoverage_audit

Analyze transition coverage for circuit

When to Use

Use this rule to identify major causes of low at-speed test (transition delay) coverage.

Description

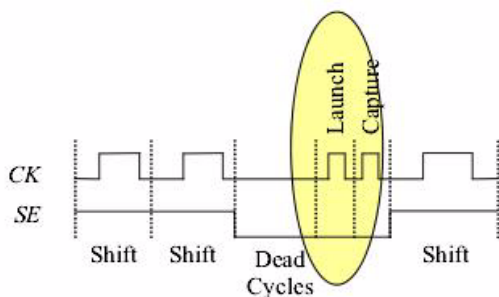
The Info_transitionCoverage_audit rule reports the major causes of low at-speed test (transition delay) coverage. However, the rule does not diagnose these causes. To diagnose the causes of low at-speed coverage, use the at-speed rule(s) corresponding to a particular cause.

You can test the transition delay using the following ways:

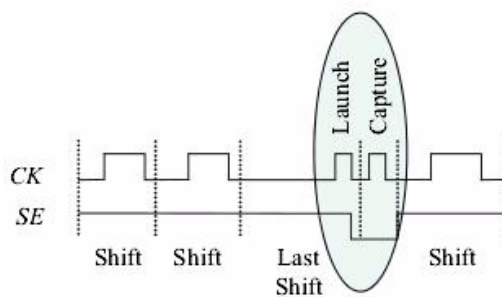
- Launch-on-Capture (LOC)
- Launch-on-Shift (LOS)

Following figure illustrates the LOC and LOS clock cycles:

Launch-on-capture



Launch-on-shift



For coverage estimation purpose, you may select the applicable approach using the [dsm_launch_method](#) parameter.

The Info_transitionCoverage_audit rule generates [transition_coverage_audit](#) report describes the predicted improvement in transition coverage number after each major step.

Default Weight

10

Language

Verilog, VHDL

Method

For each cause, the rule will find the cause and list the coverage that would be obtained if that cause was fixed.

After the incremental coverage improvement for each cause is computed, the new total coverage is listed. This total coverage will be an increasing value that approaches 100%. The following categories are the primary reason for coverage data not reaching 100%:

1. Non scan flip-flops : Non scan flip-flops reduce circuit controllability and observability. Use Async_07/Clock_11 to debug
2. Black box : Terminals of non-scan-wrapped BB's are unobservable and uncontrollable
3. 'force_no_scan' flip-flops : 'force_no_scan' and inferred no scan flip-flops cannot be used for atspeed testing
4. At speed domains of scan flip-flops: Scannable flip-flops must be clocked by atspeed testclocks. Use Atspeed_11
5. Use PIs and POs : Set dsmUsePIForAtSpeed to 'on' to use primary inputs for launch. Set dsmUsePOForAtSpeed to 'on' to use primary outputs for capture
6. Uncontrollable scan flip-flop d-pins : Uncontrollable scan flip-flop data pins prevent launch of transition edges. Use Atspeed_09
7. Clock domain crossing : Faults near domain crossings are not detectable
8. 'no_at-speed' faults : User-specified 'no_at-speed' faults are not considered detectable
9. Atspeed testmode value : Testmode values in atspeed mode restrict ATPG
10. Tristate enable : Enable pins of tristate devices are unobservable
11. 'force_ta constraints : forca_ta reduces testability of a design
12. Hanging terminals : Hanging terminals reduces controllability
13. Feedback flip-flops : Flip-flop q to d feedback may restrict launch transitions
14. False and multicycle paths : Faults on false paths and multicycle paths are not detectable
15. Undetected faults due to PG nets : Power ground values restrict ATPG

NOTE: *Flip-flops that are declared and inferred as no_scan, will not be forced scannable.*

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dft_generate_robustness_audit_report*: The default value is off. Set the value of the parameter to on to generate robustness audit report.
- *dsm_count_false_path_transition_faults*: Default value is off. Set the value of the parameter to on to consider faults on false paths while computing transition fault and test coverage. You can also set the value of the parameter to fault_coverage to consider the faults on the false paths while computing transition fault coverage only.
- *dsm_launch_method*: The default value is LOC. Set the value of the parameter to LOS to define the transition test method as launch on shift.
- *dft_show_rule_name_in_audit*: The default value is off. Set the value of the parameter to on to print the action required for each step of the coverage audit report.
- *dsmUsePIForAtSpeed*: The default value is on. Set the value of the parameter to off to disable the use of PIs to launch transition tests.
- *dsmUsePOForAtSpeed*: The default value is on. Set the value of the parameter to off to disable the use of POs to capture transition tests.
- *dsmIgnoreControlLineFaults*: The default value is off. Set the value of the parameter to on to exclude transition faults on control signals. Therefore, the rule excludes the clock, scan enable, set, and reset pins from the transition fault space.

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *false_path* (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.

- *no_atspeed* (optional): Use this constraint to exclude flip-flops from being used in at-speed testing, even if they qualify so.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *no_fault* (optional): Use this constraint to prevent faulting for a module.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The following violation message is displayed for the *Info_transitionCoverage_audit* rule:

```
[INFO] For Design Unit '<du-name>' [Step <num1>: <reason>],
transi ti on Faul t-Coverage = <num2> transi ti on Test-Coverage =
<num3>
```

Arguments

- Name of the top design name <du-name>
- Step number <num1>
- Step description <reason>, which can have any one of the following values:
 - Non scan flip-flops
 - At speed domains of scan flip-flops
 - Uncontrollable data of scan flip-flops
 - Feedback flip-flops
 - Combinational reconvergence
 - Uncontrollable logic due to hanging terminals
 - Tristate enable used for capture
 - PI and PO used for launch and capture
 - Clock domain crossing

- Fault coverage number <num2>
- Test coverage number <num3>

Potential Issues

Violation is reported due to one of the reasons mentioned in the [Arguments](#) section.

Consequences of Not Fixing

Not fixing the violation may result in low transition coverage.

How to Debug and Fix

The Info_transitionCoverage_audit rule explains the steps to increase the transition coverage. This is an informative rule and requires no debug.

To fix the violation, check the violations reported by all SpyGlass DFT ADV rules.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Debug

Reports and Related Files

[transition_coverage_audit](#) report

Info_uncontrollable

Show nets with imperfect controllability

When to Use

Use this rule to identify nets that are not fully controllable.

Rule Description

The Info_uncontrollable rule displays controllability for all nets that are not fully controllable.

Default Weight

10

Language

Verilog, VHDL

Method

Perform controllability analysis. For each node that is not fully controllable, back-annotate the three controllability values for that node at the source and at all sink pins, fed by this node, as defined by the following table:

Display	Control to 0	Control to 1	Control to z
nnn	no	no	no
nny	no	no	yes
nyn	no	yes	no
nyy	no	yes	yes
ynn	yes	no	no
yny	yes	no	yes
yyn	yes	yes	no
yyy	yes	yes	yes

NOTE: Vectors (multiple bits in the same bus) are merged into a single line on the schematic display. Consequently, the controllability display is suppressed for nodes that are merged into buses.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftTreatFlipFlopsAsScan*: The default value is off. Set the value of the parameter to on to treat all flip-flops as scannable for the Info_uncontrollable rule.
- *dftTreatLatchesAsTransparent*: The default value is off. Set the value of the parameter to on to treat all latches as transparent for the Info_uncontrollable rule.
- *dftTreatBBoxAsScanwrapped*: The default value is off. Set the value of the parameter to on to treat all black boxes as scan-wrapped for the Info_uncontrollable rule.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the DFT type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.
- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[INFO] 'Uncontrollable nets' in design <du-name> are being displayed under full-scan-ready condition

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

See [TA_09](#) rule.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[INFO] 'Uncontrollable nets' in design <du-name> are being displayed under capture condition

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

See [TA_09](#) rule.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[INFO] All nets in design <du-name> are controllable under full-scan-ready condition

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

See [TA_09](#) rule.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[INFO] All nets in design <du-name> are controllable under capture condition

The violation messages displayed for 'full-scan-ready' condition appear only if any (or all) of the rule parameters `dftTreatFlipFlopsAsScan`, `dftTreatLatchesAsTransparent`, or `dftTreatBBoxAsScanwrapped` is set to on. The messages display details depending on the set rule parameter(s).

Arguments

Name of the design unit. <du-name>)

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

See [TA_09](#) rule.

How to Debug and Fix

This is an informative rule and hence requires no debug. The rule displays the uncontrollability of entire design on per net basis. To debug root cause of bad controllability, debug the [TA_09](#) rule.

View the Modular Schematic to see how uncontrollability is distributed across design. Uncontrollability is displayed in the following format on per net basis:

(yyy.....nnn)

Here, 1st bit corresponds to controllability to 0, second bit corresponds to controllability to 1 and last bit corresponds to controllability to Z. For tristated nets this means anything other than yyy.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

You can also view the violations for the Info_testmode (under capture condition) and Info_testclock rules along with the violation for the Info_uncontrollable in the Incremental Schematic window. To do this, double-click the violation message for the Info_uncontrollable rule and open the Incremental Schematic window.

The violation messages of the Info_testmode and Info_testclock overlap the violation for the Info_uncontrollable rule in the Incremental Schematic window. This is useful in debugging the violation for the Info_uncontrollable rule.

To fix the violation, try and fix the [TA_09](#) rule violations.

Example Code and/or Schematic

See the [TA_09](#) rule.

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_undetectedCause

Display undetectable fault information.

When to Use

Use this rule to identify the number of undetectable faults in the design.

Description

The Info_undetectedCause rule reports the number of undetectable faults and displays pin controllability and pin observability back onto the structural view for undetectable faults.

Default Weight

10

Language

Verilog, VHDL

Method

Faults are declared undetectable if controllability to the complement of the stuck @ value is not possible or if the stuck @ pin is not observable.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *ATPG_credit*: The default value is 0. Set the value of the parameter to any floating point value between 0 and 1 to regulate the effect of untestable fault count on test-coverage percentage evaluation by the Info_undetectedCause rule.
- *dftGenerateTextReport*: The default value is none. Set the value of the parameter to space-separated rule name list enclosed in double quotes to specify whether the corresponding rules generate a text report.
- *dftClockEdgeFaultAnalysis*: The default value is off. Set the value of the parameter to identify the faults that could not be detected as the capture happens at opposite edge of testclock used for these faults.
- *dftPOSDETECT_credit*: The default value is 0. Set the value of the parameter to a floating-point number between 0 and 1 to regulate the effect of potentially detectable faults on the test coverage and fault coverage evaluation.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_wrap* (optional): Use this constraint to specify the black box design units or instances that will be designed with scan wrappers.
- *seq_atpg* (optional): Use this constraint to specify the case analysis conditions.
- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

The following violation message is displayed for the Info_undetectedCause rule:

```
[INFO] Design Unit '<du-name>' has <num> testable but
undetected faults. Pin count = <pin-count>
```

Arguments

- Name of the design unit. <du-name>
- Number of undetectable faults. <num>

- Total pin count in the design unit. <count>

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

Not fixing the violation may result in lower coverage.

How to Debug and Fix

Double-clicking the violation message displays the causes for the undetected fault on the structural view, in the following format:

Controllable to 0	Controllable to 1	Observability of pin	Structural Display
n	n	n	nn-(n)
n	n	y	nn-(y)
n	y	n	ny-(n)
n	y	y	ny-(y)
y	n	n	yn-(n)
y	n	y	yn-(y)
y	y	n	yy-(n)
y	y	y	yy-(y)

The Info_undetectedCause is an informative rule and hence requires no debug. It displays controllability and observability of terminals on which stuck@ fault is undetected. To debug root cause of bad controllability and bad observability, debug the [TA_09](#) rule, respectively.

You can also view the violations for the Info_uncontrollable, Info_unobservable, Info_noScan, Info_forcedScan, and Info_inferredNoScan rules along with the violation of the Info_undetectedCause rule in the Incremental Schematic window. To do this:

Double-click the violation for the Info_undetectedCause rule and open the Incremental Schematic window.

Double-click the violation messages for the Info_uncontrollable, Info_unobservable, Info_noScan, Info_forcedScan, and Info_inferredNoScan rules while holding the <Ctrl> key on the keyboard.

When the number of nodes reported is more than 100, the Hierarchy

button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

The violation messages for the Info_uncontrollable, Info_unobservable, Info_noScan, Info_forcedScan, and Info_inferredNoScan rules overlap the violation message for the Info_undetectedCause rule in the Incremental Schematic window. This is useful in debugging the violation for the Info_undetectedCause rule.

To fix the violation, fix the [TA_09](#) rule violations.

Example Code and/or Schematic

See [TA_09](#) rule.

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

[undetected_faults](#): This report lists the undetected faults and their causes. To generate this report, set the value of the [dftGenerateTextReport](#) parameter to the rule name.

To view the detailed fault report, you can refer to the [stuck_at_faults](#) report. This report is generated by the [Info_coverage](#) rule when the [dftGenerateStuckAtFaultReport](#) parameter is set to a value other none.

Info_unobservable

Show all unobservable pins

When to Use

Use this rule to detect all unobservable pins.

Description

The Info_unobservable rule displays unobservable pins.

Observability of a pin is different from the observability of the connected net. For example, consider a 2-input AND gate, with one pin grounded and the second pin connected to a net which fans out to a primary output. The connected net is observable but the second pin is not (with respect to the AND gate).

Default Weight

10

Language

Verilog, VHDL

Method

Perform observability analysis. If a node is not observable then display "N" on that pin

NOTE: *Vectors (multiple bits in the same bus) are merged into a single line on the schematic display. Consequently the display of observability data is suppressed for pins that are merged into buses.*

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftTreatFlipFlopsAsScan*: The default value is off. Set the value of the parameter to on to treat all flip-flops as scannable for the Info_unobservable rule.
- *dftTreatLatchesAsTransparent*: The default value is off. Set the value of the parameter to on to treat all latches as transparent for the Info_unobservable rule.
- *dftTreatBBoxAsScanwrapped*: The default value is off. Set the value of the parameter to on to treat all black boxes as scan-wrapped for the Info_unobservable rule.

- *dftUUMarking*: The *dftUUMarking* parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_wrap* (optional): Use this constraint to specify the black box design units or instances that will be designed with scan wrappers.
- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

When unobservable terminals/ports in a are detected under scan-ready conditions, the following message appears:

```
[INFO] 'Unobservable terminals/ports' in design <du-name> are being displayed under full-scan-ready condition
```

Arguments

To view the list of message arguments, click [Arguments](#)

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

Not fixing the violation may result in lower coverage.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

When unobservable terminals/ports in a are detected under testmode conditions, the following message appears:

```
[INFO] 'Unobservable terminals/ports' in design <design-name>  
are being displayed under capture condition
```

Arguments

To view the list of message arguments, click [Arguments](#)

Potential Issues

See [TA_09](#) rule.

Consequences of Not Fixing

Not fixing the violation may result in lower coverage.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

When all terminals/ports in a design are detected under scan-ready conditions, the following message appears:

```
[INFO] All terminals/ports in design <du-name> are observable  
under full-scan-ready condition
```

Arguments

To view the list of message arguments, click [Arguments](#)

Potential Issues

This is an informational message. Therefore, there are no potential issues with respect to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

When all terminals/ports in a design are detected under testmode conditions, the following message appears:

[INFO] All terminals/ports in design <du-name> are observable under capture condition

Arguments

To view the list of message arguments, click [Arguments](#)

Potential Issues

This is an informational message. Therefore, there are no potential issues with respect to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

NOTE: *The violation messages displayed for 'full scan-ready' condition appear only if any (or all) of the rule parameters `dftTreatFlipFlopsAsScan`, `dftTreatLatchesAsTransparent`, or `dftTreatBBoxAsScanwrapped` is set to on. The messages display details depending on the set rule parameter(s).*

Message 5

If all terminals in a design unit <du-name> are observable, the Info_unobservable rule generates the following message:

[INFO] All terminals in design '<du-name>' are observable

Arguments

Name of the design unit. <du-name>

Potential Issues

This is an informational message. Therefore, there are no potential issues with respect to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

This is an informative rule, and hence, requires no debug. The rule displays the unobservability of the entire design on per term basis. To debug root cause of bad observability, debug the [TA_09](#) rule.

View the Modular Schematic to check how unobservability is distributed across design. Unobservability is displayed as n on per term basis.

The unobservable node information display.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

You can also view the violations for the Info_testmode (under capture condition) and Info_uncontrollable rules along with the violation of the Info_unobservable rule in the Incremental Schematic window. To do this, double-click the violation message for the Info_unobservable rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_uncontrollable rules overlap the violation for the Info_unobservable rule in the Incremental Schematic window. This is useful in debugging the violation for the Info_unobservable rule.

Coloring Scheme

The Info_unobservable rule uses the following color scheme in the schematics:

Information Rules

Color	Meaning of the Color
Navy Blue	Unobservable Terminals

No fix is required as this is an informational rule.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information rules

Reports and Related Files

No related reports or files.

Info_untestable

Dislays untestable faults caused by test mode

When to Use

Use this rule to detect the number of untestable faults and the number of device pins for those faults.

Description

The Info_untestable rule reports violation for the following faults:

- Faults that are blocked during capture or captureAtspeed mode.
- Faults that are caused by testmode signal assertions and are put under untestable (UT) faults.
- Faults that are blocked due to capture or captureATspeed mode simulation of each test_mode constraint.
- Faults that are blocked because of more than one test_mode constraint during capture or captureATspeed mode simulation.

NOTE: *The CaptureAtSpeed mode will be simulated on top of 'Power - Ground' mode.*

Default Weight

10

Language

Verilog, VHDL

Method

Do fault analysis.

Iterate over all the pins and ports the categorize faults in S1, S0, S01.

Report the message.

If no test_mode constraint provided exit

Simulate power ground mode.

Iterate over each test_mode constraint simulate it over power ground.

Check faults which are getting blocked due to test_mode.

Report such blocked faults with their count.

Simulate all the test_mode constraint together

Check for which are getting blocked due to combination of test_modes.

Report such faults with their count. End

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (mandatory): Use this constraint to define the clocks of a design.
- [scan_type](#) (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- [scan_wrap](#) (optional): Use this constraint to specify the black box design units or instances that will be designed with scan wrappers.
- [force_ta](#) (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.

Operating Mode

[Capture](#)

Messages and Suggested Fix

Message 1

[INFO] All faults in '<du-name>' are testable

Arguments

Name of the design unit, <du-name>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this violation.

How to Debug and Fix

Not Applicable

Message 2

[INFO] '<number_of_faults>' faults are blocked due to test_mode constraint on '<name>' in <mode>

Arguments

- Number of blocked faults, <number_of_faults>
- Name specified in the test_mode constraint, <name>
- Operating mode capture / captureAtspeed, <mode>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no implicit consequences of not fixing this violation. However, you can use the highlighted schematic to trace the faults blocked due to test_mode.

How to Debug and Fix

Not Applicable

Message 3

[INFO] '<number_of_faults>' faults are blocked due to combination of all test_mode. constraints for top block '<module_name>' in <mode>

Arguments

- Number of blocked faults, <number_of_faults>
- Design name, <module_name>
- Operating mode capture / captureAtspeed, <mode>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no implicit consequences of not fixing this violation. However, you can use the highlighted schematic to trace the faults blocked due to multiple test_mode

constraints.

How to Debug and Fix

Not Applicable

Message 4

[INFO]test_mode constraint '<node_name>' is applied on node(s) which are unused in mode '<mode>'

Arguments

- Name in test_mode, <node_name>
- Simulation mode, <mode>

Potential Issues

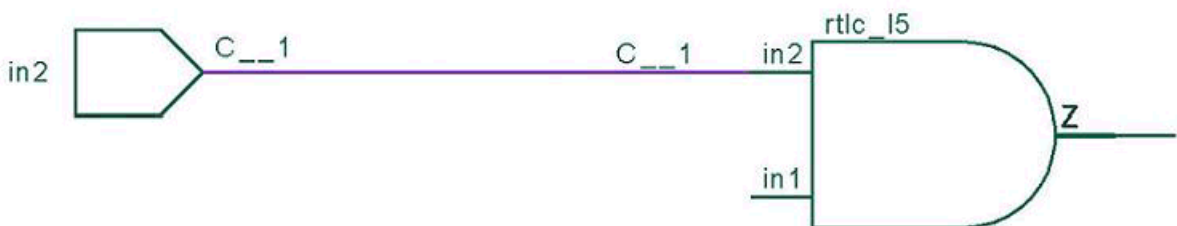
A violation is reported if you have applied the test_mode constraint on an unused design node.

Consequences of Not Fixing

This is an informational message. Therefore, there are no implicit consequences of not fixing this violation. However, you can use the highlighted schematic to trace the faults blocked due to test_mode.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic is displayed with the violating node highlighted as shown in the following figure:



This message does not require any fix as this is an informational message.

Message 5

[INFO]<number_of_faults> faults are blocked due to logical redundant nodes stuck at <stuck_at_value> in '<mode>'

Arguments

- Number of faults, <number_of_faults>
- Stuck at value, <stuck_at_value>. You can specify either 0 or 1 as input to this argument.
- Simulation mode, <mode>. Currently, only capture mode is supported.

Potential Issues

A violation is reported due to the presence of logical redundant faults, that is, either stuck-at 0 or stuck-at 1, in the design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no implicit consequences of not fixing this violation.

How to Debug and Fix

Debug and fix the violations reported by the [Info_logicalRedundant](#) rule.

Message 6

[INFO] <untestf-num> faults (over <term-num> terminal(s)) in '<du-name>' are untestable

Arguments

- Number of untestable faults, <untestf-num>
- Number of terminals, <term-num>
- Name of the design unit, <du-name>

Potential Issues

A violation is reported due to presence of test_mode in capture.

Consequences of Not Fixing

Not fixing the violation may result in faults becoming untestable, which may result in lower coverage

How to Debug and Fix

The violation message displays terminals, which have constant value during capture mode simulation.

Double-clicking the message will back annotate the untestable faults onto the structural view.

If you want to see the reason for untestable terminal, view the Modular Schematic. Select a terminal and open the Incremental Schematic. Overlay (Auxiliary violation mode) the Info_testmode rule under the capture mode. This helps in identifying the reason for constant value propagation.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

To view the violation for the Info_testmode (under capture condition) rule along with the violation of the Info_untestable rule in the Incremental Schematic window, double-click the violation for the Info_untestable rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Info_untestable rule in the Incremental Schematic window. This is useful in debugging the violation for the Info_untestable rule.

Coloring Scheme

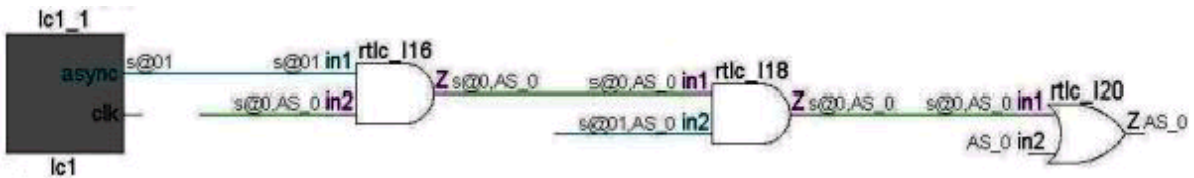
The `Info_untestable` rule uses the following color scheme in the schematics:

Color	Meaning of the Color
Dark Blue	Terminal which is s@0
Light Red	Terminal which is s@1
Red	Terminal which is s@01
Green	Instance which has s@ terminal

No fix is required as this is an informational message.

Example Code and/or Schematic

Consider the following schematic:



Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_unstable_testmode_registers

Report flip-flops, intended to be used as a configuration register, which are unable to retain their respective initialized 0/1 value during scan shift operation and create unstable test mode conditions.

When to Use

Use this rule to display initialized flip-flops which may not be able to retain their respective non-X (0 | 1) state during scan shift operation.

Description

The Info_unstable_testmode_registers rule reports flip-flops, intended to be used as configuration register, which are unable to retain their respective initialized 0/1 value during scan shift operation and create unstable test mode conditions.

Default Severity Label

Info

Parameter(s)

dft_use_stable_simulation_conditions: The default value is off. When you set the value of the parameter to on, only those non-scan flip-flops, which are not corrupted by scan shift operation retain their non-X value, during test_mode (scanshift/capture/atspeed) simulation.

Messages and Suggested Fix

Message 1

[WARNING] Ignoring unstable testmode registers analysis for design '<design-name>' as the parameter 'dft_use_stable_simulation_conditions' is off

Argument(s)

Top module name, <design-name>

Potential Issues

A violation message is reported when the parameter dft_use_stable_simulation_conditions is off.

Consequences of not fixing

Not fixing the violation will result in no stability analysis for test mode registers.

How to debug and fix

To fix the violation, specify the parameter '-dft_use_stable_simulation_conditions' while running SpyGlass.

Message 2

[WARNING] Flip-flop Q-pin '<ff-Q-pin>', initialized to unstable value '<init-value>', made 'X' during test_mode <mode>

Argument(s)

- Flip-flop Q-pin, <ff-Q-pin>
- Flip-flop initialized value, <init-value>
- Mode, <mode>

Potential Issues

A violation message is reported when the flip-flop initialized to value 0/1 is not able to retain its value during scan shift operation.

Consequences of not fixing

Not fixing the violation may result in forced 'X' value on the flip-flop Q-pin.

How to debug and fix

To fix the violation, review the RTL and constraint specification.

Message 3

[INFO] Flip-flop Q-pin '<ff-Q-pin>', initialized to stable value '<init-value>' during test_mode <mode>"

Argument(s)

- Flip-flop Q-pin, <ff-Q-pin>
- Flip-flop initialized value, <init-value>
- Mode, <mode>

Potential Issues

An informational violation message is reported when the flip-flop initialized to value 0/1 will remain stable during scan shift operation.

Consequences of not fixing

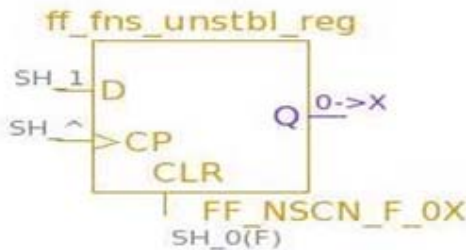
There is no implicit impact of this violation.

How to Debug and Fix

No fix is required. However, you can verify whether the flip-flop is stable at the initialized value (0 | 1) or not.

Example Code and/or Schematic

Consider the following design:



For the above design, the rule reports the following violation messages:

Violation 1

The following is a sample Warning message reported by the rule:

Flip-flop Q-pin 'top.ff_fns_unstbl_reg.Q', initialized to unstable value '0', made 'X' during test_mode

The schematic annotation (BA_DATA) for the same is `FF_NSCN_F_0X`.

Violation 2

The following is a sample Info message reported by the rule:

Flip-flop Q-pin 'top.ff_fns_stbl_reg.Q', initialized to stable value '1' during test_mode

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_unused

Display faults having no effect on system function

When to Use

Use this rule to identify the number of unused faults.

Description

The Info_unused rule displays faults that have no impact on system function.

Such faults are located on devices whose pins do not have a topological path to primary inputs, primary outputs, or both.

Default Weight

10

Language

Verilog, VHDL

Method

Step 1: Analysis to locate unused logic & mark undetectable faults

Input reachability (this marks nets)

Initialize all PIs and scanwrap module outputs to IR

Mark the output of any device with a IR marked input as IR

This marking process does not process any output that is already IR marked (to avoid feedback loops)

Output reachability (this marks nets)

Initialize all POs and scanwrap module inputs to OR

Mark the all inputs of any device with a OR marked output as OR

UD determination

For all nets

If the net is both IR and OR then skip this net (because it is used in system mode)

Mark all pins on this net

S @ 0 as UD (undetectable)

S @ 1 as UD (undetectable)

Note that a net with multiple drivers will get IR marked if any driver is IR.

NOTE: All faults considered as synthesis redundant (SR) for a design is automatically marked as unused (UU) post synthesis or for a pure netlist design.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUUMarking*: The *dftUUMarking* parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *scan_wrap* (optional): Use this constraint to specify the black box design units or instances that will be designed with scan wrappers.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

If the faults in a design unit, *<du-name>* are undetectable, the *Info_unused* rule generates the following message:

```
[INFO] <unusedf-num> faults in '<du-name>' are undetectable after reachability analysis
```

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported due to missing connection in the design.

Consequences of Not Fixing

Not fixing the violation may result in lower fault coverage. Time will be spent on code that is not being used.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

If all faults in a design unit `<du-name>` are detectable, the Info_unused rule generates the following message:

[INFO] All faults in '`<du-name>`' are detectable after reachability analysis

Arguments

- Number of unused faults `<unusedf-num>`
- Name of the design unit. `<du-name>`

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this violation.

How to Debug and Fix

This is an informative rule, and hence, requires no debug. It displays all the terminals in design which do not fan-out to output port.

When the number of nodes reported is more than 100, the Hierarchy button on the Incremental Schematic Window menu bar is enabled. Clicking this button displays the hierarchical representation of the number of nodes reported. For more information on this option, see [Viewing Flat Data Hierarchically](#).

Double-clicking the message back-annotates the unused faults onto the structural view.

The back annotation lists `<S@0 | S@1 | S@01> <!IR | !OR>` where !IR means that this device is not reachable from primary inputs and !OR means that this device is not reachable from primary outputs.

View the Modular Schematic to check the reason for the unused terminal. Select the terminal of interest and open the Incremental Schematic. Probe

the fan-out of the terminal.

Pins that are not synthesis-redundant and have no topological path to an output port

Coloring Scheme

The `Info_unused` rule uses the following color scheme in the schematics:

Color	Meaning of the Color
Yellow	Unused Terminal

No fix is required as this is an informational message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Information Rules

Reports and Related Files

No related reports or files.

Info_x_sources

Reports all the x-sources related information for the design.

When to Use

Use this rule to report all the x-sources related information for the design.

Description

The *Info_x_sources* rule categorizes and reports x-sources present in the design, hierarchical browser displays x-density and x-sources for every instance.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *dft_ignore_x_sources*: The default value is none. Set the value of the parameter to an underscore ('_') separated combination of UIO, BB, MCP, FP, NSF, NSL, CL, HN, MD, and NTL to ignore specified type of x-source(s).
- *dft_x_sources_display_limit*: The default value is low. Set the value of the parameter to high or medium to display the number of x-sources to be highlighted in the violation message by the *Info_x_sources* rule.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to define the clocks of a design.
- *false_path* (optional): Use this constraint to specify false and multi-cycle paths that you want to exclude from at-speed testing.

Operating Mode

Capture

Message Details

Message 1

[INFO] '<number_of_xsources>' x-source(s) of type '<source_type>' are present in design '<design_name>'

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Message 2

[INFO] Ignoring '<LIST>' type of X-sources(s) as specified by parameter 'dft_ignore_x_sources'

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Message 3

[INFO] '<no_of_xsources>' x-source(s) are displayed for design '<design_name>'

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Message 4

[INFO] X-Browser is displayed for design '<design_name>'

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequence of Not Fixing

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Example Code and/or Schematic

Consider the following sample violation messages reported by the Info_x_sources parameter:

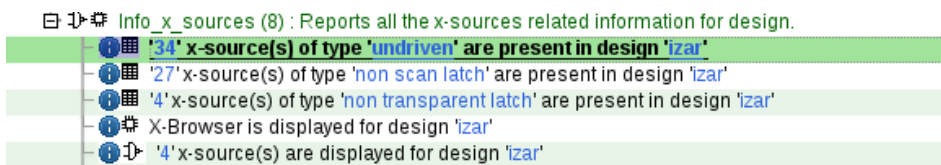


FIGURE 78. Violation Messages

Click on the first three violation messages opens the spreadsheet report, which lists the undriven nets.

Click on the fourth message to display the details of the undriven nets in the X-Browser window, as shown below:

Hierarchy	Module	X-Source Count	Impacted X-Capture Points	X-Capture Count	Captured X-sources	Total Capture Points	Module X-density(10^x)	X-Density(10^x)
[-] izar	izar	65	34	34	34	6430	2.28	2.28
[-] kernel_...	kernel_test	14	2	34	34	398	1.07	2.28
[-] kernel	kernel	19	0	0	0	6032	0.00	0.00

FIGURE 79. The X - Browser Window

Click on the last message to display the Incremental Schematic for the design:

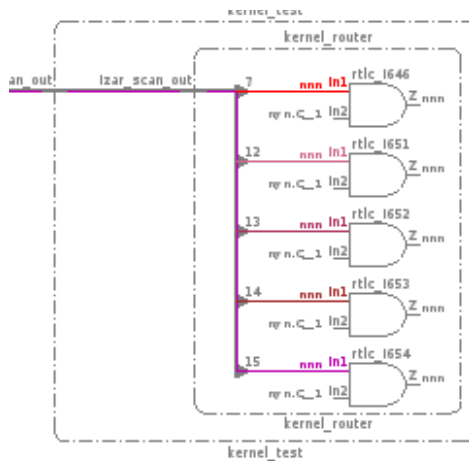


FIGURE 80. Incremental Schematic

Default Severity Label

Info

Rule Group

Information

Reports and Related Files

- [dft_x_source_report](#): This report lists all the x-sources sorted on basis of impacted capture points.

Information Rules

- [*dft_x_capture_report*](#): This report lists all X-Capture points sorted on basis of X-Capture probability.

Latch Rules

Overview

The Latch rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Purpose
Latch_01	Latches that are not transparent during power ground mode.
Latch_02	Asynchronous feedback around latches
Latch_04	Synchronous latches
Latch_06	Cascaded latches that do not have clock phase inversion
Latch_08	Latches that are not transparent in the capture mode or when testclocks are off
Latch_10	Ensure that all latch enables must only be combinational driven through primary ports.
Latch_15	Slave LSSD latches that are directly controlled by more than one master LSSD latch
Latch_16	Latches that are not retiming and non-transparent in Logic DBIST mode
Latch_18	All latch inputs should meet scannability criterion in scan-shift and should be driven by controllable sources in the capture mode
Latch_19	Ensure that all non-transparent and non-shadow latches meet the scannability criterion

Latch_01

Do not use latches unless transparent during power ground mode.

When to Use

Use this rule to identify latches that are non-transparent in the power ground mode.

Description

The Latch_01 rule reports violation for non-transparent latches simulated in the power ground mode.

Method

Simulate power and ground conditions. If any non-transparent latch is found, report a message.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Power Ground

Messages and Suggested Fix

[WARNING] A latch has been inferred for signal '<sig-name>'

Arguments

Name of the latch, <sig-name>

Potential Issues

A violation is reported when a latch is being used in the design.

Consequences of Not Fixing

Latches are sequential devices and therefore should be avoided or forced transparent during power ground mode.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the latches that are not transparent when power ground simulation is done.

You can also view the violation for the Info_pwrGndSim rule along with the violation of the Latch_01 rule in the Incremental Schematic window. To do this, double-click the violation for the Latch_01 rule and open the Incremental Schematic window.

The violation message for the Info_pwrGndSim rule overlaps the violation message for the Latch_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Latch_01 rule.

To fix the violation, either remove the latch or make it transparent.

Example Code and/or Schematic

Example 1

Consider the following sample Verilog code:

```
...
always @(lat_en or d)
    if (lat_en) q <= d;
...
```

The above example demonstrates a bad Verilog code because a latch is used in the design.

Better Verilog

Consider the following sample Verilog code:

```
...
always @(posedge clk)
    q <= d;
```

...

The above example demonstrates a good Verilog code because a flip-flop is used in the design, instead of a latch.

Bad VHDL

Consider the following sample VHDL code:

```
...
Process (lat_en, d)
  Begin
    If (lat_en) then
      Q<= d;
    End if;
  End process;
...
```

The above example demonstrates a bad VHDL code because a latch is used in the design.

Better VHDL

Consider the following sample VHDL code:

```
...
Process (clk)
  Begin
    If (clk'event and clk = '1') then
      Q<= d;
    End if;
  End process;
...
```

The above example demonstrates a good VHDL code because a flip-flop is used in the design, instead of a latch.

Default Severity Label

Warning

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_02

Ensure that no combinational loops exist from transparent latches

When to Use

Use this rule to identify combinational feedback path from latch output pin to the latch data pin.

Description

The Latch_02 rule reports violation for latches that cause combinational loops when forced transparent in capture.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode capture conditions. Walk the unblocked combinational input cone for a transparent latch data pin. Stop the walk at any node with non-x simulation value. If the walk reaches the starting latch q-pin, then generate a message.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftTreatControllableLatchTransparentForLoop](#): The default value is off. Set the value of the parameter to on to treat controllable latches as transparent for the Latch_02 rule.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (mandatory): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Combinational loop, having <num_of_nets> elements (transparent latch count: <num_of_transparent_latches> (Ex: <ex_of_transparent_latches>)), detected in '<mode_name>' mode. Loop path: <loop_path>

Arguments

- Number of nets in the loop, <num_of_nets>
- Number of transparent latches in the loop, <num_of_transparent_latches>
- Example of transparent latches in the loop, <ex_of_transparent_latches>
- Mode name as Capture, <mode_name>
- The path of the loop, <loop_path>

Potential Issues

A violation is reported because either the loop is not broken through test_mode.

Consequences of Not Fixing

Presence of combinational connection from a latch output pin to the data pin of the same latch can result in self-justified signals during ATPG and produce improperly built tests.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic displays the combinational feedback path from the output to the data pin of the latch.

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Latch_02 rule in the Incremental Schematic window. To do this, double-click the violation for the Latch_02 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Latch_02 rule in the Incremental Schematic window. This is useful in debugging the violation for the Latch_02 rule.

To fix the violation, see Example Code and/or Schematic section.

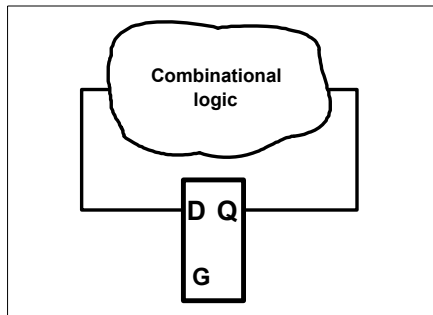
Example Code and/or Schematic

Verilog

Consider the following example:

```
module Disable_tri (en, len, data, ol);  
  input en, len, data;  
  output ol;  
  wire w;  
  reg q;  
  assign w = (data&&en) || (q&&!en);  
  always @ (len)  
    if (len) q <= w;  
endmodule
```

Consider the following figure:



Default Severity Label

Warning

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_04

Ensure that there are no synchronous latches in the design

When to Use

Use this rule to identify synchronous latches in the design.

Description

The Latch_04 reports violation for synchronous latches in the design.

Method

Simulate power and ground and any available testmode capture conditions. Walk the combinational unblocked fan-in cone for latch enables. Stop the walk at any node with a non-x simulation. If any user specified clock pin is reached, report a message.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [test_mode](#) (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

NOTE: *The Latch_04 requires you to specify at least one testclock.*

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Latch '*<sig-name>*' is synchronous with user specified clock '*<clk-name>*'

Arguments

- Latch output net name, *<sig-name>*
- Clock port name, *<clk-name>*

Potential Issues

A violation is reported when synchronous latches are used in the design.

Consequences of Not Fixing

Synchronous latches are latches enabled with clock signals or with a combination of data and clock. Even when enabled by synchronous signals, ATPG tools must be operated in full sequential mode and therefore, test coverage may decrease and generation time may increase.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the latch and the path from the clock pin to the enable pin of the latch.

You can also view the violation for the Info_path rule along with the violation of the Latch_04 rule in the Incremental Schematic window. To do this, double-click the violation for the Latch_04 rule and open the Incremental Schematic window.

The violation message for the Info_path rule overlaps the violation message for the Latch_04 rule in the Incremental Schematic window. This is useful in debugging the violation for the Latch_04 rule.

To fix the violation, make the latches transparent.

Example Code and/or Schematic

Example 1

Consider the following sample Verilog code:

```
module d_latch_rst(enable, data, q);  
    input enable, data;  
    output q;
```

Latch Rules

```
reg q;

always @ (enable or data)
  if (enable == 1'b1)
    q = data;
endmodule
```

The above example demonstrates a bad Verilog code because enable is declared as a clock.

Example 2

Consider the following sample VHDL code:

```
...
Process (enable, data)
Begin
  If (enable = '1') then
    Q<= data;
  End if;
End process;
...
```

The above example demonstrates a bad VHDL code because enable is declared as a clock.

Default Severity Label

Warning

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_05

This rule has been deprecated.

The functionality of the Latch_05 rule is covered by the [Diagnose_ScanChain](#) rule.

Latch_06

Ensure that the cascaded latches have clock phase inversion

When to Use

Use this rule to identify cascaded latches that do not have clock phase inversion.

Description

The Latch_06 rule reports violation for cascaded latches that do not have clock phase inversion.

Method

Simulate power and ground and any available testmode conditions. The unblocked data pin fan-in cone for any latch must not contain other latches unless such a latch is transparent or the enable for such a latch is the complement of the active state of the enable pin on the original latch.

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Latches '`<sig1-name>`' and '`<sig2-name>`' are cascaded without clock inversion

Arguments

- Latch output net name, `<sig2-name>`
- Driving latch net name, `<sig1-name>`

Potential Issues

A violation is reported when cascaded latches are on the same phase.

Consequences of Not Fixing

Not fixing the violation may lead to potential combinational loop and may result in write-through action.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the path between cascaded latches.

To fix the violation, either break the path or change the phase.

Cascaded latches without clock phase inversion are not allowed. If permitted, write-through action may result.

Example Code and/or Schematic

Example 1

Consider the following sample Verilog code:

```
...
always @(lat_en or d1)
    q1 <= d1;

always @(lat_en or f(q1))
    q2 <= d2;
...
```

The above example demonstrates a bad Verilog code as it shows that `q1` and `q2` are cascaded latches on same phase of `lat_en`.

Example 2

Consider the following sample Verilog code:

```

...
always @(lat_en or d1)
  q1 <= d1;

always @(!lat_en or f(q1))
  q2 <= d2;
...

```

The above example demonstrates a good Verilog code because it shows `q1` and `q2` are cascaded latches on different phases of `lat_en`.

Example 3

Consider the following sample VHDL code:

```

...
Process (lat_en, d1)
Begin
  If (lat_en) then
    Q1<= d1;
  End if;
End process;

Fq1 <= f(q1);

Process (lat_en, fq1)
Begin
  If (lat_en) then
    Q2<= d2;
  End if;
End process;
...

```

The above example demonstrates a bad VHDL code as it shows that `Q1` and `Q2` are cascaded latches on same phase of `lat_en`.

Better VHDL

Consider the following sample VHDL code:

```

...
Process (lat_en, d1)
Begin

```

```
    If (lat_en) then
        Q1<= d1;
    End if;
End process;

Fq1 <= f(q1);

Process (not lat_en, fq1)
Begin
    If (lat_en) then
        Q2<= d2;
    End if;
End process;
...
```

The above example demonstrates a bad VHDL code as it shows that Q1 and Q2 are cascaded latches on different phases of lat_en.

Default Severity Label

Warning

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_08

Ensure that the latches are transparent

When to Use

Use this rule to identify latches that are not transparent.

Description

The *Latch_08* rule reports violation for the latches that are not transparent in the capture mode.

However, the *Latch_08* rule doesn't report violation for the following cases:

- Transparent latches
- Shadow latches
- Scannable latches
- Latch is contained in a valid user-defined scan cell

Combinational ATPG tools may treat transparent latches, thereby simplifying test generation. You must exercise caution, in cases where latches are embedded with other sequential devices. In this case, some of the logic may become untestable.

Violations are reported on the sensitized source of the latch enable term. Sensitized source of the latch enable term is found by traversing through the various gates in its path.

For Latch enable source driving on both levels, violations are reported when any one of the latch is non-transparent/controllable-transparent.

Rule Exceptions

When the value of the [*dft_ignore_no_scan_latches_in_rule_checking*](#) parameter is set to on, the *Latch_08* rule ignores forced no_scan latches from rule checking.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dft_check_latch_transparency_in_shift*: The default value is off. Set the value of the parameter to on to enable the *Latch_08* rule to check for latch transparency in capture and shift mode.
- *dft_detect_shadow_latches*: The default value is on. Set the value of the parameter to off to treat all the latches normally, that is, the detection of shadow latches is disabled.
- *dftAutoFix*: The default value of the parameter is as follows:

```
" {+RULES[nothing]+TMPort[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}"
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
" {+RULES[<rulelist>]+TMPort[tmport]+TCLKPORT[tclkport]}"
```
- *dft_ignore_no_scan_latches_in_rule_checking*: The default value is off. Set the value of the parameter to on to ignore forced no_scan latches from rule checking.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *dont_touch* (optional): Use this constraint to specify the modules/nets that are not considered for AutoFix.

Operating Mode

Capture, *Scanshift* when the *dft_detect_shadow_latches* parameter is on.

Messages and Suggested Fix

Message 1

[INFO] Latch enable source <source_name> is controllable-transparent in <mode> (drives <number1> Latch(es) out of which <number2> Latch(es) are controllable transparent)

Arguments

- Source of the Latch enable term, <source_name>
- Mode for which the analysis is done, <mode>
- Total no of latches the latch enable source is driving, <number1>
- Number of latches which are controllable-transparent, <number2>

Potential Issues

A violation is reported when latch enable is controllable to active state but is currently not active.

Consequences of Not Fixing

If the violation is not fixed, ATPG will have to make latch transparent through vector shift-in

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Latch enable source <source_name> is non-transparent in <mode> (drives <number1> Latch(es) out of which <number2> Latch(es) are non-transparent)

Arguments

- Source of the Latch enable term, <source_name>
- Mode for which the analysis is done, <mode>
- Total no of latches the latch enable source is driving, <number1>
- Number of latches which are non-transparent, <number2>

Potential Issues

A violation is reported due to a non-transparent latch in the design.

Consequences of Not Fixing

Not fixing the violations results in reduced coverage due to bad controllability and observability around the latch.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

```
[WARNING] Latch enable source <source_name> is driving latch enables on both levels in <mode> (drives <number1> Latch(es) out of which <number2> Latch(es) are non-transparent/controllable transparent)
```

Arguments

- Source of the Latch enable term, <source_name>
- Mode for which the analysis is done, <mode>
- Total no of latches the latch enable source is driving <number1>
- Number of latches which are non-transparent/ controllable transparent, <number2>

Potential Issues

A violation is reported when the latch enable source is driving on both the levels.

Consequences of Not Fixing

Since latch enable source is driving latches on both the levels, none of them can be made transparent simultaneously.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

```
[WARNING] <'module_name'> has <'number'> non transparent latch(es) with floating enable pin
```

Arguments

- Name of the Top module, <module_name>
- Number of non transparent latches, <number>

Potential Issues

A violation is reported when the latch enable pin is floating.

Consequences of Not Fixing

Not fixing the violation results in reduced coverage due to bad controllability and observability around the latch.

How to Debug and Fix

Double-click on the message to open the spreadsheet which displays all the latches along with its types.

To view the schematic, double-click on a latch in the spreadsheet and then open incremental schematic.

Message 5

[INFO] Suggested latch enable pins for autofix

Potential Issues

The violation message identifies places in the RTL that gets modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic**Example 1**

Consider the following incremental schematic where the latch enable source, top.ff1, is controllable-transparent:

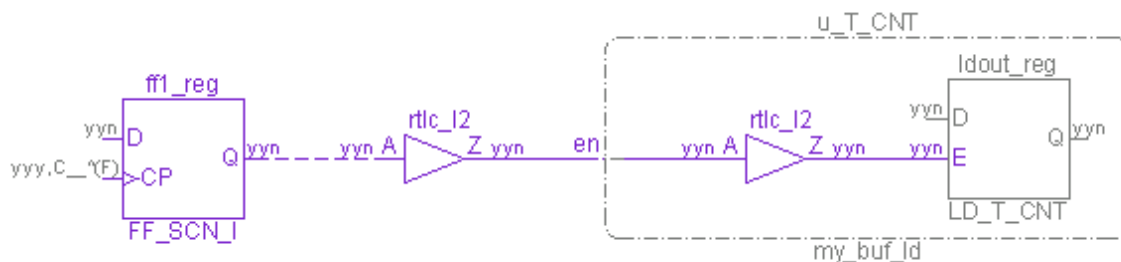


FIGURE 81. Controllable-Transparent Latches

For the above design, the Latch_08 rule reports the following violation message:

Latch enable source 'top.ff1' is controllable-transparent in Capture mode (drives 1 Latch(es) out of which 1 Latch(es) are controllable Transparent), test.v, 100

The above violation message is reported when latch enable is controllable to active state but is currently not active.

Example 2

Consider the following incremental schematic where the latch enable source, top.en_T_uncnt, is driving non-transparent latches:

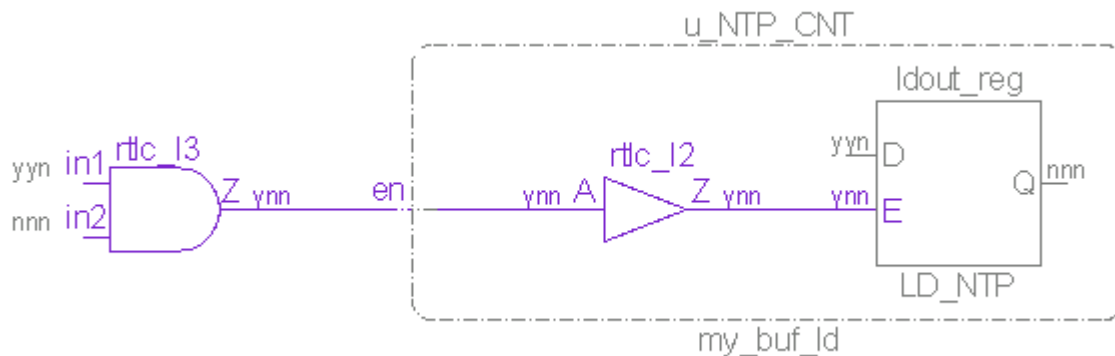


FIGURE 82. Non-Transparent Latches

Latch Rules

For the above design, the Latch_08 rule reports the following violation message:

Latch enable source 'top.en_T_uncnt' is driving non transparent latches in Capture mode (drives 1 Latch(es) out of which 1 Latch(es) are non-transparent), test.v, 106

The above violation message is reported due to the presence of a non-transparent latch in the design.

Example 3

Consider the following incremental schematic where the clock enable source, top.clk, is driving latch enables on both the levels:

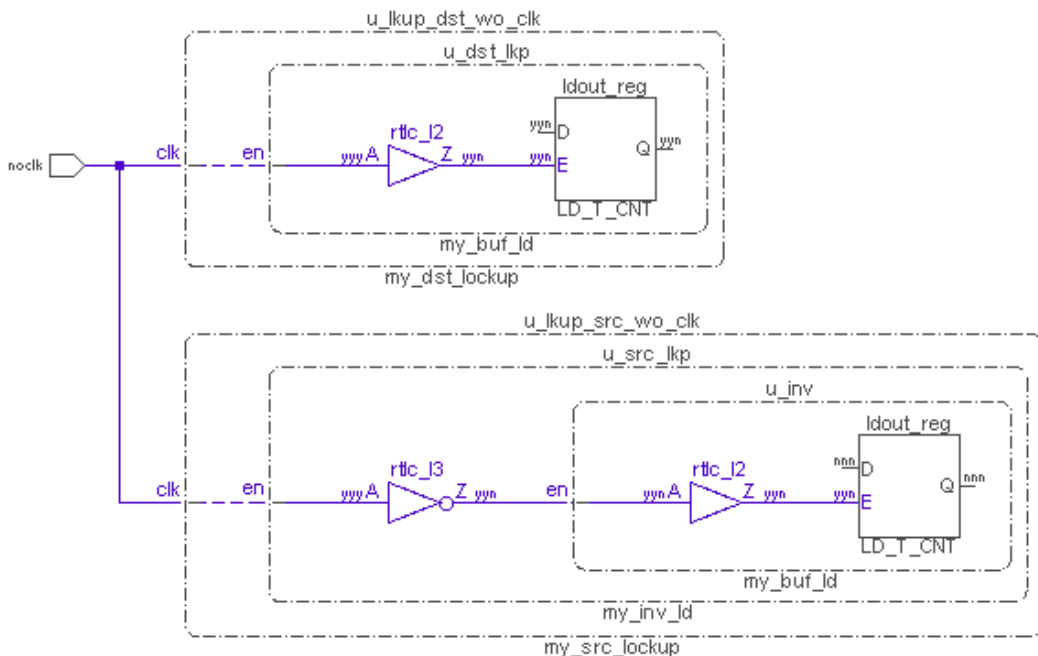


FIGURE 83. Non-Transparent/Controllable-Transparent Latches

For the above design, the Latch_08 rule reports the following violation message:

Latch enable source 'top.clk' is driving latch enables on both

Levels in Capture mode (drives 5 Latch(es) out of which 2 Latch(es) are non-transparent/control lable transparent), test.v, 53

The above violation message is reported because the latch enable source is driving on both the levels.

Default Severity Label

Warning/Info

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_10

Ensure that all latch enables must only be combinationally driven through primary ports.

When to Use

Use this rule to identify latches for which enables are not exclusively driven by primary port via combinational logic.

Description

The Latch_10 rule reports violation for latches that are not exclusively driven by primary port via combinational logic.

The Latch_10 rule reports a violation when one or more of the following are found as a driver:

- Constant value net
- Undriven net
- Flip-flop
- Non transparent latch
- Black box

Default Weight

10

Language

Verilog, VHDL

Method

Simulate capture condition

Walk the unblocked fan-in cone for every non-transparent latch enable

Stop the walk at the following and report violation

- Node with non-x simulation value
- A non-transparent latch
- A flip-flop
- A black box
- Undriven net

If the latch does not qualify for either a source or destination retiming latch, report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Latch_10 rule:

[WARNING] Enable pin of Latch '<sig1-name>' is driven by flop/latch '<sig2-name>'

Arguments

- Name of the latch. (<sig1-name>)
- Name of the flip-flop/latch that is driving 'E' pin. (<sig2-name>)

Potential Issues

A violation is reported due to incorrect / incomplete test_mode or wrong design connections.

Consequences of Not Fixing

Generally, latches are forced transparent in shift mode but this can impact testability for the enable logic as controlling them during test may be difficult

How to Debug and Fix

View the Incremental schematic for the violation message. The Incremental Schematic displays the latch and path from its enable pin to the instance, which can drive it combinationally.

Overlay Info_testmode (in auxiliary mode) in shift mode to check how the

Latch Rules


path is sensitized.

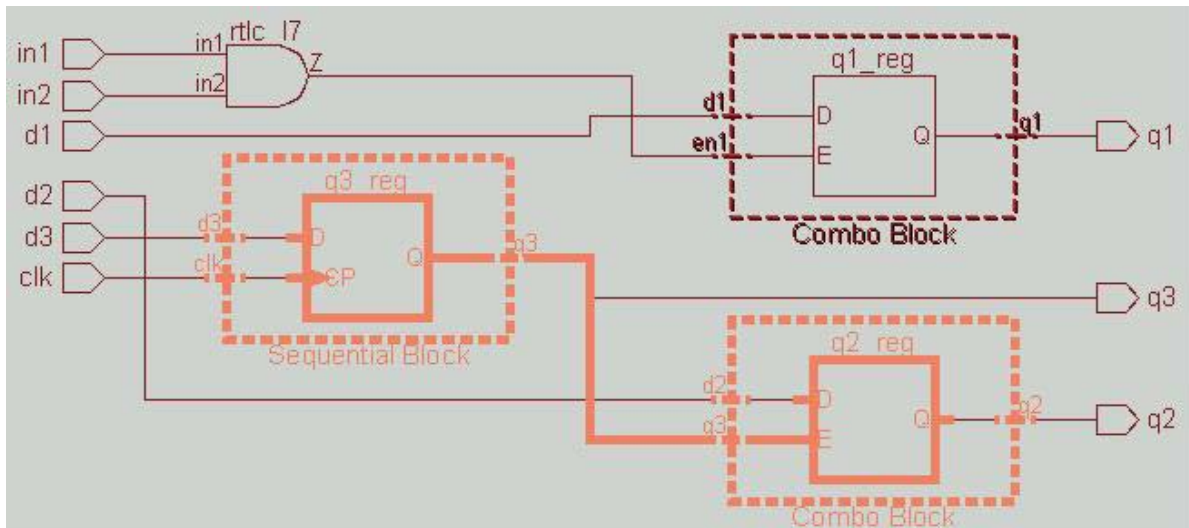
To fix the violation, use the root-level pins to control the enables. This allows the latch enables to switch in and out of transparency independent of shift mode. Use the [Info_testmode](#) rule to display how shift mode signals are distributed in the design.

Example Code and/or Schematic

Consider the following example:

 Latch_10[1]: All latch enables, except for retiming latches, must be combinationaly derived from root level inputs.

 Enable pin of latch 'test.q2' is driven by flop 'test.q3', test.v, 19



Default Severity Label

Warning

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_15

Ensure that all slave LSSD latches are fed directly by only one master LSSD Latch.

When to Use

Use this rule to check for slave LSSD latches in the design.

Description

The Latch_15 rule reports slave LSSD latches that are directly controlled by more than one master LSSD latch.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to declare the clock pins used as test clocks. Use this constraint's `-testclock` argument for the Latch_15 rule.
- *scan_type* (option): Use this constraint to specify the SpyGlass DFT ADV type, LSSD or MUXSCAN. Use this constraint's `-lssd` argument for the Latch_15 rule.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Data pin of Slave LSSD Latch '<latch-name>' is not fed directly by master

Where, <latch-name> refers to the name of the slave LSSD latch output.

Potential Issues

The violation message appears if a slave LSSD latch is not driven directly by a master.

Consequences of Not Fixing

Not fixing the violation may result in bad testability during capture, which reduces fault coverage.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the slave LSSD Latch that is not fed directly by a master LSSD latch.

To fix the violation, ensure that only one master drives the slave latch.

Message 2

[WARNING] Slave LSSD Latch '<latch-name>' is being fed by net with more than one fan in

Where, <latch-name> refers to the name of the slave LSSD latch output.

Potential Issues

The violation message appears if a slave latch is driven by multiple masters.

Consequences of Not Fixing

Not fixing the violation may result in bad testability during capture, which reduces fault coverage.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the slave LSSD Latch that is not fed directly by a single master LSSD latch.

Latch Rules

To fix the violation, ensure that only one master drives the slave latch.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Latch

Reports and Related Files

No related reports or files.

Latch_16

Ensure that all latches except retiming ones are transparent in Logic DBIST mode.

When to Use

Use this rule to identify the latches, except the retiming latches, that are non-transparent in the Logic DBIST mode.

Description

The Latch_16 rule reports a violation for a latch that is not retiming and non-transparent in Logic DBIST mode.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's -testclock argument for this rule.
- *dbist* (mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit into a state for DBIST mode.

Operating Mode

Dbist

Messages and Suggested Fix

Message 1

[WARNING] Latch '<latch-name>' is not transparent in <mode>

Arguments

- Name of the Slave LSSD latch output, <latch-name>

- Logic DBIST mode, <mode>

Potential Issues

A violation is reported due to a non-transparent latch in the design.

Consequences of Not Fixing

Not fixing the violation results in reduced coverage due to bad controllability and observability around latch.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the latch, which is not retiming and non-transparent in Logic DBIST mode.

To fix the violation, make the latch transparent by adding appropriate test_mode constraints or changing the en signal driver.

Message 2

[INFO] DBIST Mode (specified using 'dbist' sgdc command) has not been specified for design '<du-name>', hence skipping rule checking

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message is reported when rule is run without the dbist constraint. Ignore this message in case no dbist constraint is required for design.

Consequences of not fixing

Rule checking is skipped in absence of the dbist constraint.

How to Debug and Fix

To fix the violation, add necessary dbist constraints, if present.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Latch Rules

Reports and Related Files

No related reports or files.

Latch_18

All latch inputs should meet scannability criterion in scan-shift and should be driven by controllable sources in the capture mode

When to Use

Use this rule when you do not want to treat the latches in the design as X-source for APTG pattern generation, which otherwise may increase the pattern count and aborted faults.

Description

The Latch_18 rule reports violation when any of the following conditions are true:

- A set/reset pin is:
 - not tied to inactive value during the shift mode
 - not controllable during capture mode
- Enable pin is not driven by any test clock
- Data pin is not driven by controllable sources

A latch data pin is driven by a controllable source, if the sensitized path to the following are found:

- scannable flip-flop
- controllable primary input
- scan wrapped black box

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's -testclock argument for this rule.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

[WARNING] Latch '<name>' will be treated as x-source by ATPG (<reason>) in <mode >

Arguments

- Name of the latch, <name>
- Reason for treating the latch as x-source, <reason>
- Operating mode name, <mode>

Potential Issues

If latch is not driven by a controllable source (scan flip-flop or scannable black box or controllable primary port) and other inputs do not meet capture criterion in shift and are controllable in capture then ATPG may treat it as X source.

Consequences of Not Fixing

Number of patterns required by ATPG will be higher and can result in aborted faults.

How to Debug and Fix

Double-click the violation message to view the Incremental Schematic displaying the violating latch. Analyze the latch accordingly.

Example Code and/or Schematic

Consider the following violation message generated by the Latch_18 rule:

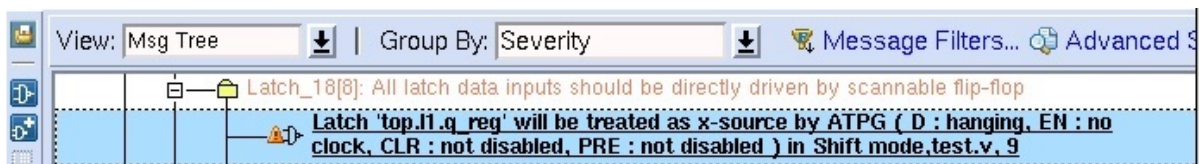


FIGURE 84. Latch_18 violation message

The above message reports that the latch, top.l1.q_reg, is treated as x-source by ATPG because of the following reasons:

Latch Rules

- D-pin is hanging
- No clock is specified in the Enable pin
- Clear (CLR) is not disabled
- Preset (PRESET) is not disabled

Double-clicking the message displays the following incremental schematic:

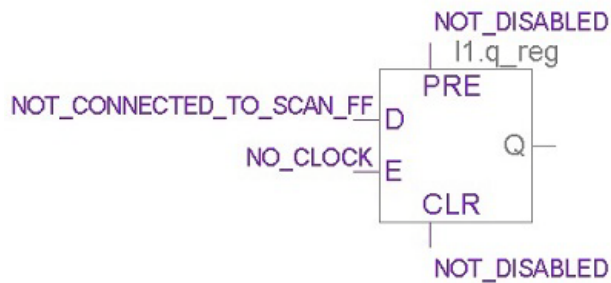


FIGURE 85. Latch_18 Incremental Schematic

Default Severity Label

Warning

Rule Group

Latch Rules

Reports and Related Files

No related reports or files.

Latch_19

Ensure that all non-transparent and non-shadow latches meet the scannability criterion

When to Use

Use this rule to check for the scannability criterion on latches which are neither transparent nor shadow.

Rule Description

The Latch_19 rule identifies all the clock and asynchronous sources of latches, which are neither transparent nor inferred as shadow. The rule reports a violation when one or more of the following conditions are true:

- Set / reset is not tied to inactive value during the scan shift mode
- Enable pin is not driven by any test clock either during scan shift or during capture mode

Pre-requisites

Ensure that the value of the [dft_scannable_latches](#) parameter is on, before running this rule.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- [test_mode](#) (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (optional): Defines the clocks of a design. Use the constraint's -testclock argument for this rule.
- [force_scan](#) (optional): Use this constraint to declare flip-flops as scannable even if they do not so qualify.
- [force_no_scan](#) (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] Latch-Enable source '`<source_name>`', which impacts '`<count>`' latches, does not get a test clock in '`<mode_name>`'.
Cause: `<cause-string>`, SPREADSHEET_PATH: `<path>`

Arguments

- Name of the latch-enable source, `<source_name>`
- Number of latch enable pin, `<count>`
- Shift or Capture mode, `<mode_name>`
- Causes, `<cause-string>`, which can be any of the following:
 - Tied to 0
 - Tied to 1
 - Constraint 'clock -testclock' is missing on the port
 - Test clock '`<Name of the user specified test clock>`' is blocked
 - No testclock is found in the combinational topological fanin cone
- CSV file path which contains the list of impacted latch enable pins, `<path>`

Potential Issues

A violation message is reported when the pin of a latch is unable to get a test clock. That is, either a test clock is missing in the fanin cone or the test clock is blocked.

Consequences of Not Fixing

If you do not fix the violation message, latches will not meet scannability criteria and will not be replaced with scan cell leading to loss of coverage.

How to Debug and Fix

To fix the violation, check the schematic and do RTL / SGDC modification.

Message 2

[WARNING] Latch-`<Clear | Preset>` source '`<source_name>`', which impacts '`<count>`' latches, is not inactive in 'Shift mode'.
Cause: `<cause-string>`, SPREADSHEET_PATH: '`<path>`'

Arguments

- Name of the latch-async name <source_name>
- Number of latch async pin, <count>
- Causes, <cause-string>, which can be any of the following:
 - Tied to 0
 - Tied to 1
 - Constraint 'test_mode' is missing on the port
 - Unconstrained
- CSV file path which contains the list of impacted latch async pins, <path>

Potential Issues

A violation message is reported when the asynchronous pin of latch is not disabled in scan shift mode. The asynchronous pin is either active or unconstrained.

Consequences of Not Fixing

If you do not fix the violation message, latches will not meet scannability criteria and will not be replaced with scan cell leading to loss of coverage.

How to Debug and Fix

To fix the violation, check the schematic and do RTL / SGDC modification.

Message 3

[INFO] Rule checking is skipped because 'dft_scannable_latches' is 'off'. Set it to 'on' for enabling scannability checks on non-transparent and non-shadow latches

Potential Issues

A violation message is reported when the value of the [dft_scannable_latches](#) parameter is set to off.

Consequences of Not Fixing

If you do not fix the violation, scannability check will not run.

How to Debug and Fix

Set the value of the [dft_scannable_latches](#) parameter to on.

Latch Rules

Example Code and/or Schematic

Not Available

Default Severity Label

Warning

Rule Group

Latch Rules

Reports and Related Files

No reports and related files generated.

PLL Rules

Overview

The PLL rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Description
<i>PLL_01</i>	Clock ports of PLL that are not directly controlled by an external signal
<i>PLL_02</i>	Reset pins of PLL that are not directly controlled by an external signal
<i>PLL_03</i>	PLL reference clocks should not directly reach to any flip-flop

PLL_01

Ensure that the reference clock of PLL is directly controlled by external signal

When to Use

Use this rule to detect PLL clock in ports that are not directly controlled by an external signal.

Description

The PLL_01 rule reports a violation for PLL clk_{in} ports that are not directly controlled by an external signal.

The rule performs following checks on each PLL:

- If clk_{in} pin is not controlled by an external signal during `-atspeed`, then rule reports [Message 1](#).
- If clk_{in} pin is not controlled by an external signal during `-capture`, then rule reports [Message 2](#).
- If clk_{in} pin is not controlled by an external signal during `-scanshift`, then the rule reports [Message 3](#).
- If clk_{in} pin is not controlled by an external signal during any of `-scanshift`, `-capture`, or `-atspeed`, then the rule reports [Message 4](#).

Prerequisites

Clock_shaper is treated as a PLL if `-pll` option is specified in its registration and all the PLL checks are applied on it.

Rule Exceptions

Even though clock `not_a_clk` is not defined as an `atspeed` testclock specifically, merely declaring it as a clock assumes the same. Therefore, the constraint options `"-testclock"` and `"-atspeed"` are not strictly required for the purpose of this rule.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate all clock in scanshift mode

If a pll clkpin does not get clock report a violation

Simulate all clock in capture mode

If a pll clkpin does not get clock report a violation

Simulate all clock in atspeed mode

If a pll clkpin does not get clock report a violation

Simulate all clock in any of scanshift, capture, or atspeed mode

If a pll clkpin does not get clock report a violation

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (optional): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, Capture, Capture (atspeed)

Messages and Suggested Fix

Message 1

[WARNING] Clkin Pin '<pin-name>' on PLL '<inst-name>' is not directly controlled during atspeed

Potential Issues

To know more about potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Clkin Pin '<pin-name>' on PLL '<inst-name>' is not directly controlled during capture

Potential Issues

To know more about potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Clkin Pin '<pin-name>' on PLL '<inst-name>' is not directly controlled during scanshift

Potential Issues

To know more about potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] Clock Pin '<pin-name>' on PLL '<inst-name>' is not directly controlled in any of, scanshift, capture, or atspeed mode

Potential Issues

A violation is reported when the PII and / or test_mode constraints are not defined properly or topological connections to pll clock are incorrect

Consequences of Not Fixing

Not fixing the violation may result in generation of capture pulse creation as it becomes difficult during atspeed testing.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the PLL whose clock pin is not controlled from primary clock port. The PLL_01 rule reports violation for all three modes, that is, scanshift, capture, and atspeed.

Probe the fan-in cone of PLL clockpin towards the primary inputs overlaying Info_testmode (Auxiliary violation mode) under the mode mentioned in the violation to see why the path is not getting sensitized to primary inputs.

To fix the violation, see example code and/or Schematic section.

Arguments

1. Name of the clock pin <pin-name>
2. Name of the PLL instance <inst-name>

Example Code and/or Schematic

Example 1

Consider the following figure:

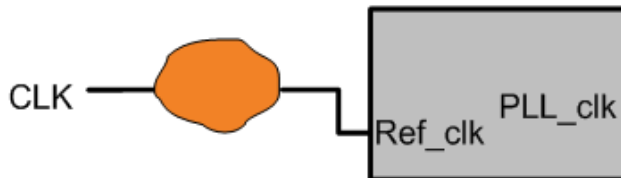


FIGURE 86. Example of Bad Ref Clock Design

The above figure illustrates a bad design example for a reference clock as the

Also, consider the following figure:

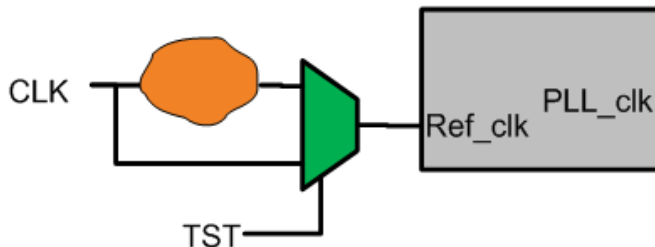


FIGURE 87. Example of bad Ref Clock Design

The above figure illustrates a bad design example for a reference clock.

Example 2

Consider the following example

```
// File: PLL.v
```

```
module PLL (clk_i, clk_o, clk_fb_i);
input clk_i, clk_fb_i;
output clk_o;
endmodule
```

```
// File: test.v
module PLL_01 (sel_to_pll, clk, not_a_clk, d, q);
input clk, not_a_clk, sel_to_pll, d;
output q;
reg q;
wire mux_clk;
assign mux_clk = (sel_to_pll) ? not_a_clk : clk;
PLL PLL_inst (.clk_i(mux_clk), .clk_o(wb_clk_i));
always @(posedge wb_clk_i)
    q <= d;
endmodule
```

```
// File: test.sgd
current_design PLL_01
clock -name "clk" -testclock -atspeed
pll -name PLL -clkin clk_i -clkout clk_o
testmode -name sel_to_pll -value 1
```

In the above example, the PLL_01 rule reports a violation since the clock signal does not reach the reference input pin of the PLL. Adding the following constraint to the constraints file test.sgd will eliminate the violation:

```
clock -name "not_a_clk"
```

Default Severity Label

Warning

Rule Group

PLL rules

Reports and Related Files

No related reports or files.

PLL_02

Ensure that the reset pin of a PLL is directly controlled by external signal

When to Use

Use this rule to identify reset pins of a PLL that are not directly controlled by an external signal.

Description

The PLL_02 rule reports violation for reset pins of a PLL that are not propagated by an external signal.

The rule checks for four cases of testmode simulation as described below:

The following checks are performed on each PLL reset pin:

- If this PLL pin is not controlled by an external signal during any of `-scanshift`, `-capture`, or `-atspeed`, then the following issue is reported:

Reset pin `<reset pin>` on PLL `<path to PLL>` is not directly controlled during any of `scanshift`, `capture`, or `atspeed` mode.

- If this PLL pin is not controlled by an external signal with `-scanshift`, then the following issue is reported:

Reset pin `<reset pin>` on PLL `<path to PLL>` is not directly controlled during `scanshift`

- If this PLL pin is not controlled by an external signal with `-capture`, then the following issue is reported:

Reset pin `<reset pin>` on PLL `<path to PLL>` is not directly controlled during `capture`

- If this PLL pin is not controlled by an external signal with `-atspeed` then the following issue is reported:

Reset pin `<reset pin>` on PLL `<path to PLL>` is not directly controlled during `atspeed`

Prerequisites

`Clock_shaper` is treated as a PLL if `-pll` option is specified in its registration and all the PLL checks are applied on it.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate a pulse on user specified reset signal in scanshift mode

If a pll reset pin does not get a pulse report a violation

Simulate a pulse on user specified reset signal in capture mode

If a pll reset pin does not get a pulse report a violation

Simulate a pulse on user specified reset signal in atspeed mode

If a pll reset pin does not get a pulse report a violation

Simulate a pulse on user specified reset signal in any of scanshift, capture, or atspeed mode

If a pll reset pin does not get a pulse report a violation

Parameter(s)*Common SpyGlass DFT ADV Rule Parameters***Constraint(s)**

- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.
- *clock* (optional): Use this constraint to declare clock pins declared as testclocks.
- *clock_shaper* (optional): Use this constraint to declare a clockshaper module to control clock pulse propagation in the design.
- *pll* (mandatory): Use this constraint to specify the PLL (Phase Lock Loops) modules.
- *reset* (mandatory): Use this constraint to specify the names of reset signals using the RESET keyword in a SpyGlass Design Constraints File.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, Capture, Capture (atspeed)

Messages and Suggested Fix

Message 1

[WARNING] Reset pin <reset pin> on PLL <path to PLL> is not directly controlled during any of scanshift, capture, or atspeed mode.

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Reset pin <reset pin> on PLL <path to PLL> is not directly controlled during scanshift.

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Reset pin <reset pin> on PLL <path to PLL> is not directly controlled during capture.

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] Reset pin <reset pin> on PLL <path to PLL> is not directly controlled during atspeed.

Potential Issues

A violation is reported when either reset constraint or test_mode is not specified properly. You may also need to modify topological connection for bypassing logic if present.

Consequences of Not Fixing

If the reset pin of a PLL is driven by a sequential logic, a feedback loop is created. The PLL creates the clocks that drive the sequential logic, which in turn drive the reset pin. Making the reset pin primary input (PI) controlled means that the reset pin is guaranteed to be controllable regardless of any timing faults.

If PLL reset is not controlled from the root level pin, it will be difficult in controlling the clock propagation through it, which will affect the coverage.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the PLL whose reset pin is not controlled from primary reset port. The PLL_02 rule reports violation for all three modes, that is, scanshift, capture, and atspeed.

Probe the fan-in cone of PLL reset towards the primary inputs overlaying Info_testmode (Auxiliary violation mode) under the mode mentioned in the violation to see why the path is not getting sensitized to primary inputs.

To fix the violation, fix pll / test_mode constraint or circuit connectivity to pll reset.

Arguments

1. Name of the reset pin of the PLL <reset pin>
2. Path to the PLL <path to PLL>

Example Code and/or Schematic**Example 1**

Consider the following figure:

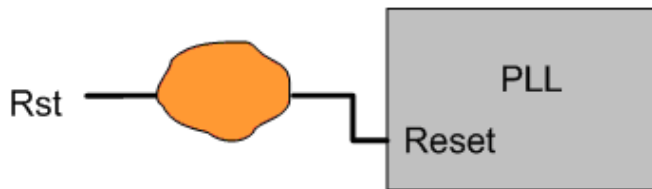


FIGURE 88. Example of Bad Reset Design

The above figure illustrates the example of a bad reset design. Also, consider the following figure:

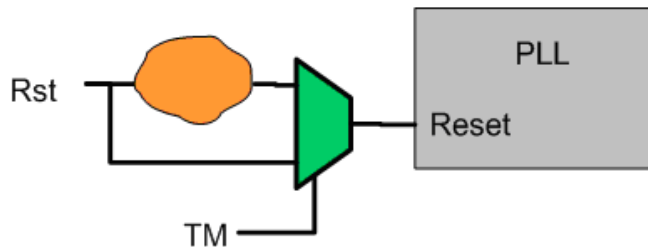


FIGURE 89. Example of Good Reset Design

The above figure illustrates the example of a good reset design.

Example 2

Consider the following example:

```
// File: pll_test.v
module pll_test (rootclk, tm, tm_r, rst, d1, out1);
```

```
input rootclk, tm, d1;
input tm_r;
input rst;
output out1;
wire c1, c2;
wire andout;
wire rootclk_and = rootclk & tm;
wire tm_r_and = tm_r & rst;
// - root cause of PLL_02 violation
myPLL u0 (.ref_clk(rootclk_and), .reset(tm_r_and),
        .clk1(c1), .clk2(c2));
FF u1 (.clk(c1), .data(d1), .out(out1));
endmodule

module myPLL (ref_clk, reset, clk1, clk2);
input ref_clk;
input reset;
output clk1, clk2;
endmodule

module FF (clk, data, out);
input clk, data;
output out;
reg out;
always @ (posedge clk) out = data;
endmodule

// pll_test.sgd
```

```
current_design pll_test
    pll -name myPLL -clkin ref_clk -clkout clk1 clk2 -reset
reset -value 0
    reset -name rst -value 0
    clock -name rootclk -testclock -atspeed
    testmode -name tm -value 1
    testmode -name tm_r -value 0
```

The constraints (see the constraint on `tm_r`) defined in this example result in a PLL_02 violation because the reset pin of the PLL cannot be controlled. If the AND gate in the design is replaced with an OR gate then the violation is removed.

```
wire tm_r_and = tm_r | rst; // Correction for PLL_02
```

Default Severity Label

Warning

Rule Group

PLL rules

Reports and Related Files

No related reports or files.

PLL_03

PLL reference clocks should not directly reach to any flip-flop.

Rule Description

The PLL_03 rule reports a violation, if a pll reference clock is driving any flip-flop.

NOTE: *Clock_shaper is treated as a PLL if -pll option is specified in its registration and all the PLL checks are applied on it.*

Constraints

- *clock* (Mandatory): Use this constraint to declare clock pins declared as testclocks. The PLL_03 rule uses -pll_reference option of the clock constraint.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Rule Parameters

Common SpyGlass DFT ADV Rule Parameters

Operating Mode

Power Ground

Message Details

Message 1

PLL reference clock <clock_name>, is directly reaching to <flip-flop-count> flip-flops. [example: <flip-flop-name>]

Message 2

PLL reference clock <clock_name>, is directly reaching to 1 flip-flop <flip-flop-name>

Arguments

1. Name of the pll reference clock, <clock_name>
2. Number of clocks which are clocked by the pll_reference, <flip-flop-count>

3. Name of the flip-flop, <flip-flop-name>

Location

The file and the line where first flip-flop hit is instantiated.

Schematic Highlight

Complete path from the pll reference clock to the flip-flop

Rule Severity

Warning

RAM Rules

Overview

The RAM rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Flags...
RAM_01	Memory instances that have more shadow logic than the specified limit
RAM_02	Memory outputs that are not registered to scannable flip-flops
RAM_03	Memory inputs that are not scannable
RAM_04	Memory outputs that are not forced to known values in the shift mode
RAM_05	Memory write enables
RAM_06	Top level pins must drive memory read/write pins
RAM_07	Memory outputs must not affect scannable flip-flops during capture
RAM_08	Write controls to register files/memories must be disabled in shift mode and ATPG controlled during capture mode
RAM_09	Memory inputs must feed scannable flip-flops during capture
RAM_10	Memory or black box input and output terminals that are not scanwrapped or bypassed.
RAM_11	Memory instances with combinational loops.

RAM_01

Memory shadow logic must be less than a user specified %

When to Use

Use this rule to identify memory instances that have more shadow logic than the specified limit.

Description

The RAM_01 rule reports violation for memory instances that have more shadow logic than the specified limit.

Prerequisite

Ensure that the memory design units specified with the *memory_type* constraints are black boxes, that is, their definitions do not exist in the design or in the specified libraries, if any.

Method

Simulate power, ground and capture mode conditions.

Count the number of nodes in the unblocked combinational fan-in/fan-out cones for all memory input/output pins. Ignore nodes with non-x simulation values.

Report a message if the ratio of shadow gates to total gate count is not less than the user specified threshold (via using *shadow_ratio*)

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *scan_wrap* (optional): Specifies black box design units or instances that are designed with scan wrappers.
- *shadow_ratio* (mandatory): Specifies the fraction of the total logic that is allowed to be shadowed by memories.

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Top module '<du-name>' has shadow ratio (<num>). Specified value is (\$shadowratio)

Arguments

- Name of the offending top-level module, <du-name>

- Actual Shadow Ratio of the design, <num>
- Constraint specified Shadow Ratio, \$shadowratio

Potential Issues

A violation is reported when the number of shadowed gates exceeds a user specified threshold.

Consequences of Not Fixing

Gates in the input or output shadow of RAMs become untestable for ATPG tools that expect combinational logic.

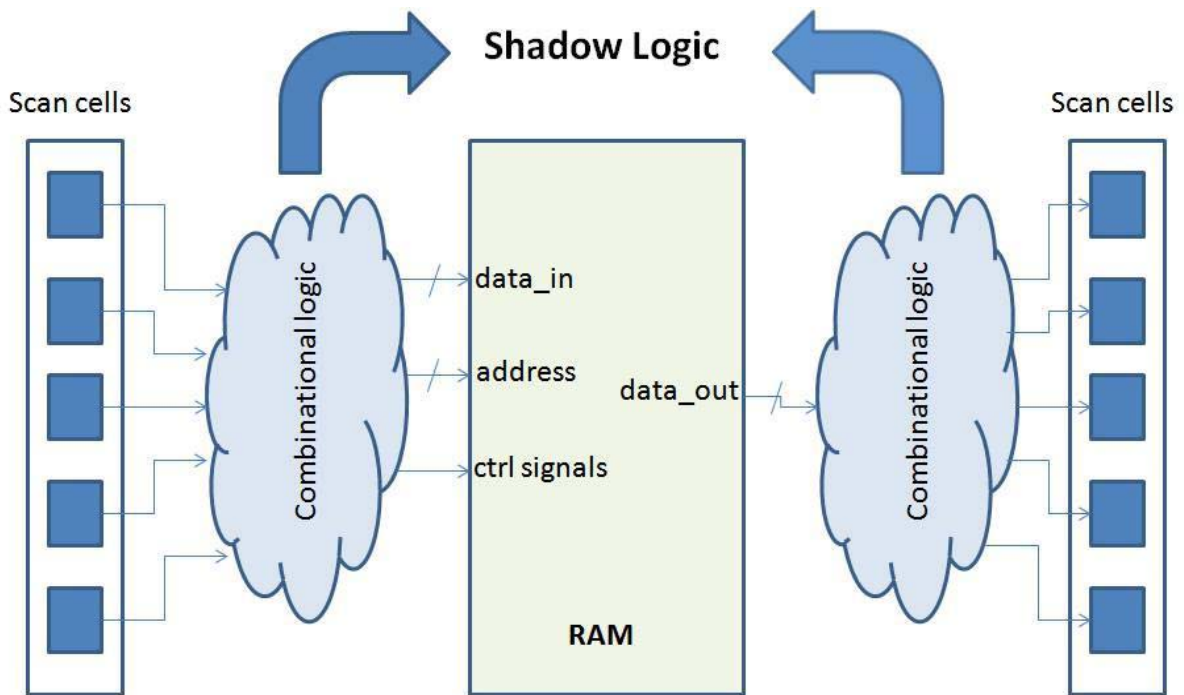
How to Debug and Fix

View the Incremental Schematic of the violation. The Schematic Viewer window highlights the memory instance, which is a black box instance.

To fix the violation, reduce the logic in the shadow of a memory.

Example Code and/or Schematic

Consider the following figure:



The RAM_01 rule reports violation for the above example because the memory instance has more shadow logic than the specified limit.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No related reports or files.

RAM_02

Ensure that the memory outputs are registered to scannable flip-flops

When to Use

Use this rule to simplify the testing of the RAM instance.

Description

The RAM_02 rule reports violation for memory outputs that are not directly registered to scannable flip-flops.

Compliance with this rule simplifies testing of the RAM itself. See [RAM_03](#) rule for compliance with input requirements.

Prerequisites

The RAM_02 rule requires that the memory design units specified with the *memory_type* constraints must be black boxes that is, their definitions must not exist in the design or in the specified libraries, if any.

System logic controlled or fed by RAM outputs and prior to registering is difficult to test due to poor controllability. Adding scan to memory outputs improves coverage and may be required for memory BIST.

Rule Exceptions

The RAM_02 rule will not flag for scanwrapped and/or module_bypassed memories as memory transparency is obtained through module_bypass and in case of scanwrap transparency is not needed.

Method

Simulate power, ground and any available testmode conditions. If the fan-out cone for any memory output instance, skipping only single-input devices and transparent latches. If anything other than a scannable flip-flop is hit, report a message.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.
- *scan_wrap* (optional): Specifies black box design units or instances that are designed with scan wrappers.
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the `dft_set_scan_wrap_on_memory` parameter to off.

Message 2

[WARNING] Memory output pin '<pin-name>' is not registered to scan flip-flop

Potential Issues

The violation message appears when memory path is blocked between scan flip-flops and memory output.

Consequences of Not Fixing

Not fixing the violation may result in loose control of the logic driven by RAM instances leading to reduced coverage.

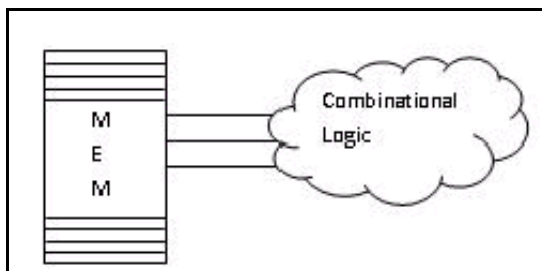
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the memory instance (black box instance) and its un-registered output pins.

To fix the violation, ensure that the schematic and the constraint specifications are correct.

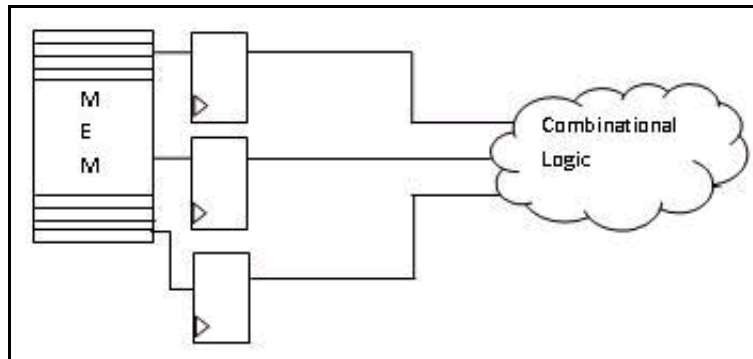
Example Code and/or Schematic

Consider the following figure:



The above figure illustrates a bad design scenario because memory output is not directly connected to a scan flip-flop.

Now, consider the following figure:



The above figure illustrates a converted good design scenario because memory outputs are directly registered to scannable flip-flops.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No related reports and files.

RAM_03

Memory inputs should be scannable

When to Use

Use this rule to identify memory inputs that are not scannable.

Description

The RAM_03 rule reports violation for memory inputs that are not scannable.

Prerequisites

Ensure that the memory design units specified using the *memory_type* constraint are black boxes, that is, their definitions do not exist in the design or in the specified libraries, if any.

Rule Exceptions

The RAM_03 rule does not report violation for scanwrapped and/or module_bypassed memories as memory transparency is obtained through module_bypass and in case of scanwrap transparency is not needed.

Also, the rule ignores clock-pin from the rule checking.

Method

Simulate power, ground and any capture conditions. The unblocked fan-in cones for all inputs on all memory instances, must only contain single-input combinational devices. If any device other than a scannable flip-flop is hit, report a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *scan_wrap* (optional): Specifies black box design units or instances that are designed with scan wrappers.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Memory input pin '<pin-name>' is not controlled by scan flip-flop

Arguments

Name of the offending pin, <pin-name>

Potential Issues

A violation is reported when combinational logic is feeding D-pin of memory.

Consequences of Not Fixing

Compliance with this rule simplifies testing of the RAM itself. See [RAM_02](#) rule for compliance with output requirements.

Memories are often made transparent in order to provide a means to observe the logic feeding the memory inputs. In cases where transparency is not an acceptable option, the memory inputs should be scannable to provide observation.

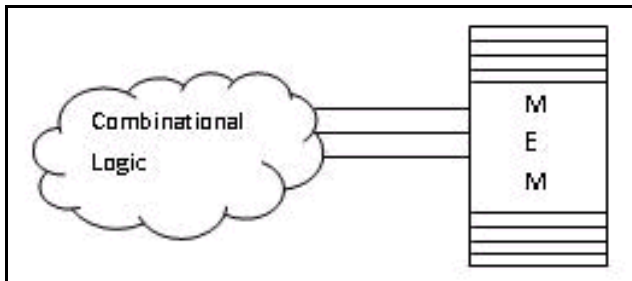
How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the memory instance (black box instance) and its non-scannable input pins.

To fix the violation, put scanwrapper on input side.

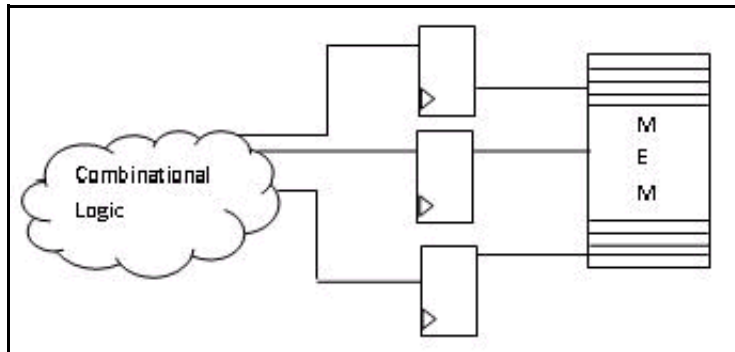
Example Code and/or Schematic

Consider the following figure:



The above figure illustrates a bad design scenario because it contains memory inputs that are not scannable.

Now, consider the following figure:



The above figure illustrates a good design scenario because it contains scannable memory inputs.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No related reports or files.

RAM_04

Memory outputs must be forced to known values in shift mode

When to Use

Use this rule to identify memory outputs that are not forced to known values in the shift mode.

Rule Description

The RAM_04 rule reports violation for memory outputs that are not forced to known values in the shift mode.

Prerequisites

Ensure that the memory design units specified using the [memory_type](#) constraint are, black boxes that is, their definitions do not exist in the design or in the specified libraries, if any.

Method

Simulate power, ground, capture and memoryforce conditions. A message is generated if any memory output remains x.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- [dft_set_scan_wrap_on_memory](#): The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_force* (mandatory): Specifies the memory design unit (black box) names.
- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *module_bypass* (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *scan_wrap* (optional): Specifies black box design units or instances that are designed with scan wrappers.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Memory output pin '<pin-name>' is not forced to a known value

Arguments

Name of the memory output pin that is not forced, <pin-name>

Potential Issues

A violation is reported when memory output is not forced to a known state during shift.

Consequences of Not Fixing

Memories are often made transparent in order to provide a means to control the logic fed by the memory. In cases where transparency is not an acceptable option, the memory outputs may be forced to particular values to prevent non-controllable values on memory from excessively reducing ATPG performance.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights memory instance (black box instance) and its output pins that are not forced to a known value.

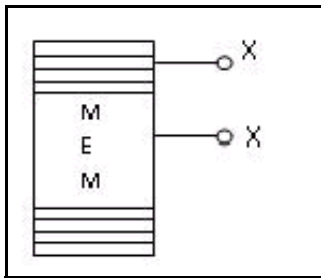
You can also view the violation for the Info_memoryforce rule along with the violation of the RAM_04 rule in the Incremental Schematic window. To do this, double-click the violation for the RAM_04 rule and open the Incremental Schematic window.

The violation message for the Info_memoryforce rule overlaps the violation message for the RAM_04 rule in the Incremental Schematic window. This is useful in debugging the violation for the RAM_04 rule.

To fix the violation, force the memory output to a known value.

Example Code and/or Schematic

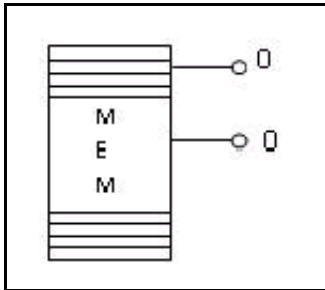
Consider the following figure:



The above figure illustrates a bad design scenario because the memory

outputs are not forced to a known value.

Now, consider the following figure:



The above figure illustrates a good design scenario because memory outputs are forced to a known value.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No related reports or files.

RAM_05

Disable RAM write enables.

When to Use

Use this rule to avoid memory write during testing.

Description

The RAM_05 reports violation for memory write enables.

Prerequisites

The RAM_05 rule requires that the memory design units specified with the *memory_type* constraints must be black boxes that is, their definitions must not exist in the design or in the specified libraries, if any.

Method

Simulate power, ground, scan conditions and memorywritedisable conditions. Report any memory with a write pin not in the inactive state.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraints

- *memory_type* (optional): Specifies the memory design unit (black box) names.

- *memory_write_pin* (mandatory): Use this to specify the write pin port name on a memory and the inactive value on that port.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that are designed with scan wrappers.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Memory write enable pin '<pin-name>' is not inactive in shift mode

Potential Issues

The violation message appears, if proper constraint is not specified and all the memory write enable pins are not disabled.

Consequences of Not Fixing

If you do not fix this violation, active write enable pins may corrupt the

RAM instances. Memory writes must be disabled to prevent scan shift operations from corrupting RAM contents.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the memory instance (black box instance) and its write-enable (user-specified) pin that is not inactive in the shift mode.

To fix the violation, ensure that the RAM write enable pin is disabled.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
...
always @(data or addr or we)
  if ( we )
    ram[addr] = data;
...

```

The above example demonstrates a bad design scenario because of the presence of memory write enables in the design.

Example 2

Consider the following Verilog code:

```
...
always @(data or addr or we)
  if ( we && ~scan )
    ram[addr] = data;
...

```

The above example demonstrates a good design scenario.

Example 3

Consider the following VHDL code:

```
...
Process (data, addr, we)
Begin
  If (we) then
    Ram(addr) <= data;
  End If;
End Process;

```

```
    End if;  
End process;  
...
```

The above example demonstrates a bad design scenario because of the presence of memory write enables in the design.

Example 4

Consider the following VHDL code:

```
...  
Process (data, addr, we)  
Begin  
    If (we and (not scan)) then  
        Ram(addr) <= data;  
    End if;  
End process;  
...
```

The above example demonstrates a good design scenario.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No reports and related files.

RAM_06

Ensure that the Top level pins must drive memory read/write pins

When to Use

Use this rule to enforce total control on the READ/WRITE pins.

Description

The RAM_06 rule flags memory read or write pins that are not driven by primary inputs.

All memory read and write control pins must be connected to primary inputs during the shift mode.

Prerequisites

The RAM_06 rule requires that the memory design units specified with the *memory_type* constraints must be black boxes that is, their definitions must not exist in the design or in the specified libraries, if any.

Also, the RAM_06 rule requires you to specify at least one memory type and a read or write pin on that type.

Method

Simulate power and ground and capture conditions. For each memory instance, walk the unblocked combinational fan-in cone from all read or write pins. If the walk reaches anything other than a top-level port, report a message.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *noBlackBoxReporting*: The default value is off. Set the value of the rule parameter to on to suppress the rule message for the nets made uncontrollable or unobservable by black boxes.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Use this constraint to specify the memory design unit (black box) names.
- *memory_write_pin* (mandatory): Use this constraint to specify the write pin port name on a memory and the inactive value on that port.
- *memory_read_pin* (mandatory): Use this constraint to specify the read pin port name on a memory and the inactive value on that port.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that are designed with scan wrappers.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the `dft_set_scan_wrap_on_memory` parameter to off.

Message 2

[WARNING] READ/WRITE pin of memory instance '`<inst>`' is driven by flip-flop/black-box/hanging-net `<memory_pin_name>`

Arguments

- Name of the memory instance, `<inst-name>`
- Memory pin name being driven by the flip-flop, black box, or hanging net `<memory_pin_name>`

Potential Issues

The violation message appears when a flip-flop, or black box, or hanging net output pin drives the READ/WRITE pin of a memory instance, instead of a primary input pin.

Consequences of Not Fixing

If you do not fix this violation, you might lose control over the READ/WRITE pin.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path from instance driving the read/write pin of the memory instance (black box instance).

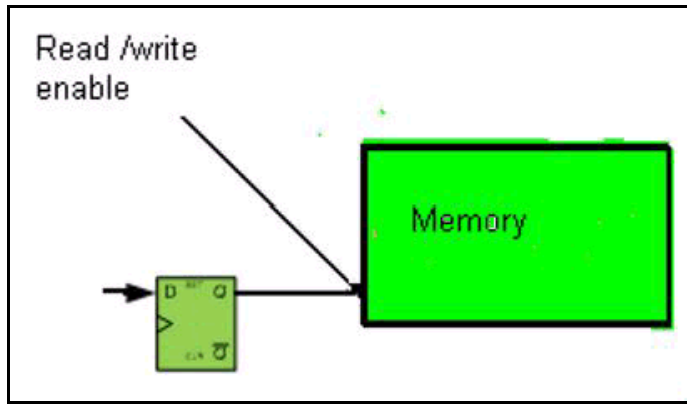
You can also view the violation for the Info_testmode rule along with the violation of the RAM_06 rule in the Incremental Schematic window. To do this, double-click the violation for the RAM_06 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the RAM_06 rule in the Incremental Schematic window. This is useful in debugging the violation for the RAM_06 rule.

To fix this violation, change the design or correct the constraints.

Example Code and/or Schematic

Consider the following example:



In the above figure a flip-flop drives the READ/ WRITE enable pin of a memory. The RAM_06 rule reports a violation because the READ/WRITE pin of the memory block is driven by output flip-flop, black box, or hanging net.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No reports and related files.

RAM_07

Ensure that the memory outputs must not affect scannable flip-flops during capture.

When to Use

Use this rule to detect memory instance or black box instance and its output pins that drive data pin of a scannable flip-flop.

Description

The RAM_07 rule reports violation for memory outputs that are not scannable.

Memories are often made transparent in order to provide a means to control the logic fed the memory outputs. In cases where transparency is not an acceptable option, the memory outputs should be scannable to provide control.

Prerequisites

The RAM_07 rule requires that the memory design units specified with the *memory_type* constraints must be black boxes, that is, their definitions must not exist in the design or in the specified libraries, if any.

Rule Exceptions

The RAM_07 rule will not flag for scanwrapped and/or module_bypassed memories as memory transparency is obtained through module_bypass and in case of scanwrap transparency is not needed.

Method

Simulate power, ground and any available capture conditions. The unblocked fan-out cones for all outputs on all memory instances must only contain single-input combinational devices. If a scannable flip-flop is hit, report a message.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *module_bypass* (optional): Use this constraint to specify modules such as memories that are designed with a bypass between data-in port and data-out port.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that are designed with scan wrappers.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the `dft_set_scan_wrap_on_memory` parameter to off.

Message 2

[WARNING] Memory output pin '`<pin-name>`' is driving Dpin of scan flip-flop

Arguments

Name of the offending pin, `<pin-name>`

Potential Issues

The violation message appears if a memory path is not blocked.

Consequences of Not Fixing

Not fixing the violation may corrupt the scan testing.

How to Debug and Fix

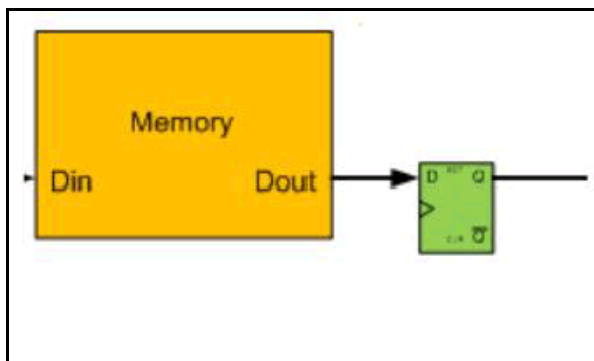
Double-click the violation message. The Incremental Schematic window highlights the memory instance (black box instance) and its output pins that are driving data pin of a scannable flip-flop.

To fix the violation, ensure that the memory path to the scan flip-flop is blocked.

Example Code and/or Schematic

Example 1

Consider the following figure:

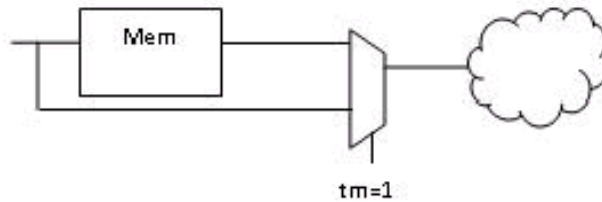


In the above example, the RAM_07 rule reports violation because the

memory output is directly connected to a scannable flip-flop.

Example 2

Consider the following figure:



The RAM_07 rule does not report violation for the above example because the memory output is not directly connected to a scannable flip-flop.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No reports and related files.

RAM_08

Ensure that the write controls to register files/memories are ATPG controlled during capture mode

When to Use

Use this rule to ensure that these write pins are ATPG controlled during the capture mode.

Description

The RAM_08 rule reports violation for memory instances that are not writable in the capture mode.

In order to generate test patterns that target the faults in the “memory shadow” logic, it has to be ensured that RAMs should be writable during the capture mode, that is, it should be possible to force active value on write controls during the capture mode.

Prerequisites

The RAM_08 rule requires that the memory design units specified with the *memory_type* constraints must be black boxes that is, their definitions must not exist in the design or in the specified libraries, if any.

Method

Simulate all testmode conditions for scan shift.

If the write pin on any memory instance is not inactive, report a message.

Simulate capture mode.

If the write pin on any memory instance is not x, report a message.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Use this constraint to specify the memory design unit (black box) names.
- *memory_write_pin* (mandatory): Use this constraint to specify the write pin port name on a memory and the inactive value on that port.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that are designed with scan wrappers.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Memory write enable pin <pin-name> is not ATPG

controllable to '<value>' in capture mode

Arguments

- Name of the enable pin, <pin-name>
- Value at which enable pin is not ATPG controllable, <value>

Potential Issues

The violation message appears, if memory enable pin is not ATPG controllable in the capture mode.

Consequences of Not Fixing

Not fixing the violation may reduce coverage and controllability over the design.

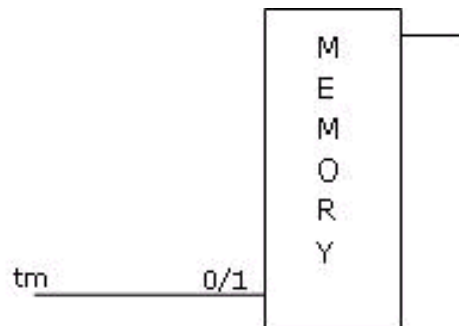
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the memory instance and its write enable pin, which is not ATPG controllable in capture mode).

To fix this violation, correct the corresponding constraint.

Example Code and/or Schematic

Consider the following figure:



In the above example, the RAM_08 rule reports violation because the write enable pin is not ATPG controllable in the capture mode.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No reports and related files.

RAM_09

Ensure that the Memory inputs must feed scannable flip-flops during capture

When to Use

Use this rule while testing logic around the memory.

Description

The RAM_09 rule reports violation for memory input nets that are not scannable.

Memories are often made transparent in order to provide a means to control the logic fed the memory outputs. In cases where transparency is not an acceptable option, the memory inputs should be scannable to provide control.

Prerequisites

The RAM_09 rule requires that the memory design units specified with the *memory_type* constraints must be black boxes that is, their definitions must not exist in the design or in the specified libraries, if any.

Rule Exceptions

The RAM_09 rule will not flag for scanwrapped and/or module_bypassed memories as memory transparency is obtained through module_bypass and in case of scanwrap transparency is not needed.

Also, the rule ignores clock-pin from the rule checking.

Method

Simulate power, ground and any available capture conditions. The unblocked fan-out cones for all inputs on all memory instances must contain at least one scannable flip-flop. If any device other than a scannable flip-flop is hit, report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- *memory_type* (optional): Specifies the memory design unit (black box) names.
- *module_bypass* (optional): Use this constraint to specify modules such as memories that are designed with a bypass between data-in port and data-out port.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that are designed with scan wrappers.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Memory input pin '<pin-name>' does not feed scan flip-flop during capture

Arguments

Name of the offending pin, <pin-name>

Potential Issues

The violation message appears, if a memory input pin does not feed a scannable flip-flop during capture.

Consequences of Not Fixing

Not fixing the violation may result in reduced observability of the design.

Compliance with this rule isolates RAM inputs so that upstream logic can be tested with no dependence on the state of the RAM.

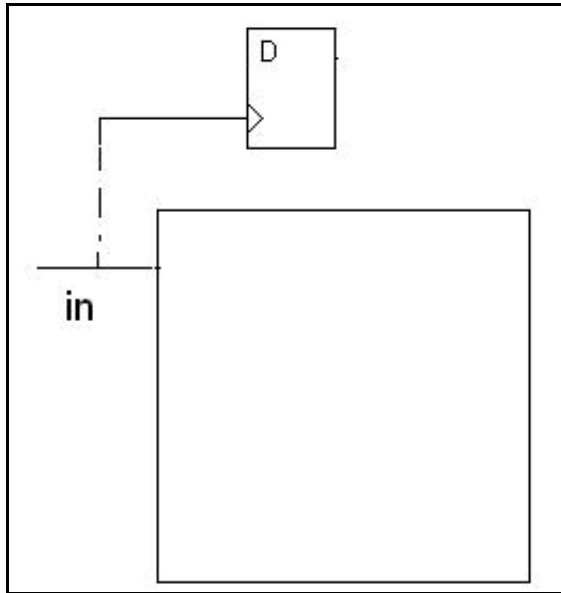
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the memory instance and offending pin.

To fix the violation, insert the scan flip-flops or test points in the design.

Example Code and/or Schematic

Consider the following figure:



The RAM_09 rule reports a violation for the above example because the input pin of the memory does not feed scannable flip-flops.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No reports and related files.

RAM_10

Ensure that the memory or black box input and output terminals are scanwrapped or bypassed

When to Use

Use this rule to test logic around the memory.

Description

The RAM_10 rule identifies the terminals of memory or black boxes that are not scanwrapped or bypassed. The rule flags violation for individual terminals. The violation message specifies which pins are neither scannable nor bypassed.

Rule Exceptions

The RAM_10 rule will not flag for scanwrapped and/or module_bypassed memories as memory transparency is obtained through [module_bypass](#) and in case of scanwrap transparency is not needed.

Also, the rule ignores clock-pin from the rule checking.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- [dft_set_scan_wrap_on_memory](#): The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- [memory_type](#) (optional): Specifies the memory design unit (black box) names.

- *module_bypass* (optional): Use this constraint to specify modules such as memories that are designed with a bypass between data-in port and data-out port.

NOTE: *The RAM_10 rule honors only the `module_bypass` constraint and not the `bypass` constraint. Thus, for bypassing any memory or black box units, you should use `module_bypass` constraint, so as to get no violations with this rule.*

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that are designed with scan wrappers.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the *dft_set_scan_wrap_on_memory* parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the *dft_set_scan_wrap_on_memory* parameter to off.

Message 2

[WARNING] Terminal '<terminal -name>' [Type '<memory>'] is neither scanwrapped nor bypassed"

Arguments

- Name of the offending terminal of the memory module or black box (input/output), <terminal-name>
- Output of memory or black box, <memory>

Potential Issues

The violation message appears, if a terminal is neither scan wrapped nor bypassed.

Consequences of Not Fixing

If you do not fix this violation, the unknown values may corrupt the design during capture mode.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the memory instance (black box) and the terminals, which are not scannable and bypassed.

You can also view the violation for the Info_testmode rule along with the violation of the RAM_10 rule in the Incremental Schematic window. To do this, double-click the violation for the RAM_10 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the RAM_10 rule in the Incremental Schematic window. This is useful in debugging the violation for the RAM_10 rule.

To fix the violation, put a scan wrap around the terminal or bypass the memory logic.

Example Code and/or Schematic

Example 1

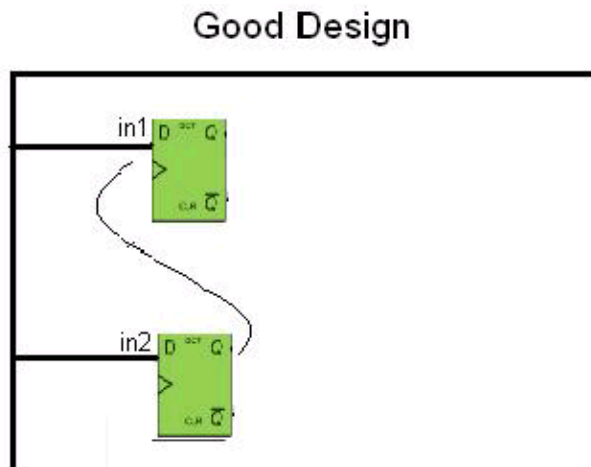
Consider the following figure:



In the above example, the RAM_10 rule reports a violation because input of the memory is not scan wrapped.

Example 2

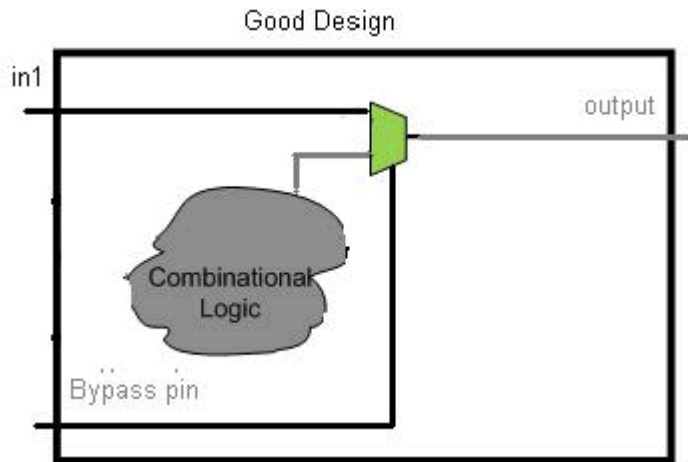
Now, consider the following figure:



The above example demonstrates a good design scenario because the memory scan inputs are scanwrapped using the *scan_wrap* constraint.

Example 3

Consider the following figure:



The above example demonstrates a good design scenario because an output pin is bypassed using the `module_bypass` constraint.

Default Severity Label

Warning

Rule Group

RAM

Reports and Related Files

No related reports or files.

RAM_11

Ensure that there are no combinational loops through memories.

When to Use

Use this rule while testing logic around the memory.

Description

The RAM_11 rule identifies memory instances whose outputs are connected to inputs through combinational logic.

NOTE: *If you specify the `-q_pin` and `-d_pin` arguments of the `memorytype` constraint, only loops from the former to the later are checked. Otherwise, paths from all the output to all inputs are checked.*

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- [dft_set_scan_wrap_on_memory](#): The default value of the parameter is on. Set the value of the parameter to off to turn off automatic scan-wrap on memories.

Constraint(s)

- [memory_type](#) (optional): Use this constraint to specify the memory design unit (black box) names.
- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- [clock](#) (optional): Use this constraint to specify the clocks in the design.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] Rule checking is skipped as parameter 'dft_set_scan_wrap_on_memory' is 'on'. Turn off the parameter to only skip those memories for which explicit scan_wrap sgdc command is set.

Potential Issues

A violation is reported when the [dft_set_scan_wrap_on_memory](#) parameter is enabled.

Consequences of Not Fixing

Not fixing the violation may result in skipping the rule checking.

How to Debug and Fix

To fix the violation, set the value of the [dft_set_scan_wrap_on_memory](#) parameter to off.

Message 2

[WARNING] Memory instance '<inst-name>' has combinational loop from its output pin '<out-pin>' to input pin '<in-pin>'

Arguments

- Name of the memory instance, <inst-name>
- Name of its output pin, <out-pin>
- Name of its input pin, <in-pin>

Potential Issues

The violation message appears, if a memory instance has a combinational loop from its output pin to its input pin.

Consequences of Not Fixing

If you do not fix this violation, ATPG tool finds it difficult to handle the loops as they make a combinational circuit, which are sequential in nature.

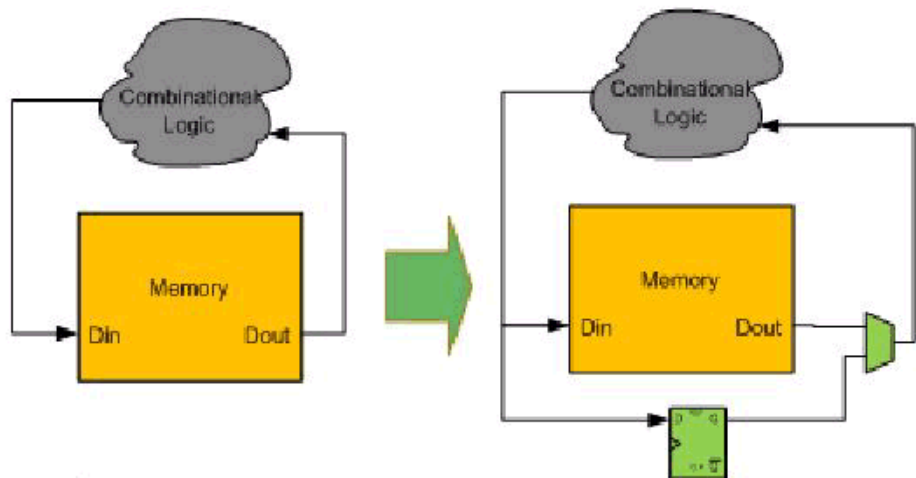
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the memory instance (black box) and the combinational loop path(s) passing through it.

To fix this violation, break the combinational loop or insert flip-flops at required places.

Example Code and/or Schematic

Consider the following figure:



In the above example, a combinational path exists between input and output pins of the memory instance, which causes the violation. To fix the problem and to improve testability, flip-flop, and multiplexer are introduced.

Default Severity Label

Warning

Rule Group

RAM

RAM Rules

Reports and Related Files

No related reports or files.

Scan Rules

Overview

The Scan rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Flags...
Scan_06	Scan chains that do not operate in parallel
Scan_07	testmode signals for which multiple testmode values have been specified
Scan_08	Flip-flops that are not scannable
Scan_11	Designs where the scan ratio exceeds the specified limit
Scan_16	Non-retiming flip-flops that are not scannable
Scan_17	Long chains of non-scannable flip-flops
Scan_18	Primary inputs that are not registered to a scan flip-flop
Scan_19	Primary outputs that are not registered to a scan flip-flop
Scan_20	Outputs of bypassed devices that are affecting scan flip-flops and those inputs of bypassed devices that are not observable
Scan_21	Pullup/pulldown enables that are not controllable by scan flip-flops
Scan_22	Scan chains where lockup latches are not present at the domain crossing
Scan_23	Instances of specified modules that are not in their bypass mode under the capture mode
Scan_24	Flip-flops that are not part of any scan chain under specified simulation conditions
Scan_25	Inverters in scan chains under specified simulation conditions
Scan_26	Scan chains that do not have a lockup latch at chain end
Scan_27	data input pins of scan flip-flop that are not connected as specified using the <code>dftScanDataConnectivityCheck</code> rule parameter during shift.
Scan_28	Scan enable pins when the state (active/inactive) of the scan enable pin of a scan flip-flop is not the same as that specified using the <code>dftScanEnablePinValue</code> rule parameter.

Scan Rules

Rule	Flags...
Scan_29	Inversions in scan chains under specified simulation conditions.
Scan_30	Report observable flip-flop list.
Scan_31	Reports flip-flop's with constant value data pin.
Scan_32	In scan mode, have all scan out inouts be outputs during scan shift
Scan_33	In scan mode, have all scan in inouts be inputs during scan shift.
Scan_34	Preferable name top test signals as per as dedicated parameters
Scan_35	Scan chains should not exceed maximum specified length
Scan_36	Scan chains should be balanced in length, that is, they should not deviate more than allowed from the expected length
Scan_38	Scan chains should start with a positive edge flip-flop
Scan_39	Ensure that each scan flip-flop in a design is part of only one scan chain
Scan_40	Ensure that the scan chains have a 'copy flip-flop' at positive to negative clock edge crossing
Scan_41	Ensure that the scanout port connected to IO-Cells do not have high impedance (Hi-z) output at any given time

Scan_06

All scan chains should operate in parallel

When to Use

Use this rule to identify scan chains that do not operate in parallel.

Description

The Scan_06 rule reports violation for scan chains that do not operate in parallel.

Method

If testmode and test clocks exist, then:

Simulate testmode for scan shift. Simulate pulses on all test clocks in parallel. Any testclock source that does not have a pulse is reported as a message.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

NOTE: *The Scan_06 requires you to specify at least one testclock.*

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Clock source '<net-name>' does not receive test clock pulse with parallel scan chains

Arguments

Name of the net corresponding to the testclock source that is not pulsed, <net-name>

Potential Issues

A violation is reported when a clock source does not get a clock pulse.

Consequences of Not Fixing

Non-parallel chains require multiple scan cycles for each test and thereby require extra test time. Parallel chains require the same amount of tester memory but may substantially reduce test application time.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the path from the clock source to a flip-flop which does not get testclock pulse when all the testclocks are simulated simultaneously under the shift mode condition.

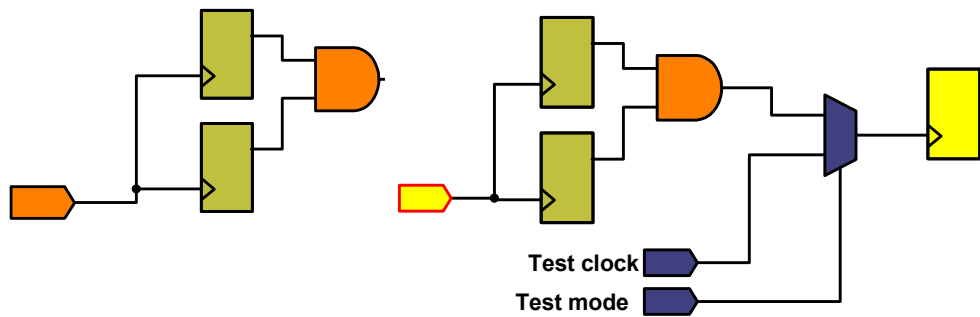
You can also view the violations for the Info_testmode (under shift condition) and Info_testclock rules along with the violation of the Scan_06 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_06 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Scan_06 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_06 rule.

To fix the violation, modify constraints or design to ensure clock propagation to all flip-flops.

Example Code and/or Schematic

Consider the following figure:



The above figure shows a good design.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_07

Ensure that there are no internal test_mode signals

When to Use

Use this rule to identify internal test mode signals.

Description

The Scan_07 rule detects *test_mode* sgdc commands which are applied on an internal node.

ATPG tools assume that the test controls are directly controllable. Registered signals violate that assumption and increase difficulty for ATPG tools. Ensure that test_mode constraints are not disturbed during the scan shift operation. Any disturbance to test_mode conditions results in incorrect DFT analysis.

Probability of controlling the internal node to a specific value decreases, if the internal node has non-forced no_scan flip-flop as one of the potential drivers. Assume that the test_mode constraint is set on a node, which is in the fanout of a flip-flop. Also assume that this flip-flop is not explicitly specified as force_no_scan. In this case, rule reports a violation that design has an invalid constraint because the flip-flop, which may participate in shift, cannot be controlled during the scan shift operation.

Method

If any testmode constraint present on a internal node, flag a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift, *Capture*, *Atspeed*

Messages and Suggested Fix

Message 1

[INFO] Internal '<mode_name>' (test_mode) constraint found on node '<node_name>' which has combinational fanin cone with '<number>' combinational driver(s)

Arguments

- Operating Mode, <mode_name>
- Name of the signal, <node_name>
- No of combinational drivers, <number>

Potential Issues

A violation is reported when internal testmode signals with combinational fanin cone are used.

Consequences of Not Fixing

Internal test_mode may be useful for doing what-if analysis or for bypassing complex logic. For example, setting test_mode on outputs of TAP Controllers rather than sequence the controller through various states using more complex test_mode defined on the inputs of the controller, can result in artificially high test coverage figures. The ATE will not be able to drive values onto internal nodes and hence the observed coverage post-synthesis/scan-insertion may be much lower than that predicted by SpyGlass.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the sequential testmode signals in the design. To fix the violation, try controlling the testmode signals from port combinationally.

Message 2

[INFO] Internal '<mode_name>' (test_mode) constraint found on node '<node_name>' which has sequential fanin cone with '<number>' sequential (black-boxes, undriven net, and/or forced no_scan flip-flops) driver(s)

Arguments

- Operating Mode, <mode_name>
- Name of the signal, <node_name>
- No of sequential drivers, <number>

Potential Issues

A violation is reported when internal testmode signals with sequential fanin cone are used.

Consequences of Not Fixing

Internal test_mode may be useful for doing what-if analysis or for bypassing complex logic. For example, setting test_mode on outputs of TAP Controllers rather than sequence the controller through various states using more complex test_mode defined on the inputs of the controller, can result in artificially high test coverage figures. The ATE will not be able to drive values onto internal nodes and hence the observed coverage post-synthesis/scan-insertion may be much lower than that predicted by SpyGlass.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the sequential testmode signals in the design. To fix the violation, try controlling the testmode signals from port combinationally.

Message 3

[INFO] Internal '<mode_name>' (test_mode) constraint found on node '<node_name>' which has sequential fanin cone with '<number>' flip-flop(s) which are not forced no_scan. List of flip-flops: '<flip_flop_names>'

Arguments

- Name of the signal, <node_name>
- No of combinational drivers, <number>

- Space separated flip-flop names, <flip_flop_names>

Potential Issues

A violation is reported when internal testmode signals with sequential fanin cone and undriven nets are used.

Consequences of Not Fixing:

Internal test_mode may be useful for doing what-if analysis or for bypassing complex logic. For example, setting test_mode on outputs of TAP Controllers rather than sequence the controller through various states using more complex test_mode defined on the inputs of the controller, can result in artificially high test coverage figures. The ATE will not be able to drive values onto internal nodes and hence the observed coverage post-synthesis/scan-insertion may be much lower than that predicted by SpyGlass.

How to Debug and Fix

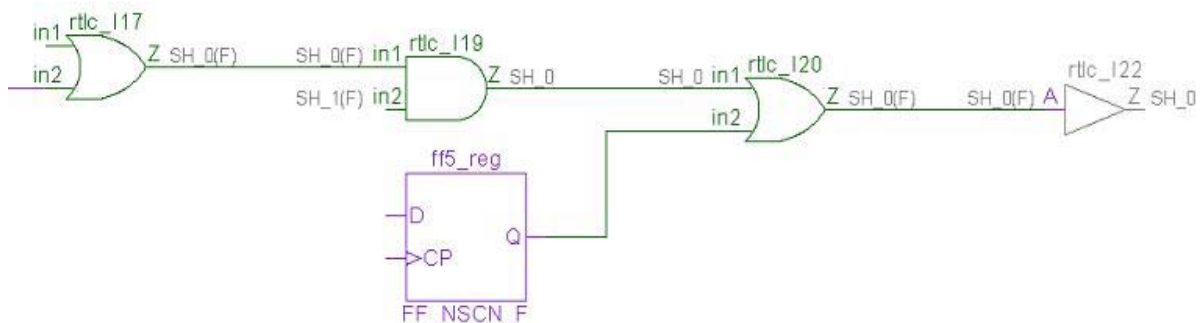
View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the sequential testmode signals in the design.

To fix the violation, try controlling the testmode signals from port combinationally.

Example Code and/or Schematic

Example 1

Consider the following schematic:



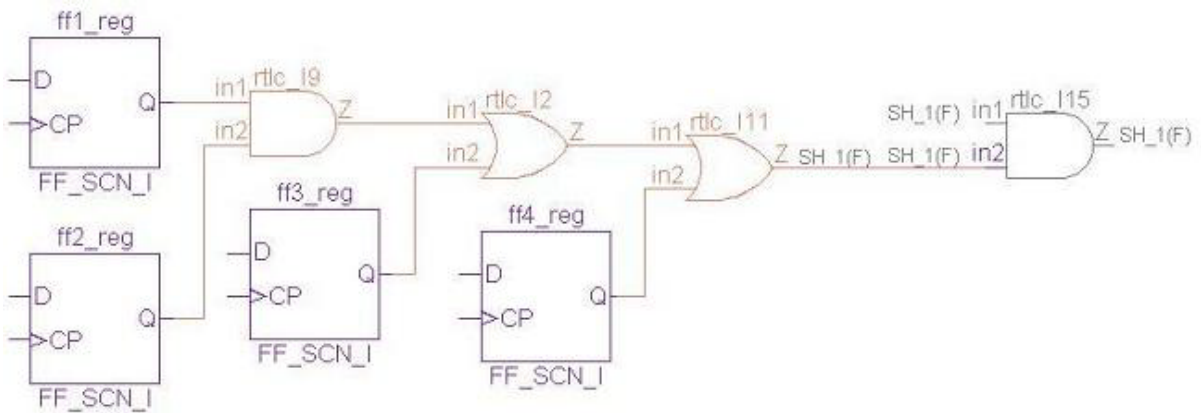
Scan Rules

For the above design, the Scan_07 rule reports the following violation message:

Internal 'Shift mode' (test_mode) constraint, found on node 'top.w5' which has sequential fanin cone with '2' sequential (black-boxes, undriven net, and/or forced no_scan flip-flops) driver(s)

Example 2

Consider the following schematic:

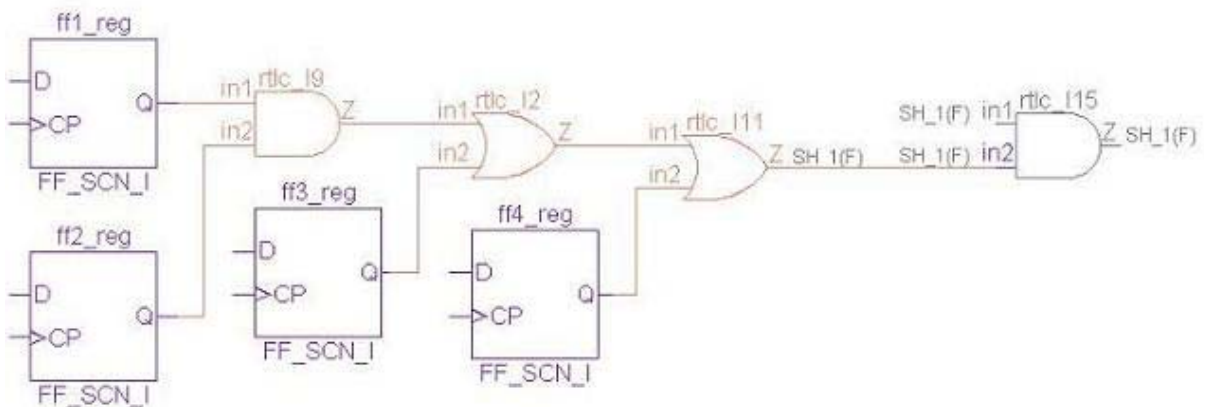


For the above design, the Scan_07 rule reports the following violation message:

Internal 'Shift mode' (test_mode) constraint, found on node 'top.w2' which has combinational fanin cone with '2' combinational driver(s)

Example 3

Consider the following schematic:



For the above design, the Scan_07 rule reports the following violation message:

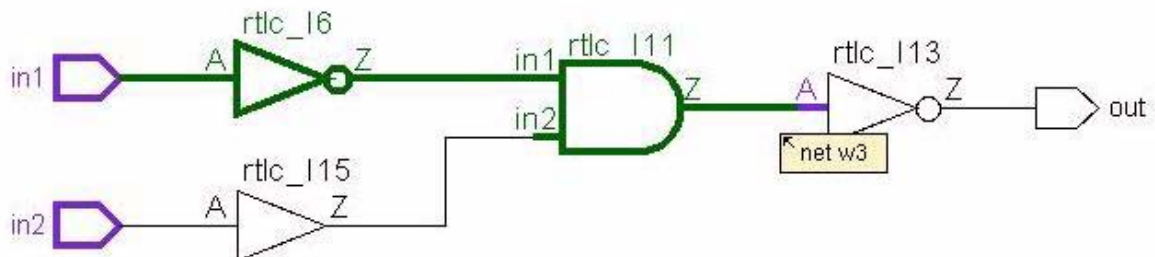
Internal 'Shift mode' (test_mode) constraint, found on node 'top.w1' which has sequential fanin cone with '4' flip-flop(s) which are not forced no_scan. List of flip-flops: 'top.ff1_reg top.ff2_reg top.ff3_reg top.ff4_reg'

Example 4

Consider the following SGDC definition:

```
test_mode -name w3 -value 0
```

Also, consider the following schematic:



Scan Rules

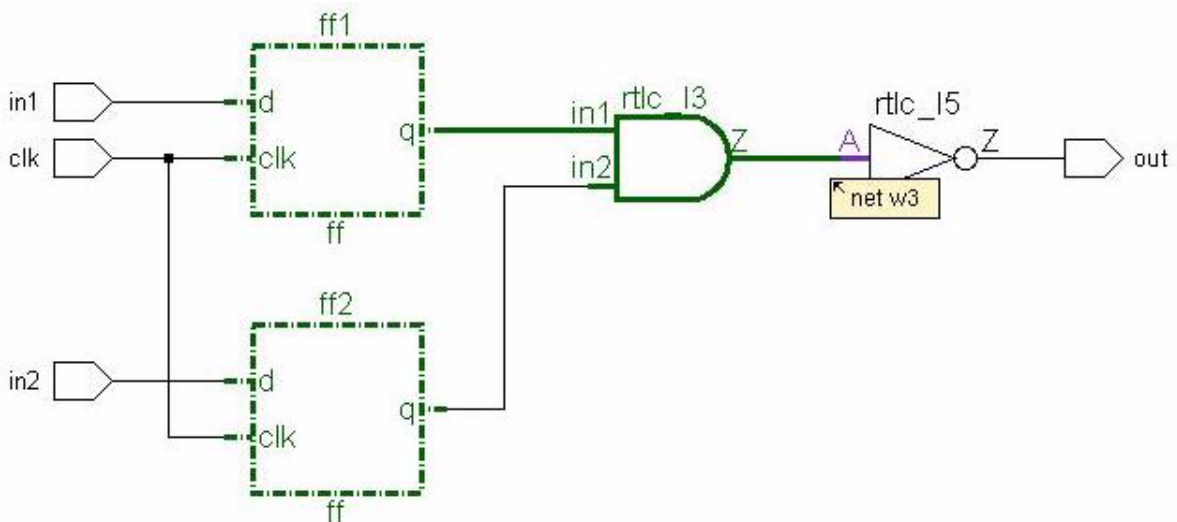
In the above example, the Scan_07 rule reports a violation because the internal testmode signal, w3, with combinational fanin cone and undriven nets is used.

Example 5

Consider the following SGDC definition:

```
test_mode -name w3 -value 0
```

Also, consider the following schematic:



In the above example, the Scan_07 rule reports a violation because the internal testmode signal, w3, with sequential fanin cone and undriven nets is used.

Example 6

The following figure illustrates the where scan enable, se, is dependent on flip-flops, FF1 and FF2:

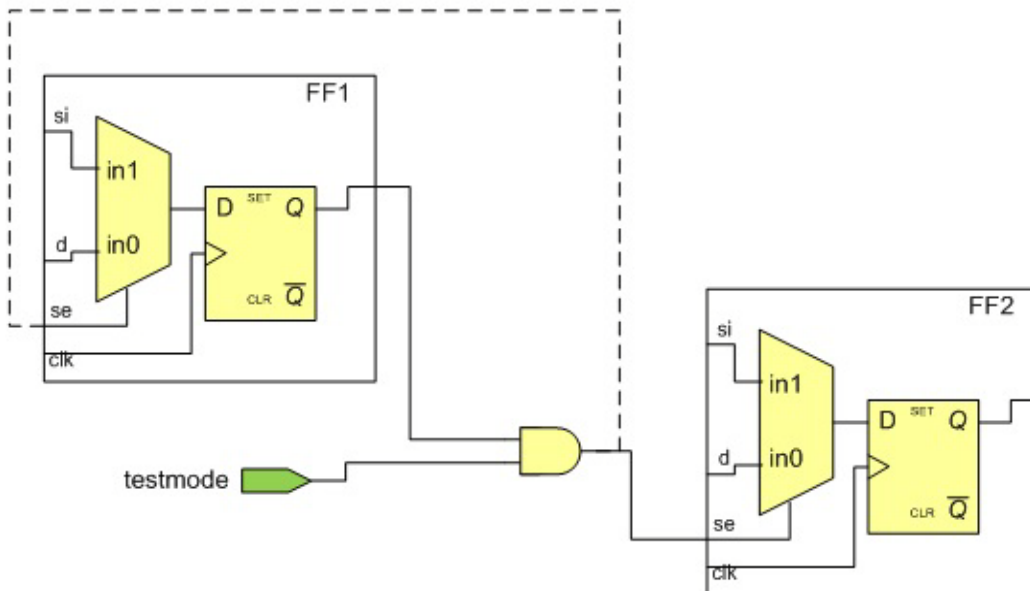


FIGURE 90. Gated Testmode

The above example depicts a bad design scenario because scan enable depends on scan flip-flop state. As a result, the scanin operation may create a flip-flop state that blocks successful completion of scanin.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan Rules

Scan_08

Ensure that all registers and flip-flops are scannable

When to Use

Use this rule while using combinational ATPG and for complete test coverage.

Description

The Scan_08 rule reports violation for flip-flops that are not scannable.

NOTE: *When the Scan_08 rule is run in conjunction with either or both [Async_07](#) or [Clock_11](#) rules, a tag can be added in the violation messages by setting the value of the [dftTagAsync07Clock11Scan08](#) rule parameter to on. This enables the correlation of the cause of unscannability.*

NOTE: *The Scan_08 rule flags all flip-flops that are not scannable whereas the [Scan_16](#) rule flags all flip-flops (excluding retiming flip-flops) that are not scannable.*

The Scan_08 rule identifies flip-flops whose clocks are not testclock controllable or whose asynchronous set or asynchronous reset pins are not inactive in shift mode. The rule message specifies the register and which pin(s) on that register prevent scannability. Use the Scan_08 rule when clock sources and/or asynchronous sources cannot be modified with testmode control.

Method

Simulate testmode for scanshift and then simulate a pulse on each testclock. Report all flip-flops with a clock that is not controlled by a test clock or with an asynchronous set or asynchronous reset that is not inactive during scan shift.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftTagAsync07Clock11Scan08](#): Default value is off. Set the value of this parameter to on to add a unique tag for every asynchronous source

such as set, reset, or clock that is visible in the violations of *Scan_08* rules. This enables the correlation of *Async_07* and *Clock_11* rules with the *Scan_08* rule.

- *dft_ignore_constant_supply_flip_flops*: The default value is off. Set the value of the parameter to on to ignore Synthesis Redundant (SR) flip-flops from rule checking.
- *dftIgnoreConstantOrUnusedFlipFlops*: The default value is off. Set the value of the parameter to on to ignore the violations reported by the *Scan_08* rule when any of the following conditions is true:
 - A constant value is specified to the D pin of a flip-flop.
 - A constant value is specified to the Q pin of a flip-flop.
 - The value at the Q pin of a flip-flop is not captured at a scan point.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *memory_type* (optional): Use this constraint to specify the memory design unit (black box) names.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, when simulated, forces the circuit in the test mode.
- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type, LSSD or MUXSCAN.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] <tag> Flip-flop '*<flip-flop-name>*' is not scannable
[<reason-list>]

Arguments

- Asynchronous source tag in the form DFT_ASYNC_CLR_id, DFT_ASYNC_SET_id or DFT_CLK_id <tag>
- Name of the flip-flop output pin, <flip-flop-name>
- List of reasons for the flip-flop not being scannable, <reason-list>
The reasons can be one of the following:
 - PRE:active
 - CP:Test-clock not reaching
 - CLR:active

Potential Issues

This violation message appears, if the clock and the reset are not controlled properly.

Consequences of Not Fixing

If you do not fix this violation, it could make flip-flop uncontrollable and unobservable.

To provide maximum test coverage, all flip-flops must be scannable in order to convert as much logic as to combinational test.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the flip-flop and its terminal due to which it is non-scannable.

You can also view the violations for the Info_testmode (under shift condition) and Info_testclock rules along with the violation of the Scan_08 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_08 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Scan_08 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_08 rule.

To fix clock problems, specify a testclock (using the cLock constraint with

-testclock argument) and a testmode signal (using the test_mode constraint) and ensure that when in shift mode, the clock to this register is controlled by a testclock.

To fix set or reset problems, add testmode signal (using the test_mode constraint) and ensure that when in shift mode, the set and reset pins to this register are inactive.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module ScanThreshold (tm, clk, si1, so1, si2, so2,
    d1, d2, d3, d4, q0, q1, q2, q3, q4);
    input tm, clk, si1, si2, d1, d2, d3, d4;
    output so1, so2, q1, q2, q3, q4;

    reg q0, q1, q2, q3, q4;
    wire clk, m1, m2, m3, m4;

    assign m1= tm ? si1 : d1;
    assign m2= tm ? q2 : d2;
    assign m3= tm ? si2 : d3;
    assign m4= tm ? q3 : d4;

    always @ (posedge clk)
    begin
        q0 <= d1;
        q1 <= m1;
        q2 <= m2;
        q3 <= m3;
        q4 <= m4;
    end
endmodule
```

The above example demonstrates a bad design scenario because not all the flip-flops in the design are scannable.

Example 2

Consider the following VHDL code:

```
Architecture arch_ScanThreshold of ScanThreshold
Begin
  Signal clk, m1, m2, m3, m4 : std_logic;
  M1 <= si1 when tm = '1' else d1;
  M2 <= q2  when tm = '1' else d2;
  M3 <= si2 when tm = '1' else d3;
  M4 <= q3 when tm = '1' else d4;

  Process (clk)
  Begin
    If (clk'event and clk = '1') then
      Q0 <= d1;
      Q1 <= m1;
      Q2 <= m2;
      Q3 <= m3;
      Q4 <= m4;
    End if;
  End process;
End arch_ScanThreshold;
```

The above example demonstrates a bad design scenario because not all the flip-flops in the design are scannable.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No reports and related files.

Scan_11

Ensure that the scan ratio must exceed threshold

When to Use

Use this rule when complete test coverage is required.

Description

The Scan_11 rule reports violation for designs where the scan ratio does not exceed the limit specified with the *scan_ratio* constraint.

Method

The scan ratio is computed as follows:

$$\frac{(\text{scannable flip-flops} + \text{forced/inferred scan flip-flops})}{(\text{total number of flip-flops} - \text{forced/inferred_no_scan flip-flops} - \text{synthesis_redundant flip-flops})}$$

If the computed number is less than a user-specified number, then flag a message.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *scan_ratio* (mandatory): Determines what fraction of the flip-flops should be scanned or be made scannable.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Specifies the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Scan ratio <num> does not exceed threshold value <limit>

Arguments

- Scan ratio of the design, <num>
- Threshold value of scan ratio (read from constraint), <limit>

Potential Issues

This violation message appears, if some of the flip-flops are not scannable.

Consequences of Not Fixing

Not fixing this violation increases the level of difficulty for ATPG tools and may result in lower fault coverage and longer test time.

Often, there are critical timing paths, or other design considerations that render full scan impractical. These cases increase the level of difficulty for ATPG tools and may result in lower fault coverage and longer test time. This rule counts the non-scan devices and issues a % scan report calculated as =scannables/total-sequentials.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the scan and non-scan flip-flops in different colors.

To fix the violation, insert test points or scan flip-flops in the design.

Example Code and/or Schematic

Currently Unavailable

Scan Rules

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_16

Ensure that all flip-flops, except retiming flip-flops, are scannable

When to Use

Use this rule while using combinational ATPG and when complete test coverage is required.

Description

The Scan_16 rule reports violation for all the flip-flops except retiming flip-flops that are not scannable.

NOTE: The [Scan_08](#) rule flags all flip-flops that are not scannable whereas the Scan_16 flags all flip-flops (excluding retiming flip-flops) that are not scannable.

Method

Perform scannability analysis. Report all non-scannable flip-flops unless that flip-flop satisfies all requirements as a retiming flip-flop.

See [Retiming or Lockup Latches](#) to know how retiming flip-flops are recognized.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftTagAsync07Clock11Scan08](#): Default value is off. Set the value of the parameter to on to add a unique tag for every asynchronous source such as set, reset, or clock that is visible in the violations of Scan_08 rules. This enables the correlation of [Async_07](#) and [Clock_11](#) rules with the [Scan_08](#) rule.
- [dft_ignore_constant_supply_flip_flops](#): The default value is off. Set the value of the parameter to on to ignore Synthesis Redundant (SR) flip-flops from rule checking.
- [dftIgnoreConstantOrUnusedFlipFlops](#): The default value is off. Set the value of the parameter to on to ignore the violations reported by the Scan_16 rule when any of the following conditions is true:

- ❑ A constant value is specified to the D pin of a flip-flop.
- ❑ A constant value is specified to the Q pin of a flip-flop.
- ❑ The value at the Q pin of a flip-flop is not captured at a scan point.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Specifies the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] <tag> Non-retiming flip-flop '<flip-flop-name>' is not scannable [<reason-list>]

Arguments

- Asynchronous source tag in the form DFT_ASYNC_CLR_id, DFT_ASYNC_SET_id <tag>
- Name of the output net of the offending flip-flop, <flip-flop-name>
- List of reasons for the flip-flop not being scannable, <reason-list>
- The reasons, <reason>, can be one of the following:

PRE:active

CP: Test-clock not
reaching

CLR: active

Potential Issues

The violation message appears, if a clock and the reset pins are not controlled properly.

Consequences of Not Fixing

Not fixing this violation makes flip-flops uncontrollable and unobservable, hence coverage is reduced.

To provide maximum test coverage, all flip-flops must be scannable in order to convert as much logic as to combinational test. Full scan tools generally treat retiming flip-flops as transparent and therefore they need not be scannable.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the non-scannable flip-flop that is not a retiming flip-flop.

You can also view the violations for the Info_testmode and Info_testclock rules along with the violation of the Scan_16 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_16 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Scan_16 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_16 rule.

To fix the violation, ensure the constraints or the connections of the design are correct.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module ScanThreshold (tm, clk, si1, so1, si2, so2,  
    d1, d2, d3, d4, q0, q1, q2, q3, q4);  
    input tm, clk, si1, si2, d1, d2, d3, d4;
```

```

output s01, s02, q1, q2, q3, q4;

reg q0, q1, q2, q3, q4;
wire clk, m1, m2, m3, m4;

assign m1= tm ? si1 : d1;
assign m2= tm ? q2 : d2;
assign m3= tm ? si2 : d3;
assign m4= tm ? q3 : d4;

always @ (posedge clk)
begin
    q0 <= d1;
    q1 <= m1;
    q2 <= m2;
    q3 <= m3;
    q4 <= m4;
end
endmodule

```

The above example demonstrates a bad design scenario.

Example 2

Consider the following VHDL code:

Architecture arch_ScanThreshold of ScanThreshold

Begin

```

Signal clk, m1, m2, m3, m4 : std_logic;
M1 <= si1 when tm = '1' else d1;
M2 <= q2  when tm = '1' else d2;
M3 <= si2 when tm = '1' else d3;
M4 <= q3 when tm = '1' else d4;

```

Process (clk)

Begin

```

If (clk'event and clk = '1') then
    Q0 <= d1;
    Q1 <= m1;
    Q2 <= m2;

```

```
        Q3 <= m3;  
        Q4 <= m4;  
    End if;  
End process;  
End arch_ScanThreshold;
```

The above example demonstrates a bad design scenario.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_17

Ensure that the sequential depth does not exceed limit between any two scan points.

When to Use

Use this rule while using sequential ATPG.

Description

The Scan_17 rule reports violation for long chains of non-scannable flip-flops.

The Scan_17 rule reports violation when the number of non-scannable flip-flops between a primary input, a scannable flip-flop, or a scan-wrapped black box and a primary output, a scannable flip-flop, or a scan-wrapped black box exceeds the limit specified by the *sequentialDepth* rule parameter (default 5).

This rule is useful when sequential ATPG tools are used that are capable of generating tests that span a limited number of cycles.

Prerequisites

The Scan_17 rule should only be run if sequential ATPG will be used. Otherwise, do not select this rule since it does require excessive run time.

Method

Walk depth first from a primary input or scannable flip-flop or output from a scanwrap module. Terminate the walk on primary outputs, scannable flip-flops, or scanwrap modules or black boxes. If the number of non-scannable flip-flops on a path exceeds the value of the *sequentialDepth* rule parameter (default 5), then report a message. Report the path endpoints and the non-scannable flip-flops encountered.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

- *sequentialDepth*: Default value is 5. Use this parameter to set the maximum limit on the number of non-scannable flip-flops allowed in path between a primary input, a black box, or a scannable flip-flop and another black box, scannable flip-flop, and a primary output.
- *showPath*: Default value is off and only the end points are highlighted. Set the value of this parameter to on to show the complete path of the schematic highlight generated by the Scan_17 rule and not just the end points.
- *noBlackBoxReporting*: Default value is off. This parameter causes the suppression of rule messages for nets made uncontrollable or unobservable by black boxes. By default, these rule messages are reported.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_use_n_cycle_capture*: The default value is 1. Set the value of the parameter to any positive integer to specify the number of capture cycles.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use to declare the clock pins used as test clocks. Use this constraint's -testclock argument for the Scan_17 rule.
- *seq_atpg* (mandatory): Use this constraint to specify the case analysis conditions.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *scan_wrap* (mandatory): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.

Operating Mode

Scanshift, Capture

Message Details

[WARNING] Sequential depth between scan points '`<name1>`' and '`<name2>`' is more than the specified limit. (`<start-obj-type>` - `>` (`<end-obj-type>`), Depth: `<num>` `>` \$sequentialDepth

Arguments

- Name of the starting object, `<name1>`
- Name of the ending object, `<name2>`
- Type of starting object, `<start-obj-type>`
Type can be Input port, scan-wrapped black-box, or scannable FlipFlop.
- Type of ending object, `<end-obj-type>`
Type can be Output port, scan-wrapped black-box, or scannable FlipFlop.
- Count of the number of non-scannable flip-flops between start and end points (`<num>`).

Potential Issues

The violation message appears if the number of non-scannable flip-flops exceeds the maximum limit.

Consequences of Not Fixing

If you do not fix this violation, ATPG may not generate pattern for sequential depth higher than specified.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the following:

- Starting point (primary input, scannable flip-flop, or scan-wrapped black box) of chain of non-scan flip-flops
- End point (primary output, scannable flip-flop, or scan-wrapped black box) of chain of non-scan flip-flops.

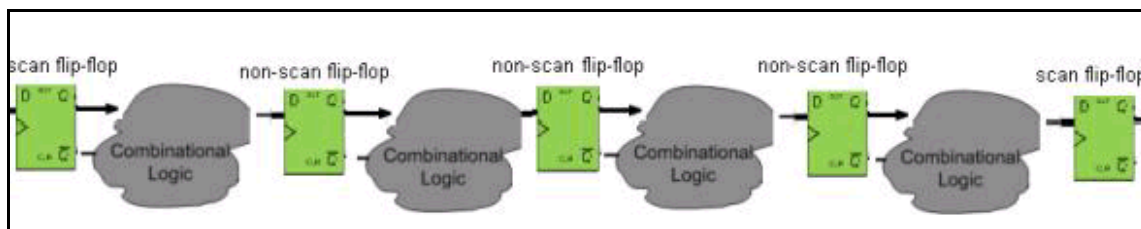
You can also view the violation for the Info_path rule along with the violation of the Scan_17 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_17 rule and open the Incremental Schematic window.

The violation message for the Info_path rule overlaps the violation message for the Scan_17 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_17 rule.

To fix the violation, make the flip-flops scannable by inserting test points in the design.

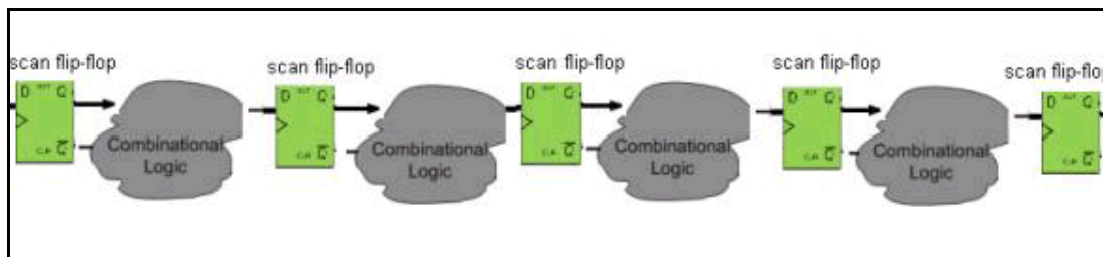
Example Code and/or Schematic

Consider the following figure:



In the above figure, if a user has specified the sequential width as 2, and the design has 3 non-scan flip-flops in a row, the Scan_17 rule reports a violation.

To fix the above violation, make all the non-scan flip-flops scannable or at least make 1 more non-scan flip-flop scannable so that the sequential depth is not exceeded, as shown in the following figure:



Default Severity Label

Warning

Rule Group

Scan

Scan Rules

Reports and Related Files

No related reports or files.

Scan_18

Ensure that all top level inputs are registered

When to Use

Use this rule when a combinational path is not required between two ports.

Description

The Scan_18 rule reports primary input ports that are not registered to a scan flip-flop.

Rule Exceptions

The Scan_18 rule ignores primary input ports that are declared as clock ports (using the *clock* constraint) or testmode ports (using the *test_mode* constraints).

Method

Find all scannable flip-flops

For each primary input

Walk the combinational fan-out cone for this input

If no scannable flip-flop was hit, report a message.

If the d-input of a scannable flip-flop is reached, do a reverse walk through single-input devices only. If the original input pin is reached by this reverse walk, then no message; otherwise report a message.

end for

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Use this constraint to declare the clock pins used as test clocks. Use this constraint's -testclock argument for the Scan_18 rule.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *scan_cell* (optional): Use this constraint to define scan cell design units and their data pins as used by the Scan_18 rule.

Operating Mode

None

Messages and Suggested Fix

[WARNING] Input Port(s) '<port-name>' is(are) not registered to a scan flip-flop

Arguments

Name of the input port, <port-name>

Potential Issues

The violation message appears, if scan flip-flops are missing at the input ports.

Consequences of Not Fixing

Not fixing this violation may result in non synchronization because of long snake paths.

Registering inputs simplifies diagnosis and use of scan rings for re-use of ATPG vectors.

This is particularly necessary for IP blocks to ensure that good fault coverage is attainable without making any assumptions about the controllability/observability of the input/output ports at the top level.

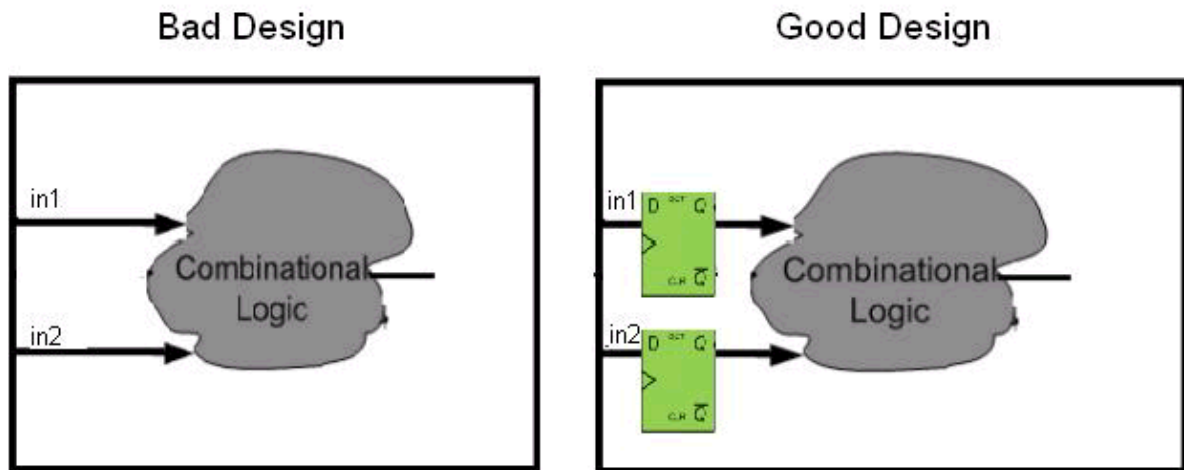
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the top-level input port.

To fix the violation, ensure to put a scan flip-flops before an input is fed to a logic.

Example Code and/or Schematic

Consider the following figure:



The above example demonstrates both the good and the bad design scenarios. For the bad design, input ports are not connected to any scan flip-flop. Therefore, the Scan_18 rule reports a violation. To fix this violation, you need to add scan flip-flops at the input ports as shown in the good design.

Default Severity Label

Warning

Rule Group

Scan

Scan Rules

Reports and Related Files

No related reports or files.

Scan_19

All top level outputs must be controlled by scan flip-flops

When to Use

Use this rule to identify primary outputs that are not registered to a scan flip-flop.

Description

The Scan_19 rule reports violation for primary outputs that are not registered to a scan flip-flop.

Method

Find all scannable flip-flops

For each primary output

Walk the single-input fan-in cone

If the walk terminates on anything other than a scannable flip-flop, report a message
end for

Language

Verilog, VHDL

Default Weight

10

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

- *force_no_scan* (optional): Excludes flip-flops from being declared scannable even if they so qualify.
- *scan_type* (optional): Use to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Output Port(s) '<port-name>' is(are) not registered to a scan flip-flop

Arguments

Name of the output port, <port-name>

Potential Issues

A violation is reported when scan flip-flop is not driving primary output port.

Consequences of Not Fixing

Unregistered output ports create diagnosis problems. Whereas, registering outputs simplifies diagnosis and use of scan rings for re-use of ATPG vectors.

This is particularly necessary for IP blocks to ensure that good fault coverage is attainable without making any assumptions about the controllability/observability of the input/output ports at the top level.

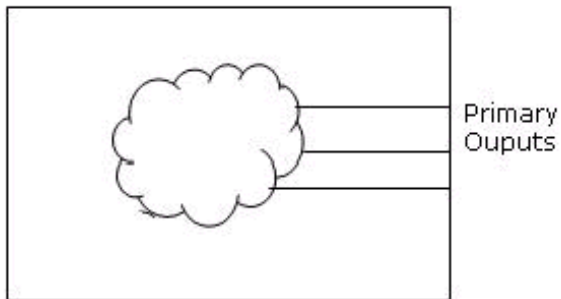
How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the top-level output port.

To fix the violation, put scan flip-flop before every primary output port.

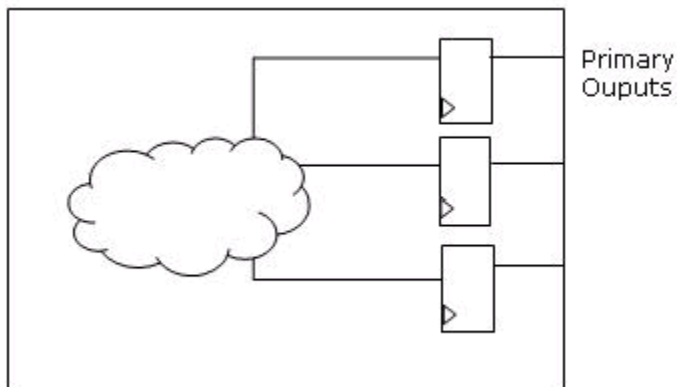
Example Code and/or Schematic

Consider the following figure:



The above figure illustrates a bad design scenario because scan flip-flops are not driving primary outputs.

Now, consider the following figure:



The above figure illustrates a good design scenario because scan flip-flops are driving primary outputs.

Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

No related reports or files.

Scan Rules

Scan_20

Ensure that outputs of bypassed devices do not affect any scan flip-flop data & inputs are observable

When to Use

Use this rule when the isolation of bypassed blocks is required.

Description

The Scan_20 rule reports violation for those outputs of bypassed devices that are affecting scan flip-flops and those inputs of bypassed devices that are not observable.

Method

Simulate capture mode conditions (testmode without -scanshift argument)

for each instance of a bypassed device

for each pin on this instance

If an output pin

Walk the unblocked combinational fan-out cone. Check that it is blocked (exclusively by testmode signals) at the first occurrence of >1 input gate. Otherwise report a message. Or if any scannable flip-flop or primary output is reached, report a message.
else

The net that is driving this input must be observable or be sourced by a scannable flip-flop. Otherwise, a message is reported.

end if

end for

end for

Default weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *bypass* (mandatory): Specified the design units (black boxes) that must be bypassed.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (mandatory): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type, LSSD, or MUXSCAN. Use this constraint's `-lssd` argument for the Scan_20 rule.

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING] Output pin '<pin-name>' of bypass device '<inst-name>' is driving scan flip-flop

Arguments

To view the arguments description, click [Arguments](#).

Potential Issues

The violation message appears due to bad connectivity and/or missing constraints specifications.

Consequences of Not Fixing

Not fixing this violation may result in reduced test coverage and controllability because of corrupt test vectors.

How to Debug and Fix

To debug and fix the violation message, click [How to Debug and Fix](#).

Message 2

[WARNING] Output pin '<pin-name>' of bypass device '<inst-name>' is driving primary port

Arguments

To view the arguments description, click [Arguments](#).

Potential Issues

The violation message appears due to bad connectivity and/or missing constraints specifications.

Consequences of Not Fixing

Not fixing this violation may result in reduced test coverage and controllability because of corrupt test vectors.

How to Debug and Fix

To debug and fix the violation message, click [How to Debug and Fix](#).

Message 3

[WARNING] Input pin '`<pin-name>`' of bypass device '`<inst-name>`' is not observable

Arguments

- Name of the input or output pin, `<pin-name>`
- Bypassed module instance name, `<inst-name>`

Potential Issues

This violation message appears if an input pin of a bypass device is not observable.

Consequences of Not Fixing

Memories that are expected to be by-passed must be connected so that their outputs cannot affect any scan cell inputs and their inputs are observable. If this not the case then scan cells may be capturing unknown data or input logic may be unobservable.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the bypass device, pin on that instance and the scan flip-flop.

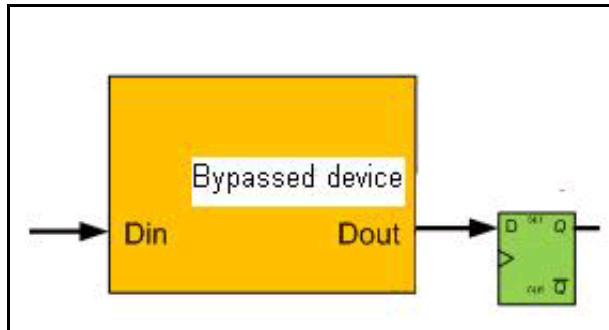
To fix the violation, make sure that the connections of the design and/or the constraints specifications are correct.

Example Code and/or Schematic

Example 1

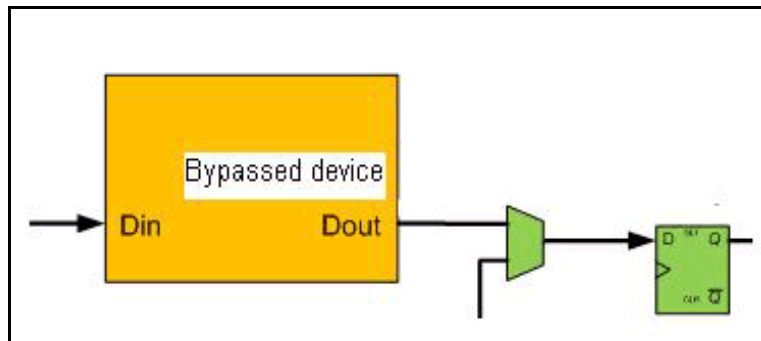
Consider the following figure:

Scan Rules



The Scan_20 rule reports violation for the above example because the memory output is directly connected to a scan flip-flop.

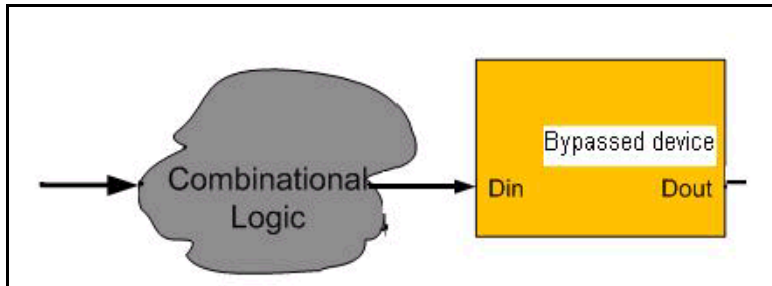
To fix the violation, correct the above design as shown in the following figure:



The Scan_20 rule does not report any violation for the above example because the memory output is not directly connected to a scan flip-flop.

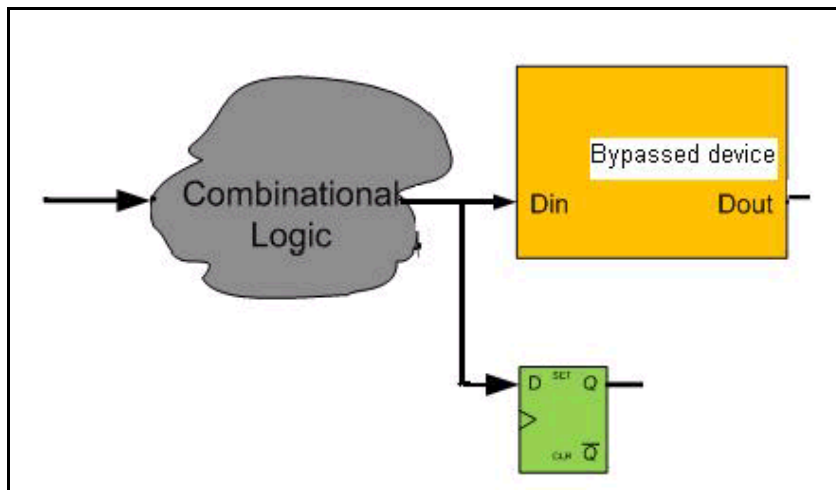
Example 2

Consider the following figure:



The Scan_20 rule reports violation for the above example because the input of the bypassed device is not observable.

To fix the violation, correct the above design as shown in the following figure:



The Scan_20 rule does not report any violation for the above example because the input device of the bypassed device is now observable due to addition of a scan flip-flop to the input port.

Default Severity Label

Warning

Scan Rules

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_21

Keeper, pullUp, and pullDown enables must be controlled by scan flip-flops

When to Use

Use this rule when the isolation of bypassed blocks is required.

Description

The Scan_21 rule reports violation for pull-up/pull-down enables that are not controllable by scan flip-flops.

The Scan_21 rule ensures that the pull-up/pull-down enables are controllable by scan flip-flops.

Method

Simulate capture mode (testmode without the -scanshift argument)

Walk the unblocked combinational fan-in cone of the enable port of any keeper, pullup, or pulldown instance. If any non-scannable flip-flop, floating net, memory module, black box or non-transparent latch is reached, then a message is reported.

Prerequisite

The Scan_21 rule requires that at least one keeper, pullup, or pulldown is specified.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *keeper* (optional): Use this constraint to specify buskeeper design units or nets connected to a (virtual) buskeeper design unit instance so that the Scan_21 rule can take appropriate action.
- *pullup* (optional): Use this constraint to specify the pullup design units so that various rules of SpyGlass DFT ADV product can take appropriate actions.
- *pulldown* (optional): Use this constraint to specify the pulldown design units so that various rules can take appropriate actions.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use to declare the clock pins used as test clocks. Use this constraint's `-testclock` argument for the *Scan_21* rule.
- *scan_type* (optional): Specifies the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

[Capture](#)

Messages and Suggested Fix

Message 1

[WARNING] Enable of Keeper '*<inst-name>*' is driven by *<type>*

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation message, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Enable of Keeper '<inst-name>' is not controlled by scan flip-flop

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation message, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Enable of Pull up '<inst-name>' is floating net

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation message, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] Enable of Pull down '<inst-name>' is floating net

Arguments

- Name of the keeper, pullup, or pulldown instance, <inst-name>
- Type of driver (black-box or non scan flip-flop), <type>

Potential Issues

The violation message appears if the required connectivity is missing or constraints specifications are incorrect.

Consequences of Not Fixing

If you do not fix this violation, controlling the enable pins during testing may be difficult.

Internal pull-ups/pull-downs enable pins should be controlled to prevent floating nets and possible power-to-ground paths.

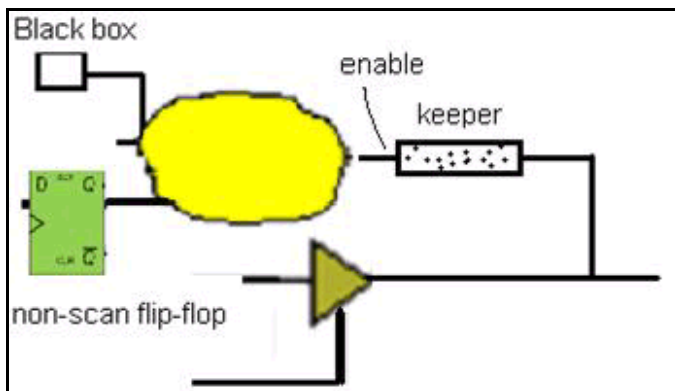
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the keeper, pullUp, or pulldown instance.

To fix the violation, correct the connections of the design or correct the constraint specifications.

Example Code and/or Schematic**Example 1**

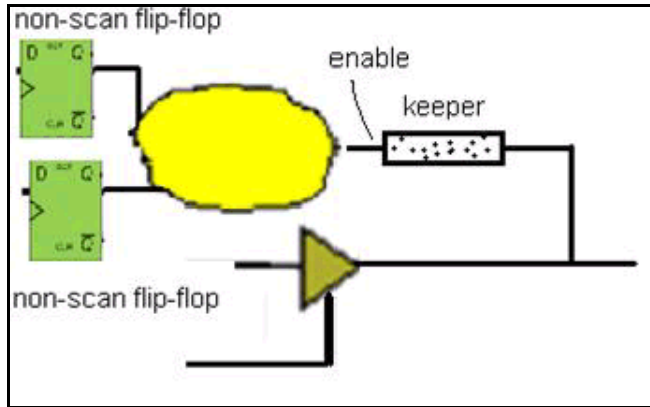
Consider the following figure:



The Scan_21 rule reports a violation for the above example because the enable pin of the keeper is driven by a black box or a non-scan flip-flop.

Example 2

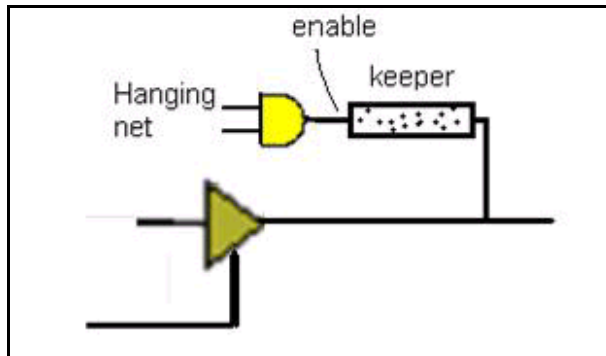
Consider the following figure:



The Scan_21 rule reports a violation for the above example because the enable pin of the keeper is driven by non-scan flip-flops.

Example 3

Consider the following figure:



The Scan_21 rule reports a violation for the above example because the enable pin of the keeper is connected to a hanging-net.

Default Severity Label

Warning

Rule Group

Scan

Scan Rules

Reports and Related Files

No related reports or files.

Scan_22

Ensure that the scan chains have lockup latches at domain crossing.

When to Use

Use this rule to identify the scan chains where lockup latches are not present at clock domain crossings.

Description

The Scan_22 rule flags scan chains where lockup latches are not present at clock domain crossings.

A lockup latch is a latch where the d-pin of the latch has a single unblocked path to a scan flip-flop and the clock of that flip-flop is also connected to the lockup latch so that the latch is enabled (transparent) when the clock is at its off state.

NOTE: *The Scan_22 rule is checked on the post-synthesis gate-level netlist only.*

Prerequisites

Specify the [scan_chain](#) constraint.

Rule Exceptions

Rule will run only if valid scan chains are found.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate all testmode (including -scanshift) and scannable conditions.

Find all chains

for each chain

At each domain change within this chain, there must be a lockup latch. Otherwise, report a message.

end for

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftShowDataInversionInScanChain*: The default value is off. Set the value of the parameter to on to report violation for the first flip-flop from the scanin point that gets inverted data which is shifted from scanin port for valid scan chains.
- *dftStrictTestClkDomains*: The default value is no. Set the value of the parameter to yes to consider the different edges of the same test clock for domain crossing check in a scan chain.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_scan_chain_report_redundant_lockup_latch*: The default value is off. Set the value of the parameter to on to report a warning, if:
 - A lockup latch is found on a scan chain
 - No clock domain crossing identified between the source and the destination flip-flop of the lockup latch.

Constraint(s)

- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *scan_chain* (optional): Use this constraint to specify the scan chains for the Scan_22 rule.
- *balanced_clock* (optional): Use this constraint to specify points within the clock distribution system where all clocks fed from that point are to be considered in the same clock domain.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Domain crossing detected between flip-flops '`<flip-flop1-name>`' [CLK1: `<clk1-name>`] and '`<flip-flop2-name>`' [CLK2: `<clk2-name>`] in scan chain (scanin=`<pin-name>`' scanout=`<pin-name>`') [under defineTag '`<tag-name>`'] without lockup latch

Arguments

- Names of the flip-flops at the domain crossing. `<flip-flop1-name>` and `<flip-flop2-name>`
- Name of the pin in scan chain. `<pin-name>`
- Name of the tag. `<tag-name>`
- Name of the clocks: `<clk1-name>` and `<clk2-name>`

Potential Issues

A violation is reported when lockup latch is missing or wrong clocks are stitched together.

Consequences of Not Fixing

Each clock domain crossing within a scan chain requires a lockup latch to guarantee reliable chain shifting. Absence of lockup latch may corrupt shift-vector

How to Debug and Fix

View the Incremental schematic of the violation message. The Incremental Schematic displays the scan_chain from scan_in point up to domain crossover point in the chain without lock up latch along with the clocks of domain crossover flip-flops. The domain crossover causes the violation.

You can also view the violation for the Info_scanchain rule along with the violation of the Scan_22 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_22 rule and open the Incremental Schematic window.

The violation message for the Info_scanchain rule overlaps the violation message for the Scan_22 rule in the Incremental Schematic window. This

is useful in debugging the violation for the Scan_22 rule.

To fix the violation, insert a lock-up latch.

Message 2

[WARNING] Redundancy check ignored for lockup latch '`<lockup latch name>`' because it is at the '`<beginning | end>`' of the `schain(scanin=> '<scanin node name>' scanout=> '<scanout node name>') [under <mode name>]`

Potential Issues

Redundancy check can only be performed on a lockup latch, if it is found between two scan-flip-flops. If the lockup latch is present at the beginning or at the end of a scan chain, the check is not performed for that lockup latch.

Consequences of Not Fixing

Unnecessary lockup latches may be present at the beginning or at the end of the scan chain. Extra logic on the scan chain will impact chain length / power / area.

How to Debug and Fix

Review whether the lockup latch is needed at the beginning or end of the scan chain.

Message 3

[WARNING] Lockup latch '`<lockup latch name>`' is `<driving or driven by>` another lockup latch '`<another lockup latch name>`' on the `schain(scanin=> '<scanin node name>' scanout=> '<scanout node name>') [under <mode name>]`

Potential Issues

Redundancy check can only be performed on a lockup latch, if it is found between two scan-flip-flops. If the lockup latch has another lockup latch, either in fanin or in fanout, the check will not happen for that lockup latch.

Consequences of Not Fixing

Unnecessary lockup latches may be present at the beginning or at the end of the scan chain. Extra logic on the scan chain will impact chain length / power / area.

How to Debug and Fix

Cascaded lockup latches are not needed. Review and remove the

redundant lockup latches.

Message 4

[WARNING] Lockup latch '<lockup latch name>', without clock domain crossing, found on the scanchain(scanin=>'<scanin node name>' scanout=>'<scanout node name>')[under mode name]

Potential Issues

Unnecessary lockup latches may be present on the scan chain.

Consequences of Not Fixing

Extra logic on the scan chain will impact chain length/power/area.

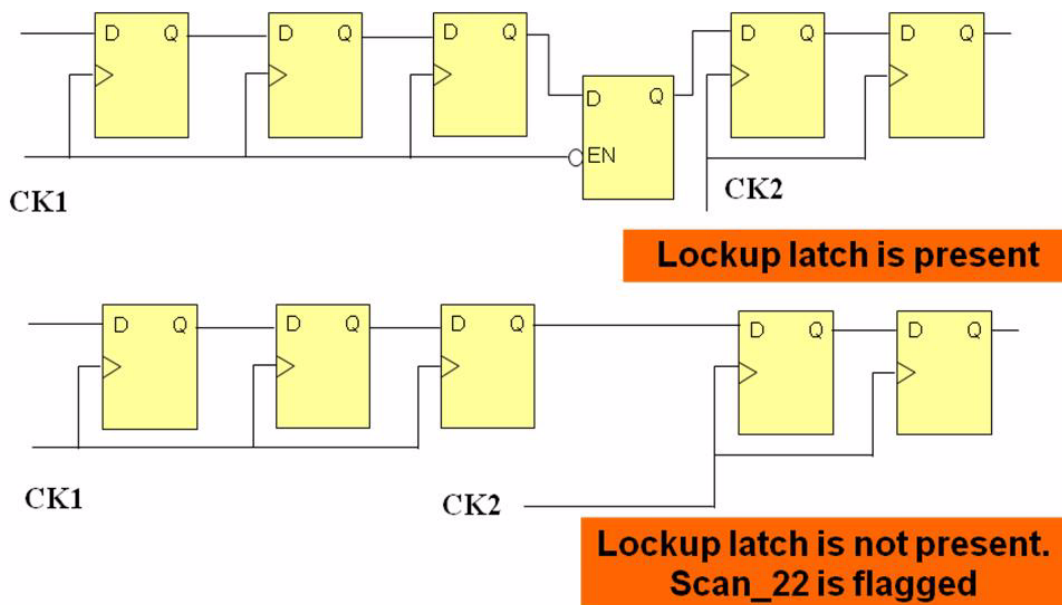
How to Debug and Fix

Remove redundant lockup latches without loss of functionality.

Example Code and/or Schematic

Example 1

Consider the following example:



Example 2

The following figure illustrates the scan chain connection from u1 to u2 crossing clock domains:

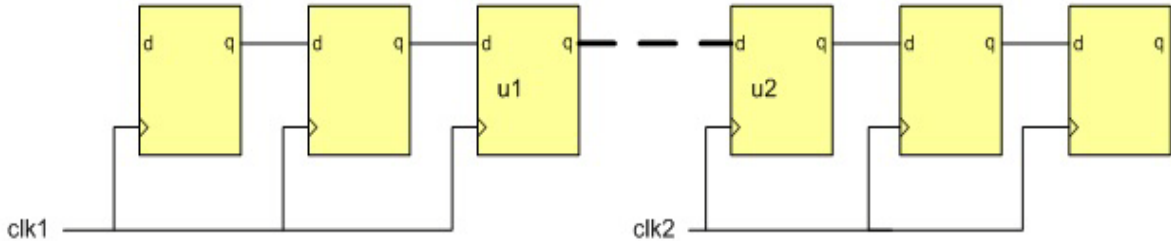


FIGURE 91. Scan chains with clock domain crossing

In the above example, if clk1 and clk2 are controlled by the same testclock during the shift mode, there is no guarantee regarding the clock timing in order to achieve reliable shifting. If u1 shifts sufficiently before u2 is clocked, then the state of u2 is always be the same as the state of u1. In such a case, all tests that depended on the state of the flip-flops after u1 would fail and all states captured in flip-flops prior to u1 would be compromised.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_23

Ensure that the module bypass conditions are satisfied.

When to Use

Use this rule to check the bypass condition for required blocks.

Description

The Scan_23 rule reports violation for module instances that are not in their bypass mode during capture.

The Scan_23 rule reports any module with the *module_bypass* constraint that is not in its by-pass state.

NOTE: *The Scan_23 rule requires the specified module to be a black box; it skips synthesizable modules.*

Method

Simulate power, ground and any available testmode capture. If the by-pass conditions for any module with module_bypass constraint are not satisfied, then report a message.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *module_bypass* (mandatory): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Instance '<inst-name>' not in by-pass mode (expected value on pin '<pin-name>' =<exp-val>, actual value=<act-val>) as specified in module_bypass constraint

Arguments

- Name of module instance, <inst-name>
- Name of Bypass Pin, <pin-name>
- Expected testmode value (1 or 0), <exp-val>
- Actual testmode value (1 or 0), <act-val>

Potential Issues

The violation message appears, if the connections or the constraint specifications are incorrect.

Consequences of Not Fixing

Not fixing this violation results in reduced controllability and observability of instances and further reduces coverage ability of the design.

Modules designed with a data-in to data-out bypass mode reduce module shadows without the necessity of scan registers.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the bypassed design unit and its bypass pin with its expected value

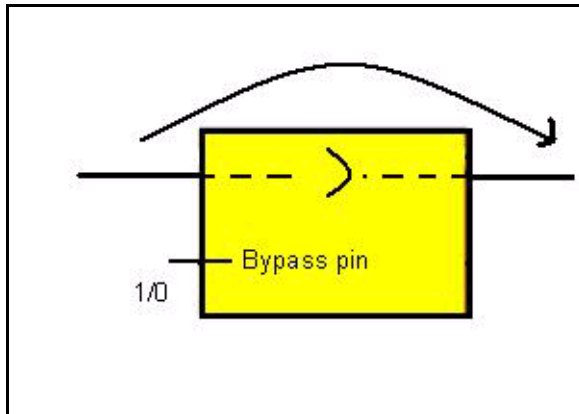
You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Scan_23 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_23 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Scan_23 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_23 rule.

To fix the violation, rectify the connections of the design and/or correct the constraint specifications.

Example Code and/or Schematic

Consider the following figure:



The Scan_23 rule reports a violation for the above example because the bypass pin is not enabled.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_24

Ensure that all flip-flops are part of some scan chain.

When to Use

Use this rule to detect flip-flops which are not part on any scan chain.

Description

The Scan_24 rule reports violation for flip-flops that are not part of any scan chain under specified simulation conditions.

The Scan_24 rule also dumps the following information about the flip-flops not present in scan chain in a tabular format.

- Cell_no
- Clock
- Clock_polarity
- Type (scannable or non-scannable)
- hier_name
- lib_cell_name

Rule Exception

The *Scan_24* rule does not report violation for the no-scan flip-flops (forced or inferred).

Default Weight

10

Language

Verilog, VHDL

Method

Simulate all testmode (including -scanshift) and scanenable conditions.

Find all chains

Flag any flip-flop not part of any chain

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *force_scan* (optional): Use this constraint to declare flip-flops as scannable even if they do not so qualify.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- *scan_chain* (optional): Use this constraint to specify the scan chains for the Scan_24 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] None of the flip-flops (flip-flop count, excluding no_scan: <num>) of design '<du-name>' are present on any scan-chain

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported when a scan chain constraint is missing.

Consequences of Not Fixing

Not fixing the violation may result in noise or incorrect calculation.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[INFO] Scan chain report file '<file-name>' is generated

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] None of the flip-flops (flip-flop count, excluding no_scan: <num>) of design '<du-name>' are part of valid scan-chains

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported when a scan chain constraint is missing.

Consequences of Not Fixing

Not fixing the violation may result in noise or incorrect calculation.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] '<num1>' of total '<num2>' flip-flops (excluding no_scan), of design '<duname>' are not part of any valid scan chain

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported when either flip-flops are not on chain path or due to wrong design connectivity.

Consequences of Not Fixing

Not fixing the violation may result in lower coverage.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 5

[INFO] All flip-flops (flip-flop count, excluding no_scan:<num>) of design '<duname>' are part of valid scan chains

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 6

[INFO] Generated XML report for Scan_24

Arguments

- Number of flip-flops that are not part of any valid scan chain, <num>

- Name of the design unit, <du-name>
- Name of the generated report file, <file-name>
- Number of affected flip-flops, <num1>
- Total number of flip-flops, <num2>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this message.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit impact of this message.

How to Debug and Fix

The Scan_24 rule is an informative rule and requires no debug. The Scan_24 rule generates the [scan_chain](#) report for the flip-flops, which are not part of any scan chain.

You can also view the violation for the Info_schain rule along with the violation of the Scan_24 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_24 rule and open the Incremental Schematic window.

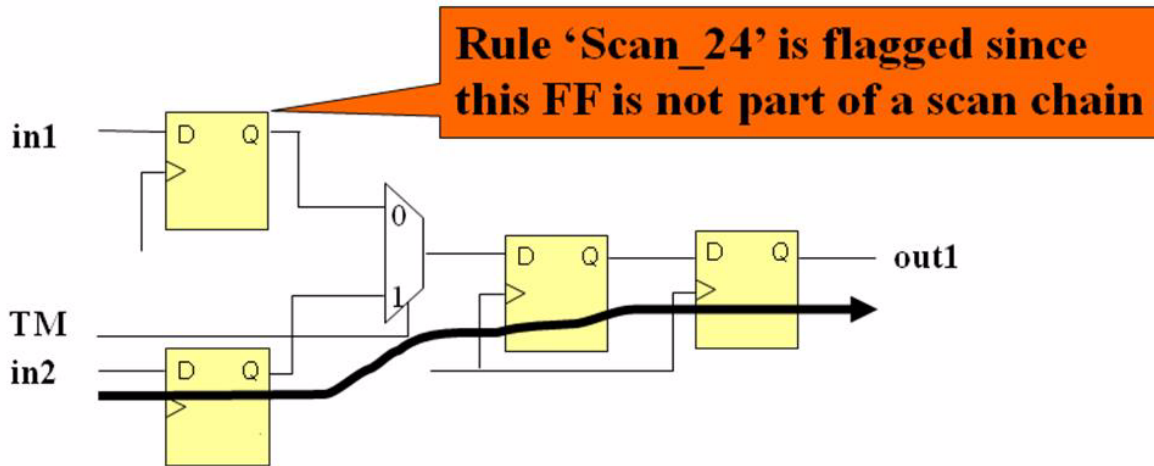
The violation message for the Info_schain rule overlaps the violation message for the Scan_24 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_24 rule.

To fix the violation, see Example Code and/or Schematic section.

Example Code and/or Schematic

Consider the following example:

```
define_tag -tag s1 -name top.TM -value 1  
scanchain -scanin in1 -scanout out1 -scanenable s1
```



Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

[scan_chain](#) report

Scan_25

Ensure that the scan chains do not contain inverters in scan path.

When to Use

Use this rule to identify a scanin/scanout port or a flip-flop containing inverters.

Rule Description

The Scan_25 rule flags inverters in scan chains under specified simulation conditions.

Prerequisites

Specify scan_chain constraint.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate all testmode (including -scanshift) and scanenable conditions.

Find all chains

Flag any inverters in a chain

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *scan_chain* (optional): Use this constraint to specify the scan chains for the Scan_25 rule.

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

The following violation message is displayed for the Scan_25 rule:

```
[WARNING] Inverters found in scan chain (scanin=>'<port1-name>' scanout=>'<port2-name>') [under defineTag '<tag-name>']
```

Arguments

- Name of the scanin port. <port1-name>
- Name of the scanout port. <port2-name>
- Simulation condition. <tag-name>

Potential Issues

A violation is reported due to wrong buffer insertion.

Consequences of Not Fixing

Inverters in scan paths can alter the vector being shifted through the scan chain.

How to Debug and Fix

View the Incremental schematic of the violation message. The Incremental Schematic displays the scan_in and scan_out and inverter between scan_in and scan_out point.

You can also view the violation for the Info_schain rule along with the violation of the Scan_25 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_25 rule and open the Incremental Schematic window.

The violation message for the Info_schain rule overlaps the violation message for the Scan_25 rule in the Incremental Schematic window. This

Scan Rules

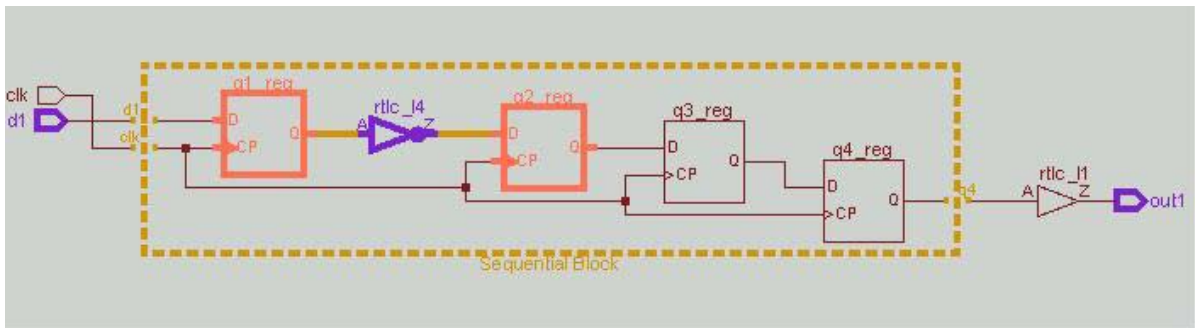
is useful in debugging the violation for the Scan_25 rule.

To fix the violation, see Example Code and/or Schematic section.

Example Code and/or Schematic

Consider the following example:

```
current_design test
clock -name clk -testclock
scanchain -scanin d1 -scanout out1
```



Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

No related reports or files.

Scan_26

Ensure that the scan chains contain lockup latch at chain end.

When to Use

Use this rule to identify scan chains that do not have a lockup latch at chain end.

Description

The Scan_26 rule reports violation for scan chains that do not have a lockup latch at chain end under specified simulation conditions.

A lockup latch is a latch where the d-pin of the latch has a single unblocked path to a scan flip-flop and the clock of that flip-flop is also connected to the lockup latch so that the latch is enabled (transparent) when the clock is at its off state.

Prerequisites

Specify the scan chain constraint.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate all testmode (including -scanshift) and scanenable conditions.

Find all chains

Flag any chain that does not have a lockup latch at chain end.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *define_tag* (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *scan_chain* (optional): Use this constraint to specify the scan chains for the Scan_26 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).

Operating Mode

Scanshift

Messages and Suggested Fix

The following violation message is displayed for the Scan_26 rule:

```
[WARNING] Scan chain(scanin=>' <port1-name>
'scanout=>' <port2-name>') [under defineTag ' <tag-name>' ] does
not have lockup latch at chain end
```

Arguments

- Name of the scanin port. <port1-name>
- Name of the scanout port. <port2-name>
- Simulation condition. <tag-name>

Potential Issues

A violation is reported due to missing lockup latch.

Consequences of Not Fixing

When used in SoC flow, may give rise to Scan_22 issues.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic displays the scan_in and scan_out points.

You can also view the violation for the Info_scanchain rule along with the

violation of the Scan_26 rule in the Incremental Schematic window. To do this, double-click the violation for the Scan_26 rule and open the Incremental Schematic window.

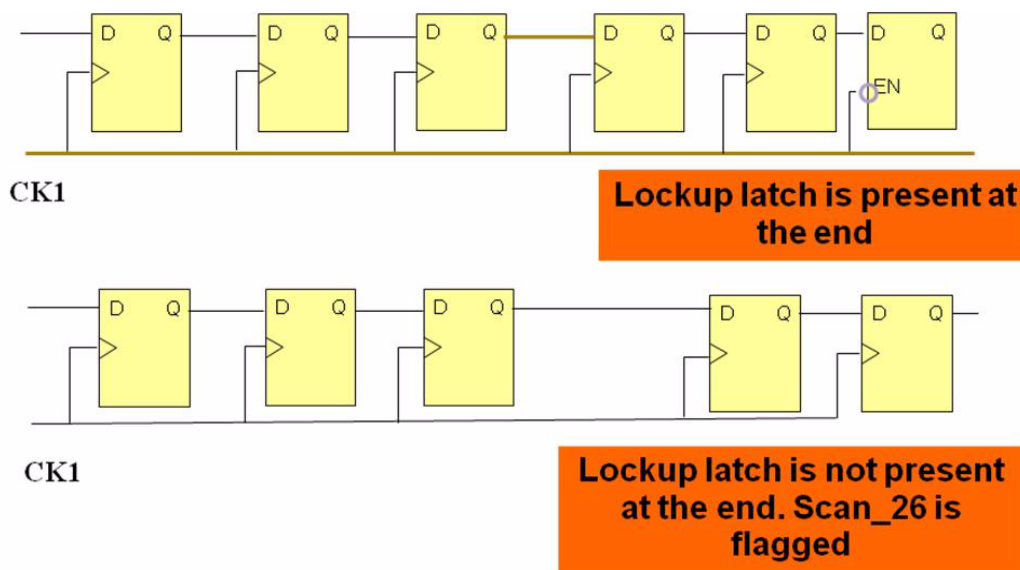
The violation message for the Info_scanchain rule overlaps the violation message for the Scan_26 rule in the Incremental Schematic window. This is useful in debugging the violation for the Scan_26 rule.

Probe the fan-in of scan_out point up to the first flip-flop from scan_out point. While probing the rule does not consider any lockup latch, which causes the violation.

To fix the violation, see Example Code and/or Schematic section.

Example Code and/or Schematic

Consider the following example:



Default Severity Label

Warning

Rule Group

Scan Rules

Scan Rules

Reports and Related Files

No related reports or files.

Scan_27

Validate the connectivity of scan data input pin of a scan flip-flop

When to Use

Use this rule to check connectivity of scan data input pin of a scan flip-flop.

Description

The Scan_27 rule reports violation for data input pins of scan flip-flop that are not connected as specified using the `dftScanDataConnectivityCheck` rule parameter during shift.

The Scan_27 rule works for both unstitched and stitched scan replaced netlists.

Method:

For each instance, if the instance is a flip-flop:

Find the scanin term list.

If the value TIED_0 is selected, check the value at each scan input term and flag a violation if the value is not 0.

If the value TIED_1 is selected, check the value at each scan input term and flag a violation if the value is not 1.

If the value FEEDBACK_DIRECT is selected, check for the direct feedback path and flag a violation if the path is not found.

If the value FEEDBACK_THRU_BUF_INV is selected, check for the direct feedback path/path with buffers and inverters and flag a violation if the path is not found.

If the value STICHED_CHAIN is selected, check whether the scan input of the scan flip-flop is connected to the output of another flip-flop, a lockup latch, or a primary input pin. If not, then flag a violation.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

- *dftScanDataConnectivityCheck*: The default value is TIED_0. Set the value of this parameter to one of the following values to specify the type of connectivity checking to be performed on the scan data input pin:
 - TIED_1
 - FEEDBACK_DIRECT
 - FEEDBACK_THRU_BUF_INV
 - STICHED_CHAIN
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Scan input pin <pin-name> of instance <inst-name> is tied to '<sim-val>' (required '0')

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation, click [Potential Issues](#).

Consequences of Not Fixing

The required values is not met (0 is expected but 1 is received).

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] Scan input pin <pin-name> of instance <inst-name> is tied to '<sim-val>' (required '1')

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation, click [Potential Issues](#).

Consequences of Not Fixing

If you do not fix this violation, the required values is not met (1 is expected but 0 is received).

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Scan input pin <in-pin-name> of instance <inst-name> does not have any direct feedback path from <Q or QN : <out-pin-name> >

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation, click [Potential Issues](#).

Consequences of Not Fixing

If you do not fix this violation, the design requirement is not met because feedback to the input is absent.

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 4

[WARNING] Scan input pin <in-pin-name> of instance <inst-name> does not contain a direct feedback path or a feedback path, with buffers and inverters, from <Q or QN : <out-pin-name> >

Arguments

To view arguments description, click [Arguments](#).

Potential Issues

To know more about potential issues pertaining to the violation, click [Potential Issues](#).

Consequences of Not Fixing

If you do not fix this violation, the feedback to the input with inverter and/or buffer is absent and design requirement is not met.

How to Debug and Fix

To debug and fix the violation, click [How to Debug and Fix](#).

Message 5

[WARNING] Scan input pin <pin-name> of instance <inst-name> in scan chain is not sourced from output of another scan flop or from a LockUp Latch or from a primary input

Arguments

- Name of the scan input pin, <pin_name>
- Name of the instantiated scan flip-flop, <inst_name>

Potential Issues

This violation message appears due to incorrect connection and/or incorrect constraint specification that results in fixed incorrect value reaching the input.

Consequences of Not Fixing

If you do not fix this violation, the input of scanned flip-flop is not sourced from the primary output pin of another flip-flop and design requirement is not met.

How to Debug and Fix

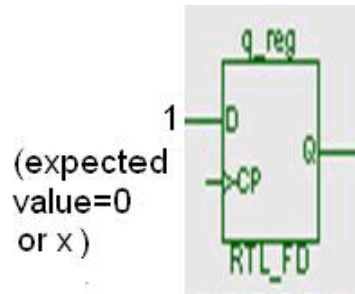
Double-click the violation message. The Incremental Schematic window highlights the instantiated scan flip-flop and the scan input pin.

To fix the violation, ensure connectivity and constraint specifications are correct.

Example Code and/or Schematic

Example 1

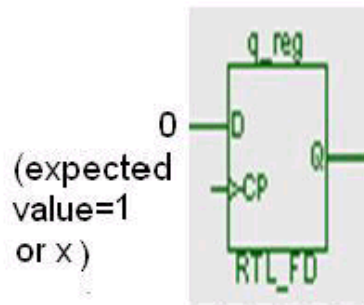
Consider the following figure:



The above figure demonstrates *Message 1*.

Example 2

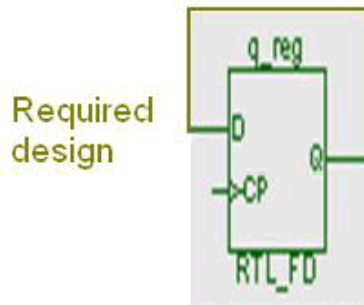
Consider the following figure:



The above figure demonstrates *Message 2*.

Example 3

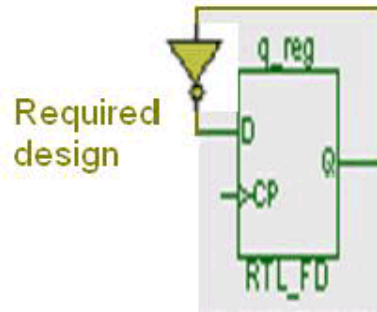
Consider the following figure:



The above figure demonstrates the required design in [Message 3](#). The Scan_27 rule reports a violation, if the feedback is absent.

Example 4

Consider the following figure:

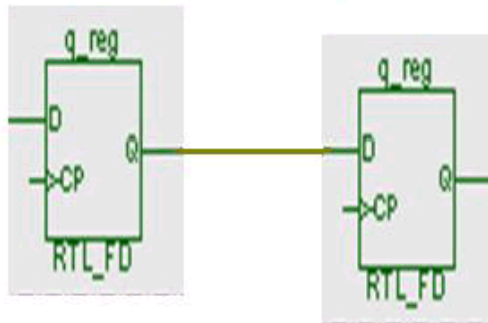


The above figure demonstrates the required design in [Message 4](#). The Scan_27 rule reports a violation, if the inverter is missing.

Example 5

Consider the following figure:

Required Design



The above figure demonstrates the required design in [Message 5](#). The Scan_27 rule reports a violation, if the input of the second flip-flop is not driven by a flip-flop.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_28

Validate the desired value at scan enable pin of scan flip-flop

When to Use

Use this rule to check if a scan chain is formed.

Description

The Scan_28 rule reports violation for scan enable pins when the state (active/inactive) of the scan enable pin of a scan flip-flop is not the same as that specified using the [dftScanEnablePinValue](#) rule parameter.

Method

For each instance, if the instance is a flip-flop:

Get the Enable pin list and the connected net for each enable pin.

Calculate the expected value reaching the scan enable pin.

Flag a violation if the value is INACTIVE and the simulation value on the connected pin signifies that the value is active.

Flag a violation if the value is ACTIVE and the simulation value on the connected pin signifies that the value is inactive.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftScanEnablePinValue](#): The default value is inactive. Set the value of the parameter to active to set the scan enable pin of a scan cell as active.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Scan enable pin '<pin-name>' of scan flip-flop instance '<inst-name>' is not tied to its <value 1> logic value. Current logic value at the Scan enable pin is '<value 2>'

Arguments

- Name of the scan enable pin, <pin-name>
- Name of the instance, <inst-name>
- Expected value at the scan enable pin, <value 1> which can be active or inactive.
- Value reaching the scan enable pin, <value 2>

Potential Issues

The violation message appears if an unexpected value reaches the scan enable pin of the scan flip-flop.

Consequences of Not Fixing

Not fixing this violation creates confusion about the existence of a scan chain. It might appear to you that scan chain is present but it may or may not exist.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the following:

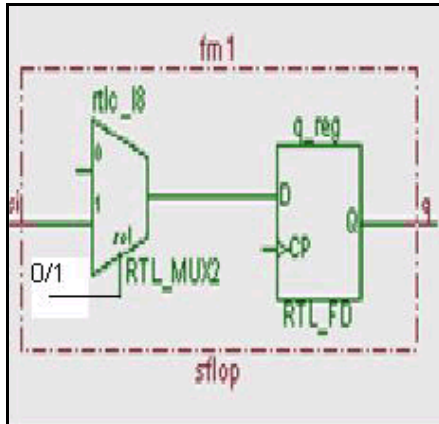
- Instantiated scan flip-flop
- Scan enable pin with the simulation value

To fix this violation, put the correct simulation value on the enable pin.

Example Code and/or Schematic

Consider the following figure:

Scan Rules



In the above figure, the highlighted enable pin of the multiplexer is not tied to the desired value. This causes the violation message to appear.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_29

Ensure that the scan chains do not have inversions in scan path

When to Use

Use this rule to check and report possible inversions in scan chains.

Rule Description

The Scan_29 rule reports violation for inversions in scan chains under specified simulation conditions.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *define_tag* (mandatory): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node
- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_29 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Path between <obj1-name> and <obj2-name> inverts the data for scan chain (scanin=>'<scanin-pin>' scanout=>'<scanout-pin>') [under <cond1>]

Arguments

- <obj1-name> and <obj2-name> can be two flip-flops, a scanin port or a flip-flop, or a flip-flop or a scanout port.
- <scanin-pin> and <scanout-pin> are the input and output pin of the scan chain, respectively.
- Scan chain enabling condition, <cond1>

Potential Issues

The violation message appears, if there is an inversion in the scan path.

Consequences of Not Fixing

Not fixing this violation makes analysis of scan vector more complicated.

How to Debug and Fix

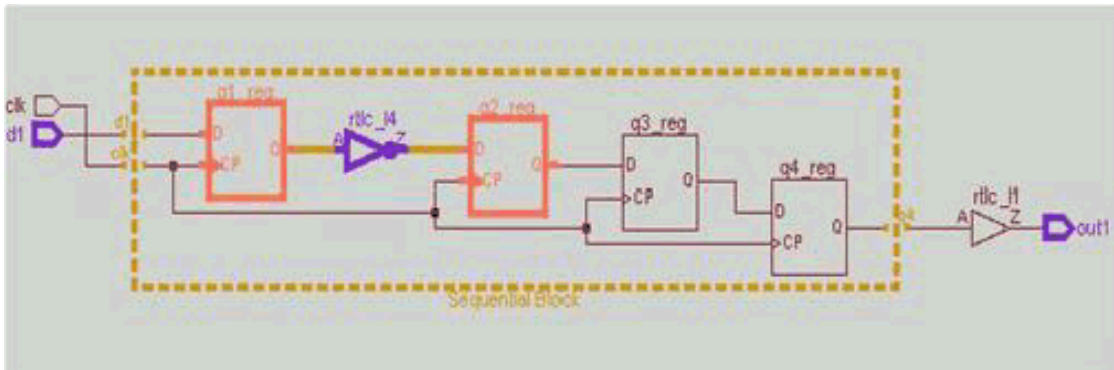
Double-click the violation message. The Incremental Schematic window highlights the path between the following:

- Two flip-flops
- Scanin port and flip-flop
- Fop and scanout port

To fix this violation, remove the inverters from the scan chain.

Example Code and/or Schematic

Consider the following figure:



In the above example, the Scan_29 rule reports violation because the scan chain has an inverter in the design.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports and files.

Scan_30

Reports observable flip-flop list

When to Use

Use this rule to know about observable flip-flops in the design.

Description

The Scan_30 rule reports an observable flip-flop list in a text report. The information is generated in the *observable_ffs* report.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] List of observable flip-flop is dumped in file <file-name> for top <name>

Arguments

- Name of the report, <file-name>
- Name of the top design unit, <name>

Potential Issues

The violation message appears if an observable flip-flop list is found for the top design unit.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

No debug or fix is required for the violation message.

Example Code and/or Schematic

Not Applicable

Default Severity Label

Info

Rule Group

Scan

Reports and Related Files

[observable_ffs](#): This report lists all observable flip-flops.

Scan_31

Reports flip-flops with constant value data pin.

When to Use

Use this rule to identify flip-flops with constant value data pin.

Description

The Scan_31 rule reports violation for flip-flops with constant value data pin.

Method

Simulate test mode under capture condition.

For each instance, if the instance is a flip-flop:

Flag a violation if a flip-flop has constant value at a terminal.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

[Capture](#)

Messages and Suggested Fix

[WARNING] Flip-flop "<flip-flop-name>" has a fixed input value of "<value1>" (Root Cause: "<value2>")

Arguments

- Name of the flip-flop with constant value data pin, <flip-flop-name>
- Value of the flip-flop, <value1>
- Root Cause, <value2>

Potential Issues

A violation is reported when a flip-flop contains a tied D-pin.

Consequences of Not Fixing

Not fixing the violation prevents observation of logic feeding D-pin as it is fixed to a value.

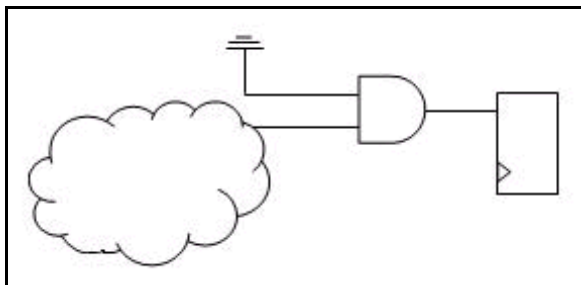
How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the flip-flops with constant value data pin.

To fix the violation, remove the tied condition.

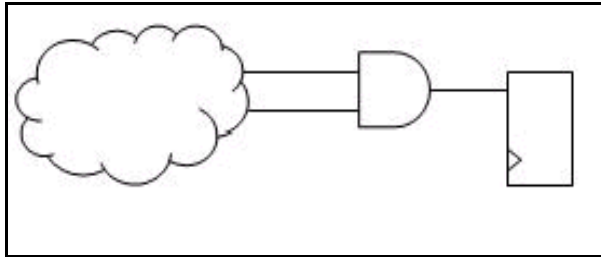
Example Code and/or Schematic

Consider the following figure:



The above figure illustrates a bad design scenario because the flip-flop has a tied data pin or a constant value data pin.

Now, consider the following figure:



The above figure illustrates a good design scenario because the flip-flop does not have a tied or constant data value pin.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_32

Reports scanout ports that are not output ports during scan mode

When to Use

Use this rule to identify all scanout ports that are not output ports during scan shift operation.

Description

The Scan_32 rule reports violation for scanout ports that are not outputs during scan mode.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_32 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Scan_32 rule uses the -testclock argument of the clock constraint.

Operating Mode

Scanshift

NOTE: *In scan mode, all scan output ports must be in output mode during scan shift operation to allow getting test values.*

Messages and Suggested Fix

[WARNING] Scanout port '<port-name>' is not acting as output only.

Arguments

Name of the port, <port-name>

Potential Issues

The violation message appears, if a scanout port does not act as output port in the scan mode.

Consequences of Not Fixing

Not fixing this violation may result in problems while controlling the design.

How to Debug and Fix

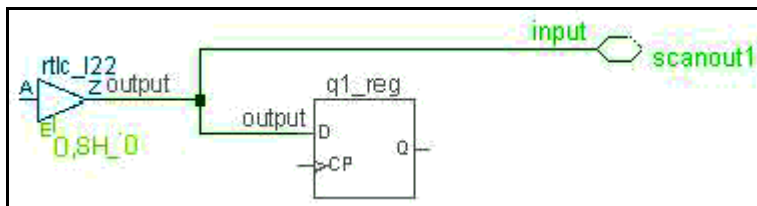
Double-click the violation message. The Incremental Schematic window highlights the following:

- Scanout port
- Path to tristate buffer with inactive enable.

To fix this violation, ensure that all scanout ports are output ports during scan mode.

Example Code and/or Schematic

Consider the following figure:



The above example the Scan_32 rule reports a violation because the design scanout1 port acts as an input port and not as an output port.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_33

Reports scanin ports that are not inputs during scan mode

When to Use

Use this rule to identify all scanin ports that are not input ports during scan mode.

Description

The Scan_33 rule reports violation for scanin ports that are not inputs during scan mode.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [scan_chain](#) (mandatory): Use this constraint to specify the scan chains for the Scan_33 rule.
- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [clock](#) (optional): Use this constraint to define the clocks of a design. The Scan_33 rule uses the `-testclock` argument of the clock constraint.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Scanin port '<port-name>' is not acting as input only

Arguments

Name of the port, <port-name>

Potential Issues

The violation message appears, if a scanin port does not act as an input port in the scan mode.

Consequences of Not Fixing

Not fixing this violation may result in problems while controlling the design.

In scan mode, all scan input ports must be in input mode during scan shift operation to allow putting test values.

How to Debug and Fix

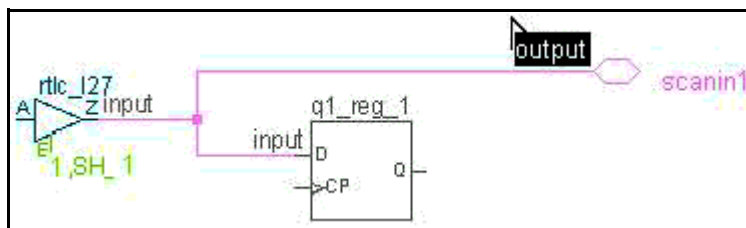
Double-click the violation message. The Incremental Schematic window highlights the following:

- Scanout port
- Path to tristate buffer with inactive enable

To fix the violation, ensure that all the scanin ports work as input ports during scan mode.

Example Code and/or Schematic

Consider the following figure:



In the above example, the Scan_33 rule reports a violation because the scanin1 port works as an output port and not as an input port.

Scan Rules

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_34

Reports top test signals not named as per dedicated parameters

When to Use

Use this rule when you want to follow strict naming convention on test signals.

Description

The Scan_34 rule ensures that names of scanin, scanout and scanenable ports of scan modules should follow standard naming conventions.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftSENames*: The default value is none. Set the value of the parameter to scanenable port name to specify the standard convention of scannable ports of scan modules. You can also specify a space-separated list of names.
- *dftSINames*: The default value is none. Set the value of the parameter to scanin port name to specify the standard naming convention of scanin ports of scan modules. You can also specify a space-separated list of names.
- *dftSONames*: The default value is none. Set the value of the parameter to scanout port name to specify the standard naming convention of scanout ports of scan modules.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_34 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Scan_34 rule uses the `-testclock` argument of the clock constraint.

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING] <port-type> port '<port-name>' of module '<mod-name>' does not follow allowed naming convention allowed names '<cor-name>'

Arguments

- Port type can be Scanin, Scanout, or Scanenable, <port-type>
- Name of the port, <port-name>
- Suggested name as per the standards, <cor-name>

Potential Issues

The violation message appears, if name of a port of a module does not follow standard convention.

Consequences of Not Fixing

Not fixing this violation may result in reduced readability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the violating module.

To fix this violation, name test signals according to the convention.

Message 2

[INFO] Scan_34 runs only on pure netlist design, rule checking

skipped for rtl design <design-name>

Potential Issues

The information message appears when rule checking is skipped because the design is a non-netlist design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_35

Ensure that the scan chains do not exceed maximum specified length

When to Use

Use this rule to avoid scan chains with more than specified length.

Description

The Scan_35 rule checks that the length of a scan chain (number of scan flip-flops) should not exceed the maximum length specified with the [*dftMaxScanChainLength*](#) parameter.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [*dftMaxScanChainLength*](#): The default value is 600. Set the value of the parameter to an integer value to specify the maximum allowed length for scan chains.
- [*dftUseOffStateOfClockInClockPropagation*](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- [*scan_chain*](#) (mandatory): Use this constraint to specify the scan chains for the Scan_35 rule.
- [*define_tag*](#) (mandatory): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- [*test_mode*](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (optional): Use this constraint to define the clocks of a design. The Scan_35 rule uses the `-testclock` argument of the clock constraint.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Scan chain (scanin=>'<start-node>' scanout=>'<end-node>') [under defineTag '<sim-cond>'] has exceeded the maximum length [Current length (<cur-len>) > Max length (<max-len>)]

Arguments

- Starting node of the scan chain, <start-node>
- Ending node of the scan chain, <end-node>
- Simulation condition, <sim-cond>
- Number of scan flip-flops in the scan chain, <cur-len>
- Allowed limit specified using `dftMaxScanChainLength` parameter, <max-len>

Potential Issues

The violation message appears, if length of a scan chain is more than the maximum allowed length.

Consequences of Not Fixing

Not fixing this violation may result in increased test cycle time.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the scan chain, with the flip-flops exceeding the limit in a different color.

To fix this violation split the scan chain into smaller chains.

Message 2

[INFO] Scan_35 runs only on pure netlist design, rule checking skipped for rtl design <design-name>

Potential Issues

The information message appears when rule checking is skipped because the design is a non-netlist design.

Consequences of Not Fixing

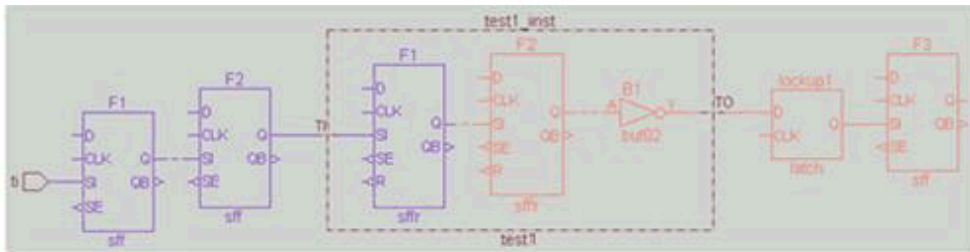
This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Consider the following figure:



Scan_35[1]: Scan chains should not exceed maximum specified length
Scan chain (scanin=>'test.ti' scanout=>'test.out') [under defineTag 'testmode'] has exceeded the maximum length [Current length (5) > Max length (3)], test.v, 3

In the above example, the Scan_35 rule reports scan chain with more than specified length.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports or files.

Scan_36

Ensure that the scan chains are balanced in length, that is, they should not deviate more than allowed from the expected length

When to Use

Use this rule to have all scan chains of balanced length.

Description

The Scan_36 rule checks that the scan chains are balanced, that is the length of a scan chain, that is, the number of scan flip-flops, should not differ from the expected length by more than the allowed deviation. The expected length is specified with the [dftExpectedScanChainLength](#) parameter. The allowed deviation is specified using the [dftAllowedScanChainLengthDeviation](#) parameter.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftAllowedScanChainLengthDeviation](#): The default value is 20. Set the value of the parameter to an integer to specify the maximum allowed deviation from the expected length for scan chains.
- [dftExpectedScanChainLength](#): The default value is 100. Set the value of the parameter to specify the maximum allowed deviation from the expected length for scan chains.
- [dftMaxScanChainLength](#): The default value is 600. Set the value of the parameter to an integer to specify the maximum allowed length for scan chains.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_36 rule.
- *define_tag* (mandatory): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Scan_36 rule uses the `-testclock` argument of the clock constraint.

Operating Mode

Scanshift

Messages and Suggested Fix

Message 1

[WARNING] Scan chain (scanin=>'<start_node>' scanout=>'<end_node>') [under defineTag '<sim_cond>'] deviates from the expected length (Current length <cur_len>, Allowed length <exp_len> +/- <len_dev>)

Arguments

To view list of arguments, click [Arguments](#).

Potential Issues

The violation message appears, if the length of scan chain exceeds the maximum allowed length.

Consequences of Not Fixing

Not fixing this violation may result in problems in simultaneous testing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the violating scan chain.

To fix this violation, replace the scan chains of unexpected length.

Message 2

[WARNING] Expected maxScanChainLength should always be in the range specified by expected mean scanchain length and its deviation (current maxScanChainLength is <chain_length>, expectedMeanScanChainLength is <exp_len>, and expectedMeanScanChainLengthDeviation is <len_dev>)

Arguments

- Starting node of the scan chain, <start_node>
- Ending node of the scan chain, <end_node>
- Simulation condition, <sim_cond>
- Number of scan flip-flops in the scan chain, <cur_len>
- Expected mean length specified using dftExpectedScanChainLength parameter, <exp_len>
- Allowed deviation limit specified using dftAllowedScanChainLengthDeviation parameter, <len_dev>
- max limit specified using dftMaxScanChainLength parameter, <chain_length>

Potential Issues

The violation message appears, if the length of a scan chain is not in the specified range.

Consequences of Not Fixing

Not fixing this violation may result in problems in simultaneous testing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the violating scan chain.

To fix this violation replace the scan chains of unexpected length.

Message 3

[INFO] Scan_36 runs only on pure netlist design, rule checking skipped for rtl design <design-name>

Potential Issues

The information message appears when rule checking is skipped because the design is a non-netlist design.

Consequences of Not Fixing

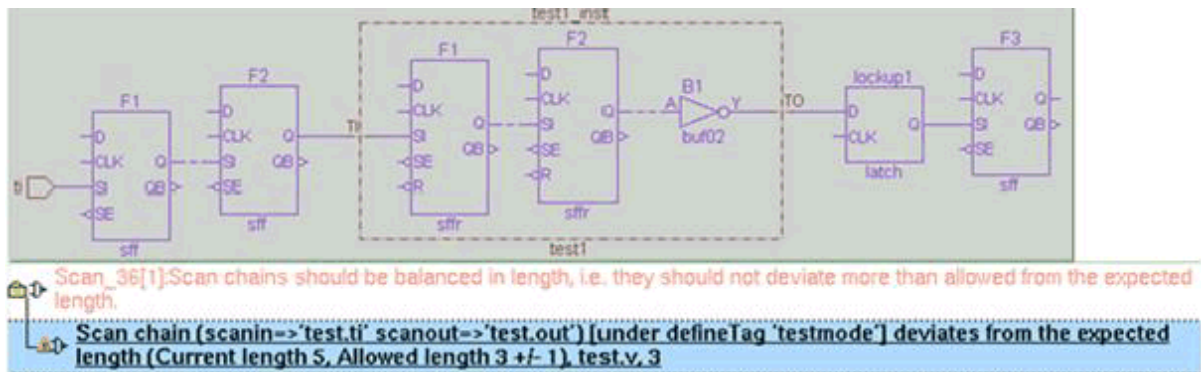
This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Consider the following figure:



The above figure highlights the scan chain, which deviates by more than expected value.

Default Severity Label

Warning

Rule Group

Scan

Reports and Related Files

No related reports and files.

Scan_38

Ensure that the scan chains start with a positive edge flip-flop

When to Use

Use this rule at the IP level, when IPs are instantiated at the SoC, to ensure easier integration of scan chains with the different blocks.

Description

This rule checks that the scan chains start with a positive edge flip-flop so that the shift operation happens on the same clock edge.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_38 rule
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

The Scan_38 rule reports the following violation message:

```
[WARNING] Scan chain (scanin=>'<scanin_signame>',
scanout=>'<scanout_signame>', condition=>'<condition>') with
'<number>' scan cells, starts with a flip-flop
('<flip_flop_name>') which triggers at '<clockedge_type>' edge of
```

```
test clock '<testclk_name>'
```

Arguments

- Scan-in signal name, <scanin_signame>
- Scan-out signal name, <scanout_signame>
- Scan chain stitching condition, <condition>
- Number of scan cells on the scan chain, <number>
- Name of the flip-flop, which is not on the positive (negative for rto) edge of clock, <flip_flop_name>
- Negative for rtz and positive for rto, <clkedge_type>
- Name of the test clock, <testclk_name>

Potential Issues

A violation is displayed because of the incorrect number of inversion on the clock path or incorrect clock polarity

Consequences of Not Fixing

Not fixing the violation results in loss of data.

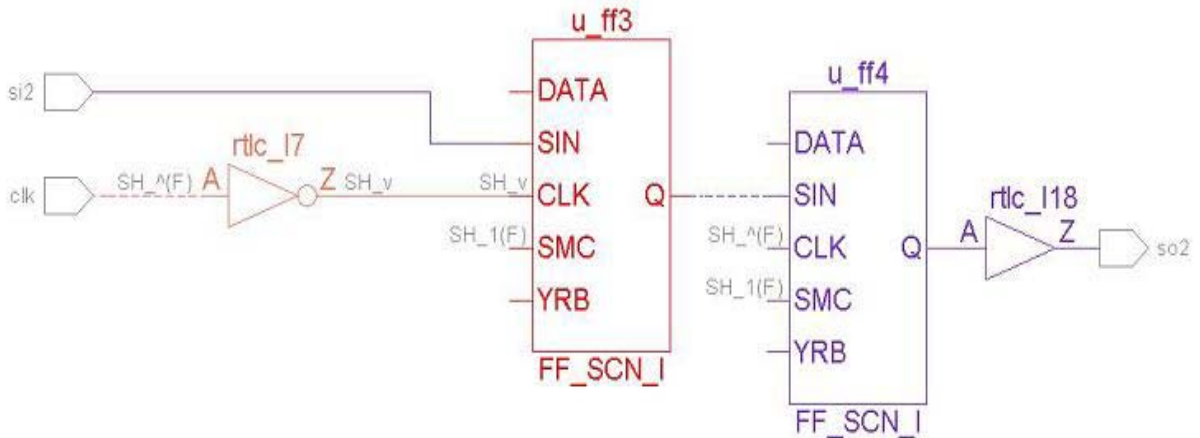
How to Debug and Fix

Correct the RTL to fix the inversion.

Examples Code and/or Schematic

Example 1

Consider the following schematic:

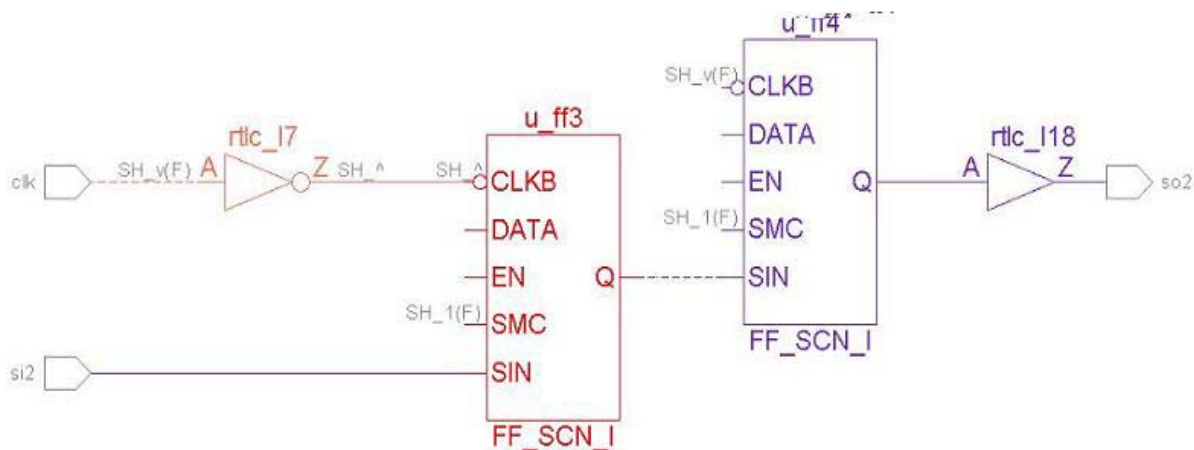


The Scan_38 rule reports following violation for the above design.

Scan chain (scanin=>'top2.si2', scanout=>'top2.so2', condition=>'testmode') with '2' scan cells, starts with a flip-flop ('top2.u_ff3') which triggers at 'negative' edge of test clock 'top2.clk'

Example 2

Consider the following schematic:



For the above design, the Scan_38 rule reports the following violation message:

Scan chain (scanin=>'top_rto.si2', scanout=>'top_rto.so2', condition=>'testmode') with '2' scan cells, starts with a flip-flop ('top_rto.u_ff3') which triggers at 'positive' edge of test clock 'top_rto.clk (rto)'

Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

None

Scan_39

Ensure that each scan flip-flop in a design is part of only one scan chain

When to Use

Use this rule to ensure that there are no overlapping scan chains in the design.

Description

The Scan_39 rule reports a violation if a scan flip-flop is part of more than one valid scan chain.

Prerequisites

The Scan_39 rule checks for only properly stitched scan chains.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_39 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Scan_39 rule uses the `-testclock` argument of the clock constraint.

Operating Mode

Scanshift

Messages and Suggested Fix

The Scan_39 rule reports the following warning message:

```
[WARNING] Scan chains with scanchain_id: '<scan_chain_id1>'
(scanin=>'<scanin_node1>', scanout=>'<scanout_node1>',
condition=>'<scan_chain_enable1>') and
scanchain_id: '<scan_chain_id2>' (scanin=>'<scanin_node2>',
scanout=>'<scanout_node2>', condition=>'<scan_chain_enable2>')
share <no_of_common_flip_flops>
```

Arguments

- ID of the first scan chain, <scan_chain_id1>
- Scan-in node of the first scan chain, <scanin_node1>
- Scan-out node of the first scan chain, <scanout_node1>
- Enable condition for the first scan chain, <scan_chain_enable1>
- ID of the second scan chain, <scan_chain_id2>
- Scan-in node of the second scan chain, <scanin_node2>
- Scan-out node of the second scan chain, <scanout_node2>
- Enable condition for the second scan chain, <scan_chain_enable2>
- Number of flip-flops which are common in the specified two chains, <no_of_common_flip_flops>

Potential Issues

A violation is reported when a flip-flop in the design is incorrectly connected to more than one scan chains.

Consequences of Not Fixing

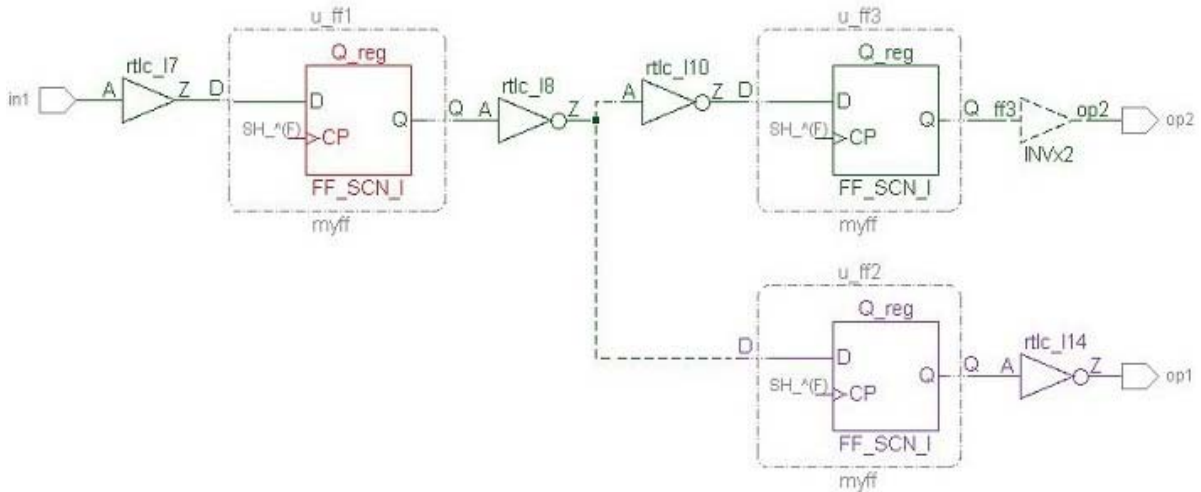
Not fixing the violation may result in related or overlapping scan chains, which are difficult to handle. This is because shift in one scan chain may cause shift in other chain or loss of data.

How to Debug and Fix

To fix the violation, reconnect the scan chain so that one scan element is part of only one scan chain.

Example Code and/or Schematic

Consider the following schematic:



For the above design, the Scan_39 rule reports the following warning message:

```
Scan chains with scanchain_id: '1' (scanin=>' top.in1',
scanout=>' top.op1', condition=>' testmode') and scanchain_id: '2'
(scanin=>' top.in1', scanout=>' top.op2', condition=>' testmode')
share '1' flip-flop
```

Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

None

Scan_40

Ensure that the scan chains have a 'copy flip-flop' at positive to negative clock edge crossing

When to Use

Use this rule to ensure reliability, if the edges of a clock are mixed on a scan chain.

Description

The Scan_40 rule checks for the presence of a copy flip-flop at the positive to negative clock edge crossing within the scan chain.

A copy flip-flop is a flip-flop which is receiving the scan-in and data pin source from the same flip-flop. For example, consider there are two flip-flops, FF(L-1) and FFL, on the same clock edge. In the capture mode, trace back the D-pin source DS of the last flop FFL. If the source DS of FFL is Q-output of FF(L-1) with zero or even number of inversion, then FFL is a copy flip-flop for FF(L-1).

Prerequisites

The Scan_40 rule checks for properly stitched scan chain only.

Default Weight

1

Language

Verilog, VHDL

Method

- i.) Trace all scan chains
- ii.) Identify last two pos edge flops FF(L-1) and FFL in any pos edge to neg edge transition in chain
- iii.) In the capture mode, trace back (through single input devices) the d pin source DS of the last flop FFL
- iv.) Issue an error message if the source DS of FFL is not the Q output of FF(L-1)
- v.) Issue an error message if the source DS of FFL is the Q output of FF(L-1) but with an odd number of inversions

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_40 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Scan_40 rule uses the `-testclock` argument of the clock constraint.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

The Scan_40 rule reports the following violation message:

```
[WARNING] Positive to negative clock edge crossing detected
between flip-flops '<posedge>' [CLK1: '<tclk_posedge>'] and
'<negedge>' [CLK2: '<tclk_negedge>'] in
schain(scanin=>'<scanin_name>'
scanout=>'<scanout_name>')[under <condition>] without 'copy
flip-flop'
```

Arguments

- Name of the scanin node, *<scanin_name>*
- Name of the scanout node, *<scanout_name>*
- Scan chain formation condition, *<condition>*. The default value for this argument is the shift method during test mode.
- Positive edge flip-flop on the chain at crossing, *<posedge>*
- Negative edge flip-flop on the chain at crossing, *<negedge>*
- Test clock driving positive edge flip-flop, *<tclk_posedge>*
- Test clock driving negative edge flip-flop, *<tclk_negedge>*

Potential Issues

Incorrect clock edge used for the crossing flip-flops or 'copy flip-flop' is missing at the crossing

Consequences of Not Fixing

Not fixing the violation may result in unreliable scan chain operation.

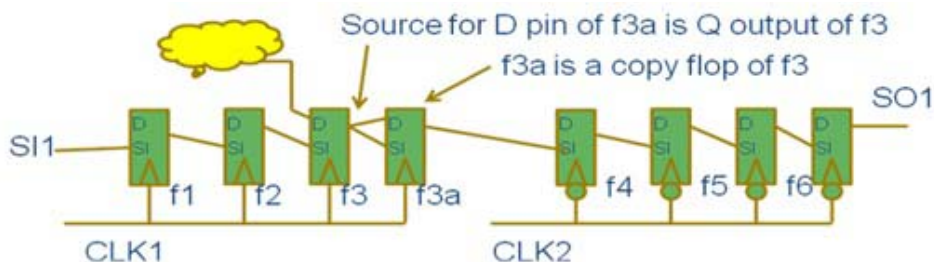
How to Debug and Fix

To fix the violation, either correct the clock edge or put the copy flip-flop at the crossing.

Example Code and/or Schematic

Example 1

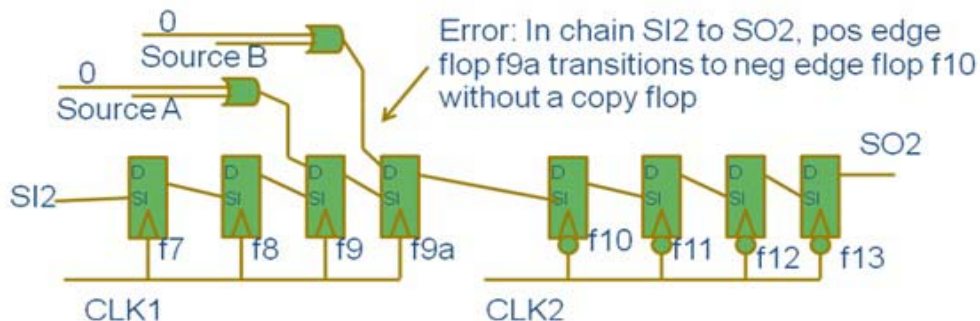
The following figure describes the scan chain from SI2 to SO2, where the source for D pin of the f3a flip-flop is connected to the Q output of the f3 flip-flop:



In the above example, the Scan_40 rule does not report any violation because a copy flip-flop, f3a, is present on the scan chain.

Example 2

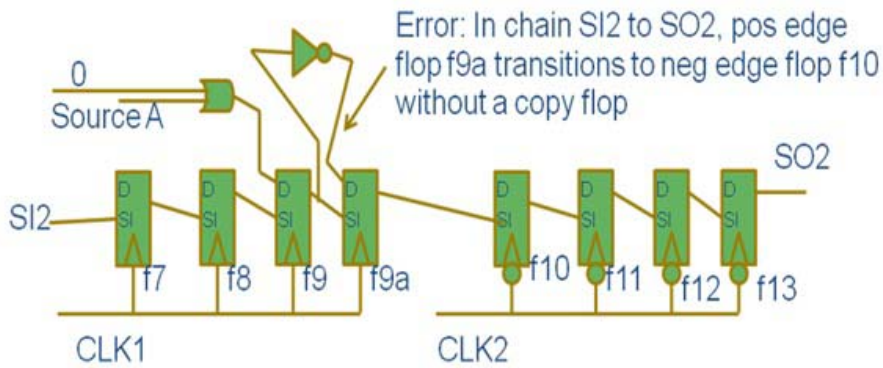
The following figure describes the scan chain from SI2 to SO2, where the positive edge flip-flop, f9a transitions to the negative edge flip-flop, f10:



In the above example, the Scan_40 rule reports a warning message because a copy flip-flop is not present at the positive to negative clock edge crossing of a scan chain.

Example 3

The following figure describes the scan chain from SI2 to SO2, where the positive edge flip-flop, f9a transitions to the negative edge flip-flop, f10:



In the above example, the Scan_40 rule reports a warning message because a copy flip-flop is not present at the positive to negative clock edge crossing of a scan chain.

Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

None

Scan_41

Scanout port should be driven by pad terminal of pad cells

When to Use

Use this rule to ensure that every scan out port is driven by a pad terminal of a pad cell.

Description

The Scan_41 rule reports violation for the following conditions:

- IO pad cell connected to scanout port has multiple pins to control the I/O direction
- Scanout port is connected to a pad terminal of pad cell having different direction
- Scanout port has non-x value
- Scanout port is not connected to a pad terminal of pad cell

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *scan_chain* (mandatory): Use this constraint to specify the scan chains for the Scan_41 rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in the test mode.

Operating Mode

Shift

Messages and Suggested Fix

Message 1

[WARNING] Scan out port <port-name> has <num> control pins

Arguments

- Name of the scan out port, <port-name>
- Number of control pins, <num>

Potential Issues

This violation message appears if the scan out port is connected to multiple IO direction control pins.

Consequences of Not Fixing

Not fixing the violation will corrupt the test vector.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the instances, their control pins connected to the scan out port, and the scan out port.

To fix the violation, it is recommended to change the RTL for such scan out ports.

Message 2

[WARNING] Scan out port <port-name> is tied to <num>.

Arguments

- Name of the scan out port, <port-name>
- Number of scan out ports tied logic value(0/1), <num>

Potential Issues

The violation message appears if the scan out port is tied to a logic 0 or 1.

Consequences of Not Fixing

Not fixing the violation will corrupt the test vector.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the instances, their control pins connected to the scan out port, and the scan out port.

To fix the violation, ensure that the scan out port is not tied to logic 0 or 1,

using constraints.

Message 3

[WARNING] Scan out port <port-name> and pad pin <pin-name> have different direction

Arguments

- Name of the scan out port, <port-name>
- Name of the pin, <pin-name>

Potential Issues

The violation message appears, if the scan out port and the pad pin have different direction.

Consequences of Not Fixing

Direction mismatch can potentially leave scan out port un-driven or can cause contention. It may lead to corruption of test vector.

How to Debug and Fix

Double-click the information message. The Incremental Schematic window highlights the scan out port.

To fix the violation, modify the design to make the direction of pad cell and scan out port consistent.

Message 4

[WARNING] Scan out port <port-name> is connected to a non-I/O pad cell instance.

Arguments

Name of the scan out port, <port-name>

Potential Issues

The violation message appears when there is an incorrect connection between pad cell and scan out port.

Consequences of Not Fixing

If you do not fix the violation, scan out port may not get the intended drive strength.

How to Debug and Fix

Double-click the information message. The Incremental Schematic window highlights the scan out port.

To fix the violation, review the pad cell connection to scan out port.

Message 5

[WARNING] Scan out port <port-name> is not connected to pad pin of IO pad cell <pad-cell-name>

Arguments

Name of the scan out port, <port-name>

Potential Issues

The violation message appears, if the scan out port is not connected to a pad pin of a pad cell.

Consequences of Not Fixing

This is an informational message. It is a good design practice to use pad cell before a scan out port.

How to Debug and Fix

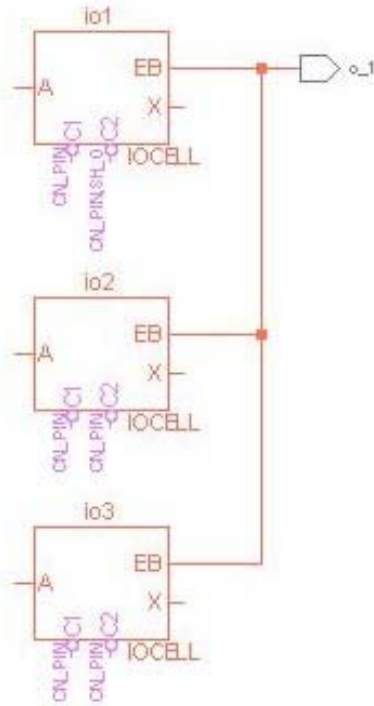
Double-click the information message. The Incremental Schematic window highlights the scan out port.

To fix the violation, review the design and insert the pad cell, if required.

Example Code and/or Schematic

Example 1

Consider the following schematic where the scan out port, `top_inout.o_1` has 6 control pins:

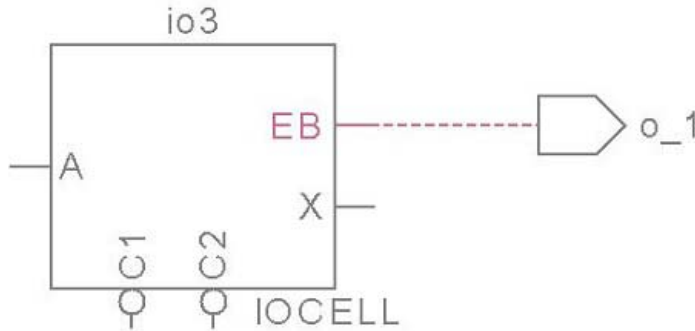


The Scan_41 rule reports following violation for the above example:

Scan out port '`top_inout.o_1`' is connected to 6 control pins

Example 2

Consider the following schematic where scan port and pad pin have different direction:

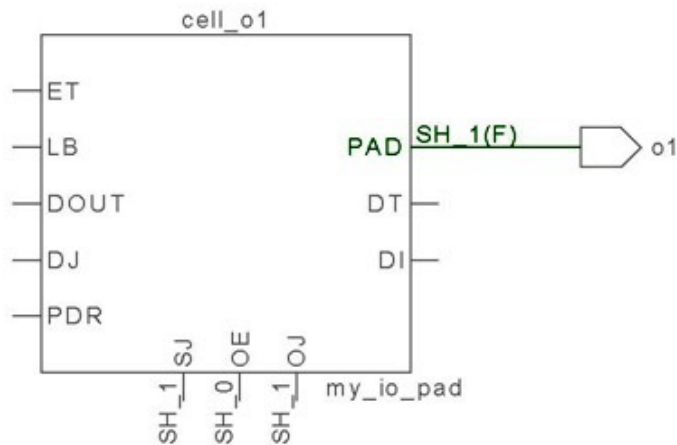


The Scan_41 rule reports following violation for the above example:

Scan out port 'top.o_1' and pad pin 'top.io3.EB' have different direction.

Example 3

Consider the following schematic where the scan out port, top_output.o1, is tied to 1:



The Scan_41 rule reports following violation for the above example:
Scan port 'top_output.o1' is tied to '1'

Default Severity Label

Warning

Rule Group

Scan Rules

Reports and Related Files

No related reports or files

Sanity Check Rules

Overview

The `Sanity Check` rule group of the SpyGlass DFT ADV DSM product has the following rules related to sanity check:

Rule	Description
<i>dftParamCheck_02</i>	Ensures that the <i>rme_selection</i> parameter points to the valid directory.
<i>dsmParamSanity_00</i>	Performs sanity checks on the defined parameter values
<i>dftDsmConstraintCheck_01</i>	Performs sanity checks on the constraints specified in SGDC.
<i>dftDsmConstraintCheck_02</i>	Performs sanity checks on the constraints specified in SGDC.
<i>dftDsmConstraintCheck_04</i>	(Verilog) Performs a sanity check as to whether the <code>expect_frequency</code> constraint is specified on a module that is a <code>clock_shaper</code> .
<i>dftDsmConstraintCheck_05</i>	(VHDL) Performs a sanity check as to whether the <code>expect_frequency</code> constraint is specified on a module that is a <code>clock_shaper</code> .
<i>dftDsmConstraintCheck_06</i>	<code>gating_cell</code> constraint should have equal width for <code>-clkoutTerm</code> , <code>-enTerm</code> and <code>-obsTerm</code> .
<i>dftDsmConstraintCheck_07</i>	Do not apply the <code>abstract_port</code> constraint on non-black box cells.
<i>dftDsmConstraintCheck_08</i>	Report conflict of simulation value with enabling value given in the constraint file
<i>dftDsmConstraintCheck_ComplexCell</i>	Constraint <code>complex_cell</code> is obsolete and should be replaced by the <code>clock_shaper</code> constraint.

dftParamCheck_02

Sanity check to ensure that parameter rme_selection points to a valid directory

When to Use

Use this rule to ensure the directory specified using the rme_selection parameter is valid.

Description

This rule reports a violation if the directory specified using the *rme_selection* parameter is valid.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

rme_selection

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[FATAL] Parameter 'rme_selection' value '<parameter value>: rme_selection' does not point to a valid run directory <dir>.

Potential Issues

The violation message is reported when you specify an invalid value to *rme_selection* parameter.

Consequences of Not Fixing

Sanity Check Rules

If you do not fix the violation, the SpyGlass run would abort.

How to Debug and Fix

Specify a valid value to the [rme_selection](#) parameter.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Sanity Rules

Reports and Related Files

No related reports or files.

dsmParamSanity_00

Performs sanity checks on the defined parameter values

When to Use

Use this rule to verify the values specified for the defined parameters.

Description

This rule flag a violation if invalid value is given to the defined parameter.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *dft_rrf_tp_count*: Default value is 5%. Set the value of the parameter to any positive integer or percentage to specify the required number of test points to be suggested for improving random pattern fault coverage.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[FATAL] Invalid value '<value>' is given to parameter '<parameter_name>'

Potential Issues

The violation message is reported when you specify an invalid value to the defined parameter.

Consequences of Not Fixing

If you do not fix the violation, the SpyGlass run would abort.

How to Debug and Fix

Specify a valid value to the defined parameter.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Sanity Rules

Reports and Related Files

No related reports or files.

dftDsmConstraintCheck_01

Performs sanity checks on the 'atspeed_clock_frequency' and 'clock_shaper' constraints.

Rule Description

This rule performs the following checks on the *atspeed_clock_frequency* and *clock_shaper* constraints.

Constraint	Check
atspeed_clock_frequency	All nodes specified in the -name field of this constraint should also be declared as an atspeed clock.
clock_shaper	Checks for the following conditions <ul style="list-style-type: none"> • The -maskcaptureATspeed field should have the same number of bits as the number of pins in -testmodepins" field. • The -scanshift field has a non-binary character. • The -captureStatic field has a non-binary character. • The -captureATspeed" field has a non-binary character. • The -scanshift field must have exactly as many bits as the number of pins in the -testmodepins field. • The -captureATspeed field must have exactly as many bits as the number of pins in the -testmodepins field.

Language

Verilog

Constraints

- *atspeed_clock_frequency* (mandatory)
- *clock_shaper* (mandatory)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Sanity Check Rules

Operating Mode

None

Message Details

The violation messages are reported in the [dft_dsm_constr-err-file](#) report.

Rule Severity

Fatal

Reports and Related Files

[dft_dsm_constr-err-file](#)

dftDsmConstraintCheck_02

Performs SGDC sanity check on the clock constraint.

Rule Description

The *dftDsmConstraintCheck_02* rule reports a violation, if an *atspeed* clock is specified without any frequencies.

You can specify the frequency of a clock using one of the following options:

- *-frequency* or *-period* option of the *clock* constraint
- *-freqList* option of the *atspeed_clock_frequency* constraint

The violation is reported only if the frequencies for a clock are not specified through any constraint.

Rule Exceptions

The *dftDsmConstraintCheck_02* rule does not report a violation if one or more of following options is specified along with the *-atspeed clock* constraints:

- *frequency* (MHz)
- *period* (Nanoseconds)
- *atspeed_clock_frequency* constraint

Language

Verilog, VHDL

Constraints

- *clock* (mandatory)
- *atspeed_clock_frequency* (mandatory)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

None

Message Details

Message

Clock frequency data is missing for atspeed clock <clkName>

Arguments

- Name of the clock node, <clkname>

Rule Severity

Warning

dftDsmConstraintCheck_04

Performs a sanity check as to whether the `expect_frequency` constraint is specified on a module that is a `clock_shaper`.

Rule Description

This rule performs a sanity check to verify whether the `expect_frequency` constraint is specified on a `clock_shaper` module.

NOTE: *Starting 4.5.0, the `dftDsmConstraintCheck_04` rule does not perform any sanity check as the `expect_frequency` constraint can now be applied on any pin of the design. Therefore, this rule will be deprecated in a future SpyGlass release.*

Language

Verilog

Constraints

`expect_frequency` (mandatory)

Rule Parameters

`dftUseOffStateOfClockInClockPropagation`

Operating Mode

None

Message Details

Message

Module `<mod_name>` specified in `-name` field is not a `clock_shaper`

Arguments

- Name of the module, which is not a `clock_shaper`, `<mod_name>`

Rule Severity

Fatal

dftDsmConstraintCheck_05

Performs a sanity check as to whether the `expect_frequency` constraint is specified on a module that is a `clock_shaper`.

Rule Description

This rule performs a sanity check to verify whether the `expect_frequency` constraint is specified on a `clock_shaper` module.

NOTE: *Starting 4.5.0, the `dftDsmConstraintCheck_05` rule does not perform any sanity check as the `expect_frequency` constraint can now be applied on any pin of the design. Therefore, this rule will be deprecated in a future SpyGlass release.*

Language

VHDL

Constraints

`expect_frequency` (mandatory)

Rule Parameters

`dftUseOffStateOfClockInClockPropagation`

Operating Mode

None

Message Details

Message

Module `<mod_name>` specified in `-name` field is not a `clock_shaper`

Arguments

- Name of the module, which is not a `clock_shaper`, `<mod_name>`

Rule Severity

Fatal

dftDsmConstraintCheck_06

gating_cell constraint should have equal width for -clkoutTerm, -enTerm and -obsTerm

Rule Description

The dftDsmConstraintCheck_06 rule reports a violation in the following cases:

- If the -clkoutTerm, -enTerm, and -obsTerm arguments of the *gating_cell* constraint do not have equal width.
- If the -clkoutTerm and -enTerm arguments of the *gating_cell* constraint are not specified.

Language

Verilog, VHDL

Constraints

gating_cell (mandatory)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

None

Message Details

Message 1

gating_cell constraint '`<name>`' has different width for clkout pin (`<clk_pin_name: <clk_pin_width>`) and `<pin-type>` (`<pin_name: <pin_width>`)

Arguments

- Name of the clock gating cell, `<name>`
- Name of the clkout pin, `<clk_pin_name>`
- Width of the clkout pin, `<clk_width>`

Sanity Check Rules

- Enable pin or Obs pin, <pin_type>
- Name of the Enable or Obs pin, <pin_name>
- Width of the Enable or Obs pin, <pin_width>

Message 2

'gating_cell' constraint 'name' has missing
'missing_pin_details'

Arguments

- Name of the clock gating cell, <name>
- Details of the missing pin, <missing_pin_details>

Rule Severity

Fatal

dftDsmConstraintCheck_07

Do not apply the abstract_port constraint on non-black box cells

When to Use

Use this rule to identify the cells that are not black box and have abstract_port constraint applied on them.

Description

The dftDsmConstraintCheck_07 rule reports violation, if the *abstract_port* constraint is applied on a cell, which is not a black box.

Rule Exception

The *abstract_port* sgdc command, which is applied on the top-level primary port is ignored.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

abstract_port (optional): Use this constraint to specify clock-domains for input or output ports of black boxes. The SpyGlass DFT ADV DSM supports only following options of the *abstract_port* constraint:

- -module
- -ports
- -clock

Operating Mode

None

Messages and Suggested Fix

[WARNING] `abstract_port` constraint is ignored for non-blackbox cell '`<cell_name>`'

Arguments

Name of the cell that is not a black box and has `abstract_port` constraint applied on it, `<cell_name>`

Potential Issues

A violation is reported when a cell, which is not a black box, has `abstract_port` constraint applied on it.

Consequences of Not Fixing

All `abstract_port` SGDC commands related to the respective cell are ignored during the SpyGlass DFT ADV analysis.

How to Debug and Fix

To debug the violation, view the Incremental Schematic and identify the cells that are not black box cells and that have `abstract_port` constraint applied on them.

To fix the violation, perform one of the following actions:

- Remove the `abstract_port` constraint from the cells that are not black box cells.
- Convert such cells to black box cells.

Example Code and/or Schematic

Consider the following Verilog example:

test.sgdc

```
current_design test
  clock -name CLK1 -domain D1 -testclock -atspeed -freq 24
  clock -name CLK2 -domain D2 -testclock -atspeed -freq 50
  abstract_port -module abs1 -ports I1 O1 -clock CLK1
  abstract_port -module abs1 -ports I2 O2 -clock CLK2
  abstract_port -module abs2 -ports I1 I2 O -clock CLK
```

test.v

```
module abs1 (CLK1, CLK2, I1, I2, O1, O2);
```

```
    output O1, O2;
    input I1, I2, CLK1, CLK2;
    assign O2 = I1 & I2;
endmodule

module abs2 (CLK, I1, I2, O);
    output O;
    input I1, I2, CLK;
    assign O = I1 & I2;
endmodule

module test(CLK1,CLK2,I1,I2,O);
    input CLK1,CLK2;
    input I1,I2;
    output O;
    reg r1,r2;
    wire n1,n2,nn1,nn2;

    always @(posedge CLK1) begin
        r1 <= I1;
        r2 <= I2;
    end

    abs1 Uabs1(.CLK1(CLK1), .CLK2(CLK2), .I1(r1), .I2(r2),
.O1(n1), .O2(n2));
    /*
    assign nn1 = n1;
    assign nn2 = n2;
    */
    abs2 Uabs2(.CLK(CLK2), .I1(/*n*/n1), .I2(/*n*/n2), .O(O));
endmodule
```

In the above example, the `dftDsmConstraintCheck_07` rule reports one violation for module `abs1` and two violations for module `abs2` in the `test.sgdc` file. This is because these modules are not black box modules but have `abstract_port` constraint applied on them.

Sanity Check Rules

Default Severity Label

Warning

Rule Group

Sanity Check

Reports and Related Files

No related reports or files.

dftDsmConstraintCheck_08

Report conflict of simulation value with enabling value given in the constraint file

Rule Description

The *dftDsmConstraintCheck_08* rule checks and reports conflict between *atspeed* testmode value and enabling value of frequency specified through the *atspeed_clock_frequency* constraint.

Constraints

- *atspeed_clock_frequency* (mandatory)
- *clock* (mandatory)
- *test_mode* (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Message Details

The following violation message is displayed for the *dftDsmConstraintCheck_08* rule:

Enabling values <enVals> of frequency <freqName> [atspeed clock : <clkNode>] and atspeed testmode conditions are conflicting

Arguments

- Enable values as given by the user, <enVals>
- Name of the frequency, <freqName>.
- The net, port, or term on which the frequency is specified, <clknode>.

Schematic highlight

Terminals or ports at which the conflict occurs.

Rule Severity

Warning

dftDsmConstraintCheck_ComplexCell

Constraint `complex_cell` is obsolete and should be replaced by the `clock_shaper` constraint

Rule Description

The `dftDsmConstraintCheck_ComplexCell` rule reports a violation message if the `complex_cell` constraint is used in the design. You can use the `clock_shaper` constraint as a replacement of the `complex_cell` constraint.

Language

Verilog

Constraints

`clock_shaper` (mandatory)

Rule Parameters

`dftUseOffStateOfClockInClockPropagation`

Operating Mode

None

Message Details

Message

Module `<mod_name>` specified in `-name` field is not a `clock_shaper`

Arguments

- Name of the module, which is not a `clock_shaper`, `<mod_name>`

Rule Severity

Warning

Scan Enable Rules

Overview

The Scan Enable rule group of the SpyGlass DFT ADV product has the following rules related to scan enable logic:

Rule	Description
<i>SE_Sanity_01</i>	User specified scan_enable_source must feed scan enable pin of a scan flip-flop
<i>SE_Sanity_02</i>	When -active_value is applied on user specified scan_enable_source, it must bring all scan flip-flop in mask-mode (scan chain mode)
<i>SE_Sanity_03</i>	In atspeed mode, user specified scan_enable_source should not get a value other than specified -active_value in atspeed mode
<i>SE_Sanity_04</i>	Avoid specifying scan_enable_source in fan-out of another scan_enable_source
<i>SE_Sanity_05</i>	scan_enable_source with -clock should be actually driven by logic clocked on specified clock.
<i>SE_01</i>	Scan Enable cannot be driven by combinational logic
<i>SE_02</i>	Scan Enable cannot be driven by sequential logic
<i>SE_03</i>	Scan Enable cannot be used as data input to a flip-flop
<i>SE_04</i>	Scan Enable cannot feed primary output
<i>SE_05</i>	Scan enable source must feed flip-flops with the same clock as is specified in the -clock field
<i>SE_06</i>	All scannable flip-flops must have their scan enable pins driven by a designated scan_enable_source

SE_Sanity_01

User specified scan_enable_source must feed scan enable pin of a scan flip-flop

When to Use

Use this rule to identify user-specified scan_enable_source that does not drive a single scan enable pin of a scan flip-flop in atspeed capture mode.

Description

The SE_Sanity_01 rule reports a violation, if a user specified scan enable source does not drive a single scan enable pin of a scan flip-flop in atspeed capture mode.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

scan_enable_source (mandatory): Defines the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_Sanity_01 rule reports following violation message:

[WARNING] scan_enable_source '`<name>`' (`<type>`) does not drive any flip-flop scan enable pin

Arguments

- Name of scan enable source, `<name>`

- Type of scan enable source, <type>

Potential Issues

A violation is reported due to incorrect SGDC specification or incorrect connectivity.

Consequences of Not Fixing

Constraint would not have any impact on any Scan Enable rules.

How to Debug and Fix

To debug the violation, check the RTL.

To fix the violation, check the connectivity for correct SGDC specification

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_Sanity_02

When -active_value is applied on user specified scan_enable_source, it must bring all scan flip-flop in mask-mode (scan chain mode)

When to Use

Use this rule to identify user-specified scan_enable_source where -active_value is specified.

Description

The SE_Sanity_02 rule reports violation for those scan_enable_source where -active_value is specified.

The generated violation message contains information on number of flip-flops in atspeed capture mode and in scan chain (mask) mode.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

scan_enable_source (mandatory): Defines the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_Sanity_02 rule reports following violation message:

[WARNING] scan_enable_source '`<name>`' `<type>`, driving '`<num1>`' flip-flop(s) scan enable pin, failed to put '`<num2>`' flip-flop(s) in atspeed capture mode

Arguments

- Name of scan enable source, <name>
- Type of scan enable source, <type>
- Number of flip-flops in scan chain (mask) mode, <num1>
- Number of flip-flops in atspeed capture mode, <num2>

Potential Issues

A violation is reported when phase between scan_enable_source and scan enable pin of flip-flop is not as expected.

Consequences of Not Fixing

Not fixing the violation results in inability to mask flip-flops during atspeed capture.

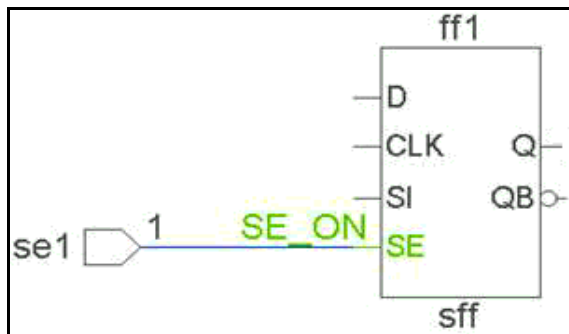
How to Debug and Fix

To debug the violation, view the Incremental Schematic and check the phase inversion.

To fix the violation, remove the inversion from the RTL on the required path.

Example Code and/or Schematic

Consider the following schematic:



Also, consider the following scan_enable_source constraint description:

Scan Enable Rules

```
scan_enable_source -name top.se1 -mode sequential -clock clk1 -active_value 1  
scan_enable_source -name top.se2 -mode combinational -clock clk2 -active_value 1
```

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_Sanity_03

In atspeed mode, user specified scan_enable_source should not get a value other than specified -active_value in atspeed mode

When to Use

Use this rule to identify if a user-specified scan_enable_source gets a conflicting value in atspeed capture mode.

Description

This rule will report a violation if a user specified scan_enable_source gets a conflicting value in the atspeed capture mode.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

scan_enable_source (mandatory): Defines the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_Sanity_03 rule reports following violation message:

[WARNING] scan_enable_source '`<name>`' (`<type>`) that drives '`<num>`' flip-flop has conflicting value (as specified using `-value`) with respect to atspeed mode. Atspeed value: `<value1>`, Specified `-active_value`: `<value2>`

Arguments

- Name of scan enable source, `<name>`

Scan Enable Rules

- Type of scan enable source, <type>
- Number of flip-flops in atspeed capture mode, <num2>
- Atspeed value, <value1>
- Specified -active_value, <value2>

Potential Issues

A violation is reported due to incorrect constraint specification or connectivity.

Consequences of Not Fixing

This will prevent polarity (application of -value) check on such scan_enable_source.

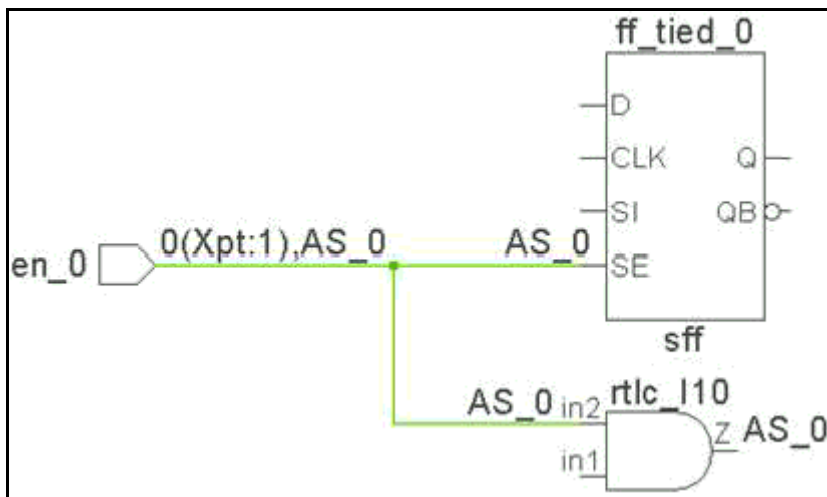
How to Debug and Fix

To debug the violation, view the Incremental Schematic and identify the root cause of the value propagation.

To fix the violation, correct the constraint specification and ensure that no values are generated in the atspeed mode on the specified scan_enable_source.

Example Code and/or Schematic

Consider the following schematic:



Also, consider the following scan_enable_source constraint description:

```
scan_enable_source -name topA.u12 -mode combinational -active_value 1  
scan_enable_source -name topA.en_1 -mode combinational -active_value 1  
scan_enable_source -name topA.en_0 -mode combinational -active_value 1
```

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_Sanity_04

Avoid specifying scan_enable_source in fan-out of another scan_enable_source

When yo Use

Use this rule to identify a user-specified scan_enable_source, which is specified in the fan-out of another user-specified scan_enable_source.

Description

The SE_Sanity_04 rule reports violation for those user-specified scan_enable_source that are found in the fan-out of another user specified scan_enable_source.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

scan_enable_source (mandatory): Defines the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_Sanity_04 rule reports following violation message:

```
[WARNING] scan_enable_source '<se_source_name1>' (<se_type1>)  
that drives '<num1>' flip-flop(s) is also driven by another  
scan_enable_source '<se_source_name2>' (<se_type2>) that drives  
'<num2>' flip-flop(s)
```

Arguments

- Scan enable source name, <se_source_name1>
- Type of scan enable source, <se_type1>
- Number of flip-flops driven by first scan enable source, <num1>
- Scan enable source name, <se_source_name2>
- Type of scan enable source, <se_type2>
- Number of flip-flops driven by second scan enable source, <num2>

Potential Issues

The violation is reported when a scan enable source is wrongly specified, that is, the scan enable source which is specified in the fan-out is not required to be specified as an SGDC command.

Consequences of Not Fixing

This rule is a sanity check rule. Therefore, there are no direct design consequences of not fixing this violation.

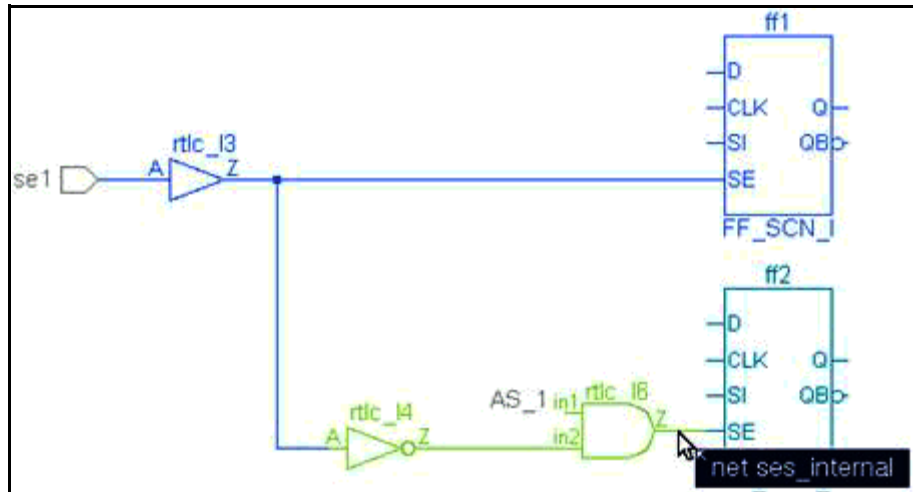
How to Debug and Fix

View the Incremental Schematic and trace the scan enable source, which is driven by another scan enable source.

To fix the violation, remove the first scan enable source from the design.

Example Code and/or Schematic

Consider the following schematic:



Also, consider the following scan_enable_source constraint description:

```
scan_enable_source -name topA.ses_internal -mode sequential
scan_enable_source -name topA.sel -mode sequential
```

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related report or files.

SE_Sanity_05

scan_enable_source with -clock should be actually driven by logic clocked on specified clock.

When to Use

Use this rule to do a sanity check on those scan_enable_source for which -clock field is specified to check whether that scan_enable_source is driven by logic which is clocked by the clock as specified with -clock field.

Description

The SE_Sanity_05 rule reports violation, if the user-specified scan_enable_source, with -clock, is not actually driven by logic clocked on specified clock.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

scan_enable_source (mandatory): Defines the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_Sanity_05 rule reports following violation message:

[WARNING] scan_enable_source '*<se_source_name>*' (*<se_type>*) that drives '*<num>*' flip-flop(s) is not driven by logic clocked on *atspeed* clock '*<at_clk_name>*' (specified with -clock)

Arguments

Scan Enable Rules

- Scan enable source name, <se_source_name>
- Type of scan enable source, <type>
- Number of flip-flops driven by the scan enable source, <num>
- Atspeed clock name, <at_clk_name>

Potential Issues

The violation is reported due to wrong design connectivity or wrong clock specification of the scan enable source.

Consequences of Not Fixing

Pipelining may not work correctly, if the violation is not fixed.

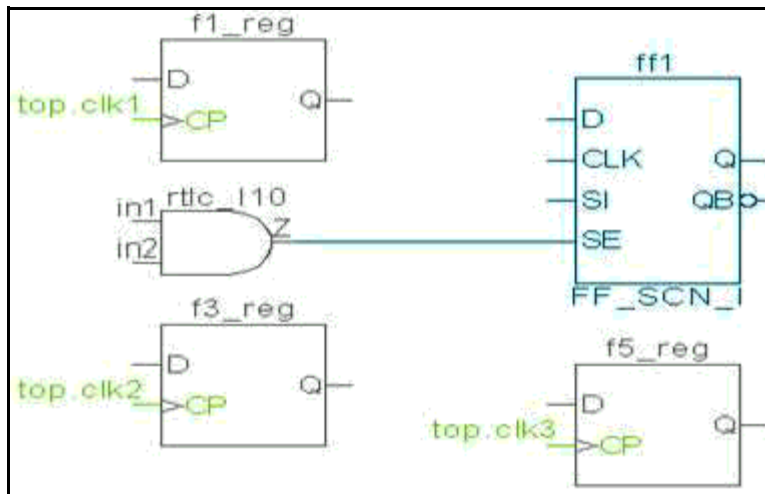
How to Debug and Fix

View the Incremental Schematic and check the clock specification for the scan enable source.

To fix the violation, change the connectivity or change the clock specification for the scan enable source.

Example Code and/or Schematic

Consider the following schematic:



Also, consider the following scan_enable_source constraint description:

```
scan_enable_source -name top.se1 -mode sequential -clock clk1
scan_enable_source -name top.se2 -mode combinational -clock clk2
```

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related report or files.

SE_01

Scan Enable cannot be driven by combinational logic

When to Use

Use this rule to identify whether the scan enable pin is driven by a combinational scan_enable_source.

Description

The SE_01 rule reports a violation, if a scan enable pin is driven by combinational scan_enable_source (primary port without scan_enable_source SGDC command or user specified scan_enable_source with -mode).

Method

If the scan enable pin of any scannable flip-flop is tagged with mode = "combinational" then issue violation

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *clock* (mandatory): Use to define the clocks of a design.
- *test_mode* (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_enable_source* (optional): Use to define the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_01 rule reports following violation message:

[WARNING] scan_enable_source '<name>' (<type>) drives '<num>' flip-flop(s) combinatorially

Arguments

- Name of the scan enable source, <name>
- Type of scan_enable_source, <type>
- Number of flip-flops, <num>

Potential Issues

A violation is reported when a scan_enable_source is driven combinatorially.

Consequences of Not Fixing

Scan enable signal is not pipelined for the flip-flops that are driven combinatorially.

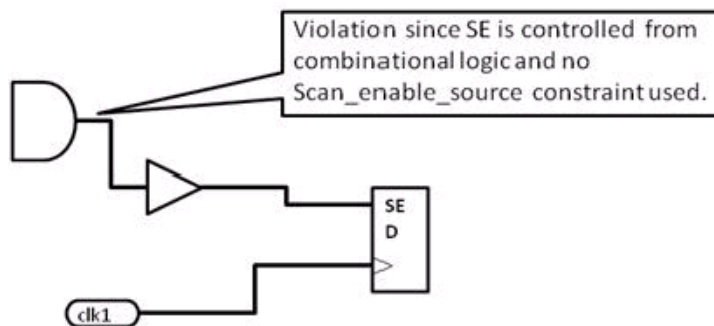
How to Debug and Fix

View the Incremental Schematic and check the path.

To fix the violation, change the connectivity or provide the correct constraint specification.

Example Code and/or Schematic

Consider the following example:



Scan Enable Rules

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_02

Scan Enable cannot be driven by sequential logic

When to Use

Use this rule to identify whether the scan enable pin is driven by sequential scan_enable_source.

Description

This rule will report violation if a scan enable pin is driven by sequential scan_enable_source (primary port with scan_enable_source SGDC command and with -mode sequential or flip-flop output pin).

Method

If the scan enable pin of any scannable flip-flop is tagged with mode = "sequential" then issue violation

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *clock* (mandatory): Use to define the clocks of a design.
- *test_mode* (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_enable_source* (optional): Use to define the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_02 rule reports following violation message:

[WARNING] scan_enable_source '`<name>`' (`<type>`) drives '`<num>`' flip-flop(s) sequentially

Arguments

- Name of the scan enable source, `<name>`
- Type of scan_enable_source, `<type>`
- Number of flip-flops, `<num>`

Potential Issues

A violation is reported when a scan_enable_source is driven sequentially.

Consequences of Not Fixing

Scan enable signals will always be pipelined and there would not be direct control from primary port.

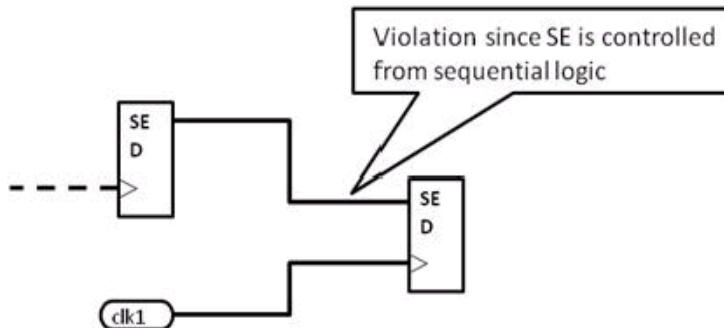
How to Debug and Fix

View the Incremental Schematic and check the path.

To fix the violation, change the connectivity or provide the correct constraint specification.

Example Code and/or Schematic

Consider the following example:



Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_03

Scan Enable cannot be used as data input to a flip-flop

When to Use

Use this rule to identify scan_enable_source that combinationally drives data-pin of a flip-flop.

Description

This rule reports a violation if a scan_enable_source (user specified or inferred) combinationally drives data-pin of a flip-flop.

Method

For each scan_enable source (from constraints or inferred)
Traverse the unblocked combinational fan-out cone
Maintain a count of the number of scannable flip-flop d-pin reached
If the count is > zero then issue a violation

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- [clock](#) (mandatory): Use to define the clocks of a design.
- [test_mode](#) (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [scan_enable_source](#) (optional): Use to define the source signal for scan chain enables.

Operating Mode

[Capture \(atspeed\)](#)

Messages and Suggested Fix

The SE_03 rule reports following violation message:

[WARNING] scan_enable_source '<name>' (<type>), feeding '<num>' flip-flop(s), feeds '<num1>' data pin(s) of flip-flop as well

Arguments

- Name of the scan enable source, <name>
- Type of scan_enable_source, <type>
- Number of flip-flops, <num>
- Number of flip-flops where data pin is being fed by the specified scan enable source, <num1>

Potential Issues

Data and scan enable pin share common path.

Consequences of Not Fixing

This may lead to a race condition.

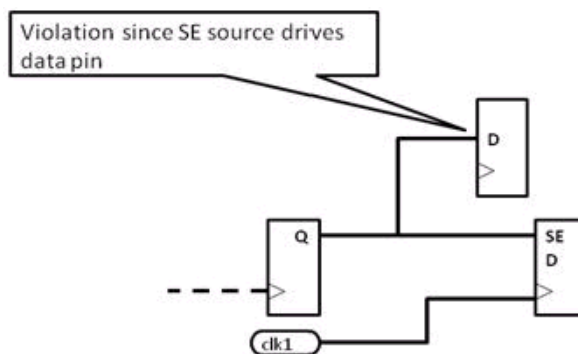
How to Debug and Fix

To debug the violation, view the Incremental Schematic and verify the path.

To fix the violation, break the path.

Example Code and/or Schematic

Consider the following example:



Scan Enable Rules

Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_04

Scan Enable cannot feed primary output

When to Use

Use this rule to identify whether a scan_enable_source combinationally drives a primary port.

Description

This rule will report violation if a scan_enable_source (user specified or inferred) combinationally drives a primary port.

Method

For each scan_enable source (from constraints or inferred)
Traverse the unblocked combinational fan-out cone
Maintain a count of the number of primary outputs reached
If this count is > zero then issue a violation

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *clock* (mandatory): Use to define the clocks of a design.
- *test_mode* (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_enable_source* (optional): Use to define the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_04 rule reports following violation message:

[WARNING] scan_enable_source '`<name>`' (`<type>`), feeding '`<num>`' flip-flop(s), "" feeds '`<num1>`' primary output port(s) combinatorially as well

Arguments

- Name of the scan enable source, `<name>`
- Type of scan_enable_source, `<type>`
- Number of flip-flops, `<num>`
- Number of flip-flops where data pin is being fed by the specified scan enable source, `<num1>`

Potential Issues

Data and scan enable pin share common path.

Consequences of Not Fixing

This may lead to a race condition during atspeed capture.

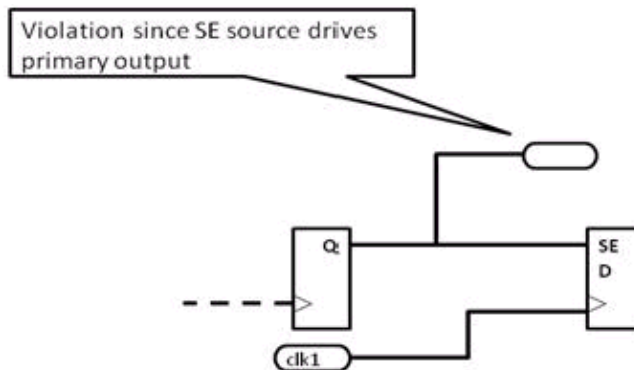
How to Debug and Fix

To debug the violation, view the Incremental Schematic and verify the path.

To fix the violation, break the path.

Example Code and/or Schematic

Consider the following example:



Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_05

Scan enable source must feed flip-flops with the same clock as is specified in the -clock field

When to Use

Use this rule to identify whether a sequential scan_enable_source drives flip-flop, which is working on a different atspeed clock.

Description

This rule reports a violation if a sequential scan_enable_source (user-specified or inferred) drives flip-flop, which is working on a different atspeed clock.

Method

For each flip-flop whose se pin is tagged with this scan_enable_source node
If this scan_enable_source constraint does not have a -clock then skip this flip-flop
If this scan_enable_source has a -clock parameter and the clock to this flip-flop is not the same as the clock declared in the scan_enable_source constraint then issue a violation
End for

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *clock* (mandatory): Use to define the clocks of a design.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_enable_source* (optional): Use this constraint to define the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_05 rule reports following violation message:

[WARNING] scan_enable_source '<name>' (<type>), driven by atspeed clock '<clk_name>', feeds '<num>' flip-flop(s) with different clock

Arguments

- Name of the scan enable source, <name>
- Type of scan_enable_source, <type>
- Clock name, <clk_name>
- Number of flip-flops whose scan enable pin is driven by the specified source, <num>

Potential Issues

A violation is reported due to incorrect connectivity of clock or scan enable pin.

Consequences of Not Fixing

Pipeline will not work correctly.

How to Debug and Fix

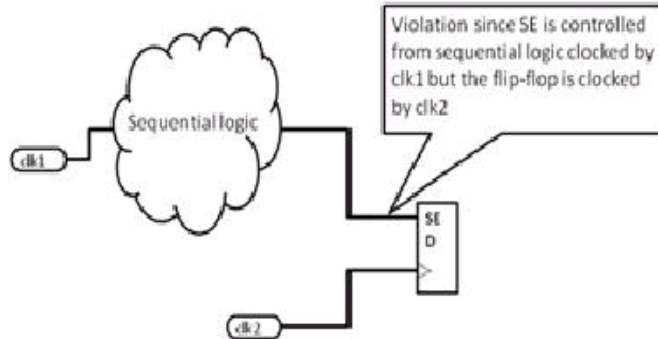
To debug the violation, check the IS and verify the path.

In order for pipelining on scan enable path to work correctly, pipeline register must be on same atspeed clock as sink flip-flops.

Example Code and/or Schematic

Example 1

Consider the following example:



Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

SE_06

All scannable flip-flops must have their scan enable pins driven by a designated scan_enable_source

When to Use

Use this rule to identify whether the inferred scan_enable_source is a proper source, that is, primary port or flip-flop output or user specified scan_enable_source

Description

This rule reports a violation if inferred scan_enable_source is not a proper source, that is, primary port or flip-flop output or user specified scan_enable_source.

Method

For each flip-flop whose se pin is tagged with this scan_enable_source node
If this scan_enable_source constraint does not have a -clock then skip this flip-flop
If this scan_enable_source has a -clock parameter and the clock to this flip-flop is not the same as the clock declared in the scan_enable_source constraint then issue a violation
End for

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *test_mode* (optional): Use to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_enable_source* (optional): Use to define the source signal for scan chain enables.

Operating Mode

Capture (atspeed)

Messages and Suggested Fix

The SE_06 rule reports following violation messages:

Message 1

[WARNING]scan_enable_source '<name>' (<type>), not a proper source (user specified or primary port or flip-flop output), feeds '<num>' flip-flop(s) scan enable pin

Arguments

- Name of the scan enable source, <name>
- Type of scan_enable_source, <type>
- Number of flip-flops whose scan enable pin is driven by the specified source, <num>

Potential Issues

A violation is reported due to missing constraint specification or scan enable path is not properly sensitized in atspeed mode.

Consequences of Not Fixing

User-specified scan enable will not control such flip-flops.

How to Debug and Fix

To fix the violation, specify the scan_enable_source constraint or sensitize the scan enable path correctly.

Message 2

[INFO]scan_enable_source '<name>' (<type>), a proper source but not user specified, feeds '<num>' flip-flop(s) scan enable pin

Arguments

- Name of the scan enable source, <name>
- Type of scan_enable_source, <type>
- Number of flip-flops whose scan enable pin is driven by the specified source, <num>

Potential Issues

A violation is reported due to missing constraint specification or scan enable path is not properly sensitized in atspeed mode.

Consequences of Not Fixing

User-specified scan enable will not control such flip-flops.

How to Debug and Fix

To fix the violation, specify the scan_enable_source constraint.

Message 3

```
[INFO]scan_enable_source '<name>' (<type>), user specified,  
feeds '<num>' flip-flop(s) scan enable pin
```

Arguments

- Name of the scan enable source, <name>
- Type of scan_enable_source, <type>
- Number of flip-flops whose scan enable pin is driven by the specified source, <num>

Potential Issues

This is an informational message. Therefore, there are no potential issues pertaining to this violation.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit consequence of not fixing the violation.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix info is required for this rule.

Message 4

```
[INFO]atspeed_scan_enable_source report is generated at  
location '<location>'
```

Arguments

Location at which the atspeed_scan_enable_source report is generated, <location>

Potential Issues

This is an informational message. Therefore, there are no potential issues pertaining to this violation.

Consequences of Not Fixing

This is an informational message. Therefore, there is no implicit consequence of not fixing the violation.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix info is required for this rule.

Message 5

[INFO] No scan_enable_source found for design '<du_name>'. Design is either rtl or there is no scan flip-flop

Arguments

Name of the design unit, <du_name>

Potential Issues

A violation is reported either due to a missing constraint or due to the absence of a scan flip-flop in the design.

Consequences of Not Fixing

SE rules will not run on such flip-flops.

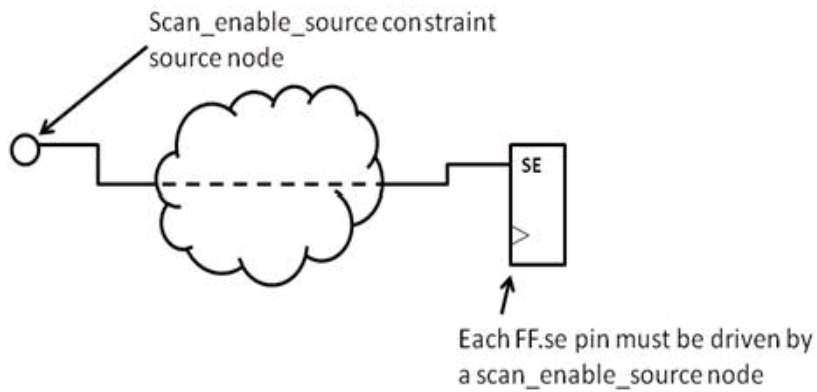
How to Debug and Fix

Check the violation message.

All scannable flip-flops must have their scan enable pins driven by a designated scan_enable_source to ensure that all flip-flops may be controlled from scan_enable_source.

Example Code and/or Schematic

Consider the following example:



Default Severity Label

Warning

Rule Group

Scan Enable Rules

Reports and Related Files

No related reports or files.

Scan Power Rules

Overview

The Scan Power rule group of the SpyGlass DFT ADV product has the following rules related to scan power:

Rule	Description
<i>SP_01</i>	No voltage domain crossing in a scan chain
<i>SP_02</i>	PMU Test Control cell (PMUWR) is not directly driving the control signal output of the PMU
<i>SP_03</i>	Control signal of the ISO cells, Power Switches and Retention cells must be driven by a PMU Test Control cell (PMUWR)
<i>SP_04</i>	PMU Test Control cell (PMUWR) must be inserted directly in the fan-in of the control signal of the ISO cells, Power Switches and Retention cells
<i>SP_05</i>	No power_management_test_control cell crossing in a scan chain

SP_01

No voltage domain crossing in a scan chain

Rule Description

The SP_01 rule flags a violation, if a given scan chain crosses more than one voltage domain (specified through `voltage_domain` SGDC command).

NOTE: *The SP_01 rule supports CPF and UPF commands.*

Constraint(s)

SGDC

- `scan_chain` (mandatory)
- `voltage_domain` (mandatory)

CPF Commands

`create_power_domain` (optional)

UPF Commands

`create_power_domain` (optional)

Rule Parameters

`dftUseOffStateOfClockInClockPropagation`

Operating Mode

`Scanshift`

Message Details

Message

Scan chain
(scanin => '<scanin_node_name>' => '<scanout_node_name>') [under
defineTag '<tag_name>'] crosses <no_of_voltage_domains> voltage
domains (<names_of_voltage_domains>)

Arguments

- The name of the scanin node `<scanin_node_name>`
- The name of the scanout node `<scanout_node_name>`

- Simulation condition or the name of a tag *<tag_name>*
- Number of voltage domains crossed *<no_of_voltage_domains>*.
- Names of all the crossed voltage domains,
<names_of_voltage_domains>

Schematic Highlight

Scan chain with different voltage domain segment in different color.

SP_02

A PMU Test Control cell (PMUWR) must be inserted directly in the fan-in of the PMU (Power Management Unit).

Rule Description

The SP_02 rule reports a violation if a PMU Test Control cell (PMUWR) is not directly driving the PMU.

Constraints

- *power_management_unit* (mandatory)
- *power_management_test_control_cell* (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Power Ground

Message Details

Message 1

<pin-type> pin '<pin-name>' (of '<parent_cell>') is not directly driven by PMU Test Control cell. Instead its fanin has '<cause-description>'

Arguments

- The type of the pin *<pin-type>*
- The name of the pin *<pin-name>*
- Parent cell of the pin *<parent_cell>*
- Cause description *<cause-description>*. This can be one of the following values:
 - hanging net
 - primary port
 - sequential or black box element

- more than one unblocked path
- all paths blocked
- undriven logic

Message 2

power_management_unit -name <pmu-name> constraint ignored as it is put on a black-box

Arguments

- Power Management Unit Name, <pmu-name>

Schematic Highlight

Path from driver to control output of PMU

Rule Severity

Warning

SP_03

Control signal of the ISO cells, Power Switches and Retention cells must be driven by a PMU Test Control cell (PMUWR)

Rule Description

The SP_03 rule reports violation if power control signals of ISO cells, Power Switches and Retention cells are not directly driven by PMU Test Control cell (PMUWR).

Constraints

SGDC

- *power_management_unit* (optional)
- *power_management_test_control_cell* (optional)

UPF

- *create_power_switch* (optional)
- *set_isolation* (optional)
- *set_isolation_control* (optional)
- *set_retention* (optional)
- *set_retention_control* (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Power Ground

Message Details

Power control ('<power_control_signal>') pin '<pin-name>' is not directly driven by PMU Test Control cell. Instead its fanin has '<cause-description>'

Arguments

- Type of power control signal *<power_control_signal>*. This argument can assume one of the following values: `power switch`, `retention control`, and `isolation control`.
- Name of the pin *<pin-name>*
- Cause description *<cause-description>*. This can be one of the following values:
 - hanging net
 - primary port
 - sequential or black box element
 - more than one unblocked path
 - all paths blocked
 - undriven logic

Schematic Highlight

Path from driver to power control signals

Rule Severity

Warning

SP_04

A PMU Test Control Cell (PMUWR) should only drive control signal of ISO cells, Power Switches and Retention cells.

Rule Description

Rule will flag a violation if a PMU Test Control cell (PMUWR) is not directly driving the control signals of ISO cells, Power Switches and Retention cells.

Constraints

SGDC

- *power_management_unit* (optional)
- *power_management_test_control_cell* (mandatory)

UPF

- *create_power_switch* (optional)
- *set_isolation* (optional)
- *set_isolation_control* (optional)
- *set_retention* (optional)
- *set_retention_control* (optional)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Power Ground

Message Details

Output pin '<pin name>' of PMU test control cell '<PMTIC Cell>' is not directly driving control signals (ISO or Power Switches or Retention). Instead its fanout has '<cause-string>'

Arguments

- Name of the pin *<pin-name>*
- Power Management Test Control Cell , *<PMTIC Cell>*

Scan Power Rules

- Cause description *<cause-description>*. This can be one of the following values:
 - hanging net
 - primary port
 - sequential or black box element
 - more than one unblocked path
 - all paths blocked
 - undriven logic

Schematic Highlight

Path from driver to power control signals

Rule Severity

Warning

SP_05

A scan chain containing flip-flops in power management test control modules should be stand alone

Rule Description

The SP_05 rule reports a violation if a scan chain, including power management test control cells, also contains flip-flops that are not in a power management test control cell.

Constraints

- *scan_chain* (mandatory)
- *power_management_test_control_cell* (mandatory)

Rule Parameters

dftUseOffStateOfClockInClockPropagation

Operating Mode

Scanshift

Message Details

Scan chain(*scan_in*=><*scan_in_point*>*scanout*=><*scan_out_point*>)[under *defineTag* <*scan_enable_condition*>] crosses *power_management_test_control_cell* at <*no-of-places*>. Flip-flop count inside PMTC cell is <*count_1*> and outside PMTC cell is <*count_2*>

Schematic Highlight

Scan chain with different pmtc cell segment in different color.

Rule Severity

Warning

SoC Rules

Overview

The SoC rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Flags/Highlights
Soc_00	Generates an abstract view of a design
Soc_04	Simulation results for the conditions specified by a tag
Soc_05	<code>testmode</code> and <code>clock</code> (with <code>-testclock</code>) constraints defined for lower-level modules are also satisfied within a higher-level module
Soc_06	Generates the hierarchical name information for specified design units/pins

Soc_00

Generates an abstract view of a design

When to Use

Use this rule to create an abstract view of a design.

For information on using the SoC abstraction flow, refer to the *SpyGlass SoC Methodology Guide*.

Description

The Soc_00 rule generates a file that contains an abstract view of a design.

Prerequisites

Specify an RTL file to run this rule.

Default Weight

10

Language

Verilog, VHDL

Method

Do all the analysis on the current design top using given constraint file if any. Generate force_ta, test_mode, and clock constraints to capture the current controllability and observability, test_mode values & clock state all the port of design. Then generate the abstract port constraint for each port to show the connectivity of that port to first level flip-flops in the design.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *clock* (optional): Use this constraint to define the clocks of a design.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.
- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

Message 1

[INFO] Abstract view for design '<design-name>' successfully created

Arguments

Name of the design, *<design-name>*

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

Use the generated SGDC file at the SoC level to use abstracted view of the current top.

Message 2

[ERROR] Abstracted sgdc file for module '<module-name>' is not generated, due to presence of unsupported System Verilog constructs in module's interface

Arguments

Name of the system verilog module, *<module-name>*

Potential Issues

There are some System Verilog constructs present in module's interface

which are not supported by spyglass for generation of abstract view.

Consequences of Not fixing

Abstract view for module is not generated.

How to Debug and Fix

To fix the violation message, stop using the module for abstraction.

Message 3

[INFO]' dft_dsm' abstract view for design '<module-name>' is not created due to unavailability of dft_dsm license

Arguments

Name of the system verilog module, <module-name>

Potential Issues

The violation message is reported because of the unavailability of the dft_dsm license.

Consequences of Not fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this message.

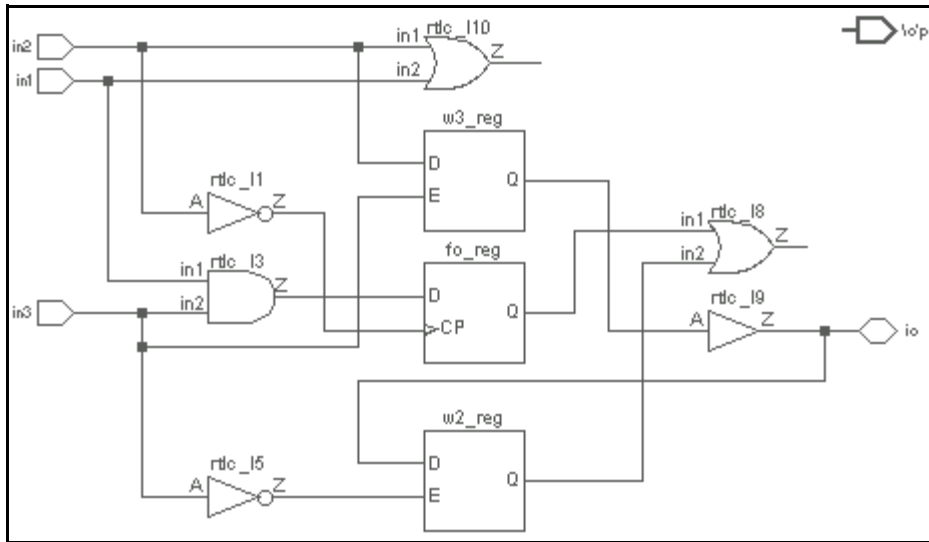
Example Code and/or Schematic

Consider the following SGDC file specification:

```
current_design "ILM_BOX"  
testmode -name in2 -value 1  
clock -name "\o'p " -testclock -atspeed  
force_ta -name in1 -value yyy  
force_ta -name in2 -observe n
```

In addition, consider the following design:

SoC Rules



For the above file specifications, the Soc_00 rule generates a file named ILM_BOX_dft_abstract.sgdc that contains an abstract view of the design, as shown below:

```
current_design "ILM_BOX"
# TEST_MODE Constraints : mode '-scanshift'
test_mode -name "in2" -value 1 -scanshift

# TEST_MODE Constraints : mode '-capture'
test_mode -name "in2" -value 1 -capture

# FORCE_TA Constraint for Controllability
force_ta -name "\o'p [0]" -control yyy
force_ta -name "\o'p [1]" -control yyy
force_ta -name "in1" -control yyy
force_ta -name "in2" -control nyn
force_ta -name "in3" -control yyy
force_ta -name "io" -control yyy

# FORCE_TA Constraint for Observability
force_ta -name "\o'p [0]" -observe y
force_ta -name "\o'p [1]" -observe y
force_ta -name "in2" -observe y
force_ta -name "io" -observe y

# CLOCK Constraints
clock -name "\o'p [0]" -value rtz -testclock
clock -name "\o'p [1]" -value rtz -testclock

# CLOCK Constraints
clock -name "\o'p [0]" -value rtz -atspeed
clock -name "\o'p [1]" -value rtz -atspeed

# ABSTRACT_PORT Constraints
abstract_port -ports "in2" -connected_inst "\ILM_BOX.fo_reg "
inst_master "RTL_FD" -inst_pin "CP" -path_logic buf -path_polarity
buf -mode shift -scope dft

# ABSTRACT_PORT Constraints
abstract_port -ports "in2" -connected_inst "\ILM_BOX.fo_reg "
inst_master "RTL_FD" -inst_pin "CP" -path_logic buf -path_polarity
buf -mode capture -scope dft
```

Default Severity Label

Info

Rule Group

Not applicable

Reports and Related Files

`<top-name>_dft_abstract.sgd`: This file contains an abstract view of a design. This file is generated in the `<current-working-directory>/spyglass_reports/abstract_view/` directory.

Soc_04

Show system state for a given tag.

When to Use

Use this rule to show the system state for each condition specified by the `define_tag` constraint.

Description

The Soc_04 rule shows the simulation results for each condition specified by the `define_tag` constraint.

The state of user-selected nodes, when a set of nodes are in a particular state, is checked. This is useful to verify that port requirements on one or more blocks in a design have required values when a set of nodes have particular values.

Prerequisites

Specify the `define_tag` constraint.

Default Weight

10

Language

Verilog, VHDL

Method

For each set of `define_tag` conditions

Simulate power, ground and the defined node/value pairs

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- `dftShowWaveForm` (optional): The default value is off. Set the value of the parameter to on to enable the generation of waveform corresponding to the rule message in the Waveform viewer of the Atrenta Console.
- `dftUseOffStateOfClockInClockPropagation`: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- *dft_show_unused_define_tag*: The default value is off. Set the value of the parameter to on to enable viewing propagation of unused define_tag through Info_define_tag/Soc_04 rules.

Constraint(s)

define_tag (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

Define_tag

Messages and Suggested Fix

Message 1

[INFO] Tag '<tag-name>' is displayed for design '<du-name>'

Arguments

To view list of arguments, click [Arguments](#).

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message

How to Debug and Fix

The Soc_04 rule assists in debugging the violations reported by the Conn_01 and Conn_02 rules. The Soc_04 rule is an informative rule and requires no debug.

No fix is required as this is an informational rule.

Message 2

[INFO] Tag '<tag name>__as_used_in_scan_chain_tracing__' is displayed for design '<du-name>'

Arguments

- Name of the tag, *<tag-name>*

- Name of the design unit, *<du-name>*

Potential Issues

A violation message is displayed for those *define_tag* tags that are used in *scan_chain* to define scan enable condition (-scanenable field).

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message

How to Debug and Fix

The Soc_04 rule assists in debugging the violations reported by the Conn_01 and Conn_02 rules. The Soc_04 rule is an informative rule and requires no debug.

No fix is required as this is an informational rule.

Message 3

[INFO] Display skipped for '<unused tag count>' unused tag(s): <unused tag list>. Set parameter 'dft_show_unused_define_tag' to 'on' to show unused tags

Potential Issues

"This is an info message which indicates that propagation for unused define_tags was skipped and is not shown.

Consequences of Not Fixing

This is an informational message.

How to Debug and Fix

To see their propagation values, set the *dft_show_unused_define_tag* parameter to on.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

SoC Rules

Rule Group

SoC Rules

Reports and Related Files

No related reports or files.

Soc_05

Performs IP verification for mode, clock, controllability and observability in the SoC flow

When to Use

Use this rule to verify whether the testmode and clock constraints defined for a lower-level black box module also satisfy the current conditions within the current module.

When this rule is run with SpyGlass DFT ADV product, it validates `atspeed_clock_frequency/ testmode / clock` in CaptureAtspeed mode.

For information on using the SoC abstraction flow, refer to the *SpyGlass SoC Methodology Guide*.

Description

The Soc_05 rule verifies following items on lower level block, if specified using `current_design` for non-top:

- `test_mode` condition in `shift` and `capture` mode, using `test_mode` constraint.

NOTE: *Specifying 'X' for value check, that is, `test_mode -value X` on a pin of a lower block, implies that any current value should match with the required value and no mismatch violation should come for that pin.*

- `test clock` reachability in `shift` mode, using `clock` constraint.
- Controllability and Observability condition in `capture` mode, using `force_ta` constraint.

Prerequisites

Specify `current_design` for lower level block with desired constraints, such as, `test_mode`, `clock`, and `force_ta`.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate mode (`shift / capture / at_speed`)

Verify that all unit modules (these are instances of modules with SGDC files) have `test_mode` values, as specified in their SGDC, satisfied.

Simulate clock tree for each top level block. Verify that all the clock thus simulate matches with the clock constraint specified using the supplied SGDC file.

Do Controllability and Observability analysis.

Verify controllability and observability values for all boundary pins are in agreement with constraint provided in the SGDC file. Observability check is skipped for input ports and controllability check is skipped for output ports.

Error message is shown for every mismatch found.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is `on`. Set the value of the parameter to `off` so that clock lines are kept at X during shift, capture, or `atspeed` mode simulation.
- *dft_soc_strict_boundary_validation*: The default value of the parameter is `off`. Set the value of the parameter to `on` so that a violation message of severity Error is reported for a mismatch in the values of clock and simulation modes at the IP boundary.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.
- *atspeed_clock_frequency* (optional): Use this constraint to specify frequencies associated with a testclock.

Operating Mode

Scanshift, Capture

Message Details

Message 1

[ERROR] <mode-name> simulation value at pin '<pin-name>' of instance '<inst-name>' (Master: '<mdu-name>') is '<sim_value1>' (Xpt: '<sim_value2>')

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported due to incorrect / incomplete [test_mode](#) constraints.

Consequences of Not Fixing

Not fixing the violation results in unsatisfied boundary conditions.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[ERROR] <mode-name> Clock pattern at pin '<pin-name>' of instance '<inst-name>' (Master: '<mdu-name>') is '<clk_value1>' (Xpt: '<clk_value2>')

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported due to incorrect / incomplete [test_mode](#) or [clock](#) constraints.

Consequences of Not Fixing

Not fixing the violation results in unsatisfied boundary conditions for clock.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[ERROR] <Control / observe> value at pin '<pin-name>' of instance '<inst-name>' (Master: '<mdu-name>') is

' <cnt_obs_val ue1>' (Xpt: ' <cnt_obs_val ue2>')

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported due to incorrect / incomplete [test_mode](#) or [clock](#) constraints.

Consequences of Not Fixing

Not fixing the violation results in unsatisfied boundary conditions for clock.

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

[ERROR] Input frequency ' <atspeed_frequency>' on pin ' <pin-name>' of instance ' <inst-name>' (Master: ' <mdu-name>') has enable condition mismatch. Actual ' <en_val ue1>' (Xpt : ' <en_val ue2>')

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported when incorrect or incomplete [atspeed_clock_frequency](#) constraint is specified.

Consequences of Not Fixing

Not fixing this violation results in unsatisfied boundary conditions for [atspeed_clock_frequency](#).

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 5

[ERROR] Pin ' <pin-name>' of instance ' <inst-name>' (Master: ' <mdu-name>') has missing frequency(s) ' <m_freq>'. Actual : ' <freq1>' (Xpt : ' <freq2>')

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported when incorrect or incomplete [atspeed_clock_frequency](#) constraint is specified.

Consequences of Not Fixing

Not fixing this violation results in unsatisfied boundary conditions for [atspeed_clock_frequency](#).

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 6

```
[ERROR] Pin '<pin-name>' of instance '<inst-name>' (Master: '<mdu-name>') has extra frequency(s) '<extra_frequencies>'. Actual: '<freq1>' (Xpt: '<freq2>')
```

Arguments

To view the list of message arguments, click [Arguments](#).

Potential Issues

A violation is reported when incorrect or incomplete [atspeed_clock_frequency](#) constraint is specified.

Consequences of Not Fixing

Not fixing this violation results in unsatisfied boundary conditions for [atspeed_clock_frequency](#).

How to Debug and Fix

For information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 7

```
[INFO] Skipping <observability / controllability> verification at pin '<pin-name>' as it is 'Input / Output' of instance '<inst-name>' (Master: '<mdu-name>')
```

Arguments

- Name of the rule-violating terminal. <pin-name>

- Name of the instance containing the rule-violating terminal <pin-name>. (<inst-name>)
- Name of the master module of the instance <inst-name> containing the rule-violating terminal <pin-name>. (<mdu-name>)
- Actual and expected values for simulation, (<sim_value1> and <sim_value2>)
- Actual and expected values for clock, (<clk1_value1> and <clk_value2>)
- Actual and expected values for controllability and observability, (<cnt_obs_value1> and <cnt_obs_value2>)
- Actual and expected enable values (<en_value1> and <en_value2>)
- Missing frequency, (<m_freq>)
- Actual and expected frequencies, (<freq1> and <freq2>)
- Actual and expected clock patterns. (<pattern1> and <pattern2>)

Potential Issues

A violation is reported due to mismatch in the *test_mode*, *clock*, and *force_ta* constraints.

Consequences of Not Fixing

Not fixing the violation results in unsatisfied boundary conditions for lower level block.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the terminal, which has mismatched (actual vs. specified) *atspeed_clock_frequency/testmode/testclock/controllability/observability* simulation.

Consider the following checks to debug the violations reported by the Soc_05 rule:

- If the violation message is about testmode simulation mismatch, overlay the Info_testmode (in auxiliary mode) rule under required mode to see how testmode simulation is distributed.
- If the violation message is about shift mode clock pattern simulation mismatch, overlay the Info_testclock (in auxiliary mode) under shift mode to see how testclock simulation is distributed.

- If the violation message is about controllability pattern mismatch, overlay the Info_uncontrollable (in auxiliary mode) to see the controllability of all nets that are not fully controllable.
- If the violation message is about observability pattern mismatch, overlay the Info_unobservable (in auxiliary mode) to see observability of all the pins that are not observable.
- If the violation message is about frequency mismatch, overlay the Info_atSpeedFrequency for a particular frequency to see the propagation of the frequency in the design.

You can also view the violations for the Info_testmode (under shift condition) and Info_testclock rules along with the violation of the Soc_05 rule in the Incremental Schematic window. To do this, double-click the violation for the Soc_05 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_testclock rules overlap the violation message for the Soc_05 rule in the Incremental Schematic window. This is useful in debugging the violation for the Soc_05 rule.

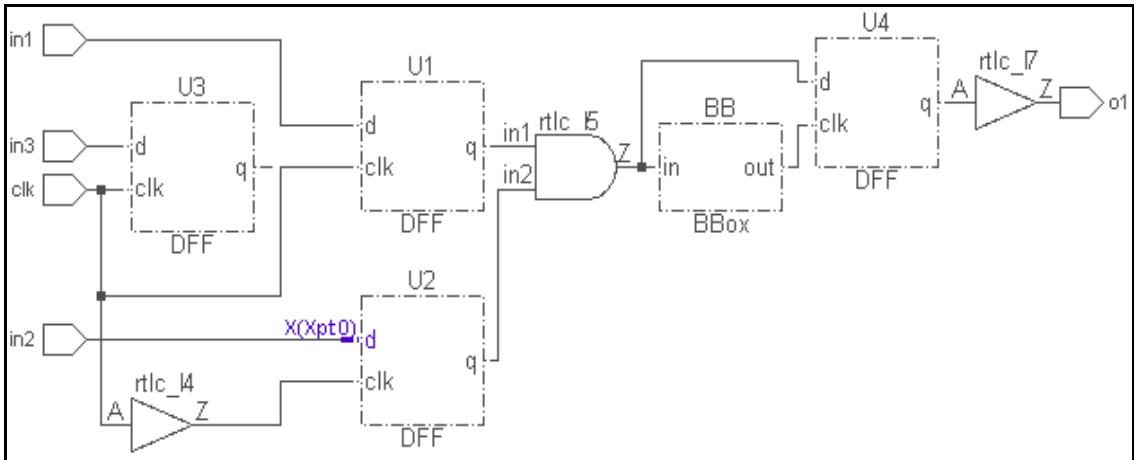
To fix the violation, see [Example Code and/or Schematic](#).

Example Code and/or Schematic

Consider the following SGDC file snippet:

```
current_design "DFF"  
    test_mode -name "d" -value 0 -scanshift
```

In addition, consider the following design:



In this case, the following violations are reported by the Soc_05 rule:

- ❌ Shift Mode simulation value at pin 'd' of instance 'top.U2' (Master: 'DFF') is 'X' (Xpt: '0'), test.sgdc, 10
- ❌ Shift Mode simulation value at pin 'd' of instance 'top.U3' (Master: 'DFF') is 'X' (Xpt: '0'), test.sgdc, 10
- ❌ Shift Mode simulation value at pin 'd' of instance 'top.U4' (Master: 'DFF') is 'X' (Xpt: '0'), test.sgdc, 10

Default Severity Label

Error

Rule Group

SoC

Reports and Related Files

No related reports or files.

Soc_06

Provides the hierarchical names of all instances appended with pin name.

When to Use

Use this rule to get the list of hierarchical names of all instances.

Description

The Soc_06 rule generates the hierarchical name information for specified design units/pins.

You need to specify constraints, such as, `require_value` and `require_path`, that can apply to endpoints, which are pins of some instances in the design. It is, thus, useful to be able to find the hierarchical names of instances of a user-specified module.

The Soc_06 rule requires you to specify the design units and their pins (optional) using the `module_pin` constraint. Then, the Soc_06 rule generates a report named `soc_06_rpt` that has the hierarchical names of all instances of design units (specified with the `-name` argument of the `modulePin` constraint) appended with pin names, which can be specified by using any of the following four options available in the `modulePin` constraint:

- `-pin`
- `-allpins`
- `-allinputs`
- `-alloutputs`

NOTE: *The Soc_06 rule runs at Nom Level.*

Method

Iterate over all the VSTOPDU's.

Read the `modulePin` constraint for the design unit.

Push the constraint information in a list.

Find all the instances in the module

Traverse on the instance list

Find the master of the instance.

If the master has been specified through the `modulePin` constraint, store this instance.

Go to Step 4

Dump all the information in the output file (dftHierarchicalNames report).

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

module_pin (optional): Use this constraint to specify the set of pins for which hierarchy of all instantiations is to be generated.

Operating Mode

None

Messages and Suggested Fix

Message 1

[INFO] File <file-name> is generated which contains all instances of the specified module-pin

You can use the generated names in other constraints like *require_value*, *require_path*, etc. that require hierarchical names for endpoints that are pins of some instances in the design.

Arguments

Name of the generated file, <file-name>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, no debug and fix information is required for this violation message.

Message 2

[WARNING] The port '<pin-name>' on module '<du-name>' does not exist in design <design-name>

Arguments

- Name of the port, <pin-name>
- Name of the module, <du-name>

Potential Issues

The violation message appears if specified pin does not exist in the design.

Consequences of Not Fixing

Not fixing this violation may result in error in the generated report.

How to Debug and Fix

The Soc_06 rule does not require any debug information.

To fix the violation, specify the correct pin name.

Example Code and/or Schematic

Consider the following example:

```
module top;
    middle middle_inst();
endmodule
module middle;
    lower lower_inst();
endmodule
module lower(A, B);
    input A;
    output B;
```



```
endmodule
```

The SOC_06 rule provides the following information based on the following inputs:

Input

The name of a module = lower

Returns

The name of all the instances as follows:

```
"top.middle_inst.lower_inst"
```

Input

The name of the module = lower

The name of the pin = A

Returns

The hier names of all instances appended with pin name:

```
"top.middle_inst.lower_inst.A"
```

Default Severity Label

Warning

Rule Group

SoC

Reports and Related Files

soc_06_rpt: This report lists the hierarchical name of all the instances appended with a pin name. The `soc_06.rpt` file is generated in the current working directory.

Testability Analysis Rules

Overview

The `Testability` rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Flags...
TA_01	Nets where controllability can be improved by adding test-points
TA_02	Nets where observability can be improved by adding test-points
TA_05	This rule has been deprecated.
TA_06	Generates a test coverage report when all SpyGlass DFT ADV-suggested test points are selected
TA_07	Testmode signals that are only controlling asynchronous set/reset pins and are restricted during capture
TA_08	Test enable/enable pins of clock gating cells that are tied to an undesired value
TA_09	Causes of uncontrollability or unobservability and estimates the number of nets whose controllability/observability is impacted.
TA_10	Add test points to improve controllability and observability.

TA_01

Add test points to improve controllability.

When to Use

Use this rule to identify the nets that are uncontrollable. It also identifies the reason for uncontrollability.

Description

The TA_01 rule reports violation for candidate nets for a test-point to improve controllability.

NOTE: *The TA_01 rule will be deprecated in a future release. Use the [TA_09](#) rule instead.*

Default Weight

10

Language

Verilog, VHDL

Method

Perform scannability analysis.

Initialize controllability analysis with all primary inputs are set to controllable for 0, 1 and z and the q-outputs of all scannable flip-flops set to controllable for 0 and 1.

Make design full-scan-ready.

Perform controllability analysis. Report all causes for un-controllability. All causes of bad controllability found by walking the non-controllable fan-in cone until either a device with uncontrollable or unconnected inputs or a closed loop is detected. Nodes with non-x simulation values are excluded.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [noBlackBoxReporting](#): The default value is off. Set the value of the parameter to on to suppress the violation messages for nets made uncontrollable or unobservable by black boxes.
- [incrementalTA](#): The default value is off. Set the value of the parameter to on to disable the incremental fault analysis.

- *dftSkipPwrGndNetsForTA*: The default value is on. Set the value of the parameter to off to disable violations for those nets that are not controllable due to power/ground nets.
- *dftSortTAMsgOnFaultCount*: The default value is off. Set the value of the parameter to on to sort the violation messages based on fault improvement count, when the uncontrollable and unobservable nets are made controllable and observable respectively.
- *dftSkipTAMsgCount*: The default value is off. Set the value of the parameter to any positive integer number to ignore those nets that result in improvement count less than the specified value.
- *dftSkipUnObservableNetsForTA*: The default value is off. Set the value of the parameter to on to ignore unobservable nets.
- *dftSkipNonXValueNetsForTA*: The default value is on. Set the value of the parameter to off to ignore nets with a non-X value.
- *schematicForAllBits*: The default value is off. Set the value of the parameter to on to generate highlight information for all bits of the rule-violating bus.
- *mergeN*: The default value is -1. Set the value of the parameter to any positive integer number to set a limit on the number of bit-slices that should be merged, and hence, reported together.
- *dftTreatFlipFlopsAsScan*: The default value is off. Set the value of the parameter to on to treat all flip-flops as scannable for the TA_01 rule.
- *dftTreatLatchesAsTransparent*: The default value is off. Set the value of the parameter to on to treat all latches as transparent for the TA_01 rule.
- *dftTreatBBoxAsScanwrapped*: The default value is off. Set the value of the parameter to on to treat all black boxes as scan-wrapped for the TA_01 rule.
- *dftUUMarking*: The *dftUUMarking* parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.

NOTE: When you set *dftTreatFlipFlopsAsScan*, *dftTreatLatchesAsTransparent*, and *dftTreatBBoxAsScanwrapped* parameters, the TA_01 rule runs under "testmode + full-scan-ready conditions". However, note that by default, the TA_01 rule runs under capture condition. For details, see [Scan-Ready Design Conditions for Testability Rules](#) section.

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *force_ta* (mandatory): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared as scannable even if they qualify so.

NOTE: *The TA_01 rule does not require special input but does make use of additional data when available. The testmode and testclock information are especially valuable in maximizing the number of scannable flip-flops and therefore maximizing testability.*

Operating Mode

Capture

Messages and Suggested Fix

Message 1

If the *incrementalTA* rule parameter is not set:

[WARNING] Net '<net-name>' [in <du-name>] is not controllable to <value> [<reason>]

Arguments

To view the arguments list for this message, click [Arguments](#).

Potential Issues

To know more about potential related to this violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

If the *incrementalTA* rule parameter is set and the *dftSortTAMsgOnFaultCount* rule parameter is not set:

```
[WARNING] Net '<net-name>' [in <du-name>] is not controllable to <cvalue> [<reason>]. Adding a test-point [Cnt = <tp-value> ] will make <num1> fault detectable [%Increase <num2>][[Controllability Improvement = '<num3>']]
```

Arguments

To view the arguments list for this message, click [Arguments](#).

Potential Issues

To know more about potential related to this violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

If both the *incrementalTA* rule parameter and the *dftSortTAMsgOnFaultCount* rule parameter are set:

```
[WARNING] Net '<net-name>' [in <du-name>] is not controllable to <cvalue> [<reason>]. Adding a test-point [Cnt = <tp-value> ] will make <num1> nets controllable [Fault Improvement = '<num2>' [%Increase <num3>]]
```

Arguments

To view the arguments list for this message, click [Arguments](#).

Potential Issues

To know more about potential related to this violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click

Consequences of Not Fixing.

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

When the TA_01 rule is run, the following message appears:

[WARNING] Testability (control / observe) analysis is being done under 'full-scan-ready' condition.

Arguments

- Name of the rule-violating net. (<net-name>)
- Name of the design unit containing the rule-violating net <net-name>
- Controllability value. <cvalue>
The controllability values are any combination of 0, 1, or Z.
- Reason for un-controllability. <reason>. Possible values are as follows:

affected by other input(s)	hanging
combinational-loop	input tied to power
input tied to ground	test-mode value = '0'
multiply driven	unconnected-input
test-mode value = '1'	
un-driven	

- Test point values.(<tp-value>). Possible values are as follows:

Controllable to Values	Test-point
0 or 1	Cnt = nn- ->yyn
0	Cnt = ny- ->yyn
1	Cnt = yn- ->yyn
Z	Cnt = yyn ->yyy

- Number of additional faults that become detectable or number of additional nets that become controllable by adding the specified testpoint. (<num1>)
- Additional percentage improvement in fault detection. (<num2>)
- Additional percentage improvement in controllability. (<num3>)

Potential Issues

Controllability is a necessary requirement for high fault coverage since ATPG tools cannot successfully process nodes that are not controllable. The TA_01 rule is necessary but not sufficient to guarantee easy ATPG because, in some cases, test points to improve observability may also be added.

Following are some of the causes of uncontrollability:

- Non-scannable flip-flops
- Feedback loops such as non-scannable clock dividers that do not have asynchronous means to initialize the loop.
- Black box outputs are declared as uncontrollable since nothing can be deduced regarding their internal structure.
- Unconnected device input pins
- Devices with inputs connected to power or ground.

Consequences of Not Fixing

Not fixing the violation may result in bad controllability. This in turn reduces testability of design thus reducing the coverage.

How to Debug and Fix

Consider the following checks to debug the violations reported by the TA_01 rule:

- If the TA_01 rule flags a violation because of non-scan flip-flops, debug the violation of the Async_07 and Clock_11 rules.
- If the TA_01 rule flags a violation because of black boxes, debug the violation of the InferBlackBox rule.
- If the TA_01 rule flags a violation because of non-transparent latches, debug the violation of the Latch_08 rule.
- If the TA_01 rule flags a violation because of undriven nets, view the Incremental Schematic of the violation message to see the undriven net.

- If the TA_01 rule flags a violation because of combinational loop, debug the violation of the Topology_01 rule.
- If the TA_01 rule flags a violation because of the constant value, view the Incremental Schematic of the violation message. Also, overlay the Info_testmode rule (auxiliary violation mode) under capture mode to check how the constant value has propagated.

To fix the violation, see Example Code and /or schematic.

Message Sorting

The TA_01 rule uses the following approach for sorting:

For each root cause:

1. SpyGlass computes two numbers.
 - Incremental gain in controllability.
 - Incremental gain in detectable fault count.
2. Then, system chooses the sorting technique based on one of the above mentioned criteria. If `dftSortTAMsgOnFaultCount` parameter is set as `on`, second criteria is used. Otherwise, the first one is used.

However, firstly all the messages of the same root cause are clubbed together and then sorting is done on these clubbed messages using any of the above mentioned criteria.

After sorting, the TA_01 rule message looks like:

Cause 1

List of sorted messages

Cause 2

List of sorted messages

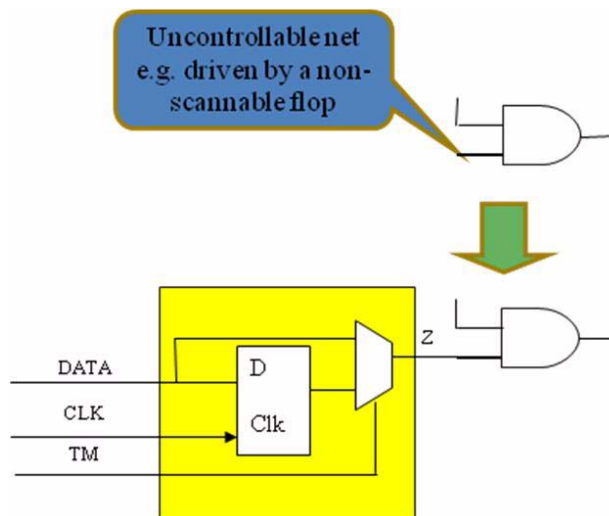
Cause3

List of sorted messages

Please note that in this rule, complete message list is not sorted. The messages with the same root cause are sorted separately.

Example Code and/or Schematic

Consider the following figure:



Verilog

Consider the following Verilog code:

```
...
always @ (posedge clk)
  q <= not(q);
...
```

The above example illustrates a bad design scenario and the TA_01 rule reports a violation for the above example.

To remove the violation, modify the above code as shown below:

```
...
always @ (posedge clk) begin
  if (reset)
    q <= 0;
  else
    q <= not(q);
end
...
```

VHDL

Consider the following VHDL code:

```

...
Process(clk)
Begin
  q <= not q;
End process;
...

```

The above example illustrates a bad design scenario and the TA_01 rule reports a violation for the above example.

To remove the violation, modify the above code as shown below:

```

...
Process(clk)
Begin
  If (reset) then
    q <= '0';
  Else
    q <= not q;
  End if;
End process;
...

```

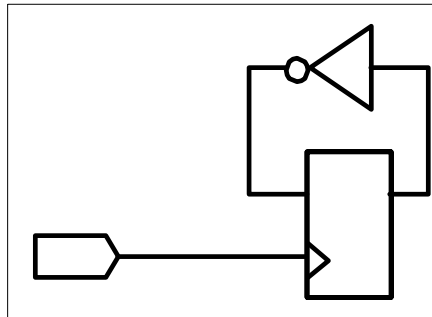


FIGURE 92. TA_01 Incremental Schematic

Schematic highlight

Instance and its output terminal to which the net that is root cause of bad controllability is connected

You can also view the violations for the Info_testmode and

Info_uncontrollable rules along with the violation of the TA_01 rule in the Incremental Schematic window. To do this, double-click the violation for the TA_01 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_uncontrollable rules overlap the violation message for the TA_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the TA_01 rule.

Default Severity Label

Warning

Rule Group

Testability Analysis Rules

Reports and Related Files

No related reports or files.

TA_02

Select test points to improve observability

When to Use

Use this rule to identify the nets that are not observable. It also identifies the reason for unobservability.

Description

The TA_02 rule reports violation for candidate nets for a test-point to improve observability.

NOTE: *The TA_02 rule will be deprecated in a future release. Use the [TA_09](#) rule instead.*

Default Weight

10

Language

Verilog, VHDL

Method

Perform scannability analysis.

Initialize controllability analysis with all primary inputs are set to controllable for 0, 1 and z and the q-outputs of all scannable flip-flops set to controllable for 0 and 1.

Perform controllability analysis.

Make design full-scan-ready.

Initialize observability with all primary outputs and the d-input to all scannable flip-flops as observable.

Perform observability analysis. Report all causes for unobservability. Nodes with non-x simulation values are excluded.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *noBlackBoxReporting*: The default value is off. Set the value of the parameter to on to suppress the violation messages for nets made uncontrollable or unobservable by black boxes.
- *incrementalTA*: The default value is off. Set the value of the parameter to on to disable the incremental fault analysis.

- *dftSkipPwrGndNetsForTA*: The default value is on. Set the value of the parameter to off to disable violations for those nets that are not controllable due to power/ground nets.
- *dftSortTAMsgOnFaultCount*: The default value is off. Set the value of the parameter to on to sort the violation messages based on fault improvement count, when the uncontrollable and unobservable nets are made controllable and observable respectively.
- *dftSkipTAMsgCount*: The default value is off. Set the value of the parameter to any positive integer number to ignore those nets that result in improvement count less than the specified value.
- *schematicForAllBits*: The default value is off. Set the value of the parameter to on to generate highlight information for all bits of the rule-violating bus.
- *mergeN*: The default value is -1. Set the value of the parameter to any positive integer number to set a limit on the number of bit-slices that should be merged, and hence, reported together.
- *dftTreatFlipFlopsAsScan*: The default value is off. Set the value of the parameter to on to treat all flip-flops as scannable for the TA_01 rule.
- *dftTreatLatchesAsTransparent*: The default value is off. Set the value of the parameter to on to treat all latches as transparent for the TA_01 rule.
- *dftTreatBBoxAsScanwrapped*: The default value is off. Set the value of the parameter to on to treat all black boxes as scan-wrapped for the TA_01 rule.
- *dftUUMarking*: The *dftUUMarking* parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.

NOTE: When you set *dftTreatFlipFlopsAsScan*, *dftTreatLatchesAsTransparent*, and *dftTreatBBoxAsScanwrapped* parameters, the TA_02 rule runs under "testmode + full-scan-ready conditions". However, by default, the TA_02 rule runs under capture condition. For details, see [Scan-Ready Design Conditions for Testability Rules](#) section

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *force_ta* (mandatory): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared as scannable even if they qualify so.

NOTE: *The TA_02 rule does not require special input but does make use of additional data when available. Testmode and testclock information are especially valuable in maximizing the number of scannable flip-flops and therefore maximizing testability.*

Operating Mode

[Capture](#)

Messages and Suggested Fix

Message 1

If the [incrementalTA](#) rule parameter is not set:

[WARNING] Net '<net-name>' [in <du-name>] is not observable to <value> [<reason>]

Arguments

To view the arguments list for this message, click [Arguments](#).

Potential Issues

To know more about potential related to this violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

If the *incrementalTA* rule parameter is set and the *dftSortTAMsgOnFaultCount* rule parameter is not set:

[WARNING] Net '<net-name>' [in <du-name>] is not observable to <ovalue> [<reason>]. Adding a test-point [Obs = y] will make <num1> fault detectable [%Increase <num2>][[Observability Improvement = '<num3>']]

Arguments

To view the arguments list for this message, click [Arguments](#).

Potential Issues

To know more about potential related to this violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

If both the *incrementalTA* rule parameter and the *dftSortTAMsgOnFaultCount* rule parameter are set:

[WARNING] Net '<net-name>' [in <du-name>] is not controllable to <cvalue> [<reason>]. Adding a test-point [Cnt = <tp-value>] will make <num1> nets observable [Fault Improvement = '<num2>' [%Increase <num3>]]

Arguments

To view the arguments list for this message, click [Arguments](#).

Potential Issues

To know more about potential related to this violation, click [Potential Issues](#).

Consequences of Not Fixing

To know more about consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 4

When the TA_02 rule is run, the following message appears:

[WARNING] Testability (control / observe) analysis is being done under 'full-scan-ready' condition.

Arguments

- Name of the rule-violating net. (<net-name>)
- Name of the design unit containing the rule-violating net <net-name>
- Observability value. <ovalue>
The Observability values are any combination of 0, 1, or Z.
- Reason for unobservability. <reason>

Possible values are as follows:

affected by other input(s)	feeds a tri-state enable
hanging	input tied to power
input tied to ground	test-mode value = '1'
test-mode value = '0'	combinational-loop
unconnected-input	un-driven
input net of disabled tristate(s)	driven by non-transparent latch(es)
input net of non-transparent latch(es)	driven by un-scannable flip-flop(s)
input net of un-scannable flip-flop(s)	input net of non scan-wrapped black box(es)
black box driven	multiply driven

- Number of additional faults that become detectable or number of additional nets that become controllable by adding the specified testpoint. (<num1>)
- Additional percentage improvement in fault detection. (<num2>)
- Additional percentage improvement in observability. (<num3>)

Potential Issues

Observability is a necessary requirement for high fault coverage since ATPG

tools cannot successfully process nodes that are not observability. The TA_02 rule is necessary but not sufficient to guarantee easy ATPG because, in some cases, test points to improve controllability may also have to be added.

Consequences of Not Fixing

Not fixing the violation may result in bad observability. This reduces the fault detection (coverage).

How to Debug and Fix

Consider the following checks to debug the violations reported by the TA_02 rule:

- If the TA_02 rule flags a violation because of non-scan flip-flops, debug the violation of the Async_07 and Clock_11 rules.
- If the TA_02 rule flags a violation because of black boxes, debug the violation of the InferBlackBox rule.
- If the TA_02 rule flags a violation because of non-transparent latches, debug the violation of the Latch_08 rule.
- If the TA_02 rule flags a violation because of tristates, view the Incremental Schematic of the violation message to see the tristate.
- If the TA_02 rule flags a violation because of the constant value, view the Incremental Schematic of the violation message. Also, overlay the Info_testmode rule (auxiliary violation mode) under capture mode to check how the constant value is affecting the observability of the net mentioned in the violation.

To fix the violation, see [Example Code and/or Schematic](#).

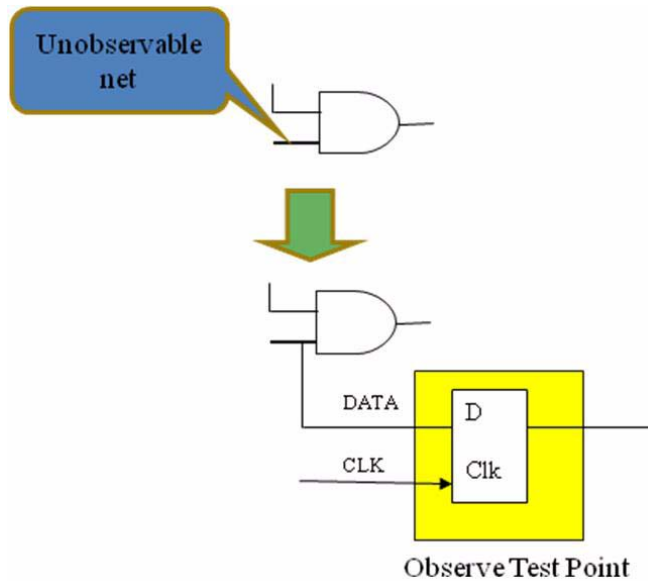
NOTE: *It is recommend to fix the TA_01 rule violations before the fixing the violation reported by the TA_02 rule.*

Message Sorting

The TA_02 rule uses the same approach as TA_01 rule for sorting. Please refer to the Message Sorting section of the [TA_01](#) rule for details.

Example Code and/or Schematic

Consider the following example:



Schematic highlight

Instance and its input terminal to which the net that is root cause of bad observability is connected

You can also view the violations for the Info_testmode (under capture condition) and Info_unobservable rules along with the violation of the TA_02 rule in the Incremental Schematic window. To do this, double-click the violation for the TA_02 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_unobservable rules overlap the violation message for the TA_02 rule in the Incremental Schematic window. This is useful in debugging the violation for the TA_02 rule.

Default Severity Label

Warning

Rule Group

Testability Analysis Rules

Reports and Related Files

No related reports or files.

TA_05

This rule has been deprecated.

TA_06

Generate testcoverage when all SpyGlass DFT ADV suggested test points are selected.

When to Use

Use this rule to identify terminals or ports that are uncontrollable or unobservable after all suggested testpoints are selected.

Description

The TA_06 rule generates a test coverage report when all SpyGlass DFT ADV suggested test points are selected. The report also includes steps to make a design full-scan-ready.

NOTE: *The TA_06 rule will be deprecated in a future release. Use the [TA_09](#) rule instead.*

Rule Exceptions

■ Following are the exceptions to this rule:

The TA_06 supports the AutoFix feature of SpyGlass. However, while using the non-selective mode of AutoFix, the violations of this rule are not removed but corresponding testpoints from the report file are inserted in the modified RTL file. For details of the AutoFix feature, see [dftAutoFix](#) rule parameter.

■ Please note the following points:

- If you specify only the `dftSkipTAMsgCount` parameter, both the [TA_06](#) and [TA_09](#) rules use it.
- If you specify both the `dftSkipTAMsgCount` and [dftSkipTestPointsOnImprovementLessThan](#) parameters, the [TA_06](#) rule uses the former parameter and the [TA_09](#) rule uses the latter one.

Default Weight

10

Language

Verilog, VHDL

Auto Test Point Insertion

The Test Points are inserted on unobservable or uncontrollable nets reported by the TA_06 rule. These are described below:

- For unobservable nets (as shown in [Figure 93](#)), an observe point is inserted. The observe point is a module (the module structure is shown in [Figure 94](#)) having a hanging output.

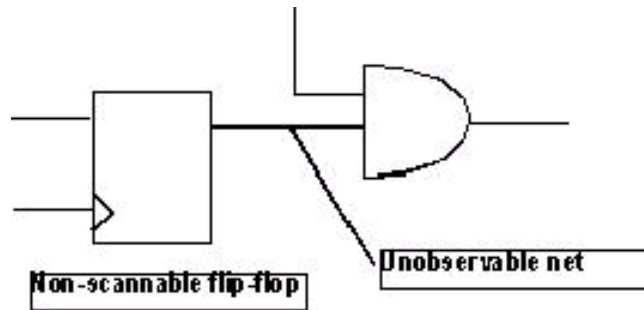


FIGURE 93. Unobservable Net

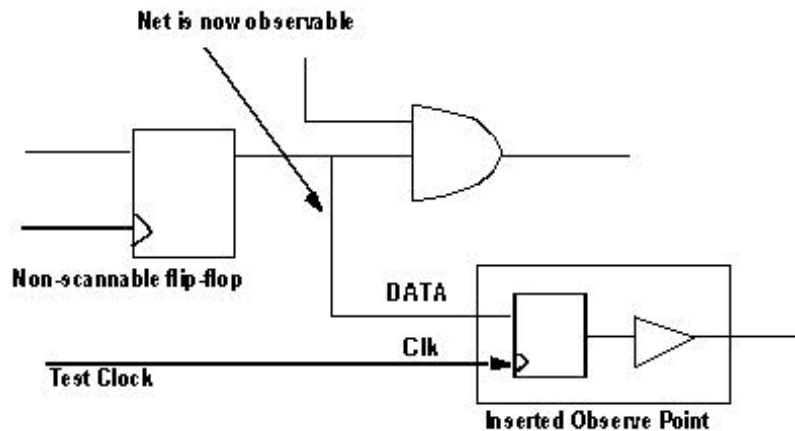


FIGURE 94. Observable Net

- For uncontrollable nets (as shown in [Figure 95](#)), a control point is inserted (the module structure is shown in [Figure 96](#)). The module renders the logic in the fan-out cone of the control point controllable. You cannot completely depend on the violation of the TA_06 rule for

control points. However, you can see a definite increase in the coverage figures

NOTE: An additional flip-flop is inserted to observe the 'SEL' of the multiplexer. This is required when the multiplexer inputs are not controllable to opposite values. In this case the output of the flip-flop will hang.

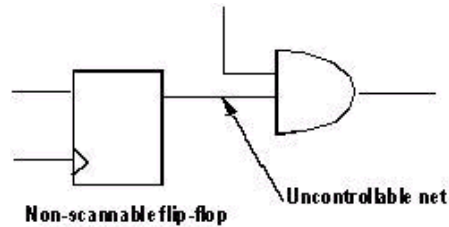


FIGURE 95. Uncontrollable Net

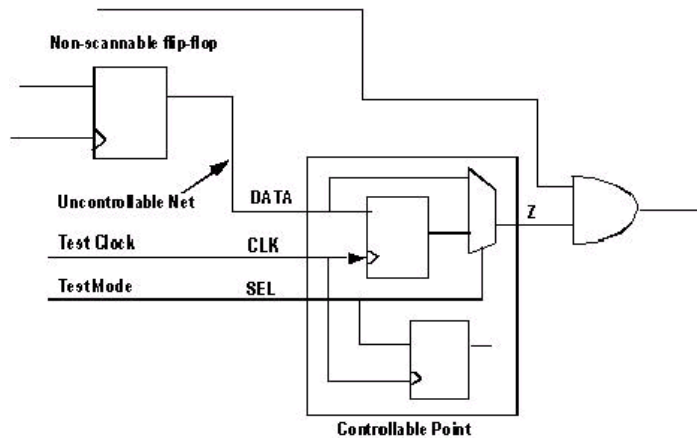


FIGURE 96. Still an Controllable Net

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)

- *noBlackBoxReporting*: The default value is off. Set the value of the parameter to on to suppress the violation messages for nets made uncontrollable or unobservable by black boxes.
- *incrementalTA*: The default value is off. Set the value of the parameter to on to disable the incremental fault analysis.
- *dftSkipPwrGndNetsForTA*: The default value is on. Set the value of the parameter to off to disable violations for those nets that are not controllable due to power/ground nets.
- *dftSkipTAMsgCount*: The default value is off. Set the value of the parameter to any positive integer number to ignore those nets that result in improvement count less than the specified value.
- *dftSkipUnObservableNetsForTA*: The default value is off. Set the value of the parameter to on to ignore unobservable nets.
- *dftSkipNonXValueNetsForTA*: The default value is on. Set the value of the parameter to off to ignore nets with a non-X value.
- *dftAutoFix*: The default value of the parameter is as follows:

```
"{+RULES[nothing]+TMPORT[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}"
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
"{+RULES[<rulelist>]+TMPORT[tmport]+TCLKPORT[tclkport]}"
```
- *dftPOSDETECT_credit*: The default value is 0. Set the value of the parameter to a floating-point number between 0 and 1 to regulate the effect of potentially detectable faults on the test coverage and fault coverage evaluation.
- *dftTreatFlipFlopsAsScan*: The default value is off. Set the value of the parameter to on to treat all flip-flops as scannable for the TA_06 rule.
- *dftTreatLatchesAsTransparent*: The default value is off. Set the value of the parameter to on to treat all latches as transparent for the TA_06 rule.
- *dftTreatBBoxAsScanwrapped*: The default value is off. Set the value of the parameter to on to treat all black boxes as scan-wrapped for the TA_06 rule.

- *dftUUMarking*: The *dftUUMarking* parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.

NOTE: *When you set *dftTreatFlipFlopsAsScan*, *dftTreatLatchesAsTransparent*, and *dftTreatBBoxAsScanwrapped* parameters, the TA_06 rule runs under “testmode + full-scan-ready” conditions. However, by default, the TA_06 rule runs under capture condition. For details, see [Scan-Ready Design Conditions for Testability Rules](#) section.*

- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *force_ta* (optional): Use this constraint to specify the controllabilities and/or observabilities for ports/pins/nets.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared as scannable even if they qualify so.
- *module_bypass* (optional): Use this constraint to specify modules such as memories that are designed with a bypass between data-in port and data-out port.
- *scan_type* (optional): Use this constraint to specify the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- *scan_wrap* (optional): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.
- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.
- *dont_touch* (optional): Use this constraint to specify the modules/nets that are not considered for AutoFix.

Operating Mode

Capture

Messages and Suggested Fix

The TA_06 rule reports the following violation messages:

Message 1

The following violation message is displayed for the TA_06 rule:

```
[INFO] Test points selection report file <report_path> for module '<du-name>' is generated
```

Arguments

- Name of the design unit, <du-name>
- Path in which the report is generated, <report_path>

Message Sorting

For TA_06 rule, all the messages of TA_01 and TA_02 rules are merged and then the list is sorted based on the user-specified sorting criteria (Incremental gain in controllability or Incremental gain in detectable fault count).

Please refer to the Message Sorting section of the [TA_01](#) rule for details.

Potential Issues

Since this is an informational rule, there are no potential issues related to this violation message.

Consequences of Not Fixing

Since this is an informational rule, there is no implicit impact of this violation message.

How to Debug and Fix

The TA_06 rule generates a text report of all the testpoints found in design and their effect on coverage number.

The schematic for the TA_06 rule highlights the terminals/ports that are uncontrollable/unobservable after all suggested testpoints are selected.

You can also view the violations for the Info_testmode (under capture condition), Info_uncontrollable, and Info_unobservable rules along with the violation of the TA_06 rule in the Incremental Schematic window. To do this, double-click the violation for the TA_06 rule and open the Incremental

Schematic window.

The violation messages for the Info_testmode, Info_uncontrollable, and Info_unobservable rules overlap the violation message for the TA_06 rule in the Incremental Schematic window. This is useful in debugging the violation for the TA_06 rule.

Consider the following checks to debug the violations reported by the TA_06 rule:

- If the testpoint mentioned is of type control point, debug the corresponding TA_01 violation.
- If the testpoint mentioned is of type observe point, debug the corresponding TA_02 violation.

Message 2

[INFO] Suggested test points for autofix to improve fault coverage

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Testability Analysis Rules

Reports and Related Files

[test_points_selected](#) report

TA_07

Ensure that the testmode signals that only control asynchronous set or reset pins are unrestricted during capture.

When to Use

Use this rule when you want control signals for asynchronous set/reset.

Description

The TA_07 rule reports violation for testmode signals that are only controlling asynchronous set/reset pins and are restricted during capture.

The logic driving the set or reset pins should be designed so that the sets and resets can be controlled from a single root-level pin. This allows root-level pins to be operated as testresets, after a scan is complete, by the ATPG tool. If the set/reset logic is controlled by a single testmode pin, and that pin only controls sets and reset pins, then consider adding scanshift to that pin. Otherwise, the design may have to be changed to get maximum test coverage.

Method

For each testmode with a scanshift parameter:

Simulate a 010 pulse on this testmode pin.

Mark all async sources that have a 010 pulse or a 101 pulse. (These are ATPG testable)

If any flip-flop pin, other than a set or reset pin, has a non-x value resulting from the scanshift pulse simulations, report the flip-flop and the pin with a message that this testmode drives an async source and other flip-flop pins. (Use of scanshift may change testmode conditions)

Return this pin back to x and repeat on the next scanshift testmode pin.

end for

If any async source did not receive a pulse from any scanshift pin, report that source as not ATPG controllable from testmode constraints. If any async source received more than one pulse report that source as not uniquely ATPG controllable.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (mandatory): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] The testmode signal at <testmode-sig-name> drives an asynchronous source and data pin of the flip flop <flip-flop-name>

Arguments

- The testmode signal name. <testmode-sig-name>
- The flip-flop name. <flip-flop-name>

Potential Issues

The violation message appears, if a testmode signal drives an asynchronous source and data pin of a flip-flop as well.

Consequences of Not Fixing

If you do not fix this violation, it may result in race condition.

How to Debug and Fix

To know more about debugging the violation, click [How to Debug and Fix](#).

Make sure that connections and/or constraint specifications are correct. To know about debug, click on How to Debug.

Message 2

[WARNING] The testmode signal at <testmode-sig-name> drives an asynchronous source and clock pin of the flip flop <flip-flop-name>

Arguments

- The testmode signal name. <testmode-sig-name>
- The flip-flop name. <flip-flop-name>

Potential Issues

The violation message appears, if a testmode signal drives an asynchronous source and clock pin of the flip-flop as well.

Consequences of Not Fixing

If you do not fix this violation, it may result in race condition.

How to Debug and Fix

To know more about debugging the violation, click [How to Debug and Fix](#).

Make sure that schematic and constraint specifications are correct. To know about debug, click on How to Debug.

Message 3

[WARNING] The testmode signal at <testmode-sig-name> drives an asynchronous source and both data pin and clock pin of the flip flop <flip-flop-name>

Arguments

- The testmode signal name. <testmode-sig-name>
- The flip-flop name. <flip-flop-name>

Potential Issues

The violation message appears, if testmode signal drives an asynchronous source and both data pin and clock pin of a flip-flop.

Consequences of Not Fixing

If you do not fix this violation, it may result in race condition.

How to Debug and Fix

To know more about debugging the violation, click [How to Debug and Fix](#).

Make sure that the schematic and/or constraint specifications are correct.

To know about debug, click on How to Debug.

Message 4

[WARNING] The source net (<type>) of flop <flip-flop-name> is not ATPG controllable from given testmode constraints

Arguments

- The flip-flop name. <flip-flop-name>
- The asynchronous signal type as set or reset. <type>

Potential Issues

The violation message appears if source net of flip-flop is not ATPG controllable from given testmode constraints.

Consequences of Not Fixing

If you do not fix this violation, the control signal is not free to take any value during capture which reduces the coverage.

How to Debug and Fix

To know more about debugging the violation, click [How to Debug and Fix](#).

Make sure that proper constraint is added to the design. To know about debug, click on How to Debug.

Message 5

The source net (<type>) of flop <flip-flop-name> is not uniquely ATPG controllable

Arguments

- The flip-flop name. <flip-flop-name>
- The asynchronous signal type as set or reset. <type>

Potential Issues

The violation message appears if a flip-flop is not uniquely ATPG controllable.

Consequences of Not Fixing

If you do not fix this violation, the control signal is not free to take any value during capture which reduces the coverage.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window

highlights the flip-flop and its set/reset pin which is fixed to a value in the capture mode.

You can also view the violation for the Info_testmode (under shift condition) rule along with the violation of the TA_07 rule in the Incremental Schematic window. To do this, double-click the violation for the TA_07 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the TA_07 rule in the Incremental Schematic window. This is useful in debugging the violation for the TA_07 rule.

To fix the violation, ensure that -scanshift argument is added to the design.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Testability

Reports and Related Files

No related reports or files.

TA_08

Reports test enable/enable pins of clock gating cells that are tied to an undesired value

When to Use

Use this rule when a design has clock gating cells for any clock blockage.

Description

The TA_08 rule reports violation for test enable/enable pins of clock gating cells that are tied to an undesired value (specified using the `dftSetCGCEnablePinsValue` rule parameter) during capture.

Rule Exceptions

The TA_08 rule does not work when the definition of the clock gating cell is present in the design but not in the SpyGlass library file.

Method

For each instance:

Skip the instance if the master of the instance is not a clock gating cell.

Get the nets connected to the defined terminals.

If the net is NULL then skip the cell.

Report a violation if the net has an undesired value under capture condition.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dftTestEnablePinNameForCGC`: The default value is a non-empty string with the text, `TE`. Set the value of the parameter to a string to set the test enable pin name of the clock gating cell for the TA_08 rule.
- `dftEnablePinNameForCGC`: The default value is a non-empty string with the text, `EN`. Set the value of the parameter to a string to set the enable pin name of the clock gating cell for the TA_08 rule.

- *dftSetCGCEnablePinsValue*: The default value is active. Set the value of the parameter to inactive to set the undesired value at the test enable/enable pins of the clock gating cells for the TA_08 rule.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *gating_cell* (optional): Use this constraint to specify the user-defined clock gating cell.

Operating Mode

Scanshift, Capture

Messages and Suggested Fix

[WARNING] '<pin-name>' pin of instance '<inst-name>' (cell name: '<cell-name>') is tied to constant '<value 1>' (desired value: '<value 2>')

Arguments

- Test enable/enable pins. <pin-name>
- The complete path of the instance. <inst-name>
- The clock gating cell name. <cell-name>
- Undesired value specified using the *dftSetCGCEnablePinsValue* rule parameter. <value 1>
- Desired value at the Test enable/enable pins. <value 2>

Potential Issues

The following message appears, if enable pin of a clock gating cell is connected to an undesired value.

Consequences of Not Fixing

If you do not fix this violation, the clock might not work properly and controllability reduces.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window

Testability Analysis Rules

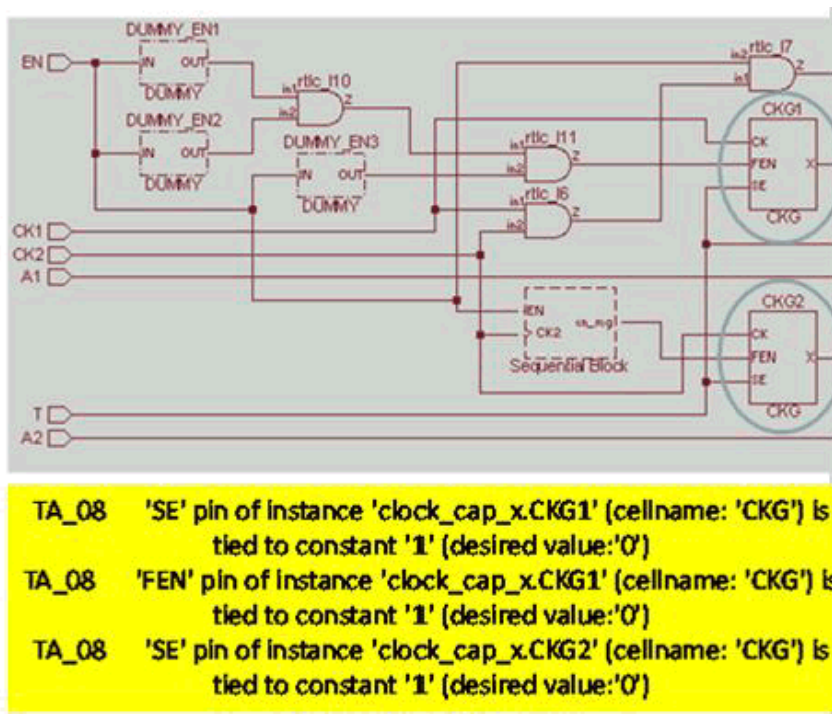
highlights the following:

- Instantiated clock gating cell.
- Test enable/enable pin along with the simulation value

To fix this violation, ensure that accurate value reaches the clock enable pin.

Example Code and/or Schematic

Consider the following figure:



In the above example, the TA_08 rule reports a violation because enable pins are connected to undesired values.

Default Severity Label

Warning

Rule Group

Testability

Reports and Related Files

None

TA_09

Reports cause of uncontrollability or unobservability and estimates the number of nets whose controllability/ observability is impacted

When to Use

Use this rule to remove any kind of unobservability.

Description

The TA_09 rule identifies the causes of uncontrollability or unobservability and estimates the number of nets whose controllability/observability is impacted.

The rule also generates a testcoverage report file when all SpyGlass DFT ADV suggested test points are selected.

Message Sorting

For TA_09 rule, the message list is sorted based on the user-specified sorting criteria (Incremental gain in controllability or Incremental gain in detectable fault count).

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *noBlackBoxReporting*: The default value is off. Set the value of the parameter to on to suppress the violation messages contributed by black boxes.
- *dft_min_scannability_ratio_for_test_points*: The default value is 0.7. Set the value of the parameter to any number between 0 and 1 to specify the minimum scannability ratio the design must be at, before test_point analysis (TA_09) starts and test points (control /observe) are identified.
- *dftAutoFix*: The default value of the parameter is as follows:


```
"{+RULES[nothing]+TMPORRT[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}"
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
"{+RULES[<ruleelist>]+TMPORT[tmport]+TCLKPORT[tclkport]}"
```

- ***dftPOSDETECT_credit***: The default value is 0. Set the value of the parameter to a floating-point number between 0 and 1 to regulate the effect of potentially detectable faults on the test coverage and fault coverage evaluation.
- ***dftTreatFlipFlopsAsScan***: The default value is off. Set the value of the parameter to on to treat all flip-flops as scannable for the Info_uncontrollable rule.
- ***dftTreatLatchesAsTransparent***: The default value is off. Set the value of the parameter to on to treat all latches as transparent for the Info_uncontrollable rule.
- ***dftTreatBBoxAsScanwrapped***: The default value is off. Set the value of the parameter to on to treat all black boxes as scan-wrapped for the Info_uncontrollable rule.
- ***dftUUMarking***: The dftUUMarking parameter has been deprecated. Use the *dftIgnoreConstantOrUnusedFlipFlops* parameter instead.
- ***dftIdentifyTestPoints***: The default value is both_control_observe. Set the value of the parameter to either control_only or observe_only to specify the type of control points selected in the TA_09 rule.
- ***dftSkipTAMsgCount***: The default value is off. Set the value of the parameter to any positive integer number to ignore those nets that result in improvement count less than the specified value.
- ***dftSkipLogicForTestPoints***: The default value is DFT_UU_SR_NonX. Set the value of the parameter to one of the following: DFT_UU, DFT_SR, DFT_NonX, or DFT_UU_SR to skip the specified logic.
- ***dftSkipTestPointsOnImprovementLessThan***: The default value is 0. Set the value of the parameter to any positive integer value to skip those nets, which result in improvement count less than the specified value

NOTE: Please note the following points:

- 📄 If you specify only the *dftSkipTAMsgCount* parameter, both the *TA_06* and *TA_09* rules use it.
- 📄 If you specify only the *dftSkipTestPointsOnImprovementLessThan* parameter, only the *TA_09* rule uses it.
- 📄 If you specify both the *dftSkipTestPointsOnImprovementLessThan* and

`dftSkipTAMsgcount` parameters, the `TA_09` rule uses the former parameter and the `TA_06` rule uses the latter one.

- **`dftUseOffStateOfClockInClockPropagation`**: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- **`force_ta`** (optional): Specifies the controllabilities and/or observabilities for ports/pins/nets.
- **`test_mode`** (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- **`clock`** (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- **`force_no_scan`** (optional): Excludes flip-flops from being declared scannable even if they so qualify.
- **`module_bypass`** (optional): Specifies modules such as memories that are designed with a bypass between data-in port and dataout port.
- **`scan_type`** (optional): Specifies the SpyGlass DFT ADV type (LSSD or MUXSCAN).
- **`scan_wrap`** (optional): Use this constraint to specify black box design units or instances that will be designed with scan wrappers.
- **`test_point`** (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.
- **`dont_touch`** (optional): Use this constraint to specify the modules/nets that are not considered for AutoFix.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] <net-name> [in <module-name>] is not controllable [<cause-string>] to <value>. Adding a test-point [Cnt = <cnt-

`initial>-><cnt-final>]` will make `<count-a>` nets controllable and `<count-b>` nets observable

Arguments

- Name of module containing the net on which violation is raised `<module-name>`
- Initial state of controllability, `<cnt-initial>`
- Final state of controllability, `<cnt-final>`
- Integer, `<count>`
- Percentage increase in fault count `<percentage>`
- Cause description, `<cause-string>`. This argument can have one of the following values: unconnected-input, multiply driven, test-mode value = '0', test-mode value = '1', input tied to power, input tied to ground, black box driven], noBBreport, affected by other input(s), hanging, input net of non scan-wrapped black box(es), input net of un-scannable flip-flop(s), driven by un-scannable flip-flop(s), input net of non-transparent latch(es), driven by non-transparent latch(es), feeds a tri-state enable, input net of disabled tristate(s), un-driven, combinational-loop.
- Controllability value, `<value>`. This argument can have one of the following values: 0, 1, or Z.

Potential Issues

The violation message appears, if a net is not controllable.

Consequences of Not Fixing

Not fixing the violation may result in reduced controllability.

How to Debug and Fix

To debug and fix the violation message, click [How to Debug and Fix](#).

Message 2

[WARNING] `<net-name>` [in `<module-name>`] is not observable [`<cause-string>`]. Adding a test-point [Obs = y] will make `<count-a>` nets controllable and `<count-b>` nets observable

Arguments

- Name of net. If the net is in fan-out cone of clock, then the fact is specified as "possibly a clock" along with name of the net, `<net-name>`

- Name of module containing the net on which violation is raised
<module-name>
- Integer, <count>
- Percentage increase in fault count <percentage>
- Cause description, <cause-string>. This argument can have one of the following values: unconnected-input, multiply driven, test-mode value = '0', test-mode value = '1', input tied to power, input tied to ground, black box driven], noBBreport, affected by other input(s), hanging, input net of non scan-wrapped black box(es), input net of un-scannable flip-flop(s), driven by un-scannable flip-flop(s), input net of non-transparent latch(es), driven by non-transparent latch(es), feeds a tri-state enable, input net of disabled tristate(s), un-driven, combinational-loop.

Potential Issues

The violation message appears, if a net is not observable.

Consequences of Not Fixing

Not fixing the violation may result in reduced observability.

How to Debug and Fix

To debug and fix the violation message, click [How to Debug and Fix](#).

Message 3

[WARNING] <net-name> [in <module-name>] is not controllable [<cause-string>] to <value>. Adding a test-point [Cnt = <cnt-initial>-><cnt-final>] will make <count> fault detectable [%increase <percentage>]

Arguments

- Name of net. If the net is in fan-out cone of clock, then the fact is specified as "possibly a clock" along with name of the net, <net-name>
- Name of module containing the net on which violation is raised
<module-name>
- Initial state of controllability, <cnt-initial>
- Final state of controllability, <cnt-final>
- Integer, <count>

- Percentage increase in fault count <percentage>
- Cause description, <cause-string>. This argument can have one of the following values: unconnected-input, multiply driven, test-mode value = '0', test-mode value = '1', input tied to power, input tied to ground, black box driven], noBBreport, affected by other input(s), hanging, input net of non scan-wrapped black box(es), input net of un-scannable flip-flop(s), driven by un-scannable flip-flop(s), input net of non-transparent latch(es), driven by non-transparent latch(es), feeds a tri-state enable, input net of disabled tristate(s), un-driven, combinational-loop.
- Controllability value, <value>. This argument can have one of the following values: 0, 1, or Z.

Potential Issues

The violation arises, if a net is not controllable.

Consequences of Not Fixing

Not fixing the violation may result in reduced controllability.

How to Debug and Fix

To debug and fix the violation message, click [How to Debug and Fix](#).

Message 4

[WARNING] <net-name> [in <module-name>] is not observable [<cause-string>]. Adding a test-point [Obs = y] will make <count> faults detectable [%Increase <percentage>]

Arguments

- Name of net. If the net is in fan-out cone of clock, then the fact is specified as "possibly a clock" along with name of the net, <net-name>
- Name of module containing the net on which violation is raised <module-name>
- Integer, <count>
- Percentage increase in fault count <percentage>
- Cause description, <cause-string>. This argument can have one of the following values: unconnected-input, multiply driven, test-mode value = '0', test-mode value = '1', input tied to power, input tied to ground, black box driven], noBBreport, affected by other input(s), hanging, input net of non scan-wrapped black box(es), input net of un-scannable

flip-flop(s), driven by un-scannable flip-flop(s), input net of non-transparent latch(es), driven by non-transparent latch(es), feeds a tri-state enable, input net of disabled tristate(s), un-driven, combinational-loop.

Potential Issues

The violation message appears, if a net is not observable

Consequences of Not Fixing

Not fixing the violation may result in reduced observability.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the violating instance and the root cause of bad controllability/observability is dumped on real net.

To fix this violation insert scan flip-flops or test-points.

Message 5

[INFO] Test points selection report file <report-file-name> for module <module-name> is generated.

Arguments

- Name of module containing the net on which violation is raised
<module-name>
- Name of the generated report file, <report-file-name>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 6

[INFO] Test points identification for design '<design_block_name>' is skipped as current scannability ratio

'<current_scannability_ratio>' is less than the required ratio of '<require_scannability_ratio>'. Use parameter `dft_min_scannability_ratio_for_test_points` <0 to 1 value range> to change the threshold ratio

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 7

[INFO] Test points constraint file '<sgdc-file-name>' for module '<module-name>' is generated

Arguments

- Name of sgdc file generated with test_point constraints, <sgdc-file-name>
- Name of design unit for which test_points are suggested, <module-name>

Potential Issues

This is an informational message. Therefore, there are no potential issues related to this violation message.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 8

[INFO] Suggested test points for autofix to improve fault coverage

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

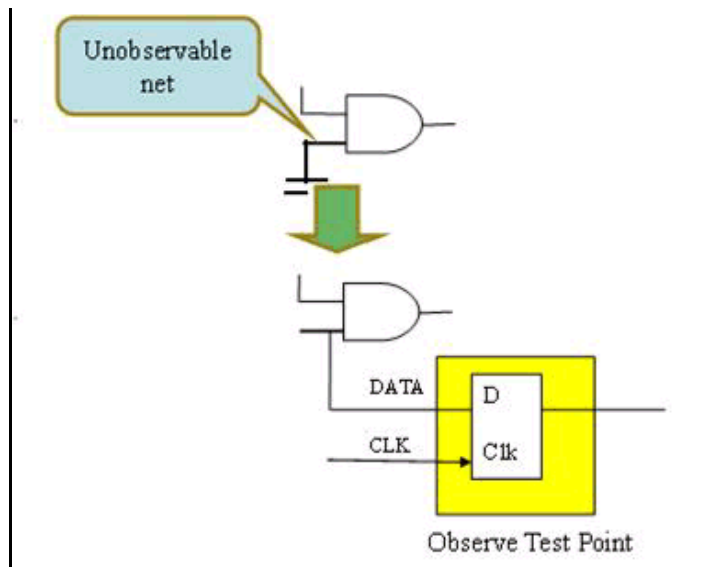
This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic**Example 1**

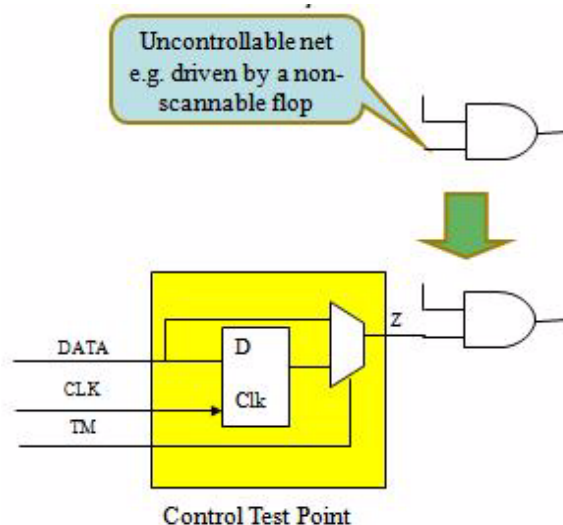
Consider the following figure:



The TA_09 rule reports violation for unobservable net and adding a observing point makes it observable.

Example 2

Consider the following figure:



The TA_09 rule reports violation for uncontrollable net and adding a test control point makes it controllable.

Default Severity Label

Warning/Info

Rule Group

Testability

Reports and Related Files

- [test_points_selected_2](#) report
- *test_point_constraints.sgdc*: This report contains test_point constraints suggested by the TA_09 rule along with the fault coverage, test coverage, and detectable faults achieved after using them. This report is generated in the `spyglass_reports/dft/` directory. Following is a sample from the *test_point_constraints.sgdc* file:

```
current_design "test.behaviour"

test_point -name test.op2 -type control
```


Testability Analysis Rules

```
# fault coverage: 70.0, test coverage: 70.0, detectable  
faults: 70
```

```
test_point -name test.bbmod2_inst2op -type control  
# fault coverage: 70.0, test coverage: 70.0, detectable  
faults: 70
```

```
test_point -name test.bbmod2_inst1op -type control  
# fault coverage: 70.0, test coverage: 70.0, detectable  
faults: 70
```

```
test_point -name test.bbmod2_inst2op -type observe  
# fault coverage: 72.0, test coverage: 72.0, detectable  
faults: 72
```

```
test_point -name test.bbmod2_inst1op -type observe  
# fault coverage: 74.0, test coverage: 74.0, detectable  
faults: 74
```

TA_10

Add test points to improve controllability and observability

When to Use

Use this rule to improve design controllability/observability.

Description

The TA_10 rule reports testpoints for improving design controllability/observability.

The `cnt_gain`, `obs_gain`, and `fault_gain` metrics in the `ta_test_point_constraints.sgdc` report and the spreadsheet report represents the increase in the controllable, observable, and detectable fault count, respectively.

Parameter(s)

- `dft_insert_ta_tp`: The default value is off. Set the value of the parameter to on to insert the testpoints reported by the TA_10 rule.
- `dft_ta_tp_for_port`: The default value is off. Set the value of the parameter to on to consider only the I/O ports as testpoint candidates.
- `dft_ta_tp_count`: The default value is 5%. Set the value of the parameter to an absolute number or percentage of flip-flops to specify the desired number of test points to be suggested for improving the stuck at fault coverage.
- `dft_ta_tp_type`: The default value is both. Set the value of the parameter to either control or observe to specify the type of testpoints that should be inserted for improving the fault coverage.
- `dft_ta_tp_criteria`: The default value is `cnt_obs_gain`. Set the value of the parameter to `fault_gain` to use `fault_gain` as the testpoint selection criteria.
- `dft_min_scannability_ratio_for_test_points`: The default value is 0.7. Set the value of the parameter to specify the minimum scannability ratio, the design must be at, before test_point analysis (TA_09/TA_10) starts and test points (control/observe) are identified.

Constraint(s)

- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *allow_test_point* (optional): Use this constraint to specify modules or instances which should be considered for suggesting test points.
- *no_test_point* (optional): Use this constraint to exclude modules or instances, which should not be considered for suggesting test points.
- *force_ta* (optional): Specifies the controllabilities and/or observabilities for ports/pins/nets.
- *test_point* (optional): Use this constraint to specify where a test point should be added in a design without the necessity of changing the source RTL.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

```
[INFO] '<test_point_count>' test points suggested for design
'<design_name>', SPREADSHEET_PATH: 'spreadsheet_path'
```

Potential Issues

The violation message is reported for the testpoints suggested for the design.

Consequences of Not Fixing

This is an informational message.

How to Debug and Fix

This is an informational message.

Message 2

```
[INFO] Test point(s) constraint file '<sgdc-file-name>'
generated
```

Arguments

Path of sgdc file with test_point constraints, <sgdc-file-name>

Potential Issues

This is an informational message.

Consequences of Not Fixing

This is an informational message.

How to Debug and Fix

This is an informational message.

Message 4

[INFO] No testpoint found for module '<top-design-name>'

Potential Issues

This is an informational message.

Consequences of Not Fixing

This is an informational message.

How to Debug and Fix

This is an informational message.

Message 5

[INFO] Test points identification for design '<top-design-name>' is skipped as current scannability ratio '<current_scannability_ratio>' is less than the required ratio of '<required_scannability_ratio>'. Use parameter dft_min_scannability_ratio_for_test_points <0 to 1 value range> to change the threshold ratio

Potential Issues

This is an informational message.

Consequences of Not Fixing

This is an informational message.

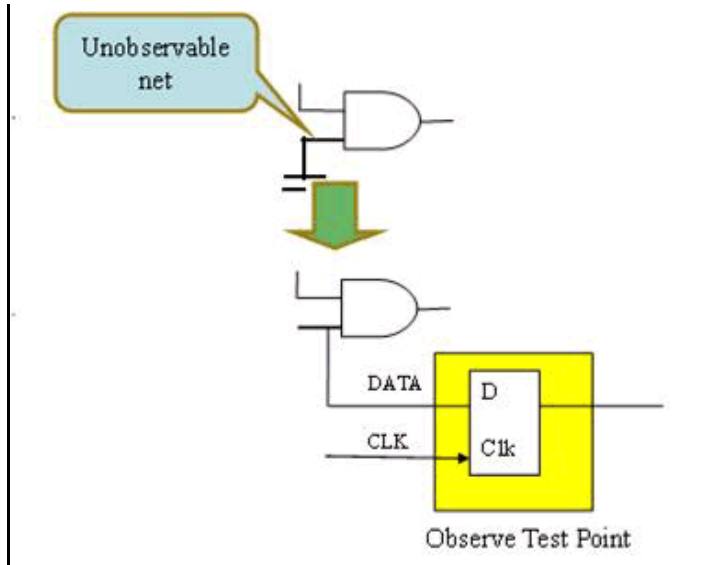
How to Debug and Fix

This is an informational message.

Example Code and/or Schematic

Example 1

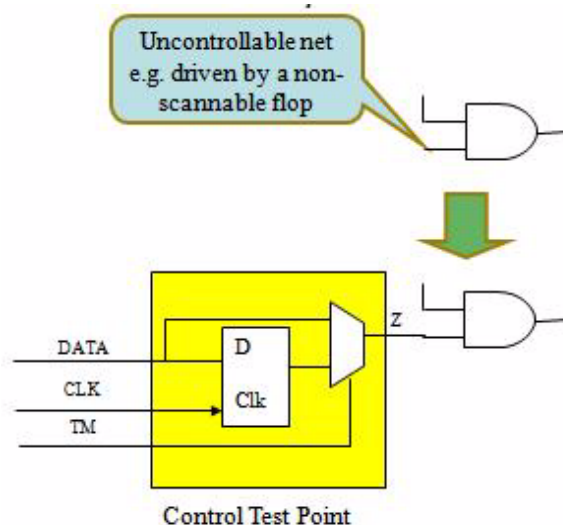
Consider the following figure:



The TA_10 rule reports violation for unobservable net and adding a observing point makes it observable.

Example 2

Consider the following figure:



The TA_10 rule reports violation for uncontrollable net and adding a test control point makes it controllable.

Default Severity Label

Warning/Info

Rule Group

Testability

Reports and Related Files

ta_test_point_constraints.sgdc: This file contains test_point constraints suggested by the TA_10 rule along with the gain achieved after using the test points.

The following is a sample *ta_test_point_constraints.sgdc* report:

```
#####
# Format :
#   A list of test_point constraints is presented
#
#   test_point -name test_point_name -type test_point_type
```

Testability Analysis Rules

```

#    fault coverage: fault_cov, test coverage: test_cov, cnt
gain: excitable_fault_gain, obs gain: observable_fault_gain,
fault gain: detectable_fault_gain
#
#####

#####
## SUMMARY :
#    Module :- test
#    Total faults considered :- 142
#    Test points considered :- 6
#    Detectable fault count without testpoint :- 51
#    Detectable fault count with all testpoints :- 123
#    Fault coverage without testpoint :- 35.92
#    Test coverage without testpoint :- 37.50
#    Fault coverage with all testpoints :- 86.62
#    Test coverage with all testpoints :- 90.44
#####
current_design "test"

    test_point -name "test.w31" -type observe
#    fault coverage: 45.77, test coverage: 47.79, obs gain:
14, fault gain: 14

    test_point -name "test.w4" -type full
#    fault coverage: 55.63, test coverage: 58.09, cnt gain:
14, obs gain: 0, fault gain: 14

    test_point -name "test.w11" -type observe
#    fault coverage: 65.49, test coverage: 68.38, obs gain:
14, fault gain: 14

    test_point -name "test.w1" -type full
#    fault coverage: 73.94, test coverage: 77.21, cnt gain:
8, obs gain: 4, fault gain: 12

    test_point -name "test.ff1" -type full
#    fault coverage: 79.58, test coverage: 83.09, cnt gain:

```

8, obs gain: 6, fault gain: 8

```
test_point -name "test.w12" -type full
# fault coverage: 86.62, test coverage: 90.44, cnt gain:
8, obs gain: 2, fault gain: 10
```


Test Compression Rules

Overview

The `Test Compression` rule group of the SpyGlass DFT ADV product has the following rules related to test compression:

Rule	Description
<i>TC_01</i>	Unequal head register pipelines.
<i>TC_02</i>	Primary input feeding more than one decompressor head register.
<i>TC_03</i>	Decompressor using more than one testclock.
<i>TC_04</i>	Unequal tail register pipeline
<i>TC_05</i>	Compressor using more than one testclock.

TC_01

Pipeline depth for head registers should equal head_register_depth

Rule Description

Test compression software produce streams of data that are assumed will reach the on-chip decompressor at the same time. Unequal head register pipelines violate this concept that will result in corrupted data reaching the internal scan chains.

Method

Simulate scanshift conditions.

For each decompressor instance

For each data input port

Traverse the sequential unblocked fan-in cone. If a flip-flop is reached then tag the decompressor with the testclock for this flip-flops' clock pin (used by TC_03), tag the decompressor with the phase inversion for this flip-flop (used by TC_04), and continue the traversal on that flip-flop's d-pin.

If more than one path is unblocked, then issue violation message #1 and stop this traversal. If the traversal stops at any node that is not a primary input or not an IO then issue violation message #2. If a primary input is reached, then tag this decompressor data pin with this primary input (used by TC_02). The number of flip-flops reached during this traversal is the number of "head register pipeline" stages on the scan path between primary scan-in ports and the decompressor inputs. If the number of stages does not equal the value specified in the head_register_depth parameter then issue violation message #3.

End for

End for

Constraints

- *compressor* (mandatory)
- *decompressor* (mandatory)

Rule Parameters

- *head_pipeline_stages*
- *dftUseOffStateOfClockInClockPropagation*

Operating Mode

Scanshift

Message Details

Message 1

Decompressor instance `<instance.port>` is driven by more than one path `<node>`

Message 2

Decompressor port `<instance.port>` is not driven by a primary input port

Message 3

Head pipeline depth for `<instance.pin>` is `<num>` does not match required value = `<depth>`

Arguments

- Instance of a port, `<instance.port>`
- Instance of a pin, `<instance.pin>`
- Any node that is neither a primary input nor an IO, `<node>`
- Number of head pipeline stages, `<num>`
- Depth of the head register, `<depth>`

Schematic Highlight

- Path from decompressor pin to stop point
- Path from decompressor to pipeline

Rule Severity

Warning

TC_02

Head registers must be driven by unique primary inputs

Rule Description

Primary ports driving the head registers must be unique in order to ensure that all input data combinations can be applied.

Method

If any primary input feeds more than 1 decompressor head register, then issue violation.

NOTE: *This check will also fail if datain pins on different decompressors are fed by the same primary input.*

Constraints

- *compressor* (mandatory)
- *decompressor* (mandatory)

Rule Parameters

- *head_pipeline_clock_phase*
- *head_pipeline_stages*
- *dftUseOffStateOfClockInClockPropagation*

Operating Mode

Scanshift

Message Details

Primary input `<input_port>` feeds head register pipelines for `<instance>` pins `<list>`

Arguments

- Primary input port that is feeding more than one decompressor register, `<input_port>`
- Decompressor instance, `<instance>`
- List of decompressor datain pins driven by the primary port, `<list>`

Schematic Highlight

Primary input and all decompressor datain ports fed from this primary input.

Rule Severity

Warning

TC_03

Head registers to the same decompressor must use the same clock and edge.

Rule Description

Head registers feeding the same decompressor must work together since the decompression function depends on synchronized data. Use of the same shift clock and phase is fundamental to this process

Constraints

- *compressor* (mandatory)
- *decompressor* (mandatory)

Rule Parameters

- *head_pipeline_clock_phase*
- *dftUseOffStateOfClockInClockPropagation*

Operating Mode

Scanshift

Message Details

Message 1

Decompressor <instance> uses more than 1 testclock <list>

Message 2

Decompressor <instance> has testclock <testclock>

Arguments

- Decompressor Instance, <instance>
- List of testclocks, <list>
- Testclock with wrong edge to a flip-flop in a head register for this decompressor, <testclock>

Schematic Highlight

- Highlight decompressor instance and testclocks.

Test Compression Rules

- Highlight testclock paths with wrong edge to a flip-flop in a head register for this decompressor.

Rule Severity

Warning

TC_04

Pipeline depth for tail registers should equal tail_register_depth

Rule Description

The TC_04 rule reports a violation, if the pipeline depth for any compressor data output pin does not equal the value specified in the tail_register_depth parameter.

Method

Simulate scanshift conditions.

For each compressor instance

For each data output port

Traverse of the sequential unblocked fan-out cone. If a flip-flop is reached then tag the compressor with the testclock for this flip-flops' clock pin (used by TC_06), tag the compressor with phase inversion for this flip-flop (used by TC_07) and continue the traversal on that flip-flop's q-pin.

If a node is reached with fan-out greater than 1 then issue violation message #1. If the traversal stops at any node that is not a primary output port then issue violation message #2. The number of registers reached during this traversal is the number of tail register pipeline stages on the scan path between compressor outputs and the primary scan-out ports. If the number of stages does not equal the value specified in the tail_register_depth parameter then issue violation message #3

End for

End for

Constraints

- *compressor* (mandatory)
- *decompressor* (mandatory)

Rule Parameters

- *tail_pipeline_stages*
- *dftUseOffStateOfClockInClockPropagation*

Operating Mode

Scanshift

Message Details

Message 1

Compressor port < instance. port> fans out to more than one path <node>

Message 2

Compressor port < instance. port> is not connected to a primary output port

Message 3

Tail pipe line depth for <instance.pin> is <num> does not match required value = <depth>

Arguments

- Instance of a port, <instance.port>
- Instance of a pin, <instance.pin>
- Any node that is neither a primary input nor an IO, <node>
- Number of tail pipeline stages, <num>
- Depth of the tail register, <depth>

Schematic Highlight

- Path from compressor pin to stop point
- Path from compressor through pipeline

Rule Severity

Warning

TC_05

Tail registers from the same compressor must be driven by the same clock and edge

Rule Description

Tail registers fed from the same compressor must work together since the compression function produces synchronized data. Use of the same shift clock and phase is fundamental to this process.

Method

For each compressor

If this compressor is tagged with more than one testclock then issue a violation #1 and skip to next compressor.

If this compressor is tagged with one testclock and an edge does not = "head_register_clock_edge" then issue a violation #2.

End for

Constraints

- *compressor* (mandatory)
- *decompressor* (mandatory)

Rule Parameters

- *tail_pipeline_clock_phase*
- *dftUseOffStateOfClockInClockPropagation*

Operating Mode

Scanshift

Message Details

Message 1

Compressor <instance> uses more than 1 testclock <list>

Message 2

Compressor <instance> has testclock <testclock>

Arguments

- Compressor Instance, *<instance>*
- List of testclocks, *<list>*
- Testclock with wrong edge to a flip-flop in a head register for this Compressor, *<testclock>*

Schematic Highlight

- Compressor instance and testclocks.
- Testclock paths with wrong edge to a flip-flop in a tail register for this compressor.

Rule Severity

Warning

Topology Rules

Overview

The Topology rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Flags...
Topology_01	Combinational loops
Topology_02	Asynchronous paths between primary ports
Topology_03	Flip-flops whose set/reset pins are registered with a flip-flop triggered by the same clock and phase.
Topology_04	Direct register-to-register connections
Topology_05	Wired nets
Topology_07_flat	Nets with parallel drivers
Topology_07_rtl	(Verilog only) Instances with parallel drivers
Topology_09	Combinational reconvergent fan-outs
Topology_10	Long logic paths
Topology_11	Clocks that have combinational paths to primary output ports in the shift mode
Topology_12	Primary inputs not connected to a pullup/pulldown device
Topology_13	Reconverging combinational paths
Topology_14	Reports if there is a convergence at the async pins of a flip-flop
Topology_15	Top level output/inout ports except scanout ports must be controlled.
Topology_16	Reports flip-flops having feedback from Q-pin to D-pin which enables flip-flops to retain their state 0 or 1 or both.
Topology_17	Reports undriven terminals and nets.

Topology_01

Ensure that the design does not contain combinational loops in the capture mode.

When to Use

Use this rule to identify the combinational path, which is forming a loop. Also, this rule helps you to detect low coverage problems for combinational ATPG tools.

Description

The Topology_01 rule reports violation for combinational loops detected in the capture mode.

Default Weight

10

Language

Verilog, VHDL

Method

If no testmode information is supplied, simulate power and ground. Walk the fan-in cone for each gate with a fan-out greater than one. The walk skips any gate with a non-x simulation value, any edge-triggered flip-flop, and any latch with a non-active enable. If any combinational gate, already encountered is reached, then report a message.

If testmode information is supplied, simulate power and ground and all available testmode for capture. Walk the unblocked combinational fan-in cone for each gate with a fan-out greater than one. The walk skips any gate with a non-x simulation value. If any combinational gate that was already encountered is reached, then report a message.

NOTE: *You should use `dftSetEffortLevel` switch to see overlapping combinational loops as well. For example, if effort level is set to 6, Topology_01 will traverse through a node that many number of times to identify overlapping loops, if they exist.*

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftAutoFix*: The default value of the parameter is as follows:

```
"{+RULES[nothing]+TMPORT[atrenta_generated_port_tm]+TCLKPORT[atrenta_generated_port_tclk]}"
```

You can specify the a list of rule names, test mode port and test clock port in the following format to generate modified RTL source files that have the suggested changes to fix the rule-violations of the specified rules:

```
"{+RULES[<rulelist>]+TMPORT[tmport]+TCLKPORT[tclkport]}"
```
- *dftSetEffortLevel*: The default value is none. Specify each rule name followed by an integer between 0 to infinity as the input to set the upper range of the effort level for a rule.
- *dftFindCombLoopThruSetResetToQ*: The default value is on. Set the value of the parameter to off to not consider the path between the reset pin and the Q pin of the flip-flop as combinational.
- *dftTreatControllableLatchTransparentForLoop*: The default value is off. Set the value of the parameter to on to treat controllable latches as transparent for the Topology_01 rule.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

The following violation messages are displayed for the Topology_01 rule:

Message 1

[WARNING] <AutoFix_ID>Combinational loop, having <num_of_nets> elements, detected in '<mode_name>' mode. Loop path:
<loop_path>

Arguments

- AutoFix ID, <AutoFix_ID>
- Number of nets in the loop, <num_of_nets>
- Mode name as Capture, <mode_name>
- The path of the loop, <loop_path>

Potential Issues

A violation is reported due to incomplete test_mode or incorrect design connections.

Consequences of Not Fixing

Combinational loops have memory and therefore can cause problems as well as lower coverage for combinational ATPG tools.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the combinational path, which is forming the loop.

To fix the violation, see [Example Code and/or Schematic](#).

Message 2

[INFO] Suggested test points to break combinational loop

Potential Issues

The violation message identifies places in the RTL that get modified or may get impacted by the automatic RTL modification, that is, AutoFix.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

See [Reports in the SpyGlass DFT ADV Product](#) section for details on fixing the violation.

Example Code and/or Schematic

Verilog

Consider the following Verilog code:

```
...
always @ (d or sel) begin
    case (sel)
        2'b00: q = 0;
        2'b01: q = d;
        2'b10: q = 1;
        2'b11: sel[0] = 0;
    endcase
end
...
```

The above example illustrates a bad design scenario and the Topology_01 rule reports a violation for the above example.

To remove the violation, modify the above code as shown below:

```
...
always @(posedge clk)
    sel = sel_tmp;

always @ (d or sel) begin
    case (sel)
        2'b00: q = 0;
        2'b01: q = d;
        2'b10: q = 1;
        2'b11: begin
            sel_tmp[0] = 0;
            sel_tmp[1] = sel[1];
        end
    endcase
end
...
```

VHDL

Consider the following VHDL code:

Topology Rules

```

...
Process (d, sel)
Begin
  Case sel is
    When ("00") => a <= '0';
    When ("01") => a <= d;
    When ("10") => a <= '1';
    When ("11") => a <= '1';
  End case;
End process;
...

```

The above example illustrates a bad design scenario and the Topology_01 rule reports a violation for the above example. To remove the violation, modify the above code as shown below:

```

...
Process (clk)
Begin
  If (clk'event and clk = '1') then
    Sel <= sel_tmp;
  End if;
End process;
Process (d, sel)
Begin
  Case sel is
    When ("00") => a <= '0';
    When ("01") => a <= d;
    When ("10") => a <= '1';
    When ("11") => begin
      Sel_tmp[0] <= 0;
      Sel_tmp[1] <= sel(1);
    end
  End case;
End process;
...

```

Schematic highlight

Combinational path which is forming loop

You can also view the violations for the Info_testmode, Info_path, and Info_uncontrollable rules along with the violation of the Topology_01 rule in the Incremental Schematic window. To do this, double-click the violation for the Topology_01 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode, Info_uncontrollable, and Info_path rules overlap the violation message for the Topology_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_01 rule.

Info_uncontrollable rule checks the effect of loops on the controllability values.

Default Severity Label

Warning

Rule Group

Topology Rules

Reports and Related Files

No related reports or files.

Topology_02

No asynchronous pin to pin paths

When to Use

Use this rule to identify the combinational path from the input port to the output port.

Rule Description

The Topology_02 rule flags asynchronous paths between primary ports.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any available testmode for capture. Walk the unblocked combinational fan-out cones from all root level ports. Stop the walk at any node with a non-x simulation value. If any root level port is reached then report a message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and

Suggested Fix

The following violation message is displayed for the Topology_02 rule:

```
[WARNING] Found a direct connection from input port
'<port1-name>' to output port '<port2-name>' (depth <num>,
between port to port)
```

Arguments

- Name of the starting input port. (<port1-name>)
- Name of the ending port. (<port2-name>)
- Number of logic levels in the port-to-port path. <num>

Potential Issues

A violation is reported when an incomplete test_mode does not break the path.

Consequences of Not Fixing

Combinational pin to pin paths may confuse both synthesis tools and timing analyzers. Since various ATE machines relate capture events to clock edges, the absence of registered values may also complicate production testing.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the combinational path from the input port to output port.

Overlay the Info_testmode (in auxiliary mode) rule under shift mode to check how this path is sensitized.

To fix the violation, ensure that paths from an input port to an output port are registered.

Example Code and/or Schematic

Verilog

Consider the following Verilog code:

```
...
input P11;
output P01;
```

Topology Rules

```
assign PO1 <= PI1;  
...
```

The above example illustrates a bad design scenario and the Topology_02 rule reports a violation for the above example.

VHDL

Consider the following VHDL code:

```
...  
PI1 : in std_logic;  
PO1 : out std_logic;  
PO1 <= PI1;  
...
```

The above example illustrates a bad design scenario and the Topology_02 rule reports a violation for the above example.

Schematic highlight

Combinational path from the input port to the output port.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Topology_02 rule in the Incremental Schematic window. To do this, double-click the violation for the Topology_02 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Topology_02 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_02 rule.

Default Severity

Label

Warning

Rule Group

Topology

Reports and Related Files

No related reports and files.

Topology_03

Avoid sequentially derived asynchronous signals with common clock

When to Use

Use this rule to identify the flip-flops that are in the same clock domain as the flip-flops in the set/reset pin fan-in cones.

Rule Description

The Topology_03 rule flags flip-flops in the same clock domain as the flip-flops in the set/reset pin fan-in cones.

Prerequisites

Specify the *clock* -testclock constraint.

Default Weight

10

Language

Verilog, VHDL

Method

Walk the set or reset pin unblocked combinational fan-in cones for each flip-flop. Stop the walk at any node with non-x simulation value. Any flip-flop reached must not have the same clock and phase as the clock to the original flip-flop.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Capture

Messages and Suggested Fix

The following violation message is displayed for the Topology_03 rule

```
[WARNING] <component_type> ' <component-name-list>' is(are) driving ' <pin-type>' pin of <num> flip-flop(s) (e.g. ' <flip-flop-name>') and they have common clock
```

Arguments

- Type of the design component as flip-flop or latch. <component_type>
- Name list of flip-flops or latches driving the set/reset pin. <component-name-list>
- Type of pin as set or reset. (<pin-type>)
- Number of affected flip-flops. <num>
- Name of one affected flip-flop. <flip-flop-name>

Potential Issues

A violation is reported due to improper reset path.

Consequences of Not Fixing

Sequentially derived reset (or set) and common clock may cause race conditions or not treated correctly by ATPG and should be avoided.

ATPG tools operate with simple timing models that are generally incapable of properly handling race conditions involved with synchronously derived set or reset signals. If the clock source for a flip-flop Q1 with a set or reset that is derived from a second flip-flop Q2 with the same clock as Q1, then ATPG tools may produce bad tests.

How to Debug and Fix

View the Incremental Schematic for the violation message. Incremental Schematic highlights the path from flip-flop A output to flip-flop B's asynchronous pin. Probe the clock pin fan-in cone of flip-flop A and flip-flop B, respectively, to see how it re-converges.

Schematic highlight

1. The clock pin.
2. Path from output pin of the flip-flop that is triggered by the clock pin, to the asynchronous pin of the other flip-flop.

You can also view the violations for the Info_testmode and Info_path rules

along with the violation of the Topology_03 rule in the Incremental Schematic window. To do this, double-click the violation for the Topology_03 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Topology_03 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_03 rule.

To fix the violation, see [Example Code and/or Schematic](#).

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module Top (clk, d1, d2, dout);
  input clk, d1, d2;
  output dout;

  reg q1, q2;

  always @ (posedge clk)
    q1 <= d1;
  always @ (posedge clk or negedge q1)
    if (q1 == 1'b0)
      q2 <= 1'b0;
    else q2 <= d2;

  assign dout = q2;
endmodule
```

The above example illustrates a bad design scenario and the Topology_03 rule reports a violation for the above example.

Example 2

Consider the following VHDL code:

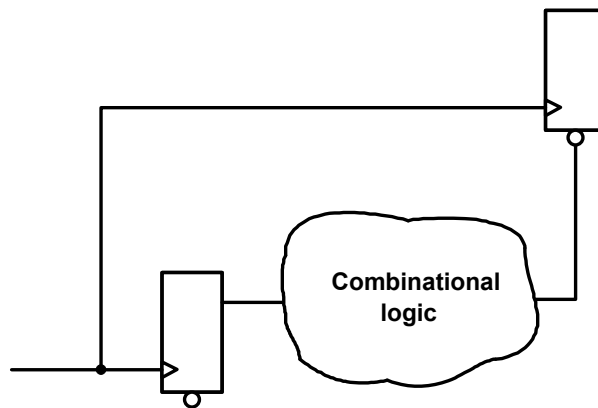
```
...
Process (clk)
Begin
```

```
    If (clk'event and clk = '1') then  
        Q1<= d1;  
    End if;  
End process;
```

```
Process (clk, q1)  
Begin  
    If (clk'event and clk = '1') then  
        If (q1 = '0') then q2 <= '0';  
        Else q2 <= d2;  
        End if;  
    End if;  
End if;
```

```
    dout<= q2;  
End process;
```

The above example illustrates a bad design scenario and the Topology_03 rule reports a violation for the above example.



Example 3

The following figure illustrates FF1, which is a capture flip-flop, and it determines values on FF2.q to make FF1.set inactive.

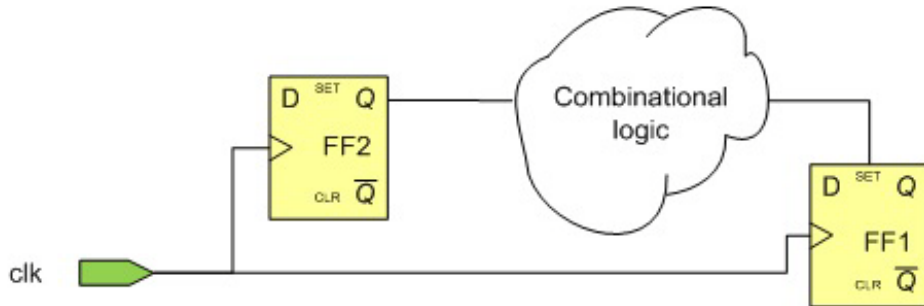


FIGURE 97. Sequentially derived asynchronous signal

In the above example, if the data on FF2.d was opposite to the requirement on FF2.q then the capture clock could cause FF1.set to become active. This would override the capture value and result in unexpected test results.

Default Severity Label

Warning

Rule Group

Topology Rules

Reports and Related Files

No related reports or files.

Topology_04

Reports direct connection between two registers

When to Use

Use this rule to check for register-to-register connections.

Description

The Topology_04 rule reports violation for direct register-to-register connections.

Method

The fan-in cone for each flip-flop data pin is walked. The walk back stops on any fan-in branch connected to a combinational device. If any sequential device is reached, a message is reported.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[WARNING] There is a direct connection between flip-flop '`<flip-flop1-name>`' and '`<flip-flop2-name>`'

Arguments

- Name of the preceding flip-flop, <flip-flop1-name>
- Name of the following flip-flop, <flip-flop2-name>

Potential Issues

This violation message appears, if there is a direct connection between two registers.

NOTE: *The message may be flagged at line where the flip-flop is reset rather than clocked.*

Consequences of Not Fixing

If you do not fix this violation, some ATE machines as well as ATPG tools cannot handle data paths between registers without intervening logic.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path from flip-flop's output pin to the data pin of another flip-flop.

To fix the violation, view the Incremental Schematic and insert flip-flops between the registers.

Example Code and/or Schematic**Example 1**

Consider the following Verilog code:

```
...
always @ (posedge clk)
  begin
    b = a;
    c = b;
  end
...
```

The above example demonstrates a bad design example because there is direct connection between the flip-flops, which results in the generation of the violation message.

Example 2

Consider the following VHDL code:

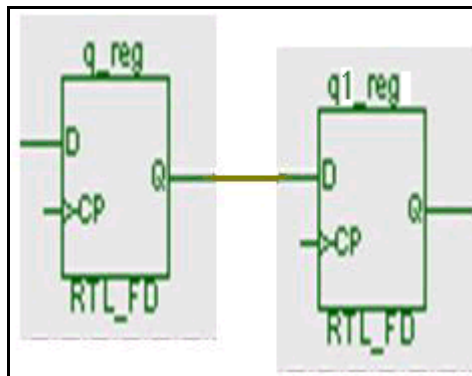
```
...
```

```
Process (clk)
Begin
  If (clk'event and clk = '1') then
    b <= a;
    c <= b;
  End if;
End process;
...
```

The above example demonstrates a bad design example because there is direct connection between the flip-flops, which results in the generation of the violation message.

Example 3

Consider the following figure:



In the above figure, two flip-flops are directly connected to each other, which results in a violation.

Default Severity Label

Warning

Rule Group

Topology

Topology Rules

Reports and Related Files

No reports and related files.

Topology_05

Ensure that the design does not have a Wire-OR and Wire-AND

When to Use

Use this rule to identify the instances and the respective terminals, which are drivers of the multiple driven net.

Description

The Topology_05 rule reports violation for the nets which may potentially have more than one active driver.

NOTE: *When the Memory Reduction feature is enabled, it may generate false Topology_05 violation. For more information on the Memory Reduction feature, refer to the Memory Reduction Feature section in the SpyGlass Console Reference Guide.*

The outputs of two or more gates may not be connected together.

Rule Exceptions

The Topology_05 rule ignores the contention caused by the following:

- INOUT port
- Inactive (in capture mode) tristate driver
- Inout terminals of black boxes when the [dft_ignore_bb_inout_term_for_topology_05](#) parameter is on.

Default Weight

10

Language

Verilog, VHDL

Method

Find nets with more than one source where source can be a tristate device, a non tristate device, a black box, or a root-level primary port. All possible combinations are listed in the following table:

Topology Rules

Case	Black Box	Tri state	Non tri state	Primary port	Message reports the following with part counts
1	n	n	n	n	n/a -- no violation
2	n	n	n	y(> 1 port)	Wired ports
3	n	n	y(> 1 gate)	n	Wired gates -- no violation if parallel case
4	n	n	y	y	Wired ports & gates
5	n	y	n	n	Tri-state bus -- no violation
6	n	y	n	y	Wired tri-state & port
7	n	y	y	n	Wired tri-state & gate
8	n	y	y	y	Wired tri-state & gate & port
9	y(>1 BB)	n	n	n	Wired black boxes
10	y	n	n	y	Wired black box(es) & port
11	y	n	y	n	Wired black box(es) & gate
12	y	n	y	y	Wired black box(es), gate & port
13	y	y	n	n	Wired black box(es) & tri-state
14	y	y	n	y	Wired black box(es), tri-state & port
15	y	y	y	n	Wired black box(es), tri-state & gate
16	y	y	y	y	Wired black box(es), tri-state, gate & port

Case 1 is shown for completeness but does not contain wired logic so no message is reported.

Case 3 is a rule-violation if the sources have different gate types (such as nand and or) or identical type but different numbers of inputs (such as nor 3 and nor 2 wired together) or identical types and input count but different input sources (such as nor(a,b) wired with nor(a,c)).

Case 5 defines a tristate bus and is not a rule-violation.

All other cases are rule-violations. In each of these rule-violations, the number of each source type, driving the wired net, is listed in the message.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *noBlackBoxReporting*: The default value is off. Set the value of the parameter to on to suppress the rule messages for nets made uncontrollable or unobservable by black boxes.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

- [dft_ignore_bb_inout_term_for_topology_05](#): The default value of the parameter is off. Set the value of the parameter to on to ignore the inout terminals of the black boxes.

Constraint(s)

None

Operating Mode

[Capture](#)

Messages and Suggested Fix

The following violation message is displayed for the Topology_05 rule:

[WARNING] Signal '[<net-name>](#)' is wired with '[<number>](#)'

Arguments

- Name of the wired net. [<net-name>](#)
- Number of wired components, such as, black box, tristate nets, and non-tristate device. [<number>](#)

Potential Issues

A violation is reported due to incorrect design connection.

Consequences of Not Fixing

Wired connections are treated by ATPG as logical elements but with functions different from tri-state buses. This may increase ATPG run time as well as the possibility of producing unusable vectors.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the multi driven net. This multi-driven net causes the violation message.

To fix the violation, see [Example Code and/or Schematic](#).

Schematic highlight

1. The multi-driven segment of the affected net.
2. Instances and their terminals that are the drivers of the affected net.

Example Code and/or Schematic

Example 1

Verilog

Consider the following Verilog code:

```
...
wand a;
wor b;
...
```

The above example illustrates a bad design scenario and the Topology_05 rule reports a violation for the above example.

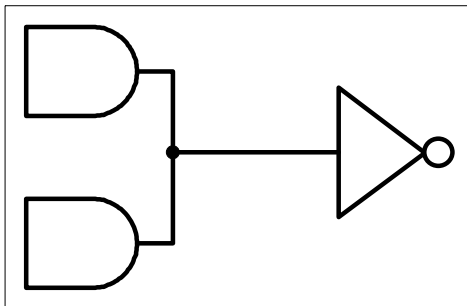
VHDL

Consider the following VHDL code

```
...
a <= int_sig1 and int_sig2;
a <= int_sig1 or int_sig2;
...
```

The above example illustrates a bad design scenario and the Topology_05 rule reports a violation for the above example.

Consider the following figure with respect to the above example code:



The rule reports the following violation message for the above example:

Signal 'WN' is wired with '1 tristate, 1 nontristate, 3 black boxes and 2 ports'

In the above example, a wired net WN that is driven by a tristate gate, a

non-tristate device, 3 black boxes and 2 ports.

Example 2

Consider the following design:

```
module Black_Box_o(input in1,in2,output o1);
endmodule

module Black_Box_io(input in1,in2,inout io1);
endmodule

module top();
wire w1,w2,w3;

Black_Box_o bb_o1(,,w1);
Black_Box_o bb_o2(,,w1);

Black_Box_io bb_io1(,,w2);
Black_Box_io bb_io2(,,w2);

Black_Box_o bb_o3(,,w3);
Black_Box_io bb_io3(,,w3);

endmodule
```

Topology Rules

[Table 3](#) lists the following information for the above design:

- Violation messages reported by the Topology_05 rule
- Potential Causes of the violation messages
- Figure number, which illustrates the respective schematics
- Whether the violation message is reported when the [dft_ignore_bb_inout_term_for_topology_05](#) parameter is set to `off`
- Whether the violation message is reported when the [dft_ignore_bb_inout_term_for_topology_05](#) parameter is set to `on`

TABLE 3 Multiple Driver Ports

Msg #	Message Text	Cause	Related Schematic	Violation reported if, dft_ignore_bb_inout_term_for_topology_05 = <code>off</code>	violation reported if, dft_ignore_bb_inout_term_for_topology_05 = <code>on</code>
1	Signal 'top.w3' is wired with '2 black box(es)', test.v, 18	Net, w3, is connected to two black boxes, bb_io3 and bb_o3	Figure 98	yes	no
2	Signal 'top.w2' is wired with '2 black box(es)', test.v, 15	Net, w2, is connected to two black boxes, bb_io1 and bb_io2	Figure 99	yes	no
3	Signal 'top.w1' is wired with '2 black box(es)', test.v, 12	Net, w1, is connected to two black boxes, bb_o1 and bb_o2	Figure 100	yes	yes

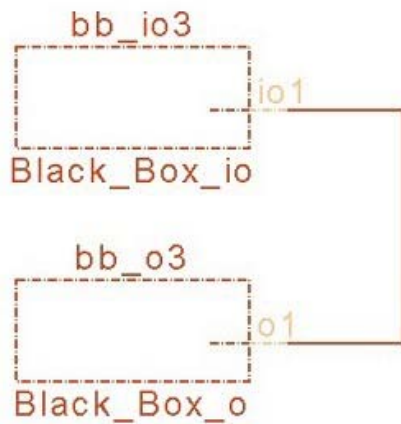


FIGURE 98. Inout and output ports as drivers

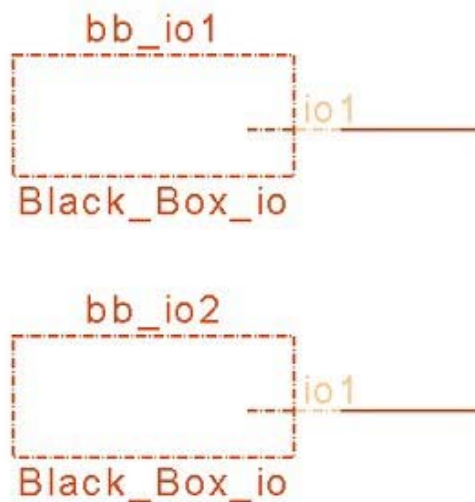


FIGURE 99. Two inout ports as drivers

Topology Rules

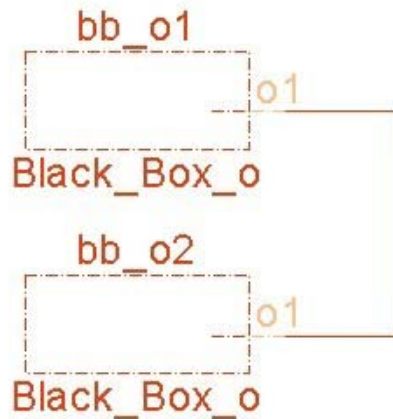


FIGURE 100. Two output ports as drivers

Default Severity**Label**

Warning

Rule Group

Topology Rules

**Reports and
Related Files**

No related reports and files.

Topology_07_flat

Do not use parallel drivers

When to Use

Use this rule when the methodology does not allow parallel drivers. This rule is run after flattening.

Description

The Topology_07_flat rule reports violation for nets with parallel drivers.

NOTE: The [Topology_07_rtl](#) rule flags parallel drivers in Verilog designs at RTL description-level where as the [Topology_07_flat](#) rule flags parallel drivers at the flattened design-level for both Verilog and VHDL designs.

Drivers wired in parallel to drive large fan-out sets cause untestable structures.

Method

A message is flagged if two or more combinational gates have the same logic type and identical input connections.

Default Weight

10

Languages

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

[Capture](#)

Messages and Suggested Fix

[WARNING] Parallel gates logic found at '<net-name>'

Arguments

Name of the net with parallel drivers, <net-name>

Potential Issues

This violation message appears, if there are parallel drivers in the design.

Consequences of Not Fixing

Not fixing this violation may lead to multiple driver case, whenever logic connectivity is changed. Also, ATPG tool finds difficulty in generating patterns. Parallel drivers represent redundancy and may confuse tools that check for bus contention.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the instances and their terminals which are forming the parallel driver of the net.

To fix the violation, remove the multiple drivers or change the logic.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module Parallel (a, b, clk, o1);
  input a, b, clk;
  output o1;

  wire a, b, out1;
  reg o1;

  and u1 (out1,a,b);

  always @ (posedge clk) begin
    o1 = out1;
  end

  and u2 (out1,a,b);
```

```
endmodule
```

The above example demonstrates a bad design scenario because of the parallel drivers in the design.

Example 2

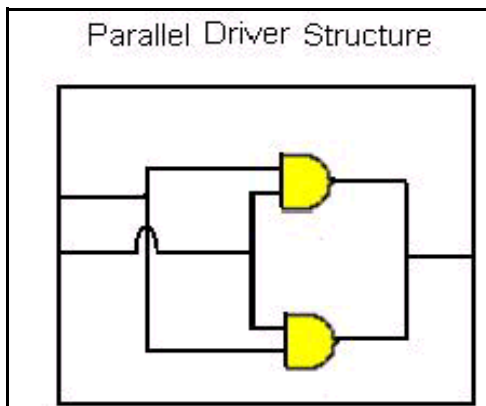
Consider the following VHDL code:

```
...  
Signal a, b, out1 : std_logic;  
U1: and port map(out1, a, b);  
  
Process (clk)  
Begin  
  If (clk'event and clk = '1') then  
    O1 <= out1;  
  End if;  
End process;  
  
U2: and port map(out1, a, b);  
...
```

The above example demonstrates a bad design scenario because of the parallel drivers in the design.

Example 3

Consider the following figure:



Topology Rules

The Topology_07_flat rule reports a violation for the above example because of the parallel driver structure.

Default Severity Label

Warning

Rule Group

Topology

Reports and Related Files

No related reports or files.

Topology_07_rtl

Do not use parallel drivers

When to Use

Use this rule when methodology does not allow parallel drivers in the design.

Description

The Topology_07_rtl rule reports violation for instances with parallel drivers.

NOTE: *The Topology_07_rtl rule flags parallel drivers in Verilog designs at RTL description-level whereas the [Topology_07_flat](#) rule flags parallel drivers at the flattened design-level for both Verilog and VHDL designs. The rule messages generated by the two rules are mutually exclusive for Verilog designs.*

Method

A message is flagged if two or more combinational gates have the same logic type and identical input connections.

Default Weight

10

Language

Verilog

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[WARNING] Parallel gates logic found between '<inst1-name>' and '<inst2-name>'

Arguments

- Name of the first instance, <inst1-name>
- Name of the second instance, <inst2-name>

Potential Issues

This violation message appears, if there are parallel drivers in a design.

Consequences of Not Fixing

If you do not fix this violation, this can lead to multiple driver case, whenever logic connectivity is changed. Drivers wired in parallel to drive large fan-out sets cause untestable structures.

Also, ATPG tool finds difficulty in generating patterns. Parallel drivers represent redundancy and may confuse tools that check for bus contention.

How to Debug and Fix

To debug and fix the violation, review the violation message, remove the multiple drivers, or change the logic.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module Parallel (a, b, clk, o1);
    input a, b, clk;
    output o1;

    wire a, b, out1;
    reg o1;

    and u1 (out1,a,b);

    always @ (posedge clk) begin
        o1 = out1;
    end
end
```

```
    and u2 (out1,a,b);  
endmodule
```

In the above example, the `Topology_07_rtl` rule reports violation because the design contains a parallel driver structure.

Example 2

Consider the following VHDL code:

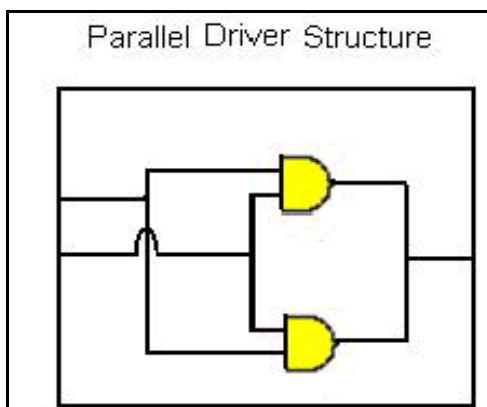
```
...  
Signal a, b, out1 : std_logic;  
U1: and port map(out1, a, b);  
  
Process (clk)  
Begin  
    If (clk'event and clk = '1') then  
        O1 <= out1;  
    End if;  
End process;
```

```
U2: and port map(out1, a, b);
```

In the above example, the `Topology_07_rtl` rule reports violation because the design contains a parallel driver structure.

Example 3

Consider the following figure:



Topology Rules

The `Topology_07_rtl` rule reports a violation for the above example because of the parallel driver structure.

Default Severity Label

Warning

Rule Group

Topology

Reports and Related Files

No reports and related files.

Topology_09

Reports the existence of combinational re-convergent fan-out

When to Use

Use this rule to identify and fix all the combinational re-convergence in the design.

Description

The Topology_09 rule reports violation for combinational reconvergent fan-outs.

NOTE: *The Topology_09 rule flags reconvergent combinational paths between all object types whereas the [Topology_13](#) rule flags reconvergent combinational paths between primary port/flip-flop output pin and asynchronous set/reset pins only.*

Method

If the intersection of the combinational fan-in cones, excluding nodes with non-x simulation values, for all inputs of a multi-input combinational device is not empty, then a message is reported. Note that only the first such reconvergence will be reported.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [reconvergenceMinDepth](#): Default value is 0. Set the value of the parameter to any natural number to set the limit on the minimum number of logic levels that a re-convergence should report, as a message.
- [showAllReconvergence](#): Default value is off. Therefore, the Topology_09 rule reports one re-convergence per path. Set the value of the parameter to on to see all the re-convergence.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Logic path from '<net1-name>' re-converges at or near '<net2-name>'

Arguments

- Name of the starting net, <net1-name>
- Name of the ending net, <net2-name>

Potential Issues

The violation appears, if there is a re-convergence between two nodes.

Consequences of Not Fixing

If you do not fix this violation, these structures may give rise to spurious signal generation. It may also cause race conditions and problems with ATPG tool.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the following:

- Start point -terminal or port where the logic which is re-converging is originating
- End point -terminal or port where the logic is re-converging

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Topology_09 rule in the Incremental Schematic window. To do this, double-click the violation for the Topology_09 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Topology_09 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_09 rule.

To fix the violation, remove the re-convergence or change the logic.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module ReconFO (a, b, c, clk, o1);
  input a, b, c, clk;
  output o1;

  wire a, b, c, out1;
  reg o1;

  assign out1 = (a&&b) || (a&&c);

  always @ (posedge clk)
    o1 = out1;
endmodule
```

The above code reports violation for the above example because of the existence of combinational re-convergent fan-out.

Default Severity Label

Warning

Rule Group

Topology

Reports and Related Files

No related reports or files.

Topology_10

Avoid long logic paths

When to Use

Use this rule to identify long logic paths.

Rule Description

The Topology_10 rule reports violation for long logic paths. That is, for each endpoint of long combinational path (path depth greater than what is specified using the *pathDepth* parameter), only one violation is reported, which will report the longest combinational path.

Use the *dft_use_old_Topology_10* parameter to report all combinational paths that were longer than value specified using the *pathDepth* parameter.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *pathDepth*: The default value is 500. Set the value of the parameter to any natural number to set the maximum limit on the number of combinational logic levels that can be present between two registers.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] In design '<top_du>', '<node_count>' node(s) found with maximum combinational logic length '<path_depth>', SPREADSHEET_PATH: '<spreadsheet_path>'

Arguments

- Name of the top design unit, <top_du>
- No of nodes reported, <node_count>
- Depth of the longest path to a particular endpoint, <path_depth>
- Path of the spreadsheet report, <spreadsheet_path>

Potential Issues

See [Potential Issues](#)

Consequences of Not Fixing

See [Consequences of Not Fixing](#)

How to Debug and Fix

See [How to Debug and Fix](#)

Message 2

The Topology_10 rule displays the following violation message when the value of the [dft_use_old_Topology_10](#) parameter is set to on:

The following violation message is displayed for the Topology_10 rule:

[WARNING] In design '<top_du>', '<no_long_paths>' paths have combinational logic length greater than '<path_depth>' (specified)

Arguments

- Name of the top design unit, <top_du>
- Number of long paths, <no_long_paths>
- Depth of the specified path, <path_depth>

Potential Issues

The violation message is reported when there is a long combinational path, that is, path depth greater than the value specified using the [pathDepth](#) parameter.

Consequences of Not Fixing

Not fixing the violation may result in timing problems due to long paths.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the long logic path. The violation message is due to the number of combinational instances in this long logic path. To verify the correctness of violation, you may manually count the combinational instances in this path.

To fix the violation, ensure that the propagation time through combinational logic is less than one tester cycle. Also, avoid the designs that skip a pulse to capture results from long data paths.

Since ATE interfaces usually assume one clock cycle, ensure that the multi-cycle paths are special cased or deleted with a potential loss in fault coverage.

Example Code and/or Schematic

Verilog

Consider the following Verilog code:

```
module LongPath (a, b, clk, o1);
  input a, b, clk; output o1;
  wire a, b, d1, d2, d3, d4, d5;
  reg o1;
  assign d1=a&&b;
  assign d2=d1&&b;
  assign d3=d2&&b;
  assign d4=d3&&b;
  assign d5=d4&&b;

  always @ ( posedge clk)
    o1=d5;
endmodule
```

The above example illustrates a bad design scenario and the Topology_10 rule reports a violation for the above example.

VHDL

Consider the following VHDL code:

```

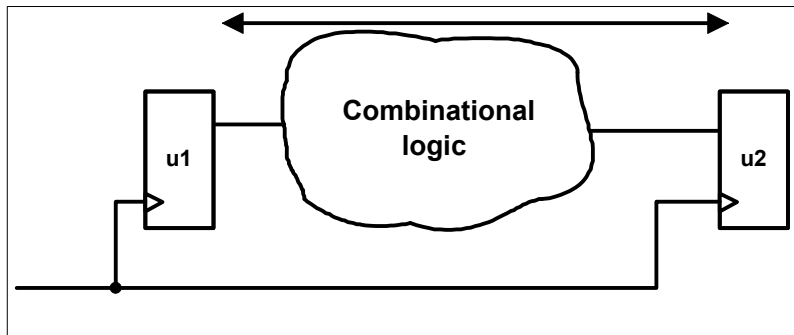
...
Signal a, b, d1, d2, d3, d4, d5 : std_logic;
d1 <= a and b ;
d2 <= d1 and b;
d3 <= d2 and b;
d4 <= d3 and b;
d5<= d4 and b;

Process (clk)
  Begin
    If (clk'event and clk = '1') then
      o1 <= d5;
    End if;
  End process;

```

The above example describes a bad design scenario and the Topology_10 rule reports a violation for the above example.

The following figure illustrates the above example:



Schematic highlight

Combinational path between two end points that have difference of combinational depth greater than the specified depth

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Topology_10 rule in the Incremental Schematic window. To do this, double-click the violation for the

Topology Rules

Topology_10 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Topology_10 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_10 rule.

Default Severity

Label

Warning

Rule Group

Topology

Reports and Related Files

No related reports or files.

Topology_11

Ensure that there are no combinational paths from a clock pin to a root level port in the capture mode

When to Use

Use this rule when you do not want clocks to reach primary ports combinationally or to detect faults in the clock path.

Description

The Topology_11 rule reports violation for clocks that have combinational paths to primary output ports in the capture mode.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *clock* (optional): Defines the clocks of a design. Use the constraint's -testclock argument for this rule.

NOTE: *The Topology_11 requires you to specify at least one testclock.*

- *memory_type* (optional): Specifies the memory design unit (black box) names.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Combinational path exists from clock pin '`<clk-name>`' to port '`<port-name>`'

Arguments

- Name of the clock pin, `<clk-name>`
- Name of the output port to which a combinational path exists, `<port-name>`

Potential Issues

The violation message appears, if a combinational path exists between a clock and a primary port.

Consequences of Not Fixing

Clocks that are combinationaly connected to output pins may block observation of logic feeding those outputs because clocks are typically at the "return to" values when test machines sample circuit outputs.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the combinational path from the clockpin to the output pin.

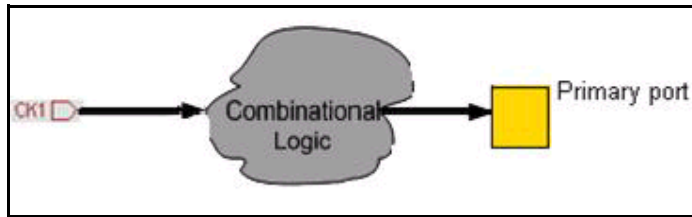
You can also view the violations for the `Info_testmode` and `Info_path` rules along with the violation of the `Topology_11` rule in the Incremental Schematic window. To do this, double-click the violation for the `Topology_11` rule and open the Incremental Schematic window.

The violation messages for the `Info_testmode` and `Info_path` rules overlap the violation message for the `Topology_11` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Topology_11` rule.

To fix the violation, ensure that clock drives a sequential logic only.

Example Code and/or Schematic

Consider the following figure:



The Topology_11 rule reports violation because a combinational path exists between a clock and primary port.

Default Severity Label

Warning

Rule Group

Topology

Reports and Related Files

None

Topology_12

Ensure that all top level inputs are connected to a pullup or pulldown device

When to Use

Use this rule to identify primary inputs that are not connected to a pullup or a pulldown device.

Description

The Topology_12 rule reports violation for primary inputs that are not connected to a pulldown or pullup device.

Method

For each top level input or inout, check that the net connected to that input has a pull-up/pull-down in its fan-out, otherwise a message is reported.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *pulldown* (Mandatory): Specifies the pulldown design units so that various rules can take appropriate actions.
- *pullup* (Mandatory): Specifies the pullup design units so that various rules of SpyGlass DFT ADV product can take appropriate actions.

Operating Mode

None

Messages and Suggested Fix

[WARNING] Input port '<port-name>' is not pinned with pull up/pull down

Arguments

Name of the input port, <port-name>

Potential Issues

A violation is reported when pull-up or pull-down logic is not present at ports.

Consequences of Not Fixing

Not fixing the violation may cause floating nets. It is often a general practice to have pull-up/pull-downs on chip level input pins.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the primary inputs that are not connected to a pullup or a pulldown device.

To fix the violation, add pullup or pulldown logic at ports.

Example Code and/or Schematic

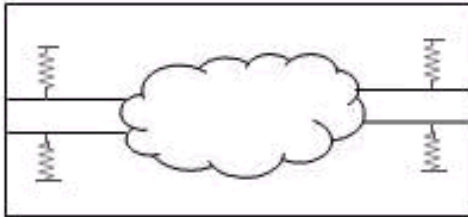
Consider the following figure:



The above figure illustrates a bad design scenario because the primary inputs in the above design are not connected to a pullup or a pulldown device.

Now, consider the following figure:

Topology Rules



The above figure illustrates a good design scenario because pullup/pulldown logic is present at ports.

Default Severity Label

Warning

Rule Group

Topology

Reports and Related Files

No related reports or files.

Topology_13

Reports the presence of combinational re-convergence to flip-flops asynchronous pins

When to Use

Use this rule to avoid unnecessary glitches on reset tree.

Description

The *Topology_13* rule reports violation for re-converging combinational paths, which start from a primary input port or the output pin of a flip-flop and re-converge at the asynchronous set/reset pin of any flip-flop.

If you want to filter violation messages that have re-convergence with different capture and shift values, you can overload message label to:

Topo_13_As_25_M2

NOTE: The *Topology_09* rule flags reconvergent combinational paths between all object types whereas the *Topology_13* rule flags reconvergent combinational paths between primary port/flip-flop output pin and asynchronous set/reset pins only.

Rule Exception

The *Topology_13* rule ignores rule checking for the following cases:

- no scan flip-flops (forced or inferred).
- Unate re-convergence, that is, paths with no phase difference

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.
- *dft_ignore_unate_reconvergence*: The default value is on. Set the value of the parameter to off to check for unate re-convergence.

- *dft_report_all_paths_between_reconvergence_start_and_end*: The default value of this parameter is off. Set the value of the parameter to on to report all paths between re-convergence start and end points.

Constraint(s)

- *test_mode* (Optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[INFO] Reconvergence report file '<report-file-name>' is generated

Arguments

Name of the report file, *<report-file-name>*

Potential Issues

The violation message appears when the text report file is generated.

Consequences of Not Fixing

This is an informational message. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Message 2

[WARNING] Logic '<start_point>' reconverges at or near '<end_point>'. (Actual Reconvergence start = '<rtl_start_point>' reconvergence end = '<rt_end_point>') [Affects '<no_of_set_pins>' SET and '<no_reset_pin>' RESET pin(s)]

Potential Issues

The violation message appears if wrong connectivity on the reset tree exists.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and glitches on the reset tree.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the entire reconvergent combinational paths.

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Topology_13 rule in the Incremental Schematic window. To do this, double-click the violation for the Topology_13 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Topology_13 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_13 rule.

To fix the violation, correct the connection on the reset tree.

Message 3

[WARNING] Logic '`<start_point>`' ('`<shift-sim-val>`', '`<capture-sim-val>`') reconverges at or near '`<end_point>`'. (Actual Reconvergence start = '`<rtl_start_point>`' reconvergence end = '`<rt_end_point>`') [Affects '`<no_of_set_pins>`' SET and '`<no_reset_pin>`' RESET pin(s)]

Potential Issues

The violation message appears if wrong connectivity on the reset tree exists.

Consequences of Not Fixing

Re-convergent fan-out might cause excessive ATPG runtime as well as the possibility of non-static logic that can give rise to bad tests and glitches on the reset tree.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the entire reconvergent combinational paths.

Topology Rules

You can also view the violations for the Info_testmode and Info_path rules along with the violation of the Topology_13 rule in the Incremental Schematic window. To do this, double-click the violation for the Topology_13 rule and open the Incremental Schematic window.

The violation messages for the Info_testmode and Info_path rules overlap the violation message for the Topology_13 rule in the Incremental Schematic window. This is useful in debugging the violation for the Topology_13 rule.

To fix the violation, correct the connection on the reset tree.

Example Code and/or Schematic

Example 1

The following figure illustrates combinational re-convergence structures driving the asynchronous pins:

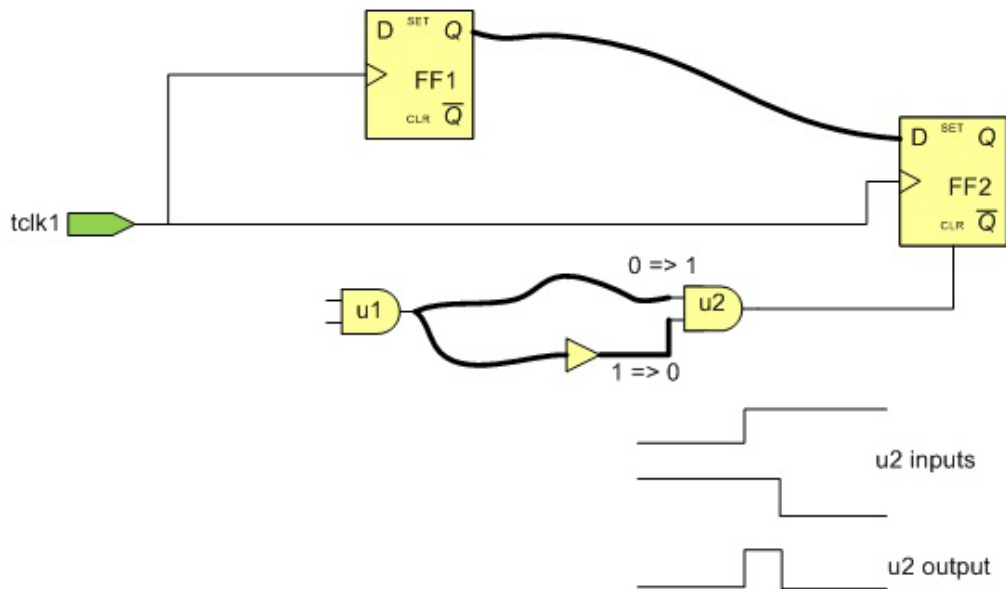
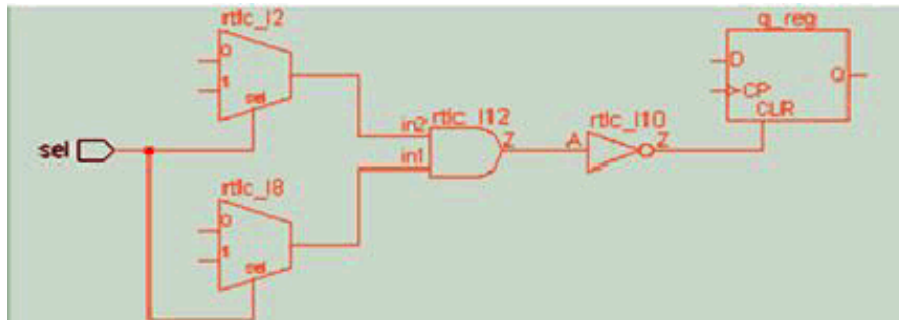


FIGURE 101. Combinational re-convergence feeding clr pin

In the above design scenario, glitches may be produced by ATPG tests, which compromise the test and thereby fault coverage.

Example 2

Consider the following figure:



The Topology_13 rule reports violation for the above example because multiplexer rtlc_l2 and rtlc_l8 are converging.

Default Severity Label

Warning

Rule Group

Topology

Reports and Related Files

- [ff_async_reconvergence.rpt](#): The Topology_13 rule generates a text file named `ff_async_reconvergence` that lists all re-convergent combinational paths that start from a primary input port or the output pin of a flip-flop and re-converge at the asynchronous set/reset pin of a flip-flop.
- **Topology_13_violation_001.csv**: The Topology_13 rule generates the spreadsheet report, `Topology_13_violation_xxx.csv`, which lists the flip-flops whose asynchronous pins are affected by the reconvergent paths. Here, `xxx` denotes a number, which is appended in the report name by SpyGlass, during report generation.

To view the `Topology_13_violation_xxx.csv` report, double-click the message generated for the Topology_13 rule. Alternatively, you can access the report from the following location:

```
../spyglass_reports/dft/Topology_13/<000>/
Topology_13_violation_<000>.csv
```

Figure 102 lists illustrates the Topology_13_violation_001.csv report:

	"Flip-Flop Name"	"Async Pin Type"
1	"top1.\dout_reg[0]"	"RESET"
2	"top1.\dout_reg[1]"	"RESET"
3	"top1.\dout_reg[2]"	"RESET"
4	"top1.\dout_reg[3]"	"RESET"
5	"top1.\dout_reg[4]"	"RESET"
6	"top1.\dout_reg[5]"	"RESET"

FIGURE 102. Topology_13_violation_001.csv Report

Topology_14

Reports if there is a convergence at the async pins of a flip-flop

When to Use

Use this rule to ensure that a single design node drives the async pins of a flip-flop.

Description

The Topology_14 rule reports a violation if an async (set/reset) pin of a flip-flop is driven by multiple design nodes.

Rule Exception

The *Topology_14* rule ignores no scan flip-flops (forced or inferred).

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is `off`. Set the value of the parameter to `on` so that all clocks, except the propagated clock, are set to their off state during clock propagation.

■

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Topology_14 rule uses the `-at speed` argument of the `clock` constraint.

- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] '<number>' design nodes are converging at node '<name>' which drives '<no_of_set_reset_pins>' pins in '<mode>' [Converging nodes: <details>]

Arguments

- Number of converging nodes, <number>
- Node name, <name>
- Number of affected set/reset pins, <no_of_set_reset_pins>
- Operating mode of the rule, <mode>
- Details of the converging node, <details>

Potential Issues

The violation is reported because more than one design nodes are driving the async pins of the reported flip-flops.

Consequences of Not Fixing

Not fixing this violation may cause glitches at the set/reset pin, resulting in an undesired functionality of the flip-flops.

How to Debug and Fix

View the incremental schematic of the violation message. It displays the design nodes that drive the async pins of the reported flip-flops.

To fix the violation, perform the following steps:

1. Verify if all the constraints are present in the SGDC file.
2. Specify any missing constraint in the SGDC file or modify the design.
3. Check message spreadsheet to view the list of affected asynchronous pins.

Example Code and/or Schematic

Consider the following schematic of the Topology_14 rule:

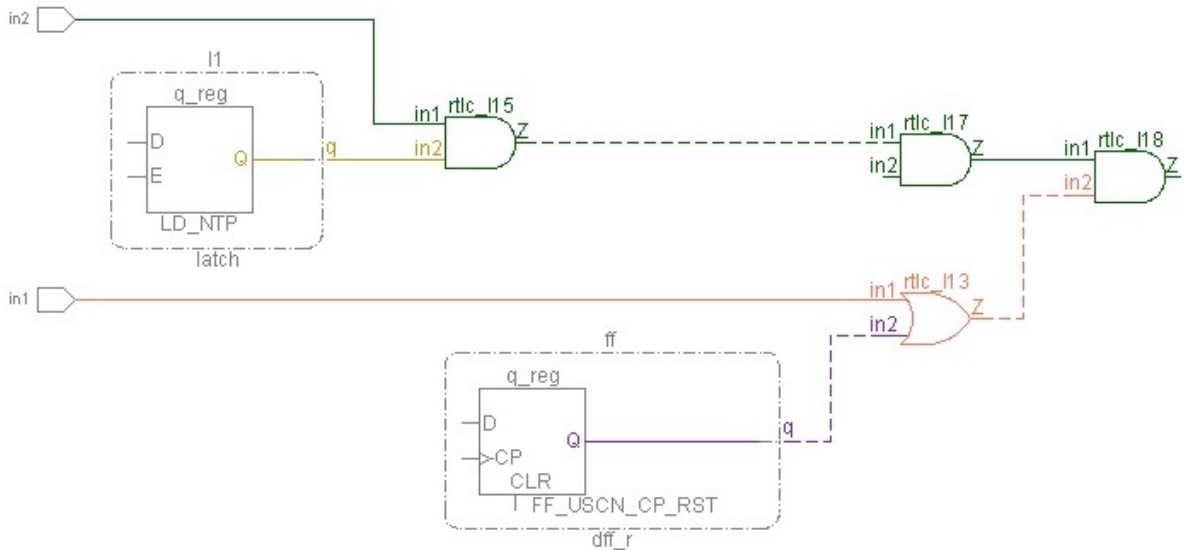


FIGURE 103. Topology_14 Schematic

The Topology_14 rule reports following violation message for the above design:

'4' design nodes are converging at node 'top.rtl_l18' which drives '0 Set and 4 reset' pins in 'Capture mode' [Converging nodes: top.ff.q_reg.Q, in1, top.i1.q_reg.Q, in2]

Double-clicking the violation message displays the following spreadsheet report:

Topology Rules

	B	C	D
	"Source gate"	"Pin name"	"Set/Reset"
1	top.rff1.q_reg	CLR	RESET
2	top.rff7.q_reg	CLR	RESET
3	top.rff8.q_reg	CLR	RESET
4	top.rff6.q_reg	CLR	RESET

FIGURE 104. Topology_14 Spreadsheet report**Default Severity Label**

Warning

Rule Group

Topology Rules

Reports and Related Files

No related reports or files.

Topology_15

Ensure that the top-level output/inout ports, except scanout ports, are controllable

When to Use

Use this rule to ensure that all the inout or output ports except scanout ports are controllable.

Description

The *Topology_15* rule reports violation, if one of the following instances are found in the fanin cone of a primary output/inout port:

- Non-scan flip-flop
- Floating net
- Floating terminals
- Combinational loops
- Black box
- Non-Transparent Latch
- Tied pins/net

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *scan_chain* (optional): Use this constraint to specify the scan chains for the *Topology_15* rule.

- *module_bypass* (optional): Use this constraint to specify modules such as memories that are designed with a bypass between data-in port and dataout port.
- *force_no_scan* (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] <name> port '<name>' has <number> uncontrollable sources (<category-wise-breakup >)in capture mode

Arguments

- Primary output/inout port, <name>
- Port name, <name>
- Total number of uncontrollable conditions, <number>
- Number of violations in each uncontrollability type, <Category-wise-breakup>

Potential Issues

The violation message is reported because at least one of the following sources make the output/inout port except scanout ports uncontrollable in the test mode:

- Non-scan flip-flop
- Floating net
- Floating terminal
- Combinational loops
- Black box
- Non-transparent latch
- Tied pins/net

Consequences of Not Fixing

Not fixing the violation message reduces the test coverage of the SoC, which contains the violating IP.

How to Debug and Fix

View the incremental schematic of the violation message. It displays the Port and the uncontrollable sources.

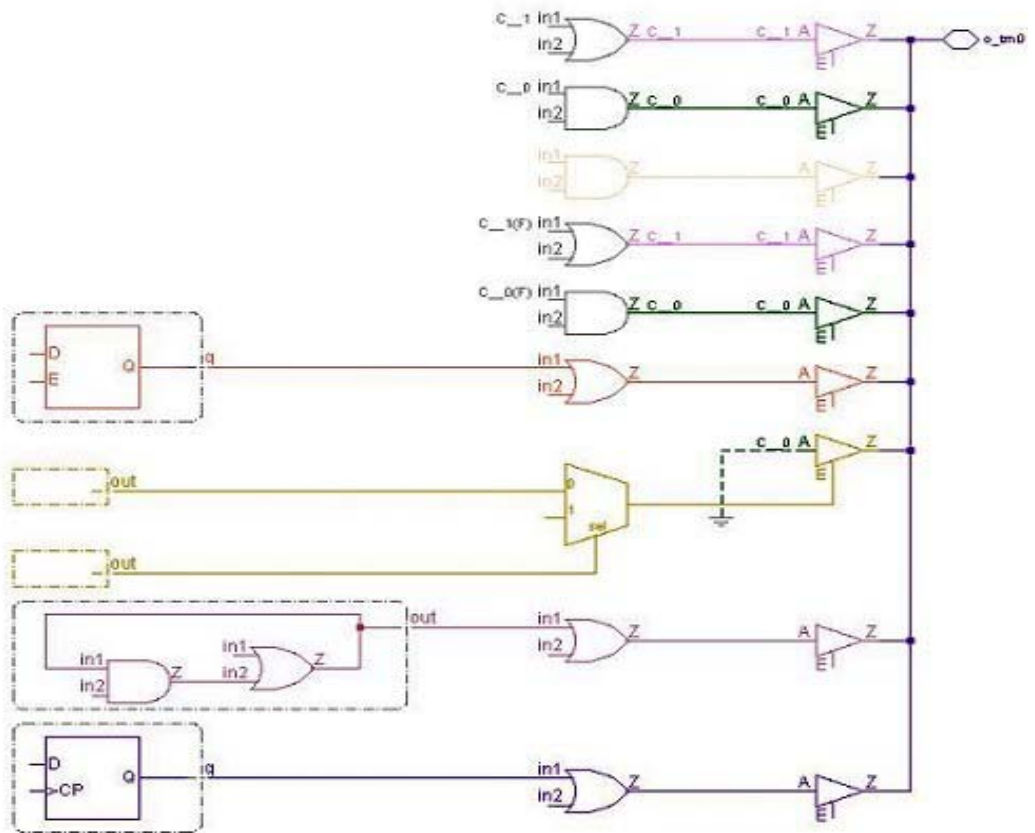
To fix the violation perform one of the following steps:

- Verify whether any of the above sources are generated from constraints in the SGDC file
- Verify whether all the library files are imported properly
- Change the design file for associated violations

Example Code and/or Schematic

The below design illustrates the uncontrollable conditions of the inout port, that is, black boxes, non-transparent latches, no-scan flip-flops, combinational loops, and tied-to-non-X value (1 or 0):

Topology Rules

**Default Severity Label**

Warning

Rule Group

Topology Rules

Reports and Related Files

No related reports or files

Topology_16

Reports flip-flops that may hold the state of 0 or 1 or both due to a feedback path from the Q-pin to its D-pin.

When to Use

To identify all the flip-flops, which may hold the state of 0 or 1 or both due to a feedback path from the Q-pin to its D-pin.

Description

The Topology_16 rule reports flip-flops having feedback path from Q-pin to D-pin. This feedback path enables flip-flops to retain their state 0 or 1 or both.

Such flip-flops may not generate a glitch or transition from a state or from both the states.

Parameter(s)

dft_state_holding_ff_identification_effort_level: The default value of the parameter is 4. Set the value of the parameter to any natural number to change the effort level for identifying the state-holding flip-flops.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Topology_16 rule uses the `-testclock` argument of the `clock` constraint.

Operating Mode

Capture

Messages and Suggested Fix

```
[INFO] Flip-flop '<flip_flop_path>' may hold state  
'<current_state>' in '<operating_mode>'
```

Arguments

- Hierarchical path of the flip-flop, <flip_flop_path>
- Held-State, that is, 0, 1, or both 0 or 1, <current_state>
- Operating Mode, <operating_mode>

Potential Issues

This is an informational rule. Therefore, there are no potential issues pertaining to this violation message.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing this violation message.

How to Debug and Fix

This is an informational rule. Therefore, there are no debug or fix is required for this violation message.

Example Code and/or Schematic

[Figure 105](#) and [Figure 106](#) describe flip-flops that retain the state-1 because the value of the D-pin is 1 whenever Q-pin is at 1.

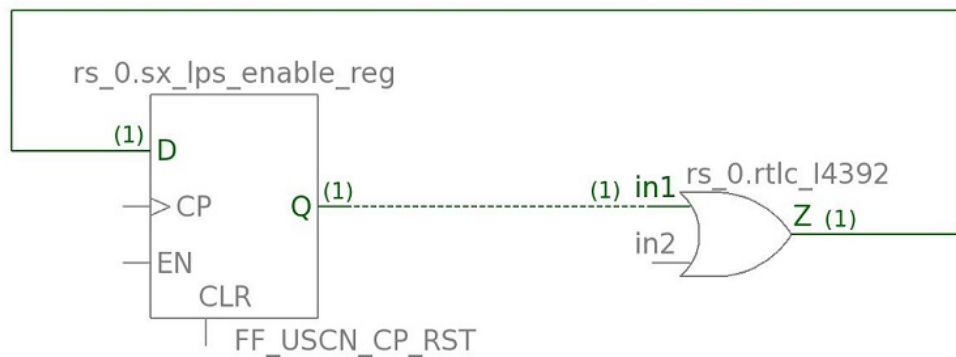
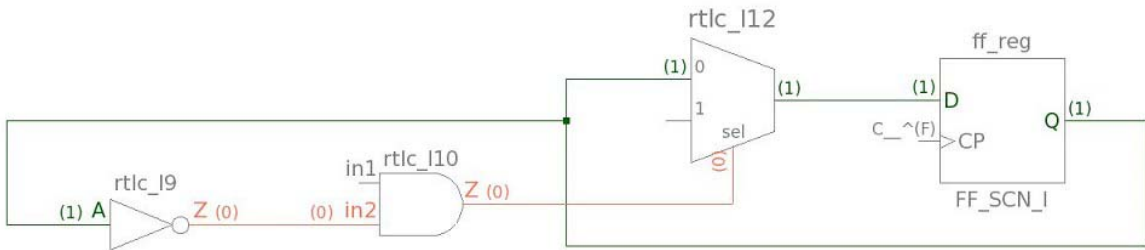


FIGURE 105.

**FIGURE 106.**

Default Severity Label

Info

Rule Group

Topology Rules

Reports and Related Files

No related reports or files

Topology_17

Reports undriven terminals and nets.

When to Use

Use this rule to detect undriven terminals and nets.

Description

This rule reports a violation, if a terminal or net is undriven.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Use this constraint to define the clocks of a design. The Topology_16 rule uses the `-testclock` argument of the `clock` constraint.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Design <design_name> has <number> undriven sources

Potential Issues

The violation message is displayed when a design has an undriven source.

Consequences of Not Fixing

Undriven terminal/nets impact the controllability and observability of the design.

How to Debug and Fix

Modify the design and connect to a suitable source.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Topology Rules

Reports and Related Files

No related reports or files

Tristate Rules

Overview

The `Tristate` rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Flags...
<i>Tristate_01</i>	Inferred tristate components
<i>Tristate_03</i>	Tristate enables that are not disabled in the shift mode
<i>Tristate_04_capture</i>	Inout ports that are not forced to only output mode in the capture mode
<i>Tristate_04_shift</i>	Inout ports that are not forced to only output mode in the shift mode
<i>Tristate_05</i>	Tristate buses that do not have a pullup or pulldown connection
<i>Tristate_06</i>	Tristate buses where more than one driver is active simultaneously
<i>Tristate_07_capture</i>	Inout ports that are not forced to only input mode in the capture mode
<i>Tristate_07_shift</i>	Inout ports that are not forced to only input mode in the shift mode
<i>Tristate_08_capture</i>	Inout ports that are not forced to only input or only output mode in the capture mode
<i>Tristate_08_shift</i>	Inout ports that are not forced to only input or only output mode in the shift mode
<i>Tristate_09</i>	Tristate buses that have bus contention or are floating
<i>Tristate_10</i>	Tristate bus enables driven by flip-flops where capture clock causes x-condition
<i>Tristate_11</i>	Tristate buses where more than one enable is active, all drivers are inactive, or some enables are in x-condition in the shift mode.
<i>Tristate_12</i>	Tristate bus driver enables that are forced on or are disabled in the shift mode

Rule	Flags...
<i>Tristate_13</i>	Lists all tristate buses sorted by the number of tristate drivers driving onto the bus
<i>Tristate_14</i>	Tristate buses where a testclock is connected to the enable pin of the tristate bus
<i>Tristate_15</i>	Primary bidirectional pin driven only by a tristate gate
<i>Tristate_16</i>	Associate a bus keeper to each tri-state
<i>Tristate_17</i>	In logic DBIST mode, preferably keep the direction of bidirectional ports fixed
<i>Tristate_18</i>	In scan mode, have bus keepers fitted with a controllable pullup

Tristate_01

Reports inferred tristate components

When to Use

Use this rule to avoid tristate components.

Description

The Tristate_01 rule reports inferred tristate components.

A message is flagged for any of the following:

- "z" assigns
- ifbuf primitives
- Transistors
- UDPs with z values
- trireg

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Tristate inferred for net '<net-name>'

Arguments

Name of the tristated net, <net-name>

Potential Issues

The violation message appears, if a tristate is inferred for a net.

Consequences of Not Fixing

Not fixing this violation may result in loss of coverage and problems in ATPG control.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the tristate buffer.

You can also view the violation for the Info_testmode rule along with the violation of the Tristate_01 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_01 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_01 rule.

To fix the violation, remove tristate devices and use multiplexers instead. Also, ensure that the assign statements do not specify z values.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
module TriState (en, d, pad);
  input en, d;
  output pad;

  reg pad;

  always
    pad = en ? d : 1'bz;
```

```
endmodule
```

The above example demonstrates a bad design scenario because of the presence of inferred tristate components.

Example 2

Consider the following VHDL code:

```
Architecture arch_TriState of TriState is  
Begin  
    Pad <= d when en = '1' else 'Z';  
End arch_TriState;
```

The above example demonstrates a bad design scenario because of the presence of inferred tristate components.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_03

Tri-state enables should be disabled during scan shifting

When to Use

Use this rule to identify tristate enables that are not disabled in the shift mode.

Description

The Tristate_03 rule reports violation for tristate enables that are not disabled in the shift mode.

Method

Simulate testmode signals. All tristate enables should be inactive, otherwise flag a message.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Tri state control of '<pin-name>' is not disabled in shift mode

Arguments

Name of the tristate output pin, <pin-name>

Potential Issues

A violation is reported when tristate enables are not disabled.

Consequences of Not Fixing

Disabling tristate enables guarantees bus contention during scan shifting.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the tristate buffer and its enable pin and shift mode simulation value on that pin.

To fix the violation, provide constraints to disable tristates or do not use them.

Example Code and/or Schematic**Example 1**

Consider the following sample Verilog code:

```
module Disable_tri (en, d1, d2, tm, o1);
    input en, d1, d2, tm;
    output o1;

    tri o1;

    assign o1 = (!tm&&en) ? d1 : 1'bz;
    assign o1 = !en ? d2 : 1'bz;
endmodule
```

The above example demonstrates a bad Verilog design because tristate enable is used in the design.

Example 2

Consider the following sample VHDL code:

```
Architecture arch_Disable_tri of Disable_tri is
Begin

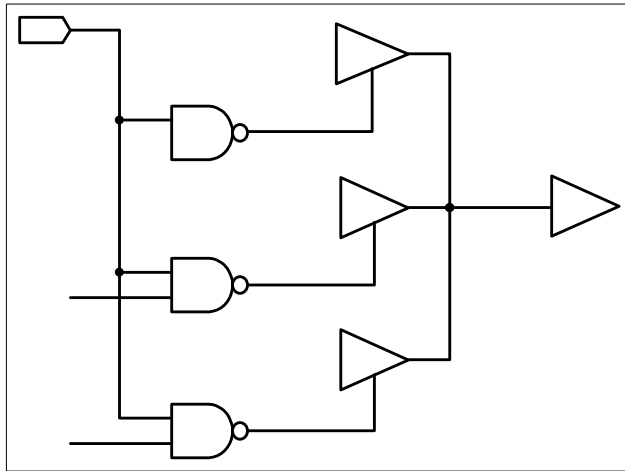
    O1 <= d1 when ((not tm and en) = '1') else 'Z';
```

```
O1 <= d2 when (not en = '1') else 'Z';  
End arch_Disable_tri;
```

The above example demonstrates a bad VHDL design because tristate enable is used in the design.

Example 3

Consider the following schematic:



The Tristate_03 rule reports a violation for the above design because the enable pins are not disabled.

Schematic highlight

Tristate buffer and its enable pin and shift mode simulation value on that pin.

You can also view the violation for the Info_testmode (under shift condition) rule along with the violation of the Tristate_01 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_01 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_01 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_01 rule.

Tristate Rules

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_04_capture

Ensure that inout ports are forced to outputs only in capture mode

When to Use

Use this rule to avoid inout ports in the capture mode.

Description

The Tristate_04_capture rule reports violation for inout ports that are not forced to output mode in during capture. The rule also reports tristate instances that assume a simulation value of X on the enable terminal in capture mode.

The Tristate_04_capture rule checks only if the tristate driver is present on an inout port. If the tristate driver is active, then the IO is considered as an output.

Method

Simulate testmode conditions (without scanshift). All root-level bidirectional pins must have enable pins forced to the active state.

Default Weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (Optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Inout net '<port-name>' direction is not output-only in capture mode

Arguments

Name of the inout port, <port-name>

Potential Issues

The violation message appears, if an inout net is not in output-only direction.

Consequences of Not Fixing

Not fixing the violation may result in design control problem and problems in ATPG testing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the inout port and its connecting net.

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_04_capture rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_04_capture rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_04_capture rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_04_capture rule.

To fix the violation, fix the direction of all input ports as output-only.

Contention between the circuit under test and test machine electronics can be minimized while maximizing observability by ensuring that, while the circuit is in the capture mode, all root-level bidirectional pins are forced to output mode.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

...

```

inout io;

always @(io or a or b)
  if ( a && b ) c = io; // read mode
  else io = a ^ b; // write mode
...

```

The *Tristate_04_capture* rule reports violation for the above example because of the presence of inout ports that are not forced to output mode during capture mode.

Example 2

Consider the following Verilog code:

```

...
inout io;

always @(io or a or b or scan or sdir)
  if ( scan ) begin
    if ( sdir==1'b0 )
      c = io;// read
    else
      io = a^b; //write
    end
  else begin
    if ( a && b )
      c = io; // read
    else
      io = a ^ b; // write
    end
  ...

```

The above example demonstrates a good design scenario.

Example 3

Consider the following VHDL code:

```

...
Port (io : inout std_logic);
Process (io, a, b)
Begin

```

```

    If (a and b) then
      C<=io;
    Else
      Io<= a or b;
    End if;
  End process;
  ...

```

The *Tristate_04_capture* rule reports violation for the above example because of the presence of inout ports that are not forced to output mode during capture mode.

Example 4

Consider the following VHDL code:

```

...
Port (io : inout std_logic);
Process (io, a, b, scan, sdir)
Begin
  If (scan) begin
    If (sdir = '0') then c <= io; --read
  Else io <= a xor b; --write;
    End if;
  Else begin
    If (a and b) then C<=io;
  Else Io<= a xor b;
    End if;
  End if;
End process;
...

```

The above example demonstrates a good design scenario.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

None

Tristate_04_shift

Ensure that the inout ports are forced to outputs only in shift mode

When to Use

Use this rule to use all inout ports in output mode in the shift mode.

Description

The Tristate_04_shift rule reports violation for those inout ports that are not forced to output mode during scan shift. The rule also reports tristate instances that assume a simulation value X on the enable terminal in shift mode.

The Tristate_04_shift rule checks only if the tristate driver is present on an inout port. If the tristate driver is active, then the IO is considered as an output.

Method

Simulate testmode conditions including scanshift. All root-level bidirectional pins must have enable pins forced to the active state.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (Optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Inout net '<port-name>' direction is not output-only in shift mode

Arguments

Name of the inout port, <port-name>

Potential Issues

The violation message appears, if an inout port is not directed as output only in the shift mode.

Consequences of Not Fixing

Not fixing the violation may result in problems in controlling design and problems in ATPG testing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the inout port and its connecting net.

You can also view the violation for the Info_testmode (under shift condition) rule along with the violation of the Tristate_04_shift rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_04_shift rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_04_shift rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_04_shift rule.

To fix the violation make all inout ports output only with help of tristates.

Contention between the circuit under test and test machine electronics can be minimized while maximizing observability by ensuring that, while the circuit is in the shift mode, all root-level bidirectional pins are forced to output mode.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
...  
inout io;
```



```

always @(io or a or b)
  if ( a && b ) c = io; // read mode
  else io = a ^ b; // write mode
...

```

The above example demonstrates a bad design scenario and the `Tristate_04_shift` rule reports violation for the above example because of the presence of inout ports that are not forced to output mode during shift mode.

Example 2

Consider the following Verilog code:

```

...
inout io;

always @(io or a or b or scan or sdir)
  if ( scan ) begin
    if ( sdir==1'b0 )
      c = io;// read
    else
      io = a^b; //write
    end
  else begin
    if ( a && b )
      c = io; // read
    else
      io = a ^ b; // write
    end
  end
...

```

The above example demonstrates a good design scenario.

Example 3

Consider the following VHDL code:

```

...
Port (io : inout std_logic);
Process (io, a, b)
Begin
  If (a and b) then

```

```
        C<=io;
    Else
        Io<= a or b;
    End if;
End process;
...

```

The above example demonstrates a bad design scenario and the `Tristate_04_shift` rule reports violation for the above example because of the presence of inout ports that are not forced to output mode during shift mode.

Example 4

Consider the following VHDL code:

```
...
Port (io : inout std_logic);
Process (io, a, b, scan, sdir)
Begin
    If (scan) begin
        If (sdir = '0') then c <= io; --read
        Else io <= a xor b; --write;
        End if;
    Else begin
        If (a and b) then C<=io;
        Else Io<= a xor b;
        End if;
    End if;
End process;
...

```

The above example demonstrates a good design scenario.

Default Severity Label

Warning

Rule Group

Tristate

Tristate Rules

Reports and Related Files

None

Tristate_05

Tristate buses should have a pull-up or pull-down connection

When to Use

Use this rule to identify tristate buses that do not have a pullup or pulldown connection.

Description

The Tristate_05 rule reports violation for tristate buses that do not have a pullup or pulldown connection.

Method

Report a message on any net driven by a tristate device but not connected to a pullup or pulldown.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[WARNING] Tri state '<net-name>' should use pull-up or pull-down

Arguments

Name of the tristated net, <net-name>

Potential Issues

A violation is reported when a tristate net does not use a pullup or a pulldown connection.

Consequences of Not Fixing

Scan testing may cause unforeseen conditions in which a tristate bus could float. This may result in unobservable faults or excessive ATPG effort.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the tristate buffer and its output net.

To fix the violation, tristate buses should be provided with a pull-up or pull-down to prevent floating condition.

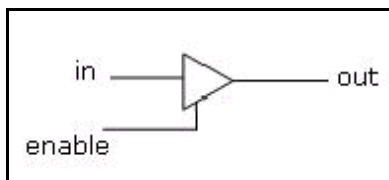
Example Code and/or Schematic

Example 1

Consider the following sample Verilog code:

```
...  
tri out;  
assign out = enable ? in : 1'bz;
```

Consider the following figure with respect to the above code:



The above example demonstrates a bad Verilog design because the tristate bus does not use a pullup or a pulldown connection.

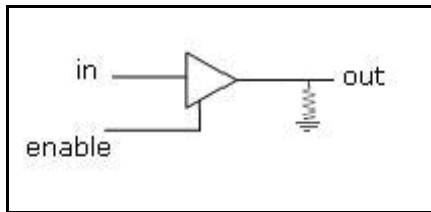
Example 2

Consider the following sample Verilog code:

```
...  
tri out;  
assign out = enable ? in : 1'bz;  
pulldown (out);  
...  

```

Consider the following figure with respect to the above code:



The above example demonstrates a good Verilog design because the tristate bus uses a pull-down connection.

Example 3

Consider the following sample VHDL code:

```
...  
Tri <= in when (enable) else 'Z';  
...  

```

The above example demonstrates a bad VHDL design because the tristate bus does not use a pullup or a pulldown connection.

Example 4

Consider the following sample VHDL code:

```
...  
Tri <= 'L';  
Tri <= in when (enable) else 'Z';  
...  

```

The above example demonstrates a good VHDL design because the tristate bus uses a pull-down connection.

Tristate Rules

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_06

Ensure that the tristate bus enables are fully decoded so that exactly one driver is active at any time

When to Use

Use this rule to identify the tristate buses where more than one driver is active simultaneously.

Description

The Tristate_06 rule reports violation for tristate buses where more than one driver is active simultaneously.

Tristate_06 rule also reports violation for floating tristate buses if the bus does not have a keeper attached to the bus. If a bus does have a keeper and the decode logic permits all enables to be off, the Tristate_06 rule does not flag a violation. Floating tristate buses are those for which all drivers are inactive.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate power and ground and any testmode conditions. For each tristate bus, walk the combinational fan-in cones to the tristate enables but abort the walk on any branch that has a non-x value. Simulate all possible input conditions on the primary inputs or flip-flop outputs reached in the walk back. If any condition makes two or more drivers active, report a message.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *busWidthLow*: The default value is 2. Set the value of the parameter to any positive integer value to specify the minimum limit of number of tristate drivers of a tristated bus.
- *busWidthHigh*: The default value is -1. Set the value of the parameter to any positive integer value to specify the maximum limit of number of tristate drivers of a tristated bus.

- *limitFaninPorts*: The default value is 12. Set the value of the parameter to any natural number to set the maximum limit on the number of objects, such as, ports, latches, flip-flops, and black box instances in the fan-in cone of a net.
- *dftReportWDriversInTriBus*: The default value is off. Set the value of the parameter to on to check if all strong drivers are off and only the weak drivers, that is, pullup, pulldown, keeper, and bus holder are connected to the bus.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *keeper* (optional): Use this constraint to specify buskeeper design units or nets connected to a (virtual) buskeeper design unit instance so that the Tristate_06 rule can take appropriate action.
- *pulldown* (optional): Use this constraint to specify the pulldown design units so that various rules can take appropriate actions.
- *pullup* (optional): Use this constraint to specify the pullup design units so that various rules of SpyGlass DFT ADV product can take appropriate actions.

Operating Mode

Capture

Messages and Suggested Fix

Message 1

[WARNING] No driver of tristate bus '<net-name>' is ON (<total_no_of_drivers> 3s-buffer(s))

Arguments

- Name of rule-violating tristate bus. <net-name>

- Total number of drivers on the bus, <total_no_of_drivers>

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 2

[WARNING] More than 1 drivers of tristate bus '`<net-name>`' are ON (<total_no_of_drivers> 3s-buffer(s))

Arguments

- Name of rule-violating tristate bus. <net-name>
- Total number of drivers on the bus, <total_no_of_drivers>

Potential Issues

For more information on potential issues related to the violation, click [Potential Issues](#).

Consequences of Not Fixing

For more information on consequences of not fixing the violation, click [Consequences of Not Fixing](#).

How to Debug and Fix

For more information on debugging and fixing the violation, click [How to Debug and Fix](#).

Message 3

[WARNING] Tristate bus '`<net-name>`' has no strong driver on, ONLY weak driver exists. (<num1> 3s-buffer(s), <num2> user specified bus keeper(s), <num3> user specified, always active, bus keeper(s))

Arguments

- Name of rule-violating tristate bus. `<net-name>`

- Number of specified type of available drivers, `<num1>`, `<num2>`, `<num3>`.

Potential Issues

A violation is reported due to incorrect or incomplete test_mode conditions or bad design.

Consequences of Not Fixing

Scan testing may momentarily create conditions that cause tristate bus contention which may cause damage to the device. System logic that guarantees only one enable can be active at a time may avoid this problem.

How to Debug and Fix

View the Incremental Schematic of the violation message. The tristate bus and the bus drivers, sequential elements/input ports, are displayed. Overlay the Info_testmode (in auxiliary mode) under shift_mode. Probe the fan-out cone of each driver to see how the path is sensitized from the driver to the enable pin of tristate.

Schematic highlight

Tristate bus and its driver instances in case tristate bus has unknown simulation value in shift mode.

Or

Tristate bus, sequential instances, and input ports which are driving tristate enable, exhaustive simulation on which will result in more than one tristate driver getting enabled.

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_06 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_06 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_06 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_06 rule.

To fix the violation, when all drivers are inactive, use bus-holder/pull-up/pull-down to prevent floating-bus condition.

Example Code and/or Schematic

Example 1

Consider the following Verilog code:

```
...
wire tri;
assign tri = enablea ? in1 : 1'bz;
assign tri = enableb ? in2 : 1'bz;
...
```

The above example illustrates a bad design scenario and the Tristate_06 rule reports a violation for the above example.

To remove the violation, modify the above code as shown below:

```
...
wire tri;
assign en2 = enable ? 1'b0 : 1'b1;
assign tri = en2==1'b0 ? in1 : 1'bz;
assign tri = en2==1'b1 ? in2 : 1'bz;
...
```

Example 2

Consider the following VHDL code:

```
...
Signal tri;
tri <= in1 when (enablea) else 'Z';
tri <= in2 when (enableb) else 'Z';
...
```

The above example illustrates a bad design scenario and the Tristate_06 rule reports a violation for the above example.

To remove the violation, modify the above code as shown below:

```
...
Signal tri;
En2 <= '0' when (enable) else '1';
tri <= in1 when (en2 = '0') else 'Z';
tri <= in2 when (en2 = '1') else 'Z';
...
```

Example 3

The following figure illustrates a wired net having an intermediate voltage:

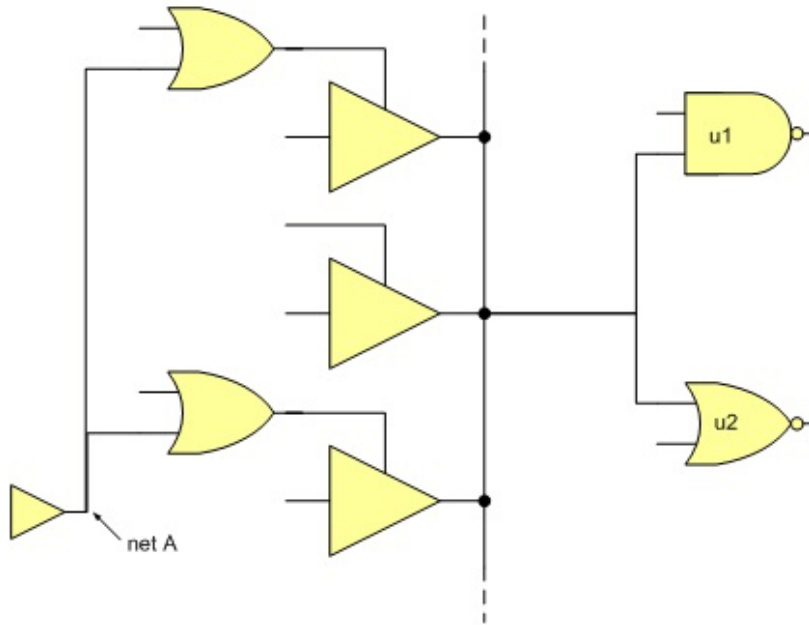


FIGURE 107. Wired net not fully decoded

The devices driven by the wired net, shown in the above figure, may not all recognize the same value. This can corrupt the expected coverage.

Default Severity Label

Warning

Rule Group

Tristate Rules

Reports and Related Files

No related reports or files.

Tristate_07_capture

Ensure that all Inout ports are inputs only in capture mode

When to Use

Use this rule to identify inout ports that are not forced to input mode in the capture mode.

Description

The Tristate_07_capture rule reports violation for inout ports that are not forced to input mode in the capture mode. The rule also reports tristate instances that assume a simulation value of X on the enable terminal in capture mode.

The Tristate_07_capture rule checks only if the tristate driver is present on an inout port. If the tristate driver is inactive, then the IO is considered as an input even if no inward path exists.

Method

Simulate testmode conditions (without scanshift). All root-level bidirectional pins must have their enable pins at their inactive value.

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (Optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Inout net '`<port-name>`' direction is not input-only in capture mode

Arguments

Name of the inout port. (`<port-name>`)

Potential Issues

A violation is reported when inout ports are not configured as input.

Consequences of Not Fixing

Not fixing the violation results in increased contention between the circuit under test and the test machine electronics. It also decreases the controllability of a circuit

How to Debug and Fix

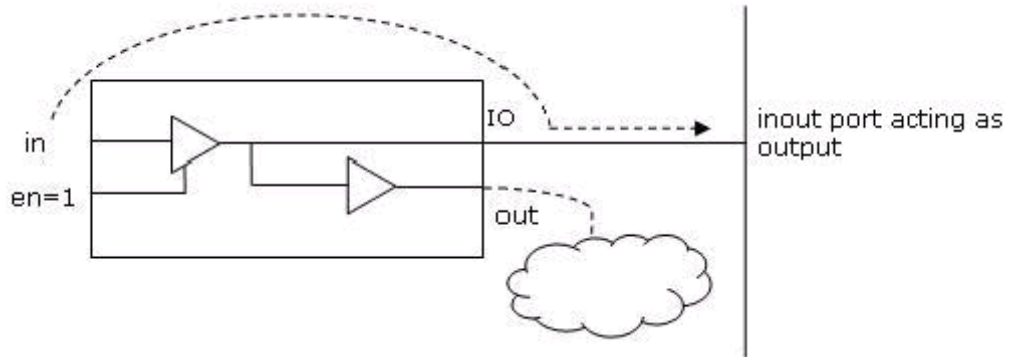
View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the tristate buffer and its output net.

To fix the violation, ensure that when the circuit is in the capture mode, all root level bidirectional pins are forced to input mode.

Example Code and/or Schematic

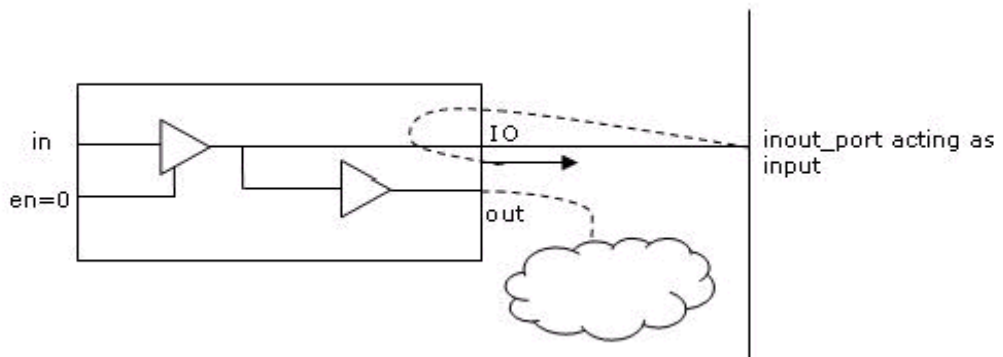
Consider the following figure:

Tristate Rules



The above figure illustrates a bad design scenario because the inout port is not configured as input.

Now, consider the following figure:



The above figure illustrates a good design scenario because inout port is configured as input.

Schematic highlight

Tristate buffer and its output net

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_07_capture rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_07_capture rule and open the Incremental Schematic

window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_07_capture rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_07_capture rule.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_07_shift

All Inout ports are inputs only in shift mode

When to Use

Use this rule to identify inout ports that are not forced to input mode in shift mode.

Description

The Tristate_07_shift rule reports violation for inout ports that are not forced to input mode in the shift mode. The rule also reports tristate instances that assume a simulation value of X on the enable terminal in shift mode.

The Tristate_07_shift rule checks only if the tristate driver is present on an inout port. If the tristate driver is inactive, then the IO is considered as an input, even if no inward path exists.

Default Weight

10

Language

Verilog, VHDL

Method

Simulate testmode conditions including scanshift. All root-level bidirectional pins must have their enable pins at their inactive value.

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (Optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

The following violation message is displayed for the Tristate_07_shift rule:

[WARNING] Inout net '<port-name>' direction is not input-only in shift mode

Arguments

Name of the inout port. (<port-name>)

Potential Issues

A violation is reported due to incomplete / incorrect test_mode conditions or wrong design connections.

Consequences of Not Fixing

Not fixing the violation may result in one of the following conditions:

- Does not minimize the contention between circuit under test and test machine electronics.
- Does not maximize the controllability of the circuit.

How to Debug and Fix

View the Incremental Schematic of the violation message. The Incremental Schematic highlights the tristate, which does not have strict '0' on its enable pin in shift mode. Overlay the Info_testmode (in auxiliary mode) rule under shift mode condition to see how shift mode conditions are propagated.

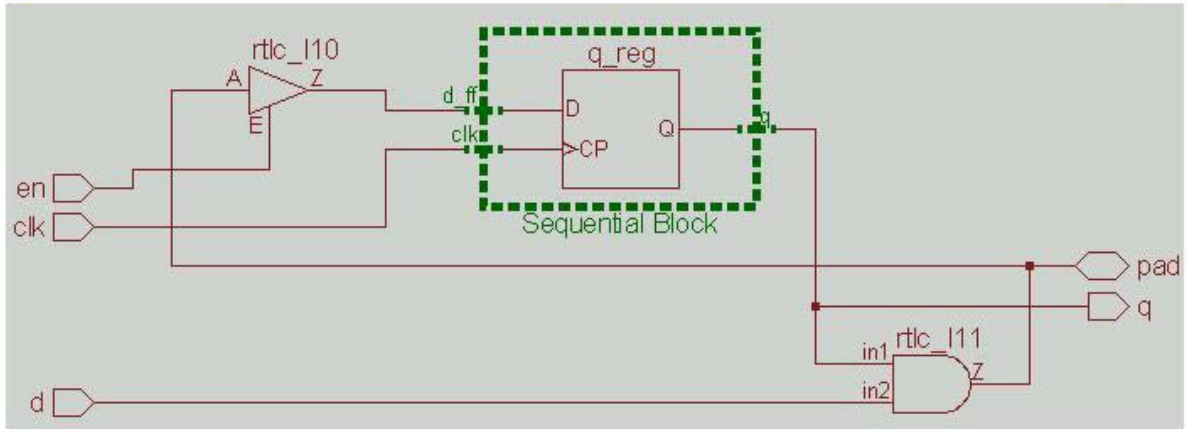
To fix the violation, use test-mode controls. Test-mode controls help prevent contention between values produced by the inout pin and the value from the test machine.

Contention between the circuit under test and the test machine electronics can be minimized and the controllability of a circuit maximized by ensuring that, while the circuit is in the shift mode, all root level bidirectional pins are forced to input mode.

Example Code and/or Schematic

Consider the following example:

```
current_design test
testmode -name en -value 0 -scanshift
```



Schematic highlight

Tristate buffer and its output net

You can also view the violation for the Info_testmode (under shift condition) rule along with the violation of the Tristate_07_shift rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_07_shift rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_07_shift rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_07_shift rule.

Default Severity Label

Warning

Rule Group

Tristate

Reports and

Related Files

No related reports or files.

Tristate_08_capture

Reports inout ports not forced to only input or only output direction in the capture mode

When to Use

Use this rule to keep all inout ports in either only input mode or only output mode in the capture mode.

Rule Description

The Tristate_08_capture rule reports violation for inout ports that are not forced or are not forced to the same direction (input or output) in the capture mode. The rule also flags Tristate instances that assume a simulation value of X on the enable terminal in capture mode.

The Tristate_08_capture rule checks only if the tristate driver is present on an inout port. If the driver is active, then the IO is considered as an output. If the tristate driver is inactive, then the IO is considered as an input, even if no inward path exists.

Method

Simulate testmode conditions (without scansift). Find the direction for each inout. If any inout has the direction “both” or if an inout is found with a direction not matching the direction of the first inout found, then report a message.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (Optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test

mode.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Design '`<du-name>`' has `<count>` inout ports (`<num1>` input; `<num2>` output; `<num3>` unknown)

Arguments

- Name of the design unit. (`<du-name>`)
- Total number of inout ports in the design unit. `<count>`
- Number of inout ports that are in only input mode. `<num1>`
- Number of inout ports that are in only output mode. `<num2>`
- Number of input ports that are in both input and output modes, or unknown mode. `<num3>`

Potential Issues

The violation message appears, if a design has inout ports that are not in the same mode.

Consequences of Not Fixing

Not fixing the violation may result in design complexity and problems in ATPG testing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the inout ports and their nets, and enable pins of connecting tristates, if any.

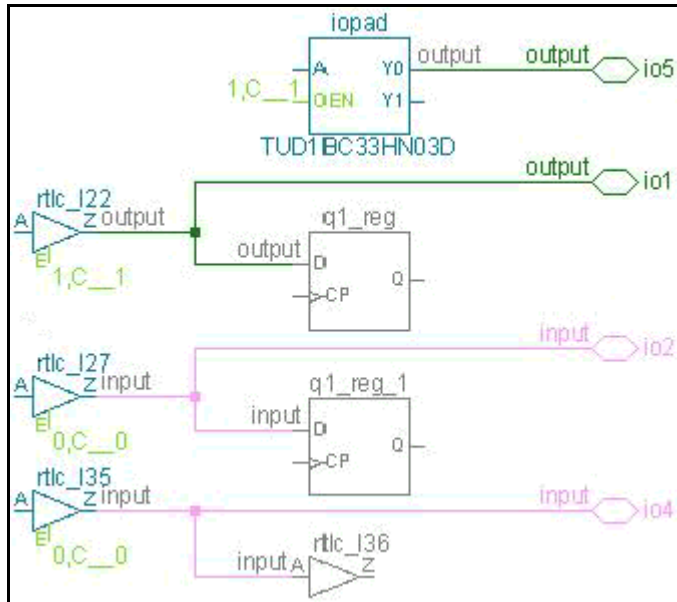
You can also view the violation for the `Info_testmode` (under capture condition) rule along with the violation of the `Tristate_08_capture` rule in the Incremental Schematic window. To do this, double-click the violation for the `Tristate_08_capture` rule and open the Incremental Schematic window.

The violation message for the `Info_testmode` rule overlaps the violation message for the `Tristate_08_capture` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Tristate_08_capture` rule.

Ensure to fix all the inout and output ports in the same mode.

Example Code and/or Schematic

Consider the following figure:



In the above example, the *Tristate_08_capture* rule reports a violation because all inout ports are not only input or only output ports. Here, *io2* and *io4* work as input ports and *io1* and *io5* work as output ports.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

None

Tristate_08_shift

Reports inout ports not forced to only input or only output mode in the shift mode

When to Use

Use this rule to keep all inout ports in the same mode, either input mode or output mode in the shift mode.

Description

The Tristate_08_shift rule reports violation for inout ports that are not forced or are not forced to the same direction (input or output) in the shift mode. The rule also reports tristate instances that assume a simulation value of X on the enable terminal in shift mode.

The Tristate_08_shift rule checks only if the tristate driver is present on an inout port. If the driver is active, then the IO is considered as an output. If the tristate driver is inactive, then the IO is considered as an input, even if no inward path exists.

Method

Simulate testmode conditions for scanshift. Find the direction for each inout. If any inout has the direction “both” or if an inout is found with a direction not matching the direction of the first inout found, then report a message.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (Optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test

mode.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Design '`<du-name>`' has `<count>` inout ports (`<num1>` input; `<num2>` output; `<num3>` unknown)

Arguments

- Name of the design unit, `<du-name>`
- Total number of inout ports in the design unit, `<count>`
- Number of inout ports that are in only input mode, `<num1>`
- Number of inout ports that are in only output mode, `<num2>`
- Number of input ports that are in both input and output modes, or unknown mode, `<num3>`

Potential Issues

The violation message appears, if a design has inout ports that are not forced to only input or only output mode in the shift mode.

Consequences of Not Fixing

Not fixing the violation may result in design complexity and problems in ATPG testing.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights all inout ports and their nets, and enable pins of connecting tristates, if any.

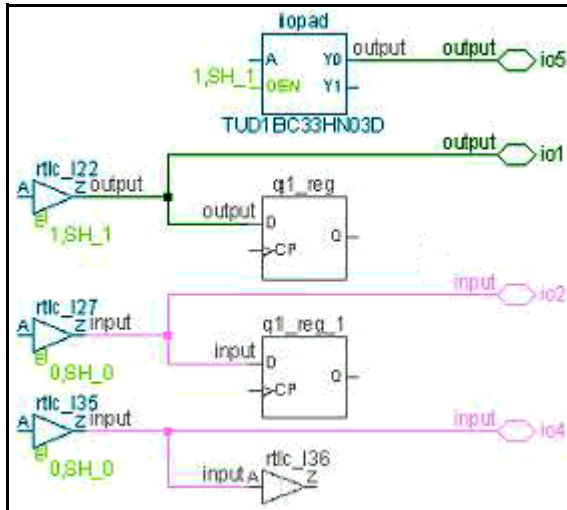
You can also view the violation for the `Info_testmode` (under shift condition) rule along with the violation of the `Tristate_08_shift` rule in the Incremental Schematic window. To do this, double-click the violation for the `Tristate_08_shift` rule and open the Incremental Schematic window.

The violation message for the `Info_testmode` rule overlaps the violation message for the `Tristate_08_shift` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Tristate_08_shift` rule.

Ensure to fix all the inout ports in same mode.

Example Code and/or Schematic

Consider the following figure:



In the above example, the *Tristate_08_shift* rule reports a violation because all inout ports are not only input or only output ports in the shift mode. Here, io5 and io1 work as output ports whereas io4 and io2 work as input ports.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_09

Ensure that the tristate buses are designed to prevent bus contention and floating during scan shift

When to Use

Use this rule when the design has tristates and to avoid poor design control.

Description

The Tristate_09 rule reports violation for tristate buses that have bus contention or are floating during scan shift.

The Tristate_09 rule checks for absence of logic that can result in floating nets or bus contention. Use waivers if any of checks are not applicable for a particular design. Also, see [Tristate_06](#) rule.

Method

Simulate testmode with scanshift

For each tri-state bus

If the bus has a value of 'x' report a message

If the bus has a value of 'z' report a message

End for

Simulate capture mode (testmode without -scanshift argument)

For each tri-state bus

If the bus has two or more "always on" drivers, report a message

If the bus has at least one tri-state driver and one or more "always on" drivers, report a message

If the bus has at least one tri-state driver and one or more black boxes, report a message

If the bus does not have one keeper or one pullUp or one pullDown, report a message

If any driver does not have an enable pin, report a message and break out.

Walk the unblocked combinational input cone for all driver enable pins.

Simulate all possible combinations on these inputs

If any combination causes all drivers (including pullup and pulldown devices) to be off or 2 or more drivers (including pullup and pulldown devices) to be on than report a message.

If any enable is never on, report a message.

end for

Default weight

1

Language

Verilog, VHDL

Parameters

- *limitFaninPorts*: The default value is 12. Set the maximum limit on the numbers of objects (ports, latches, flip-flops, and black box instances) in the fan-in cone of a net.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *keeper* (optional): Use this constraint to specify buskeeper design units or nets connected to a (virtual) buskeeper design unit instance so that the Tristate_06 rule can take appropriate action.
- *pulldown* (optional): Use this constraint to specify the pulldown design units so that various rules can take appropriate actions.
- *pullup* (optional): Use this constraint to specify the pullup design units so that various rules of SpyGlass DFT ADV product can take appropriate actions.
- *test_mode* (Optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode. Use this constraint's `-scanshift` argument for the *Tristate_09* rule.

Operating Mode*Scanshift, Capture***Messages and Suggested Fix****Message 1**

Some of the drivers of tristate bus '<net-name>' are never ON (<num1> user specified pullup(s)/pulldown(s), <num2> 3s-

buffer(s))

Arguments

To view the argument description, click [Arguments](#).

Potential Issues

The violation message appears, if some of the drivers of tristate bus are never ON.

Consequences of Not Fixing

Not fixing the violation may result in less coverage due to poor control over design.

How to Debug and Fix

To debug the violation, click [How to Debug and Fix](#).

To fix the violation, replace those tristates or change the logic.

Message 2

Tristate bus '<net-name>' has unknown value during shift mode (<num3> non-3s device(s), <num2> 3s-buffer(s))

Potential Issues

The violation message appears, if the tristate bus has unknown value during shift mode.

Consequences of Not Fixing

Not fixing the violation may result in unpredictable behavior of the design.

How to Debug and Fix

To debug the violation, click [How to Debug and Fix](#).

To fix the violation, change the logic and use tristate devices only.

Message 3

Tristate bus '<net-name>' has no (keeper or pull up or pull down)

Potential Issues

The violation message appears, if a tristate bus does not have any keeper.

Consequences of Not Fixing

Not fixing the violation may result in floating logic when none of the drivers are working.

How to Debug and Fix

To debug the violation, click [How to Debug and Fix](#).

To fix the violation, connect a keeper with the tristate bus.

Message 4

More than 1 drivers of tristate bus '`<net-name>`' are ON (`<num1>` user specified pull up(s)/pull down(s), `<num2>` 3s-buffer(s))

Potential Issues

The violation message appears, if more than one drivers of the tristate bus are ON.

Consequences of Not Fixing

Not fixing the violation may result in less coverage due to poor control over design.

How to Debug and Fix

To debug the violation, click [How to Debug and Fix](#).

To fix the violation, ensure to change the logic.

Message 5

Drivers of tristate bus '`<net-name>`' are `<num3>` non-3s device(s), `<num1>` user specified pull up(s)/pull down(s), `<num2>` 3s-buffer(s)

Potential Issues

The violation message appears, if drivers of tristate bus are not three state devices.

Consequences of Not Fixing

Not fixing the violation may result in bad design as we need only one driver to be working which is difficult without tristates.

How to Debug and Fix

To debug the violation, click [How to Debug and Fix](#).

To fix the violation, replace AND gates with tristates.

Message 6

Tristate bus '`<net-name>`' has unknown value during shift mode (`<num2>` 3s-buffer(s))

Arguments

- Name of the tristate bus, <net-name>
- Number of user-specified pullups and pulldowns, <num1>
- Number of 3-state buffers, <num2>
- Number of non-3-state buffers, <num3>

Potential Issues

The violation message appears, if tristate bus has unknown value during shift mode.

Consequences of Not Fixing

Not fixing the violation may result in less coverage as it will act as floating point.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the tristate bus and nodes driving the enable decode logic fan-in cone.

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_09 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_09 rule and open the Incremental Schematic window.

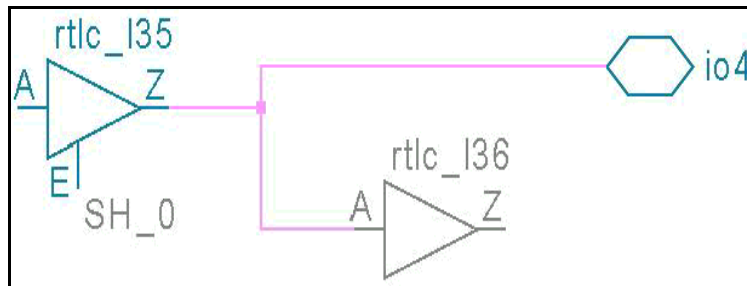
The violation message for the Info_testmode rule overlaps the violation message for the Tristate_09 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_09 rule.

To fix the violation, ensure to change the logic.

Example Code and/or Schematic

Consider the following figure:

Tristate Rules



In Tristate_09 rule reports violation for the above example because the tristate, rtlc_l35, is in floating state.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_10

Reports tristate bus enables that are driven by flip-flops where capture clock causes x-condition

When to Use

Use this rule to avoid X-condition of tristate bus.

Description

The Tristate_10 rule reports violation for those tristate bus enables driven by flip-flops where capture clock causes x-condition.

Method

Determine scannability

Perform standard testability analysis

Save these results as “scan” controllability and “scan” observability

// Mark all devices that become N-clock capture uncontrollable

Simulate power and ground and any testmode for capture.

Perform standard testability analysis

Let N = captureClockCount

If captureClockStart is not specified then set captureClockStart = 1

If captureClockEnd is not specified then set captureClockEnd = captureClockCount

For each flip-flop with imperfect d-pin controllability

set the q-pin controllability = d-pin controllability

tag this flip-flop with CC=1 //capture clock = 1 causes the q to be uncontrollable

tag all flip-flops within N-1 clocks of “this” flip-flop with (CC=1+depth from this flip-flop)

//The “tag” will be used to compare against captureClockStart and captureClockEnd

end for

Set mode to seqATPG

Recompute controllability for all “marked” devices.

// Create causes of capture clock uncontrollability report

initialize a report list to empty

for each flip-flop that has become capture clock uncontrollable

Create report entry of the full path name for this flip-flop instance

walk the capture uncontrollable fan-in cone

if the walk reaches a capture x-source

add this source as a cause for the starting flip-flop

stop the walk on this branch

end if

```

end for
// Note: any x-source may be listed for more than one flip-flop
// Note: in each of the capture check violations in this rule, the x-sources must be listed
for each of the violating flip-flops hit in a walk back.
If user supplied fully decoded list exists and user supplied mux list exists then set
MUXLIST then report a fatal error and stop processing
For each tri-state bus,
//Note: the following 4 if tests are mutually exclusive
(1)If user supplied BUS list exists,
if bus is on this list and has capture violation then report a message and go to next bus
else go to next bus
endif
(2)If user supplied bus width range exists then:
If bus is within this range then
If this bus is fully decoded (use Tristate_06 method) then go to next bus
If this bus fails capture check then report a message and go to next bus
If this bus is not within range and fails capture check then report a message and go to
next bus
(3)If user supplied fully decoded list exists then
If this bus is on the list, then go to next bus
If this bus fails capture check then report a message and go to next bus
(4)If this bus fails capture check then report a message and go to next bus
end for
Method for capture check for a bus
Simulate power, ground and testmode
For each tri_state bus driver
Walk unblocked fan-in cone for driver enable
Stop the walk at flip-flops, floating nets and black boxes
If any flip-flop, encountered in the walk back has "CC" within captureClockStart and
captureClockEnd, mark this flip-flop for message reporting.
end for
If one or more flip-flops have been marked then report all flip-flops and the actual x-
source that fed this flip-flop.

```

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *busWidthLow*: The default value is 2. Set the value of the parameter to any positive integer value to specify the minimum limit of number of tristate drivers of a tristated bus.
- *busWidthHigh*: The default value is -1. Set the value of the parameter to any positive integer value to specify the maximum limit of number of tristate drivers of a tristated bus.
- *captureClockCount*: The default value is 1. Set the value of the parameter to any positive integer value to specify the number of capture clocks for atspeed testing and various bus tests.
- *captureClockEnd*: The default value is 1. Set the value of the parameter to any positive integer value to specify the last capture clock on which the Tristate_10 rule should report message.
- *captureClockStart*: The default value is 1. Use to specify the first capture clock on which the Tristate_10 rule reports messages.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

When the *busWidthLow* and *busWidthHigh* rule parameters are set to 0 (the default value) then Tristate_10 rule operates as it currently does. When you specify these rule parameters with positive integer values ($0 \leq \$busWidthLow \leq \$busWidthHigh$), then the Tristate_10 rule performs the analysis only on buses whose driver count is within this range.

Constraint(s)

- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

Operating Mode

Capture

Messages and Suggested Fix

[WARNING] Enable of tri bus '<bus-name>' is driven by flip-flop where capture would cause x-condition

Arguments

Name of rule-violating tristate bus, <net-name>

Potential Issues

The violation message appears, if a flip-flop drives a tri-bus where capture causes x-condition.

Consequences of Not Fixing

Not fixing the violation may result in chip deterioration and design can be affected.

Capture clock pulse that result in x-conditions in scan flip-flops that feed tri-state enables driving buses can cause bus contention.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the tri-state bus and the flip-flop with the x-capture.

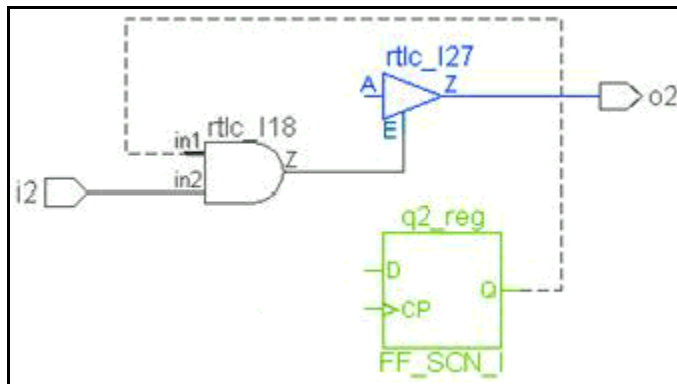
You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_10 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_10 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_10 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_10 rule.

To fix the violation, remove the x-cause or block the x-sources from the capture scan flip-flop.

Example Code and/or Schematic

Consider the following figure:



The *Tristate_10* rule reports a violation for the above example because the enable pin, *E*, of tristate, `rtlc_127`, is driven by a flip-flop, `FF_SCN_1`, where capture clock, *CP*, causes X condition.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

[dft_ff_X_source_for_tristate_enable](#): This report contains a list of all the flip-flops that could cause X condition on the tri-enables.

Tristate_11

Ensure that only one tri-state enable is active in shift mode

When to Use

Use this rule when design has tristate buses and you want complete coverage over the design.

Description

The Tristate_11 rule reports violation for those tristate buses where:

- More than one enable is active
- All drivers are inactive, or
- Some enables are in x-condition in the shift mode.

The Tristate_11 rule requires that only one tristate bus enable is active in the shift mode. All other cases are flagged.

The Tristate_11 rule is designed for buses with bus protection circuitry that guarantees contention cannot exist.

To see testmode values on the tristate enables, please superimpose the [Info_testmode](#) rule message onto a Tristate_11 rule message. Use of the incremental schematic is recommended.

Method

Simulate power, ground and any available testmode conditions for scan shift.

For each tristate bus report a message if more than one driver is enabled or if all drivers are disabled or if some driver enables are x.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [busWidthLow](#): The default value is 2. Set the value of the parameter to any positive integer value to specify the minimum limit of number of tristate drivers of a tristated bus.

- *busWidthHigh*: The default value is -1. Set the value of the parameter to any positive integer value to specify the maximum limit of number of tristate drivers of a tristated bus.
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

test_mode (optional): Use to specify the conditions that force the circuit in the test mode after simulation.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Tri-stated bus '<bus-name>' has <driver-status>

Arguments

- Tristate bus name., <bus-name>
- Driver status, <driver-status>

Potential Issues

The violation message appears, if more than one enable is active or all drivers are inactive or some enables are in x-condition in the shift mode.

Consequences of Not Fixing

If you do not fix this violation, it may result in less coverage due to poor control.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the violating tristate bus.

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_11 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_11 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_11 rule in the Incremental Schematic window.

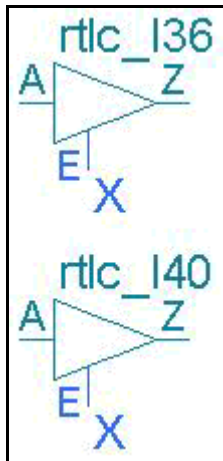
Tristate Rules

This is useful in debugging the violation for the Tristate_11 rule.

To fix the violation, ensure to change the logic as only one tristate should be enabled.

Example Code and/or Schematic

Consider the following figure:



The Tristate_11 rule reports violation for the above example because both the tristates are in X condition.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_12

Bus enables should not be forced in scan shift.

When to Use

Use the Tristate_12 rule for buses whose drivers should not have active values during scan shift.

Description

The Tristate_12 rule reports violation for those tristate bus driver enables that are forced on or are disabled in the shift mode.

Method

Simulate power, ground and testmode for scan shift

For each tristate bus within the range limits

 if any driver enable has a non-x value, report a message

end for

Language

Verilog, VHDL

Default Weight

1

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

test_mode (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

Scanshift

Messages and Suggested Fix

[WARNING] Tri-stated bus '<bus-name>' has <driver-status>

Arguments

- Tristate bus name, <bus-name>
- Driver status, <driver-status>

Potential Issues

A violation is reported when tristate enable is forced to `on` or `off`.

Consequences of Not Fixing

If violation is not fixed, test pattern will not control the enables of tristate.

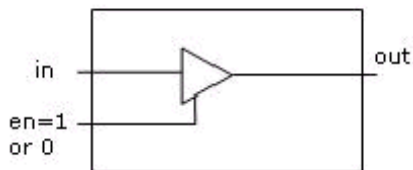
How to Debug and Fix

View the Incremental Schematic of the violation message. The Schematic Viewer window highlights the tristate bus.

To fix the violation, one application is to ensure that bus protection is off when intended.

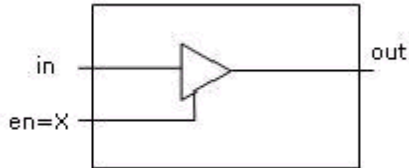
Example Code and/or Schematic

Consider the following figure:



The above figure illustrates a bad design scenario because tristate enable, `en`, is forced to `on` or `off`.

Now, consider the following figure:



The above figure illustrates a good design scenario because tristate enable is forced to an unknown value.

Schematic highlight

Tristate bus

You can also view the violation for the Info_testmode (under capture condition) rule along with the violation of the Tristate_12 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_12 rule and open the Incremental Schematic window.

The violation message for the Info_testmode rule overlaps the violation message for the Tristate_12 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_12 rule.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_13

Generates bus width report.

Description

The Tristate_13 rule generates a tristate bus report sorted by the number of tristate drivers driving onto each bus.

The Tristate_13 rule generates a report named, [dft_tristate](#).

Method

Simulate testmode under capture conditions.

Iterate over netlists for each net > 1 fan-in, report the number of active tristate drivers.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

None

Messages and Suggested Fix

[WARNING] Bus-width report 'dftBusWidth.rpt' is generated for module '<du-name>'

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message appears when the Tristate_13 rule generates the bus-width report.

Consequences of Not Fixing

There are no direct consequences of not fixing this violation message.

How to Debug and Fix

No debug or fix is required for this violation message.

Example Code and/or Schematic

Not Applicable

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

dft_tristate: Lists all tristate buses in the circuit.

Tristate_14

Ensure that the testclock are not connected to the enable pin of tristate buffer

When to Use

Use this rule to get full coverage on enable pins of tristates.

Description

The Tristate_14 rule reports violation for tristate buses where a testclock is connected to the enable pin of the tristate bus.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

Operating Mode

None

Messages and Suggested Fix

[WARNING] Enable of tri bus '`<bus-name>`' is driven by test-clock '`<tcl k-name>`'

Arguments

- Tristate bus name, <bus-name>
- Name of the testclock connected to the enable pin of the tristate bus, <tclk-name>

Potential Issues

The violation message appears, if a tri-bus is driven by test clock.

Consequences of Not Fixing

Not fixing the violation may result in reduced coverage and multiple driver stage.

The bidirectional pins of I/O have the tristate buffers in the design. If a testclock is connected to the enable pin of the tristate buffer, the ATPG tool does not generate the test pattern.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path from the testclock to the enable pin of tristate bus.

You can also view the violation for the Info_testclock rule along with the violation of the Tristate_14 rule in the Incremental Schematic window. To do this, double-click the violation for the Tristate_14 rule and open the Incremental Schematic window.

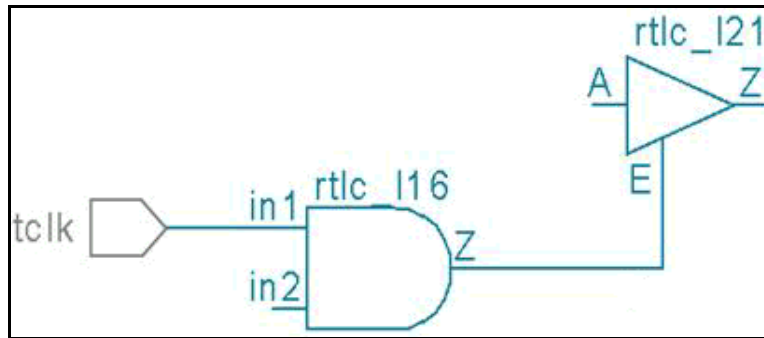
The violation message for the Info_testclock rule overlaps the violation message for the Tristate_14 rule in the Incremental Schematic window. This is useful in debugging the violation for the Tristate_14 rule.

To fix the violation, remove the testclock from enable pins of tristates.

Example Code and/or Schematic

Consider the following figure:

Tristate Rules



The *Tristate_14* rule reports violation for the above example because testclock, `tclk`, is connected to enable pin, `E`, of the tristate, `rtlc_l21`.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_15

Ensure that the primary bidirectional pin is only driven by a tristate gate

When to Use

Use this rule to use pins as inout ports.

Description

The Tristate_15 rule reports violation for those top level inout ports that are not driven by tristate gates.

Rule Exceptions

While checking for connection between an inout port and a tristate gate, SpyGlass ignores buffers if they result from a continuous assignment statement. However, user-instantiated buffers are not ignored.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[WARNING] Top level bidirectional port <port-name> not driven

by tri state gates

Arguments

Top level bidirectional port name, <port-name>

Potential Issues

The violation message appears when a bidirectional port is not driven by tristate gates.

Consequences of Not Fixing

Not fixing the violation may result in lack of inout ports and problems in ATPG testing.

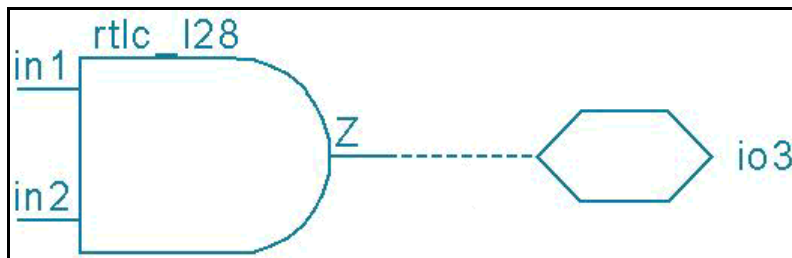
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path from bidirectional pin to non-tristate instance.

To fix the violation, assign tristate the pins which are not connected to tristates.

Example Code and/or Schematic

Consider the following figure:



The Tristate_15 rule reports violation for the above example because an inout port is not driven by a tristate gate but an AND gate.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_16

Associate a bus keeper to each tri-state

When to Use

Use this rule to ignore X condition in the design.

Description

The *Tristate_16* rule reports violation for tristates that are not associated to a bus keeper.

Default weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

- *keeper* (optional): Specifies buskeeper design units or nets connected to a buskeeper (virtual) design unit instance so that various rules can take appropriate action
- *tristate_cell* (optional): Defines the tristate cells.

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING] Tri-stated net <net-name> has no keeper associated with it

Arguments

Tristate net name, <net-name>

Potential Issues

The violation message appears when a tristated net is not associated with any keeper.

Consequences of Not Fixing

If you do not fix this violation, the design may fall into X condition.

Buskeepers are used to ensure the output value of concerned tri-states and avoid X state generation. Rational prevent test coverage dropping (due to potential generation and propagation of 'X'), as well as Logical BIST integration and IDDQ problems.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the violating tristate net.

To fix the violation, associate a keeper with bus.

Message 2

[INFO] Tri state_16 runs only on pure netlist design, rule checking skipped for rtl design <design-name>

Potential Issues

The violation message appears when rule checking is skipped because the design is a non-netlist design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

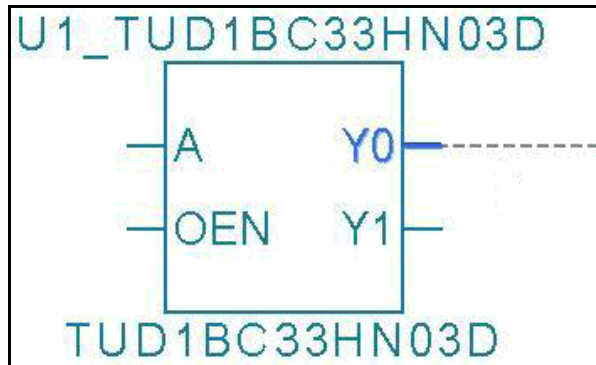
How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Consider the following figure:

Tristate Rules



The Tristate_16 rule reports violation for the above example because bus, Y0 is not attached with a keeper.

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_17

Reports bidirectional ports that do not have fixed direction in DBIST mode

When to Use

Use this rule check if bidirectional ports have a fixed direction in the DBIST mode.

Description

The Tristate_17 rule reports violation for bidirectional ports that do not have a fixed path in DBIST mode.

Default weight

1

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

dbist (mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit into a state for DBIST mode.

Operating Mode

Dbist

Messages and Suggested Fix

Message 1

[WARNING] The port '<port-name>' is used neither as input nor as output in dbist mode

Arguments

Top level bidirectional port name, <port-name>

Potential Issues

The violation message appears when a port is used neither as input nor output.

Consequences of Not Fixing

Not fixing the violation may result in problems in ATPG testing and control over design.

bidirectional pads are kept in a fixed direction to remove any implied risk of 'X' generation. Rational prevent Logical BIST integration problem with Synopsys' Tetramax.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the tristate buffer and its output net.

To fix the violation, fix the direction of port as input or output only.

Message 2

[WARNING] The port '<port-name>' is used both as input and output in dbist mode

Arguments

Top level bidirectional port name, <port-name>

Potential Issues

The violation message appears when a port is used both as input and output.

Consequences of Not Fixing

If you do not fix this violation, it may result in problems in ATPG testing and control over design.

bidirectional pads are kept in a fixed direction to remove any implied risk of 'X' generation. Rational prevent Logical BIST integration problem with Synopsys Tetramax.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the tristate buffer and its output net.

To fix the violation, fix the direction of ports in one mode only.

Message 3

[INFO] DBIST Mode (specified using 'dbist' sgdc command) has not been specified for design '<du-name>', hence skipping rule checking

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message is reported when rule is run without the dbist constraint. Ignore this message in case no dbist constraint is required for design.

Consequences of not fixing

Rule checking is skipped in absence of the dbist constraint.

How to Debug and Fix

To fix the violation, add necessary dbist constraints, if present.

Message 4

[INFO] Tristate_17 runs only on pure netlist design, rule checking skipped for rtl design <design-name>

Potential Issues

The violation message appears when rule checking is skipped because the design is a non-netlist design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Currently Unavailable

Tristate Rules

Default Severity Label

Warning

Rule Group

Tristate

Reports and Related Files

No related reports or files.

Tristate_18

Reports pins of bus keeper that are not fitted with a controllable pullup

When to Use

Use this rule in IDDQ testing to get a control over keeper.

Description

The Tristate_18 rule reports pins of bus keeper that are not fitted with a controllable pullup.

Default weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- *test_mode* (optional): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- *clock* (optional): Defines the clocks of a design. Use the constraint's `-testclock` argument for this rule.

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING] Enable pin of keeper <pin-name> does not exist

Arguments

Enable pin of the keeper, <pin-name>

Potential Issues

The violation message appears, if an enable pin of a keeper does not exist.

Consequences of Not Fixing

Not fixing the violation may result in problems in IDDQ testing as scan flip-flop will not be able to control the keeper.

If the buskeeper function is used, the enable pin must only be controllable by a scan flip-flop output. Rational prevent measurement issues in IDDQ and transition faults.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the keeper and the path from keeper's enable pin to black box/latch/flip-flop.

To fix the violation, replace the keeper with a keeper with an enable pin.

Message 2

[WARNING] Enable pin of keeper <pin-name> is not controllable

Arguments

Enable pin of the keeper, <pin-name>

Potential Issues

The violation message appears when an enable pin of keeper is not controllable.

Consequences of Not Fixing

Not fixing the violation may result in problems in IDDQ testing as scan flip-flop will not be able to control the keeper.

If the buskeeper function is used, the enable pin must only be controllable by a scan flip-flop output. Rational prevent measurement issues in IDDQ and transition faults.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the keeper and the path from keeper's enable pin to black box/latch/flip-flop.

To fix the violation, make the enable pin controllable by a scan flip-flop.

Message 3

[INFO] Tristate_18 runs only on pure netlist design, rule checking skipped for rtl design <design-name>

Potential Issues

The violation message appears when rule checking is skipped because the design is a non-netlist design.

Consequences of Not Fixing

This is an informational message. Therefore, there are no consequences of not fixing this violation message.

How to Debug and Fix

This is an informational message. Therefore, no debug or fix is required for this violation message.

Example Code and/or Schematic

Consider the following figure:



The Tristate_18 rule reports violation for the above example because pin, EN, of the keeper, KEEPER_2, is not fitted with a controllable pullup.

Default Severity Label

Warning

Rule Group

Tristate

Tristate Rules

Reports and Related Files

No related reports or files.

Integrity Checks

Overview

The MustRules rule group of the SpyGlass DFT ADV product has the following rules:

Rule	Reports
<i>BlackBoxDetection</i>	This rule has been deprecated.
<i>dftMandatory_Constraint_Check</i>	Checks the adequacy of the user-supplied mandatory constraints for the specified SpyGlass DFT ADV rules
<i>dftMultiplyDrivenPowerRail</i>	Multiply-driven power rails
<i>dftOptional_Constraint_Check</i>	Checks the adequacy of the user-supplied optional constraints for the specified SpyGlass DFT ADV rules
<i>dftSetup</i>	Sets variable values for some SpyGlass DFT ADV rules
<i>dftParamCheck_01</i>	Ensure that parameter <code>dftSetLatchFedByTCIkAsT</code> is not used along with <code>dft_scannable_latches</code> parameter
<i>dftSGDCExistence_00</i>	Ensures that constraints are specified for all top modules
<i>dftSGDCSTX_000</i>	Performs implementation level sanity checks on the SGDC constraints.
<i>dftSGDCSTX_051</i>	Multiple specifications of scalar tags with the same nodes
<i>dftSGDCSTX_053</i>	Invalid merged tag specifications
<i>dftSGDCSTX_054</i>	Multiple specifications of the same merged tag
<i>dftSGDCSTX_055</i>	Nodes that get conflicting values from two or more scalar tags, which have been merged
<i>dftSGDCSTX_057</i>	Mismatched net widths in 'from' and 'to' fields of <code>require_path</code> constraint

Integrity Checks

Rule	Reports
<i>dftSGDCSTX_058</i>	Same port or net declared as both testmode node and sysclock/testclock node
<i>dftSGDCSTX_059</i>	Same port or net declared as both testmode node and sysclock/testclock node
<i>dftSGDCSTX_060</i>	The memory_type constraint is defined for a module which is not a black box
<i>dftSGDCSTX_061</i>	Reports gating_cell constraint having different width for -clkoutTerm, -enTerm and -obsTerm
<i>dftSGDCSTX_062</i>	Checks for the existence of the 'module_pin' constraint in the design
<i>dftSGDCSTX_064</i>	Performs sanity check for macros
<i>dftSGDCSTX_069</i>	Ensure that the user-specified design state and scan_chain constraint are consistent
<i>dftSGDCSTX_070</i>	Ensure that the user-specified set_case_analysis and test_mode – functional are consistent
<i>dftSGDCSTX_071</i>	User-specified clock_shaper, complex_cell, and pll constraints should apply to at least one module or instance (in case of -register)
<i>dftSGDCSTX_072</i>	Performs a sanity check for the require_structure constraint
<i>dftSGDCSTX_073</i>	Performs sanity checks for the module_bypass constraint
<i>dftSGDCSTX_074</i>	Performs sanity checks for -constraint_message_tag field of the require_constraint_message_tag and illegal_constraint_message_tag constraints
<i>dftSGDCSTX_075</i>	Performs sanity check for the clock_shaper constraint

Rule	Reports
<i>dftSGDCSTX_076</i>	Performs sanity check for the missing required fields of the require_path, illegal_path, require_strict_path, require_value, illegal_value, require_frequency, require_constraint_message_tag and illegal_constraint_message_tag constraints
<i>dftSGDCSTX_077</i>	Performs sanity check for usages of instance based macros in require_path, illegal_path, require_strict_path, require_value, illegal_value and require_frequency
<i>dftSGDCSTX_078</i>	Sanity check for usage of macros in require_value and illegal_value constraints when -value_type field is used
<i>dftSGDCSTX_080</i>	Performs sanity checks for -filter_in_cmt and -instance_filter_in_cmt fields of require_value, require_path, illegal_value and illegal_path constraints
<i>dftUserMacroSanityCheck_01</i>	Performs sanity check for usages of user macros in the require_path, illegal_path, require_strict_path, require_value, illegal_value, require_frequency, and require_pulse constraints
<i>dftVSTOPDUSetup</i>	This is a setup rule
<i>Diagnose_ScanChain</i>	Invalid user-defined scan chains under specified simulation conditions
<i>dumpBlackBox</i>	Identifies black boxes in the design and generates scanwrap constraints for them

BlackBoxDetection

This rule has been deprecated.

The functionality of this rule is covered by the `AnalyzeBBox` rule.

dftMandatory_Constraint_Check

Reports inadequacy of the user-specified mandatory constraints

When to Use

Use this rule to check the adequacy of the specified mandatory constraints for DFT rules.

Description

The `dftMandatory_Constraint_Check` rule checks the adequacy of the user-supplied mandatory constraints for the specified SpyGlass DFT ADV rules.

Many SpyGlass DFT ADV rules require mandatory specifications of a combination of SpyGlass Design Constraints. When a constraint is mandatory for a rule, then the rule would not give a message if the said constraint is absent.

The `dftMandatory_Constraint_Check` rule checks whether the SpyGlass Design Constraints file supplied for analysis has all mandatory constraint specifications for the rules selected to run.

The incorrectly specified constraints are flagged by SpyGlass before any SpyGlass product is run. See the *SpyGlass Built-In Rules Reference* for details of rules that check incorrectly specified constraints.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dftUseOffStateOfClockInClockPropagation`: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[INPUTWARNING] One or more constraints missing in module '<du-name>'. Please see summary report in the reported file

Arguments

Design unit name, <du-name>

Potential Issues

The violation message appears when the *dftMandatory_Constraint_Check* rule finds at least one missing mandatory constraint for a design unit.

Consequences of Not Fixing

Not fixing the violation may result in a design error.

How to Debug and Fix

To fix the violation, ensure that you have specified all the mandatory constraints properly.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

InputWarning

Rule Group

Must Rules, Integrity

Reports and Related Files

[dft_mandatory_sgdc](#): This report contains details of missing mandatory constraints and the rules affected by their absence.

dftMultiplyDrivenPowerRail

Reports all the power rails that have external drivers as well

When to Use

Use this rule to avoid inconsistent simulation behavior due to conflict with the internal driver.

Description

The dftMultiplyDrivenPowerRail rule reports violation for power rails driven by external drivers.

Rule Exceptions

The dftMultiplyDrivenPowerRail rule does not flag power rails feeding an inout port, which is acting functionally as an input port.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[WARNING] Power rail '<net-name>' has external driver as well.
Number of External driver is <num1> (<driver-type>)

Arguments

- Name of the power net, <net-name>

Integrity Checks

- Number of drivers, <num>
- Driver type (port or instance), <driver-type>

Potential Issues

This violation message appears if a power rail has external drivers in the design.

Consequences of Not Fixing

Power rails cause inconsistent simulation behavior due to conflict with the internal driver.

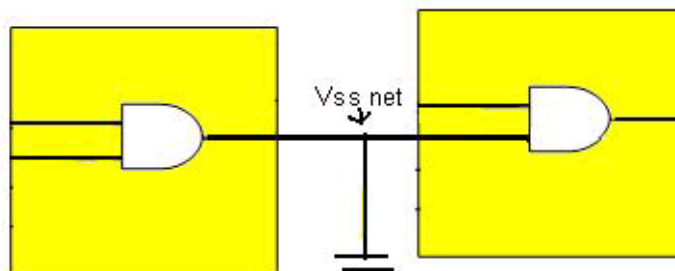
How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the instances/ports which are drivers of the power rail.

To fix the violation, it is recommended to change the RTL for such nets.

Example Code and/or Schematic

Consider the following figure:



The `dftMultiplyDrivenPowerRail` rule reports a violation for the above example because `Vss net` is driven by multiple power rails.

Default Severity Label

Warning

Rule Group

Must Rules

Reports and Related Files

No related reports or files.

dftOptional_Constraint_Check

Reports whether or not the optional constraint requirements of SpyGlass DFT ADV rules are met

When to Use

Use this rule to check the adequacy of the user-specified optional constraints.

Description

The `dftOptional_Constraint_Check` rule checks the adequacy of the user-supplied optional constraints for the specified SpyGlass DFT ADV rules.

Many SpyGlass DFT ADV rules have associated optional SpyGlass Design Constraints. If a constraint is optional for a rule, then its presence or absence affects the rule behavior.

The `dftOptional_Constraint_Check` rule checks whether the SpyGlass Design Constraints file supplied for analysis has all optional constraint specifications for the rules selected to run.

The incorrectly specified constraints are flagged by SpyGlass before any SpyGlass product is run. See the *SpyGlass Built-In Rules Reference* for details of rules that check incorrectly specified constraints.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- `dftUseOffStateOfClockInClockPropagation`: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[INFO] One or more constraints missing in module '`<du-name>`'. Please see summary report in the reported file

Potential Issues

The violation message appears, if one or more optional constraints are missing for a design

Consequences of Not Fixing

Not fixing the violation may generate noise in the design.

How to Debug and Fix

To fix the violation, ensure that you specify all optional constraints correctly

Example Code and/or Schematic

Not Applicable

Default Severity Label

Info

Rule Group

Must Rules, Integrity

Reports and Related Files

[dft_optional_sgdc](#): This report details the missing optional constraints and the rules affected by their absence.

dftSetup

Performs some pre-DFT run initial setup.

When to Use

This rule runs by default for initial pre-DFT setup.

Description

The dftSetup rule sets variable values for some SpyGlass DFT ADV rules.

The dftSetup rule sets some variable values that SpyGlass DFT ADV requires during its run and prepares all required data structures that are required.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

No violation message is generated by this rule.

Example Code and/or Schematic

Not Applicable

Default Severity Label

InputWarning

Rule Group

Must Rules, Integrity

Reports and Related Files

No related reports or files.

dftParamCheck_01

Ensure that the `dftSetLatchFedByTCIkAsT` parameter is not used along with the `dft_scannable_latches` parameter

When to Use

Use this rule to perform a sanity check to ensure that the `dftSetLatchFedByTCIkAsT` and `dft_scannable_latches` are not used together

Description

The `dftParamCheck_01` rule performs a sanity check to ensure that the `dftSetLatchFedByTCIkAsT` parameter is not used along with the `dft_scannable_latches` parameter because scannability takes precedence over transparency.

Parameter(s)

None

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

Parameter '`dftSetLatchFedByTCIkAsT`' is not effective when parameter '`dft_scannable_latches`' is set to 'on' because scannability takes precedence over transparency. For '`dftSetLatchFedByTCIkAsT`' to work set '`dft_scannable_latches`' to 'off'

Potential Issues

A violation message is reported when the `dftSetLatchFedByTCIkAsT` parameter is used along with the `dft_scannable_latches` parameter.

Consequences of Not Fixing

Not fixing the violation makes the `dftSetLatchFedByTCIkAsT` parameter ineffective and the design is not check for scannability.

How to Debug and Fix

To fix the violation, set the value of the `dft_scannable_latches` parameter to off.

Example Code and/or Schematic

Not Applicable

Default Severity Label

InputWarning

Rule Group

Must Rules, Integrity

Reports and Related Files

No related reports or files.

dftSGDCExistence_00

Ensures that constraints are specified for all top modules

When to Use

Use this rule to check if constraints are specified for each top module.

Description

The dftSGDCExistence_00 rule checks if all the top modules have a `current_design` specification. This ensures that constraints are specified for all top modules.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

Message 1

[INFO] No SGDC constraints specified for top block '`<top-block-name>`'

Arguments

Name of the top block. `<top-block-name>`

Potential Issues

The violation message appears if no SGDC constraint is specified for the reported top block.

Consequences of Not Fixing

The absence of the SGDC file may lead to false violations and/or suppression of real violations.

How to Debug and Fix

To debug the violation, refer to the SGDC file.

To fix the violation, specify a constraint for the reported top block in the SGDC file and rerun SpyGlass.

Message 2

[INFO] Either SGDC file is missing or none of the top blocks has associated SGDC constraint

Potential Issues

The violation message appears because of one of the following reasons:

- The SGDC file is not specified.
- The specified SGDC file does not exist.
- No SGDC constraint is specified for any of the top blocks in the design.

Consequences of Not Fixing

The absence of the SGDC file may lead to false violations and/or suppression of real violations.

How to Debug and Fix

To debug the violation, perform the following steps:

1. Check if the specified SGDC file is valid.
2. Check if an SGDC constraint is specified for any of the top blocks in the design.

To fix the violation, specify a valid SGDC file containing a constraint specification for a top block in the design. Then run SpyGlass again.

Example Code and/or Schematic

Not Applicable

Integrity Checks

Default Severity Label

Info

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dftSGDCSTX_000

Performs implementation level sanity checks on the SGDC constraints.

When to Use

Use this rule to perform implementation level sanity checks on the SGDC constraints.

Description

This rule performs implementation level sanity checks on the SGDC constraints, thereby, validating the specified SGDC inputs.

Parameter(s)

None

Constraint(s)

test_mode (mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Messages and Suggested Fix

The dftSGDCSTX_000 rule reports the following violation message:

```
[FATAL] Invalid specification provided for '-value' of
test_mode constraint for design '<design-name>'. Only '0' or
'1' can be specified along with -power_ground option"
```

Potential Issues

The violation message is reported when you assign an invalid value to the test_mode constraint.

Consequences of Not Fixing

If you do not fix the violation, the SpyGlass run would abort.

How to Debug and Fix

Specify a valid value to the -value argument of the test_mode constraint.

Default Severity Label

Fatal

Integrity Checks

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dftSGDCSTX_051

Ensure that there is no duplicate declaration of a node for the same scalar tag.

When to Use

This rule runs sanity check on duplication of a node. Run this rule immediately after SGDC command file parsing.

Description

The dftSGDCSTX_051 rule reports violation for multiple specifications of scalar tags with same nodes.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

[INFO] Node <node-name> has been redefined for tag <tag-name>.

Arguments

- Name of the node. <node-name>
- Name of the tag. <tag-name>

- SGDC file name and line number of place where initial declaration has occurred. <file-name>, <num>

Potential Issues

The violation message appears, if a node is redefined for a tag in the design.

Consequences of Not Fixing

This is an informational rule. Therefore, there are no direct consequences of not fixing the violation.

How to Debug and Fix

No debug is required for this message.

To fix the violation, ensure that each tag is defined only once in the design.

Example Code and/or Schematic

Consider the following example:

```
define_tag -tag t1 -name n1 -value 1
define_tag -tag t1 -name n2 -value 1
define_tag -tag t1 -name n1 -value 1
define_tag -tag t2 -name n1 -value 1
```

Here, the node `n1` has been specified with the same scalar tag `t1` in two places.

Default Severity Label

Info

Rule Group

SoC, Integrity

Reports and Related Files

No related reports or files.

dftSGDCSTX_053

Reports incorrectly merged tags

When to Use

Use this rule for sanity check on the SGDC command for `define_tag` constraint.

Description

The dftSGDCSTX_053 rule reports violation for invalid merged tag specifications.

The dftSGDCSTX_053 rule reports the following cases:

- Both `-name` and `-merge` arguments specified in a *define_tag* constraint.
- If one or more of the constituent scalar tags are invalid (possibly marked so by some previously run dftSGDCSTX rule).
- If one or more specified constituent tags do not exist (that is, has no `define_tag` constraint specification of its own).

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

define_tag (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

None

Messages and Suggested Fix

Message 1

[INFO] Merged tag '<mtag-name>' cannot have -name field

Arguments

Name of the merged tag, <mtag-name>

Potential Issues

The violation message appears, if a merged tag is found to have a name argument.

Consequences of Not Fixing

If you do not fix the violation, the constraints are not applied for rule checking and are ignored.

How to Debug and Fix

No debug is required for this violation message.

To fix the violation, ensure to specify the name argument of the merged tag.

Message 2

[INFO] Illegal tag '<tag-name>' used in merged tag <mtag-name>

Arguments

- Name of the merged tag, <mtag-name>
- Name of the illegal tag, <tag-name>

Potential Issues

This message appears, if your design has an illegal tag name for the merged tag argument.

Consequences of Not Fixing

If you do not fix the violation, the constraints are not applied for rule checking and are ignored.

How to Debug and Fix

No debug is required for this violation message.

To fix the violation, ensure to specify the design with an illegal tag name for the merged tag argument.

Example Code and/or Schematic

Consider the following example:

```
define_tag -tag t12 -merge t1 t2 -name n2 -value 1
```

The dftSGDCSTX_053 rule reports a violation for the above example because `-name` and `-merge` arguments are specified in a *define_tag* constraint.

Default Severity Label

Info

Rule Group

Integrity, SoC

Reports and Related Files

No related reports or files.

dftSGDCSTX_054

Ensure that the merge tag specification do not span multiple lines.

When to Use

This rule runs sanity check on the merge tags. Run this rule immediately after SGDC command file parsing.

Description

The dftSGDCSTX_054 rule reports violation for multiple specifications of the same merged tag.

A merged tag should be defined just once and in one line. Hence, if there are two or more *define_tag* constraints with `-merge` argument, each on different lines, and with the same tag name, then the dftSGDCSTX_054 rule flags a message.

However, you can span very long merged tag specifications to multiple lines by adding a `'\'` at the end every time you want to move onto a new line.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

define_tag (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

None

Messages and Suggested Fix

[INFO] Merged tag <mtag-name> redefined.

Where, <mtag-name> refers to the name of the merged tag.

Potential Issues

The violation message appears, if the design has a merged tag name which is redefined.

Consequences of Not Fixing

If you do not fix the violation message, the [define_tag](#) constraint is ignored.

How to Debug and Fix

No debug is required for this violation message.

To fix the violation, ensure that a merged tag has unique specification only.

Example Code and/or Schematic

Consider the following example:

```
define_tag -tag t12 -merge t1 t2
define_tag -tag t12 -merge t3
```

The dftSGDCSTX_054 rule reports a violation for the above example because multiple specifications are assigned for the same tag, t12.

Default Severity Label

Info

Rule Group

Integrity, SoC

Reports and Related Files

No related reports or files.

dftSGDCSTX_055

Ensure that there are no conflicting values for the same node when used in a merged tag

When to Use

Use this rule to run sanity check on merged tags. Run this rule immediately after SGDC command file parsing.

Description

The dftSGDCSTX_055 rule reports violation for nodes that get conflicting values from two or more scalar tags, which have been merged.

The dftSGDCSTX_055 rule checks for conflicting usage of any name-value pair while merging tags.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[define_tag](#) (optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.

Operating Mode

None

Messages and Suggested Fix

[INFO] Requirement for node <name> is conflicting in merged tag <mtag-name>

Arguments

- Name of the node, <name>
- Name of the merged tag, <mtag-name>

Potential Issues

The violation message appears when the design has a merged tag name with conflicting values.

Consequences of Not Fixing

If you do not fix this message `define_tag` constraint is ignored.

How to Debug and Fix

No debug is required for this violation message.

To fix the violation, ensure that the conflict does not occur during defining tags.

Example Code and/or Schematic

Consider the following example:

```
define_tag -tag t1 -name n1 -value 1
define_tag -tag t2 -name n1 -value 0
define_tag -tag t12 -merge t1 t2
```

The `dftSGDCSTX_055` rule reports violation for the above example because there is a conflicting usage of name-value pair while merging tags.

Default Severity Label

Info

Rule Group

SoC, Integrity

Reports and Related Files

No related reports or files.-

dftSGDCSTX_057

Reports mismatches in node widths.

When to Use

Use this rule to run sanity check on nodes. This rule runs immediately after design read (HDL parsing) and elaboration.

Description

The dftSGDCSTX_057 rule reports violation for *require_path* constraints specified with `-parallel` argument where the width of nodes specified in the `-from` argument and the `-to` argument do not match.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagatation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

require_path (optional): Use this constraint to define a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument.

Operating Mode

None

Messages and Suggested Fix

[INFO] Mis-matched net widths in 'from' and 'to' field for specified 'require_path' constraint

Potential Issues

The violation message appears when a mismatch is found in `-from` and `-to` arguments of the `require_path` constraint.

Consequences of Not Fixing

If you do not fix the violation message, the [require_path](#) constraint is ignored.

How to Debug and Fix

No debug is required for this violation message

To fix the violation, correct the `require_path` constraint specifications.

Example Code and/or Schematic

Consider the following example:

```
require_path -tag s101 -from top.SEF[2:0]  
-to top.FGT[3:0] - parallel
```

In the above example, there is a mismatch between `-from` and `-to` arguments with `-parallel` argument specified.

Default Severity Label

Info

Rule Group

SoC, Integrity

Reports and Related Files

No related reports or files.

dftSGDCSTX_058

Ensure that the testmode and testclock are separate ports.

When to Use

Use this rule to run sanity checks on ports and functions before the analysis phase. This rule checks immediately after design read (HDL parsing) and elaboration.

Description

The dftSGDCSTX_058 rule reports violation for ports or nets declared both as a testmode node and a sysclock/testclock node when the testmode sequence does not end with "X".

NOTE: While the dftSGDCSTX_058 rule and the [dftSGDCSTX_059](#) rule perform the same function, the dftSGDCSTX_058 rule works before the analysis phase and the [dftSGDCSTX_059](#) rule works after flattening.

Default Weight

10

Language

VHDL, Verilog

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

None

Messages and Suggested Fix

[FATAL] Testmode '`<name>`' and clock '`<name>`' are conflicting

Where,

`<name>` refers to the name of the port/net.

Potential Issues

The violation message appears when the design shows a conflict between the testmode clock and the testclock.

Consequences of Not Fixing

Not fixing the violation may block the clock line for all further analysis due to 0 or 1. This may further result in a fatal design error.

How to Debug and Fix

No debug is required for this violation message.

To fix the violation message, put X at the end of `test_mode` constraint so that the signal is free to be pulsed as a testclock.

Example Code and/or Schematic

Consider the following example, where both `test_mode` and `-testclock` are same ports:

```
clock -name abc
test_mode -name abc -value 1010101010
```

The `dftSGDCSTX_058` rule reports violation for the above example. To fix this error, put X in the last value of `test_mode` constraint:

```
test_mode -name abc -value 1010101010x
```

Default Severity Label

Fatal

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dftSGDCSTX_059

Ensure that the testmode and testclock are separate ports.

When to Use

Use this rule to run sanity check on ports. This rule is run immediately after design read (HDL parsing) and elaboration.

Description

The dftSGDCSTX_059 rule reports ports or nets declared both as a testmode node and a sysclock/testclock node.

NOTE: While the [dftSGDCSTX_058](#) rule and the [dftSGDCSTX_059](#) rule perform the same function, the [dftSGDCSTX_058](#) rule works before the analysis phase and the [dftSGDCSTX_059](#) rule works after flattening.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

[test_mode](#) (mandatory): Specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

Operating Mode

None

Messages and Suggested Fix

[FATAL] Testmode '<name>' and clock '<name>' are conflicting
Where,

<*name*> is the name of the port/net.

Potential Issues

The violation message appears when the design shows a conflict between the testmode clock and the testclock.

Consequences of Not Fixing

Not fixing the violation may block the clock line for all further analysis due to 0 or 1. This may further result in a fatal design error.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, put X at the end of *test_mode* constraint so that the signal is free to be pulsed as a testclock.

Example Code and/or Schematic

Consider the following example, where both *test_mode* and *-testclock* are same ports:

```
clock -name abc
test_mode -name abc -value 1010101010
```

The *dftSGDCSTX_058* rule reports violation for the above example. To fix this error, put X in the last value of *test_mode* constraint:

```
test_mode -name abc -value 1010101010x
```

Default Severity Label

Fatal

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dftSGDCSTX_060

Reports memory_type constraint defined for a module which is not a black box

When to Use

Use this rule to perform sanity check on the memory_type constraint. This rule checks immediately after design read (HDL parsing) and elaboration

Description

The dftSGDCSTX_060 rule reports violation, if *memory_type* constraint is not applied on a black box and thus is ignored from further analysis.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

memory_type (mandatory): Specifies the memory design unit (black box) names.

Operating Mode

None

Messages and Suggested Fix

[WARNING] 'memory_type' constraint '<mod-name>' is defined on a module which is not a black box and hence ignored for rest of dft analysis

Where,

<*mod-name*> refers to the module name on which the `memory_type` constraint is applied.

Potential Issues

The violation message appears when the design has an argument specified on a module which is not a black box.

Consequences of Not Fixing

If you do not fix this violation, the `memory_type` constraint is ignored.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, either remove `memory_type` constraint or apply it on a black box.

Example Code and/or Schematic

Consider the following example where the rule reports a violation because `my_memory` is not a black box:

```
memory_type -name <my_memory>
```

To fix this violation, apply the `memory_type` constraint on a black box.

Default Severity Label

Warning

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dftSGDCSTX_061

Reports `gating_cell` constraint having different width for `-clkoutTerm`, `-enTerm` and `-obsTerm`

When to Use

Use this rule to run sanity check on the `gating_cell` constraint.

Description

The `dftSGDCSTX_061` rule reports a violation in the following cases:

- If the `-clkoutTerm`, `-enTerm`, and `-obsTerm` arguments of the `gating_cell` constraint do not have equal width.
- If the `-clkoutTerm` and `-enTerm` arguments of the `gating_cell` constraint are not specified.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

`gating_cell` (Mandatory): Use to specify the user-defined clock gating cell.

Operating Mode

None

Messages and Suggested Fix

Message 1

[FATAL] `gating_cell` constraint '`<name>`' has different width

for clkout pin (<clk_pin_name: <clk_pin_width>) and <pin-type> (<pin_name: <pin_width>)

Arguments

- Name of the clock gating cell, <name>
- Name of the clkout pin, <clk_pin_name>
- Width of the clkout pin, <clk_width>
- Enable pin or Obs pin, <pin_type>
- Name of the Enable or Obs pin, <pin_name>
- Width of the Enable or Obs pin, <pin_width>

Potential Issues

The violation message appears, if *gating_cell* constraint has different width for clkout pin and enable pin or Obs pin.

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, correct the *gating_cell* constraint.

Message 2

[FATAL] 'gating_cell' constraint 'name' has missing 'missing_pin_details'

Arguments

- Name of the clock gating cell, <name>
- Details of the missing pin, <missing_pin_details>

Potential Issues

The violation message appears if the *-clkoutTerm* and *-enTerm* arguments of the *gating_cell* constraint are not specified.

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, specify the missing arguments in the `gating_cell` constraint.

Example Code and/or Schematic

Consider the following example:

```
gating_cell -name<du-name>  
-clkoutTerm clkout[3:0]  
-enTerm en[4:0]
```

The `dftSGDCSTX_061` rule reports a violation for the above example because `clockoutTerm` and `enTerm` have different widths.

Default Severity Label

Fatal

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dftSGDCSTX_062

Checks for the existence of the 'module_pin' constraint in the design

When to Use

Use this rule to check for the existence of the module_pin constraint in the design.

Description

The dftSGDCSTX_062 rule reports a violation, if the module specified using the -name field does not exist in the design.

Parameter(s)

- *Common SpyGlass DFT ADV Rule Parameters*
- *dftUseOffStateOfClockInClockPropagation*: The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraint(s)

module_pin (mandatory): Use this constraint to specify the set of pins for which hierarchy of all instantiations is to be generated.

Operating Mode

None

Messages and Suggested Fix

[WARNING] '`<mod_name>`' [SubModule] is never instantiated within environment '`<design_top_name>`'

Arguments

- Name of the module, `<mod_name>`
- Name of the top design unit, `<design_top_name>`

Potential Issues

The violation message appears, if the module specified using -name field of the *module_pin* constraint does not exist in the design.

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, correct the module_pin constraint of the design.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

SoC, Integrity

Reports and Related Files

No related reports or files.

dftSGDCSTX_064

Performs sanity checks for macros

When to Use

Use this rule to perform sanity check for macros.

Description

This rule checks for validity of the logical names supplied using `sgdc` constraints.

The `dftSGDCSTX_064` rule performs the sanity check on a macro in a constraint only when the rule using that constraint is enabled.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *require_path* (mandatory): Use this constraint to define a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument.
- *require_strict_path* (mandatory): Use this constraint to define a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument.
- *require_pulse* (mandatory): Use this constraint to define a check that requires a pulse sequence to be established on a certain node, when the circuit is simulated using the condition specified by the `-tag` argument.
- *expect_frequency* (mandatory): Use this constraint to specify a certain frequency value is expected at a node in the design.
- *require_value*: Use this constraint to define a check that requires a logic value to be established on a certain node when the circuit has been simulated using the condition specified by the `-tag` argument.

Operating Mode

All

Messages and Suggested Fix

Message 1

[ERROR] No white-box instance or module named <mod_name> found in design <du_name>

Arguments

- Name of module or hierarchical path of an instance which is defined as scope for a macro, <mod-Name>
- Name of the top design unit, <du_name>

Potential Issues

The violation message appears when the specified white box instance or module name is not found in the design. This may happen, if you have specified a wrong module name.

Consequences of Not Fixing

If you do not fix this violation message, the macro will not expand into respective design nodes and, therefore, the constraint will not work as intended.

How to Debug and Fix

To fix the violation, check for any typos while specifying the module name and fix them.

Message 2

[ERROR] Global scope defined for macro <macro_name> for except field in design <du_name>

Arguments

- Macro in constraint, <macro_name>
- Name of the top design unit, <du_name>

Potential Issues

The violation message appears when you have specified a wrong module name.

Consequences of Not Fixing

If you do not fix the violation message, the dftSGDCSTX_064 rule excludes all the design nodes described by the macro or the logical name.

How to Debug and Fix

To fix the violation message, check for missing scope or remove the macro from both include and exclude fields.

Message 3

[ERROR] Macro <macro_name> defined only in except field for design <du_name>

Arguments

- Macro in constraint, <macro_name>
- Name of the top design unit, <du_name>

Potential Issues

The violation message appears, if you have missed the macro in the include field of the constraint.

Consequences of Not Fixing

If you do not fix this violation message, the macro will not expand into respective design nodes and, therefore, the constraint will not work as intended.

How to Debug and Fix

To fix the violation, inspect and add any missing macro from the include field or remove the macro from the exclude fields also.

Message 4

The dftSGDCSTX_064 rule reports following violation message for an unrecognized macro:

[ERROR] Macro <macro_name> is not recognized

Arguments

Macro in constraint, <macro_name>

Potential Issues

The violation message appears when the specified white box instance or module name is not found in the design. This may happen, if you have specified a wrong module name.

Consequences of Not Fixing

If you do not fix this violation message, the macro will not expand into respective design nodes and, therefore, the constraint will not work as intended.

How to Debug and Fix

To fix the violation, check for any typos while specifying the module name and fix them.

Message 5

The dftSGDCSTX_064 rule reports following violation message for an empty field in the design:

```
[ERROR] <field_name> field is effectively empty for design  
<du_name>
```

Arguments

- Constraint fields (-from, -to, or -name), <field name>
- Name of the top design unit, <du_name>

Potential Issues

A violation message appears, if you have missed scope or some macros.

Consequences of Not Fixing

If you do not fix this violation message, the macro will not expand into respective design nodes and, therefore, the constraint will not work as intended.

How to Debug and Fix

To fix the violation message, review the specified constraints.

Message 6

```
[WARNING] Include and exclude fields for macro <macro-name> are  
disjoint for field <field> in design <du_name>
```

Arguments

- Macro in constraint, <macro_name>
- Name of the top design unit, <du_name>

Potential issues

The violation message appears, when the specified white box instance or module name is not found in the design. The rule may also flag this violation message, if you have missed scope or some macros.

Consequences of Not fixing

If you do not fix this violation message, the except field for the specified constraints will be ineffective.

How to debug and Fix

To fix the violation message, perform one or more of the following tasks:

- Check for missing scope
- Remove the macro from both include and exclude fields
- Review the specified constraints.

Example Code and/or Schematic

Not Available

Default Severity Label

Error

Rule Group

SoC, Integrity

Reports and Related Files

No related reports or files.

dftSGDCSTX_069

Ensure that the user-specified design state and scan_chain constraint are consistent

When to Use

Use this rule to check whether the user-specified design state and the [scan_chain](#) constraint are consistent.

Description

The dftSGDCSTX_069 rule reports a violation for a design unit whose design state is defined as post_scan_stitched but no [scan_chain](#) constraint is provided.

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

[scan_chain](#) (mandatory): Use this constraint to specify the scan chains for the *dftSGDCSTX_069* rule.

Operating Mode

None

Messages and Suggested Fix

[WARNING] For design '<du-name>', no 'scan_chain' constraint is supplied even though specified design state is 'post_scan_stitched'. This forces all the flip-flops to be treated as inferred no_scan. Please provide scan chain information or switch back the design state to 'pre_scan_stitched'

Arguments

Name of the design unit, <du-name>

Potential Issues

The violation message appears when a design state for a design unit is specified as post_scan_stitched and no scan chain is specified.

Consequences of Not Fixing

If you do not fix the violation, all the flip-flops in the design are inferred as no scan.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, either change the design state to `pre_scan_stitched` or specify the `scan_chain` constraint for the design unit.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Error

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files.

dftSGDCSTX_070

Ensure that the user-specified `set_case_analysis` and `test_mode -functional` are consistent

When to Use

Use this rule to check whether the user-specified `set_case_analysis` and `test_mode -functional` are consistent.

Description

The dftSGDCSTX_070 rule reports a violation, if you have specified both the `set_case_analysis` and the `test_mode -functional` constraint on a same node but with conflicting values.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- `set_case_analysis` (mandatory): Use this constraint to specify functional mode simulation value on a node.
- `test_mode` (mandatory): Use this constraint to specify functional mode simulation value on a node.

Operating Mode

None

Messages and Suggested Fix

```
[WARNING] Conflicting functional mode simulation value
specified on node '<node name>' set_case_analysis:
<set_case_analysis value>, test_mode -functional: <test_mode
functional value>
```

Arguments

- Node on which the constraint is specified, `<node name>`
- Value specified for the `set_case_analysis` constraint, `<set_case_analysis value>`

- Value specified for the `test_mode` -functional constraint, `<test_mode functional value>`

Potential Issues

The violation message appears when both the `set_case_analysis` and the `test_mode` -functional constraint are specified on the same node with the conflicting values.

Consequences of Not Fixing

If you do not fix this violation, value specified for the `test_mode` - functional constraint will be used for the subsequent analysis.

How to Debug and Fix

No debug is required for this violation message.

To fix this violation, use either the `test_mode` -functional or `set_case_analysis` constraint.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files.

dftSGDCSTX_071

User-specified 'clock_shaper' 'complex_cell' and 'pll' should apply to at least an instance, in case of 'clock_shaper -register, or to a module.

When to Use

Use this rule to ensure that the specified clock_shaper, complex_cell, or pll constraint is applied to at least an instance (if the -register argument is used) or to a module.

Description

This rule reports a violation, if the user-specified clock_shaper, complex_cell, and pll constraint are not applied to at least an instance, in case of the -register argument, or to a module

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

Message 1

Constraint '<constraint_name> -name <module_name>' is skipped for design '<current_du>' because design does not contain a module by name '<module_name>'

Arguments

- Name of the constraint, that is, clock_shaper, complex_cell, or pll, <constraint_name>
- Name of the module used in the sgdc specification, <module_name>
- Name of current design unit, <current_du>

Potential Issues

The violation message is reported when an incorrect module name is specified in the sgdc file.

Consequences of Not Fixing

The specified constraint is ignored for analysis, if you do not fix the violation.

How to Debug and Fix

To fix the violation, check the name of the module and correct it in the sgdc file.

Message 2

Constraint '<constraint_name> -name <module_name> -register' is skipped for design '<current_du>' because design does not contain a flip-flop by name '<register_name>'

Arguments

- Name of the constraint, that is, clock_shaper, complex_cell, or pll, <constraint_name>
- Name of the instance used in the sgdc specification, <module_name>
- Name of current design unit, <current_du>

Potential Issues

The violation message is reported when an incorrect instance name is specified in the sgdc file.

Consequences of Not Fixing

The specified constraint is ignored for analysis, if you do not fix the violation.

How to Debug and Fix

To fix the violation, check the name of the registers and correct them in the sgdc file.

Message 3

Constraint '<constraint_name> -name <module_name>' is skipped for design '<current_du>' because the specified module '<module_name>' is either present inside another clock_shaper or is present outside the design '<current_du>'

Arguments

- Name of the constraint, that is, clock_shaper, complex_cell, or pll, <constraint_name>
- Name of the instance used in the sgdc specification, <module_name>
- Name of current design unit, <current_du>

Potential Issues

The violation message is reported if the user-specified clock_shaper is ignored because even though it is present in the design but is either outside the current_top DU hierarchy or is inside another user-specified clock_shaper.

Consequences of Not Fixing

The specified constraint is ignored for analysis, if you do not fix the violation.

How to Debug and Fix

To fix the violation, check the name of the registers and correct them in the sgdc file.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files.

dftSGDCSTX_072

Performs sanity check for the `require_structure` constraint

When to Use

Use this rule to perform a sanity check for the `require_structure` constraint

Description

The `dftSGDCSTX_072` rule reports a violation when the `-from_one_of` argument is used along with the `-type SIMULATION` argument in the `require_structure` constraint.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

The `dftSGDCSTX_072` rule reports the following violation message:

```
[FATAL] '-from_one_of' is not supported with '-type SIMULATION'
for constraint 'require_structure'
```

Potential Issues

A violation message is reported when the `-from_one_of` field is used with the `-type SIMULATION` field of the `require_structure` constraint.

Consequences of Not Fixing

The specified constraint is ignored for analysis, if you do not fix the violation.

How to Debug and Fix

The `require_structure` constraint does not support simulation based verification with the `-from_one_of` argument. To fix this violation, set the

Integrity Checks

value of the -type argument of the The *require_structure* constraint to STRUCTURAL.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files.

dftSGDCSTX_073

Performs sanity checks for `module_bypass`

When to Use

Use this rule to perform sanity check for the `module_bypass` constraint

Description

The dftSGDCSTX_073 rule checks for validity of input output term pair and bypass condition supplied using the `module_bypass` constraint.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

`module_bypass` (mandatory)

Operating Mode

None

Messages and Suggested Fix

Message 1

[WARNING]The -iport '<iport_name>' and -oport '<oport_name>' specified in module_bypass constraint have already been specified in other module_bypass constraint [on line number '<line_no>' in file '<file_name>'] but with different invert polarity. Same pair of iport and oport on same design unit can not be specified with different invert polarity. Software will skip this constraint for this run.

Arguments

- Name of the input port specified in constraint, `<iport_name>`
- Name of the output port specified in constraint, `<oport_name>`
- Line number where other conflicting constraint is specified, `<line_no>`

- Name of the file where other conflicting constraint is specified, *<file_name>*

Potential Issues

A violation message is reported when an incorrect iport and oport pair is specified in the sgdc file.

Consequences of Not Fixing

The specified constraint is ignored for analysis, if you do not fix the violation.

How to Debug and Fix

To fix the violation, correct the iport and oport pair in the sgdc file.

Message 2

```
[WARNING]The bypass condition in module_bypass constraint does not match with other module_bypass constraint [on line number '<line_no>' in file '<file_name>']. Different bypass condition can not be specified for same design unit. Software will skip this constraint for this run.
```

Arguments

- Line no where other conflicting constraint is specified, *<line_no>*
- Name of the file where other conflicting constraint is specified, *<file_name>*

Potential Issues

A violation message is reported when an incorrect bypass condition is specified in the sgdc file.

Consequences of Not Fixing

The specified constraint is ignored for analysis, if you do not fix the violation.

How to Debug and Fix

To fix the violation, correct the bypass condition in the sgdc file.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Warning

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files.

dftSGDCSTX_074

Performs sanity checks for `-constraint_message_tag` field of the `require_constraint_message_tag` and `illegal_constraint_message_tag` constraints

When to Use

Use this rule to perform sanity checks on the `constraint_message_tag_expression` field.

Description

The `dftSGDCSTX_074` rule reports a violation, if the `-constraint_message_tag` field of the `require_constraint_message_tag` and/or `illegal_constraint_message_tag` constraint:

- points to a tag which is not present in one of the following constraints:
 - `require_path`
 - `require_value`
 - `illegal_value`
 - `illegal_path`
- Contains the tag-modifier other than `:PASS` or `:FAIL`
- The expression contains operator other than `'&&'` (and) and `'||'` (or)

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- `require_path` (Optional): Use this constraint to define a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument.
- `require_value` (Optional): Use this constraint to define a check that requires a logic value to be established on a certain node when the circuit has been simulated using the condition specified by the `-tag` argument.

- *illegal_path* (Optional): Use this constraint to define a connectivity check for an illegal path from a pin specified with the -from argument to a pin specified with the -to argument.
- *illegal_value* (Optional): Use this constraint to define a check that requires a logic value not to be established on a certain node when the circuit has been simulated using the condition specified by the -tag argument.

Operating Mode

All

Messages and Suggested Fix

Message 1

[FATAL] Invalid constraint_message_tag name is used in constraint_message_tag_expression '<expression>'. Reason: <reason>

Potential Issues

A violation message is reported when invalid constraint_message_tag is specified in the constraint_message_tag_expression.

Consequences of Not Fixing

See [Consequences of Not Fixing](#)

How to Debug and Fix

See [How to Debug and Fix](#)

Message 2

[FATAL] Syntax error found in constraint_message_tag_expression '<expression>'. Reason: <reason>

Potential Issues

A violation message is reported when incorrect constraint_message_tag_expression is specified.

Consequences of Not Fixing

See [Consequences of Not Fixing](#)

How to Debug and Fix

See [How to Debug and Fix](#)

Message 3

[FATAL] Invalid constraint_message_tag name '<tag>' is used, valid name can only contain 'a-z', 'A-Z', '0-9', and '_'

Potential Issues

A violation message is reported when invalid constraint_message_tag name is specified.

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

To fix the violation, review the specified constraints.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCSTX_075

Performs sanity check for the clock_shaper constraint

When to Use

Use this rule to perform sanity check for the clock_shaper constraint

Description

The *dftSGDCSTX_075* rule reports a violation you have specified the `-scan_set` and/or `-scan_reset` arguments without using the `-scan_clock` argument of the *clock_shaper* constraint.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

clock_shaper

Operating Mode

None

Messages and Suggested Fix

The *dftSGDCSTX_075* rule reports the following violation message:

[FATAL] -scan_clock' is required with '-scan_set' and '-scan_reset' arguments of 'clock_shaper'

Potential Issues

The violation message is reported when you have specified the `-scan_set` and/or `-scan_reset` arguments without using the `-scan_clock` argument of the *clock_shaper* constraint.

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

To fix the violation, specify the `-scan_clock` argument, if you have specified `scan_set` and/or `-scan_reset` arguments of the *clock_shaper*

Integrity Checks

constraint.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCSTX_076

Performs sanity check for the missing required fields of the `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, `require_constraint_message_tag` and `illegal_constraint_message_tag` constraints

When to Use

Use this rule to perform a sanity check for the missing required fields of the `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, `require_constraint_message_tag` and `illegal_constraint_message_tag` constraints.

Description

The `dftSGDCSTX_076` rule reports a violation when a set of optional fields are not defined for the supported constraint.

[Table 4](#) describes the supported constraints and the corresponding arguments on which the `dftSGDCSTX_076` rule runs a sanity check:

TABLE 4 Constraint List for the `dftSGDCSTX_076` rule

Constraint	Argument
<code>require_path</code>	-from, -from_type, -from_one_of, -from_one_of_type, require_path, -to, -to_type, -to_one_of, -to_one_of_type
<code>illegal_path</code>	-from, from_type, -to, -to_type
<code>require_strict_path</code>	-from, -from_type, -from_one_of, -from_one_of_type, -to, -to_type, -to_one_of, -to_one_of_type
<code>require_value</code>	-name, -type
<code>illegal_value</code>	-name, -type
<code>require_constraint_message_tag</code>	-name, -type
<code>illegal_constraint_message_tag</code>	-name, -type

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

The *dftSGDCSTX_076* rule reports the following violation message:

[FATAL] '<constraint-name>' constraint must have at least one of the following fields defined '<list of optional fields>'

Potential Issues

Constraint is not valid without mentioned missing fields.

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

Review the constraint and specify the missing fields.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCSTX_077

Performs sanity check for usages of instance based macros in `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, and `require_pulse` constraint

When to Use

Use this rule to perform a sanity check for the usage of instance-based macros `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, and `require_pulse` constraints.

Description

The `dftSGDCSTX_077` rule reports a violation if the use of instance-based macros is detected in the `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value` and `require_frequency` constraints.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

The `dftSGDCSTX_077` rule reports the following violation message:

[FATAL] 'constraint-name' constraint does not support instance based macros '<list of inst-macros used>'

Potential Issues

The violation message is reported when you have used instance-based macros in any of the following constraints:

- `require_path`
- `illegal_path`
- `require_strict_path`

Integrity Checks

- *require_value*
- *illegal_value*
- *require_frequency*

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error.

How to Debug and Fix

Review the constraint to remove instance-based macro reported in the violation message.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCSTX_078

Sanity check for usage of macros in `require_value` and `illegal_value` constraints when `-value_type` field is used

When to Use

Use this rule to perform a sanity check for the usage of unsupported macros in the `-value_type` argument of the [require_value](#) and [illegal_value](#) constraints.

To view the list of macros supported by the `require_value` and `illegal_value` constraints, see [List of Macros Supported by the require_value and illegal_value Constraints](#).

Description

The `dftSGDCSTX_078` rule reports a violation if unused macros are used in the `-value_type` argument of the [require_value](#) and [illegal_value](#) constraints.

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

The `dftSGDCSTX_078` rule reports the following violation message:

```
[FATAL] 'constraint-name' constraint does not support '<list_of_unsupported_macros>' macros when '-value_type' is used to specify the value,
```

where,

`constraint-name` can be either [require_value](#) or [illegal_value](#).

Potential Issues

The violation message is reported when you have unused macros in the -

value_type argument of the any of the following constraints:

- *require_value*
- *illegal_value*

Consequences of Not Fixing

Not fixing the violation may result in a fatal design error and the software run aborts.

How to Debug and Fix

Review the constraint to use only the supported macros. Else, use the -value argument instead of the -value_type argument of the require_value and illegal_value constraints.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCSTX_080

Performs sanity checks on design-object-type/conditional-checks(filtering) based fields

When to Use

Use this rule to perform sanity checks on the following:

- **Design-object-type based arguments**, such as, `-type`, `-except_type`, `-from_type`, `-to_type`, `-from_one_of_type`, `-to_one_of_type`, `-filter_in_type`, `-filter_in_type_from`, and `-filter_in_type_to`
- **Conditional-checks (filtering) based arguments**, such as, `-filter_in_cmt`, `-instance_filter_in_cmt`, `-filter_in_cmt_from`, `-filter_in_cmt_to`, `-instance_filter_in_cmt_from`, `-instance_filter_in_cmt_to`, and `-cmt`

Description

The dftSGDCSTX_080 rule reports violation for the following:

- If the `-filter_in_cmt`, `-filter_in_cmt_from`, `-filter_in_cmt_to`, `-instance_filter_in_cmt`, `-instance_filter_in_cmt_from`, `-instance_filter_in_cmt_to`, and `-cmt` arguments points to a tag which is not present in either of the following constraint:
 - `require_value`
 - `require_path`
 - `illegal_value`
 - `illegal_path`
 - `require_strict_path`
- Contains a tag modifier other than PASS or FAIL
- The expression contains operator other than '&&' (and) and '||' (or)
- Contains cyclic dependency among constraints
- Contains disabled tag (corresponding rules of the constraints are inactive)

Integrity Checks

- If `-type`, `-except_type`, `-from_type`, `-to_type`, `-from_one_of_type`, `-to_one_of_type`, `-filter_in_type`, `-filter_in_type_from`, `-filter_in_type_to` contains unrecognized macro, that is, neither any static macro nor dynamic macro (tcl-based or *define_macro* based).
- Cyclic dependency among the *define_macro* and other constraints wherever design-object-type based field is supported
- Contains disabled tag (tags disabled due to inactive corresponding rules)
- If dependency on any connectivity constraint(s) found for `test_mode` constraint, because connectivity constraint(s) require `test_mode` simulation data for analysis

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

- *require_value* (optional) : Use this constraint to define a check that requires a logic value to be established on a certain node when the circuit has been simulated using the condition specified by `-tag` argument.
- *require_path* (optional) : Use this constraint to define a connectivity check for path from a pin specified with the `-from` argument to a pin specified with `-to` argument.
- *illegal_value* (optional) : Use this constraint to define a check that requires a logic value not to be established on a certain node when the circuit has been simulated using the condition specified by `-tag` argument.
- *illegal_path* (optional) : Use this constraint to define a connectivity check for an illegal path from a pin specified with the `-from` argument to a pin specified with `-to` argument.
- *define_macro*: Specifies the dynamic macros.
- *test_mode* (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

- *require_strict_path* (mandatory): Use this constraint to define a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument.

Operating Mode

All

Messages and Suggested Fix

Message 1

[FATAL] Invalid constraint_message_tag name is used in argument '`<argument-name>`' expression '`<expression>`'. Reason: `<reason>`

Argument(s)

- Argument name, `<argument_name>`
- Filtering expression, `<expression>`
- Reason for invalid filtering expression, `<reason>`

Potential Issues

A violation message is reported when the filtering expression contains invalid `constraint_message_tag`.

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Message 2

[FATAL] Syntax error found in argument '`<argument-name>`' expression '`<expression>`'. Reason: `<reason>`

Argument(s)

- Filtering-based argument name, `<argument-name>`
- Filtering expression, `<expression>`
- Reason for incorrect filtering expression, `<reason>`

Potential Issues

A violation message is reported when the filtering expression is incorrect.

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Message 3

[FATAL] Invalid constraint_message_tag name '<tag>' is used in argument '<argument-name>' expression, valid name can only contain 'a-z', 'A-Z', '0-9', and '_'

Argument(s)

- constraint_message_tag, <tag>
- Filtering-based argument name, <argument-name>

Potential Issues

A violation message is reported when invalid constraint_message_tag name is specified for one of the argument.

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Message 4

[FATAL] Constraint_message_tag '<cmt>' contains disabled tag '<disabled_cmt>' in the argument '<argument-name>' expression in current design '<design>'

Argument(s)

- constraint_message_tag, <cmt>
- Disabled constraint_message_tag, <disabled_cmt>
- Filtering-based argument name, <argument-name>
- Top-module name, <design>

Potential Issues

A violation message is reported when disabled constraint_message_tag names are used in the expressions of one of the filtering-based argument. The tags are disabled due to corresponding rules being not run.

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Message 5

[FATAL] Macro '`<macro-name>`' provided in '`<argument-name>`' argument of constraint is not recognized.

Argument(s)

- Macro name, `<macro-name>`
- Design-object-type based argument name, `<argument-name>`

Potential Issues

A violation message is reported if the macro name specified is unrecognized, i.e.- macro is neither static nor dynamic, that is, Tcl-based or [define_macro](#) is specified.

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Message 6

[FATAL] Dependency on connectivity constraint(s) found for test_mode constraint in current design '`<design>`' [`<dependency-string>`]

Argument(s)

- Top-module name, `<design>`
- Dependency string, `<dependency-string>`

Potential Issues

A violation message is reported if dependency on any connectivity constraint(s) found for the [test_mode](#) constraint, because connectivity constraint(s) require test_mode simulation data for analysis.

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Message 7

[FATAL] Cyclic dependency found among constraint_message_tags and/or macros in current design '`<current-design>`' [`<cyclic-dependency-string>`]

Argument(s)

- Top-module name, `<current-design>`
- Cyclic dependency string, `<cyclic-dependency-string>`

Potential Issues

A violation message is reported when cyclic dependency is found among:

- Constraints
- User-defined macros (specified by `define_macro` constraint)
- Constraints and user-defined macros

Consequences of not fixing

Not fixing the violation may result in a fatal design error.

How to debug and fix

To fix the violation, review the specified constraints.

Example Code and/or Schematic

Dependency Among Constraints

There should not be any cyclic dependency among multiple constraints based on filtering expression. Cyclic dependency will be reported as FATAL severity message and software execution will stop.

Consider the following constraint specifications for understanding the cyclic dependency among constraints

```
illegal_value -name obj1 -value 0_or_1 -
constraint_message_tag X1 -filter_in_cmt "X1:PASS"

illegal_value -name obj2 -value 0_or_1 -except_type TIED -
constraint_message_tag X2 -filter_in_cmt "X3:PASS"
```

```

illegal_value -name obj3 -value 0_or_1 -except_type TIED -
constraint_message_tag X3 -filter_in_cmt "X2:PASS"

illegal_value -type "FLIP_FLOP_CLOCK" -value 0_or_1 -
constraint_message_tag X4 -filter_in_cmt "X7:FAIL"

require_value -name obj1 -value 1 -except_type TIED
-constraint_message_tag X5 -filter_in_cmt "X4:FAIL"

require_value -tag t1 -name obj1 -value 0
-constraint_message_tag X6 -filter_in_cmt "X5:PASS"

require_path -from sig1 -to sig2
-constraint_message_tag X7 -filter_in_cmt_from "X6" -
filter_in_cmt_to "X6"

```

For the above example, the following cyclic dependency violations are reported:

Cyclic dependency found among constraint_message_tags in current design 'top' [X1->X1]

Cyclic dependency found among constraint_message_tags in current design 'top' [X2->X3->X2]

Cyclic dependency found among constraint_message_tags in current design 'top' [X4->X7->X6->X5->X4]

Dependency Among UDMs

There should not be any cyclic dependency among multiple define_macro constraints. Cyclic dependency will be reported as FATAL severity message and software execution will stop.

Consider the following examples where FATAL violations are reported due to cyclic dependency:

Example 1:

Consider the following constraint specification:

```
define_macro -macro macro1 -type macro1
```

In the above example, the dependency tree is macro1 -> macro1.

Example 2:

Consider the following constraint specifications:

```
define_macro -macro macro1 -type macro2
```

```
define_macro -macro macro2 -type macro1
```

In the above example, the dependency tree is macro1 -> macro2 -> macro1.

Example3:

Consider the following constraint specifications:

```
define_macro -macro macro1 -type macro2
```

```
define_macro -macro macro2 -type macro3
```

```
define_macro -macro macro3 -type macro1
```

In the above example, the dependency tree is macro1 -> macro2 -> macro3 -> macro1.

Dependency Among UDMs and Constraints

There should not be any cyclic dependency among define_macro and other constraints. Cyclic dependency will be reported as FATAL severity message and software execution will stop.

Consider the following examples where FATAL violations are reported due to cyclic dependency:

Example 1:

Consider the following constraint specifications:

```
define_macro -macro macro1 -cmt X2
```

```
require_value -type macro1 -value 0 -constraint_message_tag X1
```

```
require_value -name obj -value 0 -constraint_message_tag X2 -  
filter_in_cmt "X1:FAIL"
```

In the above example, the dependency tree is macro1 -> X2 -> X1 -> macro1.

Example 2:

Consider the following constraint specifications:

```
define_macro -macro macro1 -type macro2
```

```
define_macro -macro macro2 -cmt X1
require_value -type macro1 -value 0 -constraint_message_tag
X1
```

In the above example, the dependency tree is macro1 -> macro2 -> X1 -> macro1.

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCDefineMacroCheck_01

The `define_macro` constraint sanity checks

When to use

Use this rule to perform sanity check on the macro names specified using the [define_macro](#) constraint.

Description

The `dftSGDCDefineMacroCheck_01` rule reports a violation if the macro-name specified using the [define_macro](#) constraint is not valid.

A macro name can be invalid due to one of the following reasons:

- An empty macro name is specified.
- A predefined-macro, static or dynamic, is specified.
- Invalid characters are used, that is, valid name can only contain: 'a-z', 'A-Z', '0-9', and '_'.

This rule performs sanity checks before the Analysis phase. To perform the sanity checks after the Flattening phase use the [dftSGDCDefineMacroCheck_02](#) rule.

Parameter(s)

None

Constraint(s)

[define_macro](#) : Specifies the dynamic macros

Operating Mode

None

Messages and Suggested Fix

[FATAL] Invalid macro name '`<udm_name>`' is used in specification of `define_macro`. Reason: `<reason>`

Argument(s)

- User-defined macro name, `<udm_name>`

- Reason for invalid udm_name, <reason>

Potential Issues

A violation message is reported if the macro-name specified using the [define_macro](#) constraint is not valid.

Consequences of not fixing

Not fixing the violation results in a fatal error.

How to debug and fix

To fix the violation, check the macro-name specified using the [define_macro](#) constraint.

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftSGDCDefineMacroCheck_02

The `define_macro` constraint sanity checks

When to use

Use this rule to perform sanity check on the macro names specified using the [define_macro](#) constraint.

Description

The `dftSGDCDefineMacroCheck_01` rule reports a violation if a dynamic macro can not be created for the user-defined macro name.

Parameter(s)

None

Constraint(s)

[define_macro](#): Specifies the dynamic macros

Operating Mode

None

Messages and Suggested Fix

[ERROR] Unable to create macro '`<udm_name>`'

Argument(s)

- User-defined macro name, `<udm_name>`

Potential Issues

A violation message is reported if dynamic macro can not be created for a specified `udm_name`.

Consequences of not fixing

If the violation message is not fixed, the dynamic macro is ignored from the rule checking.

How to debug and fix

To fix the violation, check the [define_macro](#) constraint specification.

Default Severity Label

Fatal

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files

dftUserMacroSanityCheck_01

Performs sanity check for usages of user macros in the `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, and `require_pulse` constraints

When to Use

Use this rule to perform a sanity check for the usage of user macros in the `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, and `require_pulse` constraints.

Description

The `dftUserMacroSanityCheck_01` rule reports a violation, if the use of user macros is detected in the `require_path`, `illegal_path`, `require_strict_path`, `require_value`, `illegal_value`, `require_frequency`, and `require_pulse` constraints.

Parameter(s)

Common SpyGlass DFT ADV Rule Parameters

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

Message 1

```
[FATAL] User macro '<macro_name>' ignored because of change in design '<du_name>'. Previous node count:
inst: '<previous_instance_count>'
term: '<previous_terminal_count>' port: '<previous_port_count>'.
Current node count: inst: '<previous_instance_count>'
term: '<previous_terminal_count>' port: '<previous_port_count>'
```

Potential Issues

The violation message is reported when the design has changed.

Consequences of Not Fixing

If the violation message is not fixed, UDM will be ignored.

How to Debug and Fix

Re-define the user macros.

Message 2

[FATAL] Instances present in user macro(s)
'<list_of_user_defined_macros>' were ignored for constraint
'<top_du_name>'

Potential Issues

The violation message is reported when the instances present in macros are not supported for rule checking.

Consequences of Not Fixing

If the violation message is not fixed, instances inside the UDM will be ignored.

How to Debug and Fix

The [require_path](#), [illegal_path](#), [require_strict_path](#), [require_value](#), [illegal_value](#), [require_frequency](#), and [require_pulse](#) constraints do not support instances as an argument. Instead, specify the corresponding pin for that instance.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Fault

Rule Group

Integrity Rules, Must Rules

Reports and Related Files

No related reports or files.

dftVSTOPDUSetup

This is a set up rule.

When to Use

This is a pre-requisite and set up rule and is run always.

Description

Run this rule to complete the pre-run setup. The rule marks those synthesizable modules as leaf for which following constraints are specified

- [clock_shaper](#)
- [module_bypass](#)

For such cells, internal structure may be visible in GUI but back annotation data is not visible on the internal nodes.

Parameter(s)

[Common SpyGlass DFT ADV Rule Parameters](#)

Constraint(s)

- [clock_shaper](#)
- [module_bypass](#)

Operating Mode

None

Messages and Suggested Fix

The dftVSTOPDUSetup rule reports the following violation message:

```
[WARNING] SGDC '<sgdc_name>' forces a white-box module '<module_name>' to be treated as a leaf cell
```

Potential Issues

The violation message is reported if a [clock_shaper](#) and/or [module_bypass](#) constraints have been specified on a synthesizable cell.

Consequences of Not Fixing

The marked synthesizable cells are treated as leaf for DFT analysis.

How to Debug and Fix

Review the constraint specification.

Example Code and/or Schematic

The following figure shows the impact of treating a synthesizable cell, `u_clock_shaper_buf`, as a leaf cell for the DFT analysis:

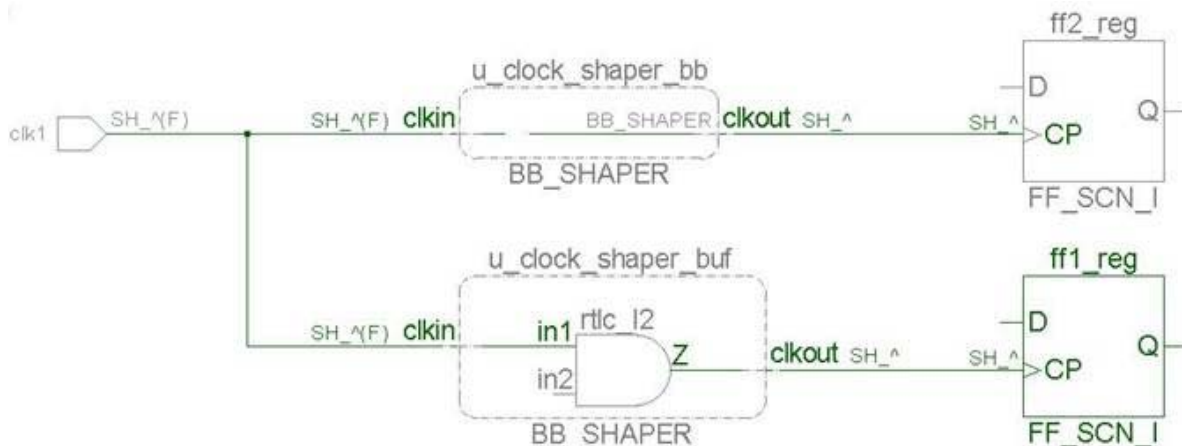


FIGURE 108. cell marked as leaf cell

In the above example, even though other input of AND gate does not have a value '1', clock will still pass through as it has been defined using the [clock_shaper](#) constraint. There will not be any schematic back annotation available for nodes inside the cell. Additionally, simulation conditions may not be consistent for clock propagation because there is no check for the internal logic.

Default Severity Label

Warning

Rule Group

Integrity Rules, Must Rules

Integrity Checks

Reports and Related Files

No related reports or files.

Diagnose_ScanChain

Ensure that the scan chain must exist between the user-specified scanin and scanout points

When to Use

Use this rule to check scan chain integrity.

Description

The Diagnose_ScanChain rule reports invalid user-specified scan chains under specified simulation conditions.

A path between the scanin and scanout points is a valid scan chain path if it does not contain any of the following under the specified simulation condition:

- A combinational loop
- A blocked terminal
- A multi-driven net
- A floating net
- A black box instance
- A non-lockup latch
- A non-transparent latch
- A non-scan flip-flop
- Instances having more than one un-blocked input
- A net having non-X simulation value

Scan chains are specified using the [scan_chain](#) constraint and the simulation conditions are specified using the [define_tag](#) constraint.

The *Diagnose_ScanChain* rule reports violation when a scan chain breaks due to presence of a non-transparent latch in the shift mode. This implies that the *Diagnose_ScanChain* rule indirectly checks for latch transparency in the shift mode for latches present on a scan chain path.

Prerequisites

You can specify the [scan_chain](#) constraint only when scan chain is stitched in the design.

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dft_max_flops_in_diagnose_scan_chain_violation_schematic](#): The default value is 3. Set the value of the parameter to all or any non-negative integer value to control the maximum number of flip-flops shown in schematic for the violation messages reported by the *Diagnose_ScanChain* rule.
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Constraints

- [define_tag](#) (mandatory): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- [force_scan](#) (optional): Use this constraint to declare flip-flops as scannable even if they do not so qualify.
- [force_no_scan](#) (optional): Use this constraint to exclude flip-flops from being declared scannable even if they so qualify.
- [test_mode](#) (optional): Use this constraint to specify the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.
- [scan_chain](#) (mandatory): Use this constraint to specify the scan chains for the Info_scanchain rule.

Operating Mode[Scanshift](#)

Messages and Suggested Fix

Message 1

[WARNING] Scan chain (scanin=>'<port1-name>' scanout=>'<port2-name>') [under <tag-name>] is blocked after tracing '<num_scan_cells>' scan cells, starting from scanout. Cause: <reason>

Arguments

- Name of the scanin port, <port1-name>
- Name of the scanout port, <port2-name>
- Simulation condition, <tag-name>
- Number of scan cells traced. <num_scan_cells>
- Cause of blocking the scan chain. <reason>

Potential Issues

The violation message appears, if the `scan_chain` constraint specification is incomplete or connectivity in the design is wrong.

Consequences of Not Fixing

If you do not fix this violation, required testing cannot happen using this scan chain.

How to Debug and Fix

Double-click the violation message. The Incremental Schematic window highlights the path from scanout port up to the instance which breaks the scan path.

You can also view the violations for the `Info_testmode` (under shift condition), `Soc_04`, and `Info_testclock` rules along with the violation of the `Diagnose_ScanChain` rule in the Incremental Schematic window. To do this, double-click the violation message for the `Diagnose_ScanChain` rule and open the Incremental Schematic window.

The violation messages of the `Info_testmode`, `Soc_04`, and `Info_testclock` overlap the violation for the `Diagnose_ScanChain` rule in the Incremental Schematic window. This is useful in debugging the violation for the `Diagnose_ScanChain` rule.

To fix the violation, ensure that the `scan_chain` constraint specifications are correct and/or design connectivity is right.

Message 2

[WARNING] No scan chain created having <scanin/scanout> = '<name_of_si_so_node>' because corresponding <scanout/scanin> not found in given constraint

Arguments

Name of the scanin or scanout node, <name_of_si_so_node>

Potential Issues

The violation message appears if the scan_chain constraint specification is incorrect.

Consequences of Not Fixing

Required testing cannot happen using the scan chain specified by this constraint.

How to Debug and Fix

Review constraint and make sure scanin and scanout points to intended design nodes.

Message 3

[WARNING] Scan chain (scanin=>'<scan_in_net>' scanout=>'<scan_out_net>') [under <condition>] detection did not happen because of '<reason>'

Arguments

- Define tag or shift mode, <condition>
- Invalid/incomplete condition specification, <reason>

Potential Issues

The violation message appears if the condition specified in the scan_chain constraint is incorrect

Consequences of Not Fixing

Required testing cannot happen using the scan chain specified by this constraint.

How to Debug and Fix

Review constraint and make sure that condition for scan chain constraint is valid.

Message 4

[WARNING] Scan chain (scanin=>'<scan_in_net>' scanout=>'<scan_out_net>') [under <condition>] is ignored.
Cause: Both scanin and scanout point to same design net '<net_name>'

Arguments

Define tag or shift mode, <condition>

Potential Issues

The violation message appears if the both scanin and scanout are same in the scan_chain constraint. In case of wildcard, both scanin and scanout patterns usually matching the same design net.

Consequences of Not Fixing

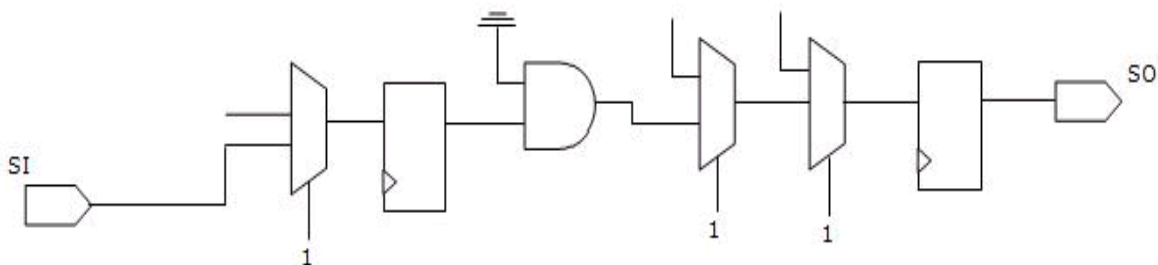
Required testing cannot happen using the *scan_chain* specified by this constraint.

How to Debug and Fix

Review constraint and make sure scanin and scanout points to intended design nodes. In case of wildcard, use more specific patterns so that they match different design nets.

Example Code and/or Schematic

Consider the following figure:



In the above figure, if the upper input pin of AND gate is 1, then the scan chain is completed and valid between scan-in and scan-out points.

Integrity Checks

Default Severity Label

Warning

Rule Group

MustRules

Reports and Related Files

No related reports or files.

dumpBlackBox

Identify black boxes in a design and dump corresponding 'scan_wrap' constraints in a file

When to Use

Use this constraint to identify the black boxes in the design and generate a SpyGlass Design Constraints format file having scan_wrap constraints for each such black box.

Description

The dumpBlackBox rule provides a list of all possible scan_wrap constraints for a given top-module. You can then keep the ones that he/she desires, and comment out the rest. This generated file can then be passed in the second run along with any other constraint file.

The dumpBlackBox rule generates the [scan_wrap Report for the dumpBlackBox Rule](#) in the current working directory. This report is in the SpyGlass Design Constraints format and has a scan_wrap constraint for each black box under every top-level design unit. You can then review this file and comment out unwanted scan_wrap constraints. You can then supply this file (along with other SpyGlass Design Constraints files) in the next SpyGlass DFT ADV run.

NOTE: *The dumpBlackBox rule is ignored, if you run it along with the [Info_scanwrap](#) rule. Please use Info_scanwrap rule as dumpBlackBox rule will be deprecated in a future release.*

Default Weight

10

Language

Verilog, VHDL

Parameter(s)

- [Common SpyGlass DFT ADV Rule Parameters](#)
- [dftUseOffStateOfClockInClockPropagation](#): The default value of the parameter is on. Set the value of the parameter to off so that clock lines are kept at X during shift, capture, or atspeed mode simulation.

Integrity Checks

Constraint(s)

None

Operating Mode

None

Messages and Suggested Fix

The following violation message is displayed for the dumpBlackBox rule:

[INFO] Scanwrap constraint file <file-name> is generated

Where <file-name> is the name of the generated file.

Potential Issues

A violation is reported when a black boxes is present in the design.

Consequences of Not Fixing

Not fixing the violation may result in lower coverage.

How to Debug and Fix

The dumpBlackBox rule generates an SGDC file for all the black boxes found in the design and requires no debug.

To fix the violation, provide description of black boxes or put scan_wrap around them.

Example Code and/or Schematic

Currently Unavailable

Default Severity Label

Info

Rule Group

Integrity Checks

Reports in the SpyGlass DFT ADV Product

The SpyGlass DFT ADV product generates rules and reports that you can view from the *Reports* menu in the Atrenta Console or by using the `set_option report <report-name>` command followed by the report name.

The following table lists the available SpyGlass DFT ADV product reports:

Report Name	Purpose
<i>add_fault</i>	Lists all the ports and pins which are marked as add fault using the add_fault SGDC constraint
<i>bist_ready_summary</i>	The bist_ready_summary.rpt file is generated by the BIST_01, BIST_02, BIST_03, BIST_04, and BIST_05 rules. This report contains a summary of these rules.
<i>dft_ff_set_reset_active</i>	Lists all flip-flops and the combination that makes the Async pins of these flip-flops simultaneously active
<i>dft_ff_set_reset_sequential_in_capture</i>	Lists all flip-flops whose set/reset pins are driven by sequential elements/black box in capture mode

<i>dft_ff_set_reset_sequential_in_shift</i>	Lists all flip-flops whose set/reset pins are driven by sequential elements/black box in shift mode
<i>dft_ff_X_source_for_tristate_enable</i>	Lists all the flip-flops that could cause X on Tri- Enable's
<i>dft_latch_enable</i>	Contains the details regarding info latch mapping
<i>dft_mandatory_sgdc</i>	Lists all the missing mandatory constraints needed by a dft rule
<i>no_fault</i>	Lists all blocks for which all faults, internal and boundary pins, are treated as no fault
<i>dft_optional_sgdc</i>	Lists all the missing optional constraints needed by a SpyGlass DFT ADV rule.
<i>dft_summary</i>	Lists a summary of autofix performed by the SpyGlass DFT ADV product
<i>dft_tristate</i>	Lists all the tristate buses in the circuit
<i>dft_initialized_ffs</i>	list all the initialized flip-flops after an initialization sequence is applied
<i>potentially_detected_faults</i>	Lists all the potentially detectable faults
<i>scan_chain</i>	Lists all flip-flops that are or are not a part of some scan chains
<i>scan_wrap</i>	Contains a scanwrap list of all black boxes and their related information
<i>soc_06_rpt</i>	Lists the hierarchical name of all the instances appended with a pin name
<i>stil_file</i>	Lists all STIL constructs
<i>stuck_at_faults</i>	Lists all stuck-at faults
<i>stuck_at_coverage</i>	Contains the summary and details of stuck at fault numbers
<i>stuck_at_coverage_audit</i>	Contains the coverage audit summary
<i>test_points_selected_2</i>	Lists suggested test points
<i>undetected_faults</i>	Lists undetected faults and their causes

add_fault

The add_fault.rpt file is generated by the [Info_addFault](#) rule. This report contains a lists all the ports and pins which are marked as add fault using the [add_fault](#) SGDC constraint.

A sample format of add_fault report is shown below:

```
#####
# Format :
# Design Name : design
# 'number of add-fault faults' <sgdc command>
# Port | Terminal: Fault name

#####

#####

Design Name : "test"
'4' faults are marked as 'add_fault' due to constraint
'add_fault -name bb2_af_mod'
  Terminal: test.bb2_af_mod_il (Master = bb2_af_mod).ip
  Terminal: test.bb2_af_mod_il (Master = bb2_af_mod).op
'4' faults are marked as 'add_fault' due to constraint
'add_fault -name wb2_af_mod'
  Terminal: test.wb2_af_mod_il.rtlc_I1 (Master = RTL_BUF).A
  Terminal: test.wb2_af_mod_il.rtlc_I1 (Master = RTL_BUF).Z
'4' faults are marked as 'add_fault' due to constraint
'add_fault -name test.bb1_af_inst_il'
  Terminal: test.bb1_af_inst_il (Master = bb1_af_inst).ip
  Terminal: test.bb1_af_inst_il (Master = bb1_af_inst).op
'4' faults are marked as 'add_fault' due to constraint
'add_fault -name test.wb1_af_inst_il'
  Terminal: test.wb1_af_inst_il.rtlc_I1 (Master =RTL_BUF).A
  Terminal: test.wb1_af_inst_il.rtlc_I1 (Master =RTL_BUF).Z
'6' faults are marked as 'add_fault' due to constraint
'add_fault -register_suffix fftype1'
  Terminal: test.ff_fftype1_reg (Master = RTL_FD).D
  Terminal: test.ff_fftype1_reg (Master = RTL_FD).CP
```

```

Terminal: test.ff_fftypel_reg (Master = RTL_FD).Q
'2' faults are marked as 'add_fault' due to constraint
'add_fault -fault test.buf02_i1.A'
Terminal: test.buf02_i1 (Master = buf02).A
'4' faults are marked as 'add_fault' due to constraint
'add_fault -net test.w2'
Terminal: test.buf02_i2 (Master = buf02).Y
Terminal: test.buf02_i3 (Master = buf02).A
'2' faults are marked as 'add_fault' due to constraint
'add_fault -net_output ip9'
Terminal: test.buf02_i4 (Master = buf02).A
'2' faults are marked as 'add_fault' due to constraint
'add_fault -net_input op9'
Terminal: test.buf02_i4 (Master = buf02).Y
'14' faults are marked as 'add_fault' due to constraint
'add_fault -clock_control clk2'
Terminal: test.ff3 (Master = sffs).D
Terminal: test.ff3 (Master = sffs).CLK
Terminal: test.ff3 (Master = sffs).SI
Terminal: test.ff3 (Master = sffs).SE
Terminal: test.ff3 (Master = sffs).S
Terminal: test.ff3 (Master = sffs).Q
Terminal: test.ff3 (Master = sffs).QB
'14' faults are marked as 'add_fault' due to constraint
'add_fault -set_control set'
Terminal: test.ff2 (Master = sffs).D
Terminal: test.ff2 (Master = sffs).CLK
Terminal: test.ff2 (Master = sffs).SI
Terminal: test.ff2 (Master = sffs).SE
Terminal: test.ff2 (Master = sffs).S
Terminal: test.ff2 (Master = sffs).Q
Terminal: test.ff2 (Master = sffs).QB
'14' faults are marked as 'add_fault' due to constraint
'add_fault -reset_control reset'
Terminal: test.ff1 (Master = sffr).D
Terminal: test.ff1 (Master = sffr).CLK
Terminal: test.ff1 (Master = sffr).SI
Terminal: test.ff1 (Master = sffr).SE

```

```
Terminal: test.ff1 (Master = sffr).R
Terminal: test.ff1 (Master = sffr).Q
Terminal: test.ff1 (Master = sffr).QB
```

SUMMARY:

```
'74' faults are marked as 'add_fault' for design "test"
#####
```

atpg_conflict_testpoints

The atpg_conflict_testpoints.rpt file is generated by the [Info_atpg_conflict](#) rule. This report specifies test point location along with conflict value at that point. The conflict value reported for testpoint is the number of conflicts present at that node once we have applied all testpoints reported before that particular testpoint.

A sample format of atpg_conflict_testpoints report is shown below:

```
#####
# Format:
#   Index      Testpoint type      Conflict value      Design nodes
#
#####

Testpoint(s) for "test"
Testpoint(s) considered:- 6

      Index      Testpoint type      Conflict value      Design nodes
-----
1          Control-1          8          Net: "test.w3"
Pin(s): "test.i28.A0"
2          Control-0          8          Net: "test.w3"
Pin(s): "test.i32.A0"
3          Control-1          8          Net: "test.w3"
Pin(s): "test.i36.A0"
4          Control-0          6          Net: "test.w3"
Pin(s): "test.i40.A0"
5          Control-0          4          Net: "test.ip3"
Pin(s): "test.i11.A1"
6          Control-0          4          Net: "test.w1"
Pin(s): "test.i20.A1"
```

The atpg_conflict_testpoints.rpt file is generated in the <current-working-

directory>/spyglass_reports/dft_dsm/ directory.

bist_ready_summary

The bist_ready_summary.rpt file is generated by the BIST_01, BIST_02, BIST_03, BIST_04, and BIST_05 rules. This report contains a summary of these rules.

A sample format of the bist_ready_summary report is shown below:

```
#####
# Purpose :
#   The report summarizes all the BIST_XX rule violations
#   BIST_01:
#   Reports the number of flip-flops for which fanin cone
#   width is higher than the value specified by
#   'flopInFaninCount'
#   BIST_02:
#   Reports the number of gates for which fanin count is
#   higher than the value specified by 'gateInputCount'
#   BIST_03:
#   Reports the number of flip-flops for which the initial
#   value is not set even after the initialization sequence
#   is applied
#   BIST_04:
#   Reports the number of observable points which are
#   driven by unknown ('X') sources
#   BIST_05:
#   Reports the number of TIE_X cell outputs which are not
#   bypassed
#
#   NOTE : For more details, refer to the corresponding
#   rule violations
#####

#####

# Format :
#   Top design unit: <top_design_name>
#
#   BIST_01:
```

```

#      <N> flip-flops have more than <flopInFaninCount>
#      sources (flip-flops and black boxes) in fanin
#
#      BIST_02:
#      <N> gates have more than <gateInputCount> fanins
#
#      BIST_03:
#      <N> flip-flops remain uninitialized
#
#      BIST_04:
#      <N> observable points are driven by 'X' sources
#
#      BIST_05:
#      <N> TIE_X cell outputs are not bypassed
#
#      NOTE : If a particular rule is disabled in the current
#      run, then there will not be a section corresponding to
#      that rule
#####

#####

Top design unit: top1

BIST_01:
  '2' flip-flops have more than '150' sources (flip-flops
and black boxes) in fanin

BIST_02:
  '3' gates have more than '8' fanins

BIST_03:
  '2' flip-flops remain uninitialized

BIST_04:
  '4' observable points are driven by 'X' sources

```

```

BIST_05:
  '1' TIE_X cell output is not bypassed
#####

#####

```

Top design unit: top2

```

BIST_01:
  No flip-flop has more than '150' sources (flip-flops and
black boxes) in fanin

```

```

BIST_02:
  No gate has more than '8' fanins

```

```

BIST_03:
  No flip-flop remains uninitialized

```

```

BIST_04:
  No observable point is driven by 'X' sources

```

```

BIST_05:
  All TIE_X cell outputs are bypassed
#####

```

The bist_ready_summary.rpt file is generated in the <current-working-directory>/spyglass_reports/dft/ directory.

dft_connectivity_check_summary

The dft_connectivity_check_summary.rpt file is generated by the [Connection Rules](#). This report contains a summary of these rules.

```
#####
# Purpose :
#   The report summarizes all the connectivity check results
#
#   require_value:      Checked by Soc_01, Soc_01_Info
#   Reports the number of 'require_value' constraints
#   passed and failed
#
#   require_path:      Checked by Soc_02, Soc_02_Info
#   Reports the number of 'require_path' constraints passed
#   and failed
#
#   require_structure:  Checked by Soc_07, Soc_07_Info
#   Reports the number of 'require_structure' constraints
#   passed and failed
#
#   require_strict_path: Checked by Soc_08
#   Reports the number of 'require_strict_path' constraints
#   passed and failed
#
#   illegal_path:      Checked by Soc_09
#   Reports the number of 'illegal_path' constraints passed
#   and failed
#
#   NOTE : For more details, refer to the corresponding
#   rule violations
#####

#####

# Format :
#   Top design unit: <top_design_name>
#
#   require_value:
```

```

#         <N> require_value constraints passed
#         <N> require_value constraints failed
#
#     require_path:
#         <N> require_path constraints passed
#         <N> require_path constraints failed
#
#     require_structure:
#         <N> require_structure constraints passed
#         <N> require_structure constraints failed
#
#     require_strict_path:
#         <N> require_strict_path constraints passed
#         <N> require_strict_path constraints failed
#
#     illegal_path:
#         <N> illegal_path constraints passed
#         <N> illegal_path constraints failed
#
#     NOTE : If a particular rule is disabled in the current
#           run, then there will not be a section corresponding to
#           that rule
#####
#####

Top design unit: top1

require_value:
    '1' require_value constraint passed
    No require_value constraint failed

require_path:
    '2' require_path constraints passed
    No require_path constraint failed

require_structure:
    No require_structure constraint passed

```

```
'1' require_structure constraint failed

require_strict_path:
  '2' require_strict_path constraints passed
  '2' require_strict_path constraints failed

illegal_path:
  '1' illegal_path constraint passed
  '2' illegal_path constraints failed
#####

#####

Top design unit: top2

require_value:
  No valid require_value constraint specified

require_path:
  No valid require_path constraint specified

require_structure:
  No valid require_structure constraint specified

require_strict_path:
  No valid require_strict_path constraint specified

illegal_path:
  No valid illegal_path constraint specified
#####
```

Scan Clock Tree Report

The *Scan Clock Tree* report is generated by the [Info_testclock](#) rule in the following location:

```
$wdir/spyglass_reports/dft//*
```

This report lists the scan clock tree within a mode.

In general, following is the naming convention of this report:

```
<clock_name>__<modename>_mode.rpt
```

When the clock name contains the escape character, '\', the clock name is modified to use it in filename. However, no change is done in case of the following special characters:

- '-' (hyphen)
- '_' (underscore)
- '[' ']' (both type of square brackets)

Also, the forward slash character, '/', is replaced by '.'. Other special characters are replaced by '_'. Therefore, the following naming convention is followed for the generated report:

```
<clock_name>_<modename>_mode_du<duNumber>_clock_id_<clockId>.rpt
```

NOTE: *This report is generated only when the [Info_testclock](#) rule is enabled and the clock constraints are defined.*

A sample `clk1_capture_mode.rpt` report is shown below:

```
#####
# Format :
# Scan Clock tree is shown in a tree-like format. The nodes
# are indented according to their position in hierarchy,
# `-- is used to indicate hierarchical cell, |-- is used
# for rest of the nodes. The level is also printed in
# braces before every cell.
#
# RX/TX Lists have the clock domains interaction with the
# current clock along with number of interactions
# a. RXLIST - list of clock domains driven by this clock
# b. TXLIST - list of clock domain driving instances of
# this clock
#
# The (+ve)/(-ve) suffix to a leaf-level cell output node
# name indicates the clock polarity reaching at that node.
# (BLKD) suffix is used to indicate clock blocking cells.
```

```

#
#   A leaf on the tree can be one of:
#   a. A register inferred from the RTL - the hierarchical
#   name of the registered instance is shown along with the
#   polarity of the clock at that point.
#
#   b. A black box - the hierarchical name of the black box
#   instance is annotated with the polarity of the clock.
#
#   c. A latch inferred from the RTL - the hierarchical name
#   of the latch instance is shown along with the polarity
#   of the clock.
#
#   d. A sequential leaf cell - the hierarchical name of the
#   sequential instance, which is neither known to be a flop
#   nor a latch, is annotated with the polarity of the
# clock.
#
#   e. A clock blocking cell - the hierarchical name of the
#   instance which is blocking this clock from propagating
#   forward with the polarity mentioned as BLKD.
#
#   The intermediate nodes of a tree can
#   a. A combinational cell - the hierarchical name of the
#   combinational cell found while walking along a path from
#   the root of the tree to a leaf is shown, along with the
#   number of leaf instances driven by this cell and the sum
#   of the number of leaf instances of all it's children
#   intermediate nodes and it's own leaf instances. Also,
#   the list of all the gating signals upto this hierarchy
#   is given in the next column.
#

```

```
#####
```

```

CLOCK:  clk1
MODE:   capture mode
RXLIST: 2:  top.wl2 (count: 1)

```



```

,    clk2 (count: 1)

top.clk1    [local:2, cumulative:12]
  |-- (1) top.D1.qbar_reg (+ve)
  |-- (1) top.D1.q_reg (+ve)
  `-- (1) top.rtlc_I33 [local:4, cumulative:10, Gating:1:
top.in1]
  |-- (2) top.D3.qbar_reg (+ve)
  |-- (2) top.D3.q_reg (+ve)
  |-- (2) top.D2.qbar_reg (-ve)
  |-- (2) .... ()
  `-- (2) top.rtlc_I39 [local:6, cumulative:6,
Gating:1: top.in2]
  |-- (3) top.D4.qbar_reg (+ve)
  |-- (3) top.D4.q_reg (+ve)
  |-- (3) top.D6.qbar_reg (+ve)
  |-- (3) .... ()
  |-- (1) top.rtlc_I49 (BLKD) [local:0, cumulative:0,
Gating:1: top.in4]

```

dft_ff_set_reset_active

The `dft_ff_set_reset_active.rpt` file is generated by the [Async_06](#) rule. This report contains a list of all flip-flops and the combination that makes the Async pins of these flip-flops simultaneously active.

A sample format of `dft_ff_set_reset_active` report is shown below:

```

#####
Format :

# The numbered elements represents the Flip-Flop for
# which both Async pins (set/reset) become active
# simultaneously. These are followed by the net-value
# pairs which results in above mentioned scenario.
#####

```

```
1. top.ff2
   top.ff1 (1)
   top.rst (1)
```

The `dft_ff_set_reset_active.rpt` file is generated in the
<*current-working-directory*>/spyglass_reports/dft/> directory.

dft_ff_set_reset_sequential_in_capture

The `dft_ff_set_reset_sequential_in_capture.rpt` file is generated by the [Async_02_capture](#) rule. This report contains a list of all flip-flops whose set/reset pins are driven by sequential elements/black box in capture mode.

A sample format of `dft_ff_set_reset_sequential_in_capture` report is shown below:

```
#####
Format :

# Listed below are the sequential elements and
# black boxes in the design that are present in the
# fanin cone of asynchronous flip flops. The file
# displays name of the sequential elements and all the
# names of the flops that are affected by them.
#####
current_design 'top'
  Flip-Flop 'top.ff1' affects :
    Clear pin of 1 flip-flops.
      top.ff2 (reset Signal : top.reset_seq)
```

The `dft_ff_set_reset_sequential_in_capture.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/>` directory.

dft_ff_set_reset_sequential_in_shift

The `dft_ff_set_reset_sequential_in_shift.rpt` file is generated by the [Async_02_shift](#) rule. This report contains a list of all flip-flops whose set/reset pins are driven by sequential elements/black box in shift mode.

A sample format of `dft_ff_set_reset_sequential_in_shift` report is shown below:

```
#####
Format :

# Listed below are the sequential elements and
# black boxes in the design that are present in the
```

```
# fanin cone of asynchronous flip flops. The file
# displays name of the sequential elements and all the #
# names of the flops that are affected by them.
#####
current_design 'top'
    Flip-Flop 'top.ff1' affects :
        Clear pin of 1 flip-flops.
            top.ff2(reset Signal : top.reset_seq)
```

The `dft_ff_set_reset_sequential_in_shift.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/>` directory.

dft_ff_X_source_for_tristate_enable

The `dft_ff_x_source_for_tristate_enable.rpt` file is generated by the [Tristate_10](#) rule. This report contains a list of all the flip-flops that could cause X on Tri Enable's.

A sample format of `dft_ff_X_source_for_tristate_enable` report is shown below:

```
#####
          Design Unit      : top
          Run-Flow         : (Null)
          Capture-cycle Start : 1
          Capture-cycle End  : 3
#####
```

Listed below are flops which combinationally drive the enables of tri-state enables. These are tri-stated enables that are 'uncontrollable' at some, possibly all, capture clock cycles in the range mentioned above. The cause for the uncontrollability, in the 'Nth' capture clock cycle, is the data from the flop listed under 'Depth = <N>'.

NOTE1 : If a flop is listed for 'Depth = <N>', then it is also a cause for 'Depth = <M>', where (M > N)

NOTE2 : For each flop the file & line where it was inferred are also reported. Also, the key [KEY = f_<some_number>] can be used to jump to other references to the flop in the file.

```
-----
CC1-SOURCES FOR FLOP: top.u_chain.ff2_reg [KEY = f_0]
(test.v,18)
```

```
    Depth = 2
-----
```

```
        D: top.u_chain.u_ffb01.Q2 [KEY = f_8]
```

```
CC1-SOURCES FOR FLOP: top.u_chain.u_ffb01 [KEY = f_8]
(test.v,12)
```

```
    Depth = 1
-----
```

```
        D1: top.u_chain.u_ffb01.Q1 [KEY = f_8]
```

```
        D2: top.u_chain.u_ffb01.Q2 [KEY = f_8]
```

```
CC1-SOURCES FOR FLOP: top.u_chain.u_ffb03 [KEY = f_9]
(test.v,23)
```

```
    Depth = 2
-----
```

```
        D1: top.u_chain.u_ffb01.Q2 [KEY = f_8]
```

```
    Depth = 3
-----
```

```
        D1: top.u_chain.u_ffb01.Q2 [KEY = f_8]
```

```
        D2: top.u_chain.u_ffb01.Q2 [KEY = f_8]
-----
```

The data below lists, for each flop, the causes of 'D-pin' uncontrollability. This uncontrollability would get transmitted to another flop, <N> sequential depth ahead, after <N> capture clock cycles. The data here should be 'cross-referenced' with the data above to get all the 'sources of uncontrollability' at the <Nth> capture clock cycle.

```
-----
BAD CONTROLLABILITY CAUSE(S) FOR FLOP:
```

```
    top.u_chain.u_ffb01 [KEY = f_8]          (test.v,12)
```

```
        D1:
```

```
            top.u_chain.u_ffb01.D1 [Undriven-Terminal] (X-Source)
```

```
D2:  
    top.u_chain.w1 [Hanging Net] (X-Source)
```

The `dft_ff_x_source_for_tristate_enable.rpt` file is generated in the
<*current-working-directory*>/spyglass_reports/dft/> directory.

dft_latch_enable

The dft_latch_enable.rpt file is generated by the *Info_latchMapping* rule. This report contains the details regarding info latch mapping.

A sample format of dft_latch_enable report is shown below:

```
#####
# Format :
# <>ABBREVIATIONS & DEFINITIONS OF TERMS :
# 1. LD_T_F      : Transparent as enable pin has non-X
# simulation value in capture
# 2. LD_T_CNT    : Transparent as enable pin is controllable
# to its active value
# 3. LD_T_SHDW   : Inferred as 'Shadow' latch
# 4. LD_T^off    : 'Off' state of test clock is forcing it
# transparent
# 5. LD_T^on     : Transparent as latch is getting free
# running test clock
# 6. LD_S_LKUP   : Lockup (source) latch
# 7. LD_D_LKUP   : Lockup (destination) latch
# 8. LD_NTP      : Non-transparent and non-lockup latch

# Column 1: S. No
# Column 2: Latch Transparency / Scannability Status (shift
# mode)
# Column 3: Latch Transparency / Scannability Status (capture
# mode)
# Column 4: Test clock reaching the enable pin (shift mode)
# Column 5: Test clock reaching the enable pin (capture mode)
# Column 6: Latch Name

#####

Latch status for design 'top':
[s.no] shift      capture  shift   capture   latch
         _status  _status  _clock  _clock   name
-----
[1] LD_T^off LD_T^off -- top.clk top.u_ld1.ld1_reg
```

```
[1] LD_T^off LD_T^off -- top.clk top.u_ld1.ld1_reg
[2] LD_NTP LD_T^off -- top.clk top.u_ld2.ld1_reg
[3] LD_T^off LD_T^off -- top.clk top.u_ld11.ld1_reg
[4] LD_NTP LD_T^off -- top.clk top.u_ld12.ld1_reg
[5] LD_NTP LD_NTP top.clk top.clk top.u_ld_3.ld1_reg
[6] LD_NTP LD_NTP top.clk top.clk top.u_ld_4.ld1_reg
```

#####

Latch status for design 'top2':

[s.no]	shift	capture	shift	capture	latch
	_status	_status	_clock	_clock	name
[1]	LD_T^off	LD_T^off	--	top2.clk	top2.u_ld1.ld1_reg
[2]	LD_NTP	LD_T^off	--	top2.clk	top2.u_ld2.ld1_reg
[3]	LD_T^off	LD_T^off	--	top2.clk	top2.u_ld11.ld1_reg
[4]	LD_NTP	LD_T^off	--	top2.clk	top2.u_ld12.ld1_reg
[5]	LD_NTP	LD_NTP	top2.clk	top2.clk	top2.u_ld_3.ld1_reg
[6]	LD_NTP	LD_NTP	top2.clk	top2.clk	top2.u_ld_4.ld1_reg

#####

The dft_latch_enable.rpt file is generated in the
 <current-working-directory>/spyglass_reports/dft/> directory.

dft_mandatory_sgdc

The dft_mandatory_sgdc.rpt file is generated by the [dftMandatory_Constraint_Check](#) rule. This report contains a list of all the missing mandatory constraints needed by a dft rule.

A sample format of dft_mandatory_sgdc report is shown below:

```
#####

# Purpose :
#   This file has been generated by DFT-Rule
#   'dft_Mandatory_Constraint_Check'. It has details of all
#   missing mandatory constraints, and the rules affected
#   by their absence.'Mandatory' constraints are essential
#   to that rule's checking.
#####

#####

# Format :
#   For each top-level module, and for each 'enabled'
#   rule, the 'mandatory' constraints required are listed.
#   Alongside the constraint name is a '['+' or a '['-' to
#   indicate whether that constraint was found or not.
#####
#####

                RULE                MANDATORY CONSTRAINT
#####

Current_Design : test
-----

Diagnose_testmode

                                test_mode[-]
-----
```

Info_stilFile

clock -testclock[+]
test_mode[-]

Info_memorywritedisable

memory_write_disable[-]

Info_memoryforce

test_mode[-]

BIST_03

test_mode[-]

Clock_17

clock -testclock[+]
test_mode[-]

Clock_22

test_mode[-]

Clock_24

clock -testclock[+]
test_mode[-]

RAM_01

shadow_ratio[-]

RAM_04

test_mode[-]

RAM_05	memory_write_pin[-]

RAM_06	memory_read_pin[-] memory_write_pin[-]

RAM_08	memory_write_pin[-]

Scan_07	test_mode[-]

Scan_11	scan_ratio[-]

Scan_23	module_bypass[-]

Soc_01	require_value[-]

Soc_02	require_path[-]

Soc_04	define_tag[-]

Soc_05	clock -testclock[+] test_mode[-]

Soc_01_Info	require_value[-]

Soc_02_Info	require_path[-]

TA_07	test_mode[-]

Tristate_04_capture	test_mode[-]

Tristate_04_shift	test_mode[-]

Tristate_07_capture	test_mode[-]

Tristate_07_shift	test_mode[-]

Tristate_08_capture	test_mode[-]

Tristate_08_shift	

test_mode[-]

The dft_mandatory_sgdc.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

dft_memory_report

This report provides information on all the memory instances in the design.
The following figure shows a sample of the report:

```

1 #####
2 #
3 # This file has been generated by SpyGlass:
4 #   Report Created by: meetu
5 #   Report Created on: Sun Jan 14 20:15:23 2018
6 #   Working Directory: /remote/us01home54/meetu/INFO_MEMORIES_TEST_LATEST/case02
7 #   Report Location : xx_yy_latest/test/test_dft_goal/spyglass_reports/dft/dft_memory_report.txt
8 #   SpyGlass Version : SpyGlass_vN-2018.09-Alpha
9 #   Policy Name      : dft(SpyGlass_vM-2017.03)
10 #   Comment         : Generated by rule Info_memories
11 #
12 #####
13 #####
14 #####
15 # This file has been generated by DFT rule 'Info_memories'.
16 # This report has two sections.
17 # - The first section contains the total memory count in the design
18 #   followed by the number of memory instances per module
19 # - The second section contains memory instance specific information
20 #   - shift_clock
21 #   - capture_clock
22 #   - functional clock(s) for all clock pins of all memory instances
23 #####
24
25 Total Memory Instance Count                : 3
26 Total Instance Count for memory module mem_single_port : 3
27
28
29 memory_instance
30 -instance_name                "top.mem1"
31 -master_name                  "mem_single_port"
32 -scanwrapped
33 -sgdc_memory
34
35 -clock_pin_i_name             "clk"
36 -clock_pin_i_shift_clock_name "tclk1"
37 -clock_pin_i_capture_clock_name "tclk1"
38 -clock_pin_i_fastest_functional_clock_name "fclk1"
39 -clock_pin_i_fastest_functional_clock_frequency 200.7
40
41 -clock_pin_i_func_clock_001_logical_name "tclk1"
42 -clock_pin_i_func_clock_001_root_name "tclk1"
43 -clock_pin_i_func_clock_001_frequency 200.4
44 -clock_pin_i_func_clock_002_logical_name "fclk1"
45 -clock_pin_i_func_clock_002_root_name "fclk1"
46 -clock_pin_i_func_clock_002_frequency 200.7
47
48 memory_instance
49 -instance_name                "top.mem2"
50 -master_name                  "mem_single_port"
51 -scanwrapped
52 -sgdc_memory
53
54 -clock_pin_i_name             "clk"
55 -clock_pin_i_shift_clock_name "_NO_CLOCK_"
56 -clock_pin_i_capture_clock_name "_NO_CLOCK_"
57
58 -clock_pin_i_func_clock_001_logical_name "NOT_SPECIFIED"
59 -clock_pin_i_func_clock_001_root_name "tclk2"

```

FIGURE 1. The dft_memory_report Report

dft_multibit_flipflops

The dftmax_ultra_configuration.rpt file is generated by the [Info_DftDebugData](#) rule. The following figure shows a sample dft_multibit_flipflops report.

```

#####
# Purpose :
# Report Type : Summarize all MultiBit Flip-flops in the

```

```

# design, their storage bits count and their number of
# instantiations.
#####

#####
# Design : top
#####

Number of Multi-bit Cell(s) present in the design top: 1
#++++++
MultiBit Cell Name  # of Storage Bits  # of Instantiations
                    per Cell
ff_2_RD             2                  2
#####
# End Design : top
#####

```

dftmax_ultra_configuration

The dftmax_ultra_configuration.rpt file is generated by the [Info_dftmax_configuration](#) rule. The following figure shows a sample *dftmax_ultra_configuration* report.

```
#####
# Begin data for design unit: test
#####
Number of Partitions: 2
Number of Top Level Pins: 20
OCC in Design: Yes
<> SI/SO ALLOCATION

Partition Name          | #DFP          | X-Density
| #SI                   | #SO           |
-----
my_partition1          | 29            | Low
| 6                     | 6             |
default_partition     | 11            | Low
| 3                     | 3             |

<> COMPRESSION CONFIGURATION METRICS

#SI      | #DFP      | Compression Ratio | #Internal Chains | Max Chain
Size | #External Cycles/Pattern | Codec Penalty Ratio
-----
6      | 29      | 0                | 0                | 0
| 0      |         |                  |                  |
3      | 11      | 0                | 0                | 0
| 0      |         |                  |                  |

<> PARTITION SIZES
Partition : my_partition1
BLOCK          | #FF
-----
mod_ff_i1     | 29

Partition : default_partition
BLOCK          | #FF
-----
mod_ff_i2     | 11

#FFs in external chain of other partitions
-----
external_my_partition1      3
#####
# End data for design unit: test
#####
```

FIGURE 2. dftmax_ultra_configuration

dftmax_ultra_configuration_command

The `dftmax_ultra_configuration_command.rpt` report is generated by the [Info_dftmax_configuration](#) rule. The following figure shows a sample `dftmax_ultra_configuration_command` report:

```
#####
# Begin data for design unit: test
#####

# enable streaming compression in global configuration
set_dft_configuration -streaming_compression enable

# define and configure user defined partitions
define_dft_partition my_partition1 -include [ list mod_ff_i1 ]
current_dft_partition my_partition1
set_scan_configuration -chain_count 6
set_streaming_compression_configuration -chain_count 20 -inputs
6 -outputs 6
define_dft_partition my_partition2 -include [ list ]
current_dft_partition my_partition2
set_scan_configuration -chain_count 0
set_streaming_compression_configuration -chain_count 0 -inputs
0 -outputs 0

# define and configure the default partitions
# the default partition picks up all remaining blocks
define_dft_partition default_partition -include [ list
mod_ff_i2 ]
current_dft_partition default_partition
set_scan_configuration -chain_count 3
set_streaming_compression_configuration -chain_count 7 -inputs
3 -outputs 3

#####
# End data for design unit: test
#####|
```

FIGURE 3. `dftmax_ultra_configuration_command`

dft_self_gating_logic_summary

The `dft_self_gating_logic_summary.rpt` file is generated by the `Info_self_gating_logic` rule. This report contains the summary of identified self-gating logic in the design.

The `dft_self_gating_logic_summary.rpt` file is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

The following is a sample `dft_self_gating_logic_summary` report:

```
#####
# Format :
#   The summary of the identified self-gating logic and metric
#   on the
#   proportion of the self-gated flops in the design is given.
#
#   Each self-gating logic consists of the following :
#   - A clock-gating cell (CGC)
#   - One or more FF clocked by same CGC
#   - Each FF contains XOR/XNOR logic between either Q & D or
#   Qbar & D pins
#   - All XOR/XNOR logic outputs are connected to CGC-EN pin
#   via OR-logic
#   (OR and XOR/XNOR tree can be a combination of other logic
#   gates)
#####

#####
# Begin data for design unit: RISC_CORE
#####
Percentage of FFs that are self-gated      : 93.6%
Number of FFs that are self-gated         : 250
Number of FFs in the design               : 267
Number of self-gating groups              : 23
Smallest number of FFs in a self-gating group : 1
Largest number of FFs in a self-gating group : 30
Average number of FFs in a self-gating group : 10

#####
# End data for design unit: RISC_CORE
#####|
```

FIGURE 4. `dft_self_gating_logic_summary`

dft_self_gating_ff

The dft_self_gating_ff.rpt file is generated by the Info_self_gating_logic rule. This report contains the list of FFs corresponding to each identified self-gating logic in the design.

The dft_self_gating_ff.rpt file is generated in the <cwd>/spyglass_reports/dft_dsm/ directory.

The following is a sample dft_self_gating_ff report:

```
#####
# Format :
#   Following details for each self-gating (SG) group in the
design is given:
#   - Name of SG GROUP ( same as the name of self-gated CGC
enable term )
#   - Total FF Count ( it is the count of self-gated flops
in the SG-GROUP )
#   - Name of all self-gated FFs present in the SG-GROUP
#
#   Each self-gating group consists of the following :
#   - A clock-gating cell (CGC)
#   - One or more FF clocked by same CGC
#   - Each FF contains XOR/XNOR logic between either Q & D or
Qbar & D pins
#   - All XOR/XNOR logic outputs are connected to CGC-EN pin
via OR-logic
#   (OR and XOR/XNOR tree can be a combination of other logic
gates)
#####

#####
# Begin data for design unit: test
#####
<> SG GROUP : test.cgc_bbox_il.sys_en
  Total FF Count : 4
    1. test.ff1_reg
    2. test.ff2_reg
    3. test.ff3_reg
    4. test.ff4_reg

#####
# End data for design unit: test
#####
```

FIGURE 5. dft_self_gating_ff

dft_self_gating_test_points

The dft_self_gating_test_points.rpt file is generated by the Info_self_gating_logic rule. This report contains the list of test-points corresponding to each identified self-gating logic in the design.

The dft_self_gating_test_points.rpt file is generated in the <cwd>/spyglass_reports/dft_dsm/ directory.

The following is a sample *dft_self_gating_test_points* report:

```

#####
# Format :
#   Following details for each self-gating (SG) group in the design is
#   given:
#   - Name of SG GROUP ( same as the name of self-gated CGC enable
#   term )
#   - List of test-points for SG-GROUP ( TPs are observe points on
#   the o/p of the
#
#                               XOR logic in the self-gating
#   logic)
#   - Number of non-self-gated FFs in fanin and fanout of self-gating
#   logic|
#
#   Each self-gating logic consists of the following :
#   - A clock-gating cell (CGC)
#   - One or more FF clocked by same CGC
#   - Each FF contains XOR/XNOR logic between either Q & D or Qbar &
#   D pins
#   - All XOR/XNOR logic outputs are connected to CGC-EN pin via OR-
#   logic
#   (OR and XOR/XNOR tree can be a combination of other logic gates)
#####

#####
# Begin data for design unit: test
#####

<> SG GROUP : test.cgc_bbox_il.sys_en
    1. test.xor1_out
    2. test.xor2_out
    3. test.xor3_out
    4. test.xor4_out
    FF Count in fanin  : 0
    FF Count in fanout : 0

#####
# End data for design unit: test
#####

```

FIGURE 6. dft_self_gating_test_points

no_fault

The no_fault.rpt file is generated by the *Info_noFault* rule. This report contains a lists all the ports and pins which are marked as "no fault" via user specified sgdc commands.

A sample format of no_fault report is shown below:

```
#####
# Purpose :

# List of all faults which are marked as 'no fault' due to
# presence of 'no_fault' sgdc command. These faults are not
# considered in fault coverage calculations

#####

#####
# Format :
# Design Name : design
# 'number of no-fault faults' <sgdc command>
#      Port | Terminal: Fault name
#####

#####
Design Name : "top"
'28' faults are marked as 'no_fault' due to constraint
'no_fault -name block1'
Terminal: test.inst1.ff1_reg (Master = RTL_FD).D
Terminal: test.inst1.ff1_reg (Master = RTL_FD).CP
Terminal: test.inst1.ff1_reg (Master = RTL_FD).Q
Terminal: test.inst1.ff2_reg (Master = RTL_FD).D
Terminal: test.inst1.ff2_reg (Master = RTL_FD).CP
Terminal: test.inst1.ff2_reg (Master = RTL_FD).Q
Terminal: test.inst1.rtlc_I5 (Master = RTL_OR).in1
Terminal: test.inst1.rtlc_I5 (Master = RTL_OR).in2
Terminal: test.inst1.rtlc_I5 (Master = RTL_OR).Z
Terminal: test.inst1.rtlc_I6 (Master = RTL_AND).in1
Terminal: test.inst1.rtlc_I6 (Master = RTL_AND).in2
```



```

Terminal: test.inst1.rtlc_I6 (Master = RTL_AND).Z
Terminal: test.inst1.rtlc_I8 (Master = RTL_NOT).A
Terminal: test.inst1.rtlc_I8 (Master = RTL_NOT).Z
'28' faults are marked as 'no_fault' due to constraint
'no_fault -name block2'
Terminal: test.inst2.ff1_reg (Master = RTL_FD).D
Terminal: test.inst2.ff1_reg (Master = RTL_FD).CP
Terminal: test.inst2.ff1_reg (Master = RTL_FD).Q
Terminal: test.inst2.ff2_reg (Master = RTL_FD).D
Terminal: test.inst2.ff2_reg (Master = RTL_FD).CP
Terminal: test.inst2.ff2_reg (Master = RTL_FD).Q
Terminal: test.inst2.rtlc_I5 (Master = RTL_OR).in1
Terminal: test.inst2.rtlc_I5 (Master = RTL_OR).in2
Terminal: test.inst2.rtlc_I5 (Master = RTL_OR).Z
Terminal: test.inst2.rtlc_I6 (Master = RTL_AND).in1
Terminal: test.inst2.rtlc_I6 (Master = RTL_AND).in2
Terminal: test.inst2.rtlc_I6 (Master = RTL_AND).Z
Terminal: test.inst2.rtlc_I8 (Master = RTL_NOT).A
Terminal: test.inst2.rtlc_I8 (Master = RTL_NOT).Z

```

SUMMARY:

```

'56' faults are marked as 'no_fault' for design "test"
#####

```

The no_fault.rpt file is generated in the
 <current-working-directory>/spyglass_reports/dft/> directory.

dft_optional_sgdc

The dft_optional_sgdc.rpt file is generated by the [dftOptional_Constraint_Check](#) rule. This report contains a list of all the missing optional constraints needed by a SpyGlass DFT ADV rule.

A sample format of dft_optional_sgdc report is shown below:

```

#####
Format :
# For each top-level module, and for each 'enabled'
# rule, the 'optional' constraints required are listed.

```

```

# Alongside the constraint name is a '['+' or a '['-'
# to indicate whether that constraint was found or not.
#####
#####
RULE                                OPTIONAL CONSTRAINT
#####
Current_Design : izar
-----
Info_uncontrollable

                                force_scan[-]
                                module_bypass[-]
                                force_no_scan[-]
                                scan_type[-]
                                force_ta[-]
                                scan_wrap[+]
                                test_point[-]
                                clock -testclock[+]
                                test_mode[+]
-----
Info_unobservable

                                force_scan[-]
                                module_bypass[-]
                                force_no_scan [-]
                                scan_type[-]
                                force_ta[-]
                                scan_wrap[+]
                                test_point[-]
                                clock -testclock[+]
                                test_mode[+]
-----
Info_untestable

                                force_scan[-]
                                module_bypass[-]
                                force_no_scan [-]
                                scan_type[-]
                                force_ta[-]
                                scan_wrap[+]
                                test_point[-]

```

```

                                clock -testclock[+]
-----
Info_coverage
                                force_scan[-]
                                module_bypass[-]
                                force_no_scan [-]
                                scan_type[-]
                                force_ta[-]
                                scan_wrap[+]
                                test_point[-]
                                clock -testclock[+]
                                test_mode[+]

```

The `dft_optional_sgdc.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/>` directory.

dft_summary

The `dft_summary.rpt` file is generated by the `dftAutoFix` parameter. This report contains a summary of autofix performed by the SpyGlass DFT ADV product.

A sample format of the `dft_summary` report is shown below:

```

#####
AutoFix done details
#####

Modified RTL Directory           : <Path of modified rtl>

Total Number of modified files  : 1

Total Number of modified modules : 1

Total Number of created files   : 0

Total Number of created modules : 0

```

The `dft_summary.rpt` file is generated in the
`<current_working_directory>/spyglass_reports/rme/>` directory.

dft_tristate

The dft_tristate.rpt file is generated by the [Tristate_13](#) rule. This report contains a list of all the tristate buses in the circuit.

A sample format of dft_tristate report is shown below:

```
#####
Format :

# Module name followed by 'triBus' constraints for all
# the tristate buses are tabulated
#####
current_design 'mytop'
    triBus -name mytop.t1 -busWidth 2
    Instances on this bus are:
        mytop.rtlc_I36
        mytop.rtlc_I40
#####
# SUMMARY :
# Bus width = 2 has 1 bus(es).
#####current
t_design 'mytop2'
    triBus -name mytop2.o2 -busWidth 1
    Instances on this bus are:
        mytop2.rtlc_I27
    triBus -name mytop2.o1 -busWidth 1
    Instances on this bus are:
        mytop2.rtlc_I23
#####
SUMMARY :
# Bus width = 1 has 2 bus(es).
```

The dft_tristate.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

dft_initialized_ffs

The `dft_initialized_ffs.rpt` file is generated by the `BIST_03` rule. This report contains a list of all the initialized flip-flops after an initialization sequence is applied.

A sample format of `dft_initialized_ffs` report is shown below:

```
#####
# Format :
# Flip Flop name is given.
# If the flip flop is initialized, the reason
# is also given.
#####
# Listed below are flip-flops present in the design
# 'izar' which are in known (non-X) state after initialization.
  Flipflop(s)
  izar.kernel_test.kernel_wrap.wrap_rtck.wrapper_1_out'
  is(are) initialized (REASON: D pin has known value(0))
  Flipflop(s)
  'izar.kernel_test.kernel_wrap.wrap_ipt_test_debug_tdo.wrapper_1_out' is(are) initialized (REASON: D pin has known value(0))
  Flipflop(s)
  'izar.kernel_test.kernel_wrap.wrap_ipt_test_membist_tdo.wrapper_1_out' is(are) initialized (REASON: D pin has known value(0))
  Flipflop(s)
  'izar.kernel_test.kernel_wrap.wrap_ipt_test_membist_done.wrapper_1_out' is(are) initialized (REASON: D pin has known value(0))
  Flipflop(s)
  'izar.kernel_test.kernel_wrap.wrap_ipt_test_membist_fail_0.wrapper_1_out' is(are) initialized (REASON: D pin has known value(0))
  Flipflop(s)
  'izar.kernel_test.kernel_wrap.wrap_ipt_test_membist_fail_1.wrapper_1_out' is(are) initialized (REASON: D pin has known value(0))
```

The `dft_initialized_ffs.rpt` file is generated in the
`<current-working-directory>/spyglass_reports/dft/>` directory.

forced_scan

The *forced_scan* report is generated by the *Info_forcedScan* rule.

This report lists all the flip-flops and latches that are marked scannable using the *force_scan* constraint.

The following is a sample *forced_scan* report:

```
#####
# Format :
#   Design Name : design
#   'number of forced-scan objects' <sgdc command>
#   FF | Latch | BB: Object name
#####

#####
  Design Name : "top"

  '3' objects are marked as 'force_scan' due to constraint
'force_scan -clock_control clk01'
    FLOP: top.u_B_1.u_A_1.ff1_reg
    FLOP: top.u_B_1.u_A_1.xbist_ff_reg
    FLOP: top.u_B_1.u_A_1.BIST_reg

  '3' objects are marked as 'force_scan' due to constraint
'force_scan -reset_control rst11'
    FLOP: top.u_B_1.u_A_2.ff1_reg
    FLOP: top.u_B_1.u_A_2.xbist_ff_reg
    FLOP: top.u_B_1.u_A_2.BIST_reg

  '3' objects are marked as 'force_scan' due to constraint
'force_scan -set_control set21'
    FLOP: top.u_B_2.u_A_1.ff1_reg
    FLOP: top.u_B_2.u_A_1.xbist_ff_reg
    FLOP: top.u_B_2.u_A_1.BIST_reg

  '4' objects are marked as 'force_scan' due to constraint
'force_scan -register_suffix ff'
    FLOP: top.u_B_1.u_A_1.xbist_ff_reg
    FLOP: top.u_B_1.u_A_2.xbist_ff_reg
    FLOP: top.u_B_2.u_A_1.xbist_ff_reg
    FLOP: top.u_B_2.u_A_2.xbist_ff_reg

SUMMARY:
  '13' objects are marked as 'force_scan' for design "top"
#####
```

FIGURE 7. The forced_scan Report

inferred_no_scan_ffs

The `inferred_no_scan_ffs.rpt` file is generated by the [Info_inferredNoScan](#) rule. This report contains a list of all flip-flops that are inferred as noscan by SpyGlass DFT ADV. These flip-flops can be inferred as scan by using [force_scan](#) constraint.

A sample format of `inferred_no_scan_ffs` report is shown below:

```
#####
# Purpose :
# It lists all flip-flops which are inferred as no_scan by Spyglass DFT.
# Reasons for no_scan inference:
# 1. Flip-flop output has 'clock-testclock' constraint specified
# 2. Flip-flop output has 'test_mode' constraint specified
# 3. Flip-flop output has non-X test_mode value through sequential propagation
#    and parameter 'dft_infer_sequential_propagation_as_no_scan' is 'on'
# 4. Flip-flop is not on a properly stitched scan chain when design state is post_scan_stitched
#####

#####
# Format :
# Listed below are the flip-flops that should be declared as force_no_scan
#####

current_design 'top'
force_no_scan -name top_ff_ns_stbl #Inferred 'no_scan' by Spyglass-DFT for further analysis as this flip-flop output gets a non-X value in testmode through sequential propagation
force_no_scan -name top_ff_ns_unstbl #Inferred 'no_scan' by Spyglass-DFT for further analysis as this flip-flop output gets a non-X('0') (though unstable during scan/shift operation) in testmode through sequential propagation
#####

#SUMMARY:
# Total 2 flip-flops should be declared as 'force_no_scan' in design 'top'.
#####
```

FIGURE 8. The `inferred_no_scan_ffs` Report

The `inferred_no_scan_ffs.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/>` directory.

dft_x_source_report

The `dft_x_source_report.rpt` file is generated by the [Info_x_sources](#) rule. This report lists all the x-sources sorted on basis of impacted capture points. The following is a sample `dft_x_source_report` report:

```
#####
# Purpose :
# Reports all X-Sources sorted on basis of impacted capture points
#####

#####
# Format :
# Rank : Rank of x-source
# Impacted Capture Point Count : Num of capture points capturing a x-source
# X-Source Obs Prob : Probability of x-source being observable
# Type : Type of x-source
# X-Source Name : Hierarchical name of x-source
#####

Design Name : "izar"
-----

-----+-----+-----+-----+-----
Rank | Impacted Capture | X-Source | Type | X-Source Name
     | Point Count      | Obs Prob |      |
-----+-----+-----+-----+-----
  1 |          1 | 0.000 | HN | izar.izar_scan_out[7]
  2 |          1 | 0.000 | HN | izar.izar_scan_out[15]
  3 |          1 | 0.000 | HN | izar.izar_scan_out[14]
  4 |          1 | 0.000 | HN | izar.izar_scan_out[13]
  5 |          1 | 0.000 | HN | izar.izar_scan_out[12]
  6 |          1 | 0.000 | HN | izar.izar_scan_out[11]
  7 |          1 | 0.000 | HN | izar.izar_scan_out[10]
  8 |          1 | 0.000 | HN | izar.izar_scan_out[9]
  9 |          1 | 0.000 | HN | izar.izar_scan_out[8]
 10 |          1 | 0.000 | HN | izar.kernel_test.izar_wrap_scan_out[0]
 11 |          1 | 0.000 | HN | izar.izar_scan_out[6]
 12 |          1 | 0.000 | HN | izar.izar_scan_out[5]
```

FIGURE 9. The `dft_x_source_report` Report

The `dft_x_source_report.rpt` file is generated in the

`<current-working-directory>/spyglass_reports/dft/` directory.

dft_x_capture_report

The `dft_x_capture_report.rpt` file is generated by the [Info_x_sources](#) rule. This report lists all X-Capture points sorted on basis of X-Capture probability.

The following is a sample `dft_x_capture_report` report:

```
#####
# Purpose :
# Reports all X-Capture points sorted on basis of X-Capture probability
#####
# Format :
# Rank          : Rank of X-Capture Node
# X-Capture Probability : Probability of a X being captured by x-capturing node
# Captured X-Source Count : X-Sources being captured by capture node
# Name          : Hierarchical name of X-capture node
#####

Design Name : "izar"
-----+-----+-----+-----
Rank | X-Capture | Captured | X-Capture
   | Probability | X-Source Count | Node Name
-----+-----+-----+-----
 1 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[7] .D
 2 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[15] .D
 3 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[14] .D
 4 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[13] .D
 5 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[12] .D
 6 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[11] .D
 7 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[10] .D
 8 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[9] .D
 9 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[8] .D
10 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[16] .D
11 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[6] .D
12 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[5] .D
13 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[4] .D
14 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[3] .D
15 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[2] .D
16 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[1] .D
17 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[0] .D
18 | 1.000 | 1 | izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.\q_reg[24] .D
```

FIGURE 10. The `dft_x_capture_report` Report

The `dft_x_capture_report.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/<` directory.

no_scan

The no_scan report is generated by the [Info_noScan](#) rule. This report lists all the flip-flops, latches, black boxes, or memories that are marked no_scan using the [force_no_scan](#) constraint.

The following is a sample no_scan report:

```
#####
# Format :
#   Design Name : design
#   'number of no-scan objects' <sgdc command>
#   FF | Latch | BB: Object name
#####

#####
Design Name : "top"
'3' objects are marked as 'force_no_scan' due to constraint
'force_no_scan -clock_control clk01'
  FLOP: top.u_B_1.u_A_1.ff1_reg
  FLOP: top.u_B_1.u_A_1.xbist_ff_reg
  FLOP: top.u_B_1.u_A_1.BIST_reg

'3' objects are marked as 'force_no_scan' due to constraint
'force_no_scan -reset_control rst11'
  FLOP: top.u_B_1.u_A_2.ff1_reg
  FLOP: top.u_B_1.u_A_2.xbist_ff_reg
  FLOP: top.u_B_1.u_A_2.BIST_reg

'3' objects are marked as 'force_no_scan' due to constraint
'force_no_scan -set_control set21'|
  FLOP: top.u_B_2.u_A_1.ff1_reg
  FLOP: top.u_B_2.u_A_1.xbist_ff_reg
  FLOP: top.u_B_2.u_A_1.BIST_reg

'4' objects are marked as 'force_no_scan' due to constraint
'force_no_scan -register_suffix ff'
  FLOP: top.u_B_1.u_A_1.xbist_ff_reg
  FLOP: top.u_B_1.u_A_2.xbist_ff_reg
  FLOP: top.u_B_2.u_A_1.xbist_ff_reg
  FLOP: top.u_B_2.u_A_2.xbist_ff_reg

SUMMARY:
'13' objects are marked as 'force_no_scan' for design "top"
#####
```

FIGURE 11. The no_scan Report

observable_ffs

The observable_ffs report.rpt file is generated by the [Scan_30](#) rule. This report lists all observable flip-flops.

A sample format of observable_ffs report is shown below:

```
#####

# Format :
#   The name of the top module, followed by the number
#   of observable flip-flops and their individual
#   details.
#####

Observable flip-flops for 'top1' module

Total number of observable flip-flops: 4

Flip flop: top1.U0_CFD3EX1H
Flip flop: top1.U1_CFD3EX1H
Flip flop: top1.U2_CFD3EX1H
Flip flop: top1.U3_CFD3EX1H
```

The observable_ffs.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

potentially_detected_faults

The potentially_detected_faults.rpt file is generated by the [Info_potDetectable](#) rule. This report contains a list of potentially detectable faults.

A sample format of potentially_detected_faults report is shown below:

```
#####

# Format :
#   The potentially detectable faults of the pins has
#   been listed.(e.g. top.mod.u.A => S@1 means that
#   stuck-at-one
```

```

# fault is potentially detectable at this pin.)
#####
#      MODULE : top
#####

top.op2 => S@10
top.in2 => S@10
top.rst => S@10
top.ff2_reg.D => S@10
top.ff2_reg.CP => S@10
top.ff2_reg.PRE => S@10
top.ff2_reg.CLR => S@10
top.ff2_reg.Q => S@10
top.rtlc_I2.in1 => S@10
top.rtlc_I2.in2 => S@10
top.rtlc_I2.Z => S@10
top.rtlc_I5.A => S@10
top.rtlc_I5.Z => S@10

```

The potentially_detected_faults.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

scan_chain

The scan_chain.rpt file is generated by the [Scan_24](#) and [Info_schain](#) rule. This report contains a list of all flip-flops that are or are not a part of some scan chains.

A sample format of scan_chain report is shown below:

```

#####
# Format :
# Section 1: Scan chain details in tabular format.
# This is followed by the summary of section 1.
# Section 2: Flip-flops detail that are not part of
# any scan chain. This is followed by the summary
# of section 2.
#####
# In section 1: For each scan chain, ordered list of

```

scan cells are reported with the following information.

```
# 1. cell_order      : cell index number
                        where '0' is the scan cell
                        closest to the scan-out(SO) pin
# 2. scanchain_ID    : scan chain id in which the scan
                        cell resides
# 3. cell_type       : type of cell(latch/port/flop
                        /black-box/memory_instance)
# 4. shift_clock     : clock for each scan
                        cell(shift_clock)
# 5. clock_polarity  : polarity of the global
                        clock(shift_clock),rising
                        edge(+) and falling edge(-)
# 6. hier_name       : hierarchical name of the cell
# 7. lib_cell_name   : Library name of the cell
# 8. scanout         : scan output port of the scan
                        cell(Q/QB/dedicated_so_pin)
```

In section 2: Flip-flops which are not part of any valid scan chain are reported with the following information.

```
# 1. cell_no         : serial index number of
                        the cell
# 2. clock           : source clock driving
                        the cell
# 3. clock_polarity  : clock polarity of the
                        cell rising edge(+)
                        and falling edge(-)
# 4. type            : scannability status of
                        the cell
# 5. hier_name       : hierarchical name of
                        the cell
# 6. lib_cell_name   : Library cell name of
                        the cell
```

```
-----
# Section 1 of top module 'top1': BEGIN
scanchain_id      : chain_1
scanout           : top1.sout
```



```

scanin          : top1.sin
clock domains   : 1 (top1.tclk)
scanchain length : 4

```

cell_order	scan_chain_id	cell_type	shift_clock
0000	chain_1	<flop>	top1.tclk
0001	chain_1	<flop>	top1.tclk
0002	chain_1	<flop>	top1.tclk
0003	chain_1	<flop>	top1.tclk

clock_polarity	hier_name	library_cell_name	scanout
(+)	top1.U3_CFD3EX1H	CFD3EX1H	Q
(+)	top1.U3_CFD3EX1H	CFD3EX1H	Q
(+)	top1.U3_CFD3EX1H	CFD3EX1H	Q
(+)	top1.U3_CFD3EX1H	CFD3EX1H	Q

```

#####
# Summary :
#   Top module name           : 'top1'
#   Scan chain count          : 1
#   Maximum chain length      : 4
#   Minimum chain length      : 4
#   Total Flip flop count     : 5527
#     Scannable Flip flop count : 55400(100.00 %)
#       On valid scan chains    : 30101(54.44 %)
#       Outside valid scan chains : 25299(45.56 %)
#     Unscannable Flip flop count : 100(0.00 %)
#     No-Scan Flip flop count    : 20(0.00 %)
#       Forced no_scan Flip flop count: 1(0.00 %)
#       Inferred no_scan Flip flop count : 19(0.00 %)
#       test_mode | clock -testclock constraint : 11(0.00 %)
#       Outside scan chain      : 8(0.00 %)
#     Synthesis Redundant (SR) flip-flop count : 7(0... %)
#####
# Section 1: END

```

```
# Section 2 of top module 'top1': BEGIN
#####
# Summary :
#     All flip-flops(4) are part of valid scanchains
#####
# Section 2: END
```

The scan_chain.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

scan_wrap

The scan_wrap.rpt file is generated by [dumpBlackBox](#) and [Info_scanwrap](#) rules. This report contains a scanwrap list of all boxes and their related information.

scan_wrap Report for the Info_scanwrap Rule

A sample format of scan_wrap report generated by the [Info_scanwrap](#) rule is shown below:

```
#####
# Purpose :
#     The report lists all the black-boxes present in design.
#     The report is divided into four sections.
#     Section I:
#     Reports all the black-boxes for which scan_wrap is specified and
#     enabled
#     Section II:
#     Reports all the memory instances for which scan_wrap is inferred
#     when -dft_set_scan_wrap_on_memory=on. Assumption:
#     1.All the memories will be BIST (Built in self test) inserted.
#     Section III:
#     Reports all the black-boxes for which scan_wrap is not required
#     because:
#     1.All of their input nets are observable.
#     2.All of their output pins are blocked in test_mode so that
```

```

# they do not drive any downward logic.
# Section IV:
# Reports all the potential candidates for scan_wrap.
# Each such black-box module has an additional information about
# controllabilty, observability and fault improvement on scan-
# wrapping that black-box module.

# Improvement achieved by scan-wrapping all candidates is reported
# in summary section.
# NOTE : Specifying a black-box as scan_wrap makes the outputs
# controllable and inputs observable.
#####

#####
# Format :
# SUMMARY:
# 'M' black-boxes found in design 'designName' for which scan_wrap is
# specified and enabled
# 'N' memory instances found in design 'designName' for which
# scan_wrap is inferred
# 'O' black-boxes found in design 'designName' for which scan_wrap is
# not required
# 'P' potential scan_wrap candidates found in design 'designName'
# All candidates scan-wrapped # Control Gain : XX ,Observe Gain : XX,
# Fault Gain : XX
# Section I: scan_wrap constraint defined and enabled
# #scan_wrap -name blackBoxModuleName
# #instanceName

# Section II: inferred scan_wrap on memories
# [-dft_set_scan_wrap_on_memory on|off]
# #scan_wrap -name moduleName
# #instanceName

# Section III: scan_wrap not required
# Module Name: blackBoxModuleName
# #instanceName

```

```

# Section IV: potential candidates for scan_wrap
# scan_wrap -name blackBoxModuleName # Control Gain : XX, Observe
# Gain : XX, Fault Gain :XX
# #instanceName

# The gain in controllability, observability and fault testability
# will be obtained if only that module is placed in a scan-wrapper.
#####

#####
current_design "top"
#SUMMARY:
# 2 black boxes found in design 'top' for which scan_wrap is specified
and enabled
# 3 memory instances found in design 'top' for which scan_wrap is
# inferred
# No black box found in design 'top' for which scan_wrap is not required
# 1 scan_wrap candidate found in design 'top'
# All candidate scan-wrapped # Control Gain : 8 , Observe Gain : 2 ,
# Fault Gain : 12

#Section I: scan_wrap constraint defined and enabled
#scan_wrap -name "memory1"
#top.m1
#top.m2

#Section II : inferred scan_wrap on memories
# [-dft_set_scan_wrap_on_memory on]
#scan_wrap -name "memory2"
#"top.mm1"
#scan_wrap -name "art_hssp_64x16"
#top.mem1
#top.mem2

#Section III: scan_wrap not required
# No black box found for which scan_wrap is not required

```

```
#Section IV: potential candidates for scan_wrap
  scan_wrap -name "bbox" # Control Gain : 8 , Observe Gain : 2 , Fault
  Gain : 12 "top.bb1"
#####
```

scan_wrap Report for the dumpBlackBox Rule

A sample format of the scan_wrap report generated by the [dumpBlackBox](#) rule is shown below:

```
#####
# Format :
#   The module name followed by the 'scanwrap'
#   constraints for the blackboxes are given.
#   Finally, the total number of black boxes in the
#   design is given.
#####
current_design top1
#####
# scanwrapped black box(es):
#   master mybb -- instance: "top1.U1"
#   master mybb -- instance: "top1.U2"
#   master mybb1 -- instance: "top1.U4"
#####
# SUMMARY :
#   No blackbox, without scanwrap, found in design
#   'top1'.
#   3 blackbox(es), with scanwrap, found in design
#   'top1'.
```

NOTE: The [dumpBlackBox](#) rule will be deprecated in a future release. Therefore, please use the [Info_scanwrap](#) rule instead of the [dumpBlackBox](#) rule.

The scan_wrap.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

soc_06_rpt

The soc_06.rpt file is generated by the *Soc_06* rule. This report lists the hierarchical name of all the instances appended with a pin name.

A sample format of soc_06_rpt report is shown below:

```
#####
# Format :
# Module - <module_name>
# <instance_name>.<pin_name> --- <DIRECTION>
# <instance_name>
#####
Module - mxx_zz (under design: top)
    top.u3
    top.u2
    top.u1
Module - myy_cc (under design: top)
    top.u33.in2----- INPUT
    top.u33.in1----- INPUT
    top.u33.op----- OUTPUT
    top.u22.in2----- INPUT
    top.u22.in1----- INPUT
    top.u22.op----- OUTPUT
    top.u11.in2----- INPUT
    top.u11.in1----- INPUT
    top.u11.op----- OUTPUT
```

The soc_06.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

soft_error_metrics

The soft_error_metrics.rpt file is generated by the *Info_soft_error_generation* rule. The soft_error_metrics report lists Soft Error Metrics defined in the ISO-26262 Standard.

A sample soft_error_metrics report is shown below:

```

#####
# Purpose :
#   Soft Error Metrics Report.
#   Reports Soft Error Metrics defined in ISO-26262 Standard.
#####

#####
# Format :
#   SPFM number is calculated by using below formula.
#           Sum(lambda_reg[i] x (1 - DC[i]) x (1 - probability_safe[i]))
#   SPFM = 1 - -----
#           Sum(lambda_reg[i])
#
#   LFM number is calculated by using below formula.
#           Sum(lambda_sm[i] x (1 - DCL[i]))
#   LFM = 1 - -----
#           Sum(lambda_reg[i] + lambda_sm[i] - lambda_reg[i] x (1 - DC[i]) x (1 - probability_safe[i]))
#
#   PMHF number is calculated by using below formula.
#   PMHF = Sum(lambda_reg[i] x (1 - DC[i]) x (1 - probability_safe[i])) + Sum(lambda_sm[i] x (1 - DCL[i]))
#####

#####
# Parameter values :
#   'ser_control_sequential_depth'      : 5
#   'ser_observe_sequential_depth'     : 5
#   'ser_observe_nsr_initial_value'    : 0.5
#   'ser_propagation_difference_threshold: 5
#####

#####
# Design information :
#   Top module : 'top'
#####

SPFM = 0.711250
LFM = 0.178297
PMHF = 4.849022

```

The `soft_error_metrics.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/` directory.

soft_error_registers_spfm

The `soft_error_registers_spfm.rpt` file is generated by the `Info_soft_error_generation` rule. The `soft_error_registers_spfm` report lists the soft error metric contribution to ISO-26262 Standard of highest contributing registers. Lower than expected SPFM value can be corrected by analyzing the `soft_error_registers_spfm` report.

A sample `soft_error_registers_spfm` report is shown below:

Index	Safety	SPFMContribution	lambda_reg	DC	Probability	Contribution %	Cumulative- Contribution %	Module Name	Register
1	N	0.010000	0.010000	0.000000	1.000000	0.015552	0.015552	RTL_FDCE	izar.kernel.arm7
2	N	0.010000	0.010000	0.000000	1.000000	0.015552	0.031104	RTL_FDCE	izar.kernel.arm7
3	N	0.010000	0.010000	0.000000	1.000000	0.015552	0.046656	RTL_FDCE	izar.kernel.arm7
4	N	0.010000	0.010000	0.000000	1.000000	0.015552	0.062208	RTL_FDCE	izar.kernel.arm7

The `soft_error_registers_spfm.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/` directory.

stil_file

The `stil_file.rpt` file is generated by the [Info_stilFile](#) rule. This report is a file containing STIL constructs.

A sample format of `stil_file` report is shown below:

```
STIL 1.0 {
    Design P2000.9;
}

Header {
    Title "SpyGlass - STIL output";
    DATE "Tue Apr 27 16:46:22 2010";
    History {
    }
}

Signals {
    "ipg_hard_async_reset_b" In;
    "ipg_clk" In;
    "ips_module_en[0..5]" In;
    "ips_rwb" In;
    "ips_byte_7_0" In;
    "ips_byte_15_8" In;
    "ips_byte_23_16" In;
}
```



```

"ips_byte_31_24" In;
"ips_addr[2..11]" In;
"ips_wdata[0..31]" In;
"clk32k" In;
"ipp_ind_refclk" In;
"ipp_ind_bt6" In;
"ipp_ind_bt2" In;
"ipp_ind_bt1" In;
"ssi_fsync" In;
"ssi_clk" In;
"ssi_rx" In;
"a7s_dbgreq" In;
"a7s_dbgen" In;

```

The `stil_file.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/>` directory.

stuck_at_faults

The `stuck_at_faults.rpt` file is generated by the [Info_coverage](#) rule. This file is basically a text report for all stuck at faults.

NOTE: *Ensure that the `dftGenerateStuckAtFaultReport` parameter is set to a value other than none to enable the generation of the `stuck_at_faults` report.*

A sample format of `stuck_at_faults` report is shown below:

```

#####
# Purpose :
#   Report type           : Stuck-At coverage
#   Stuck-At faults at ports and terminals.
#####

```

<>ABBREVIATIONS & DEFINITIONS OF TERMS :

1. hierarchical instance: An instance which has other hierarchical instances and/or basic cells within it.

2. DT fault: Faults which are testable AND detectable. It is possible that a fault is testable, but not detectable.
3. !DT/ND fault: A fault which was deemed testable but found to be undetectable. (Note that faults classified as TI or BL are not included here.) (Please see Info_coverage / Info_undetectedCause violations).
 - ND_PIO : PRIMARY_INPUT_OUTPUT
 - ND_NSF : NON_SCAN_FF
 - ND_BB : BLACK_BOX
 - ND_LT : NOT_TRANSPARENT_LATCH
 - ND_CL : COMBINATIONAL_LOOP
 - ND_TM : TEST_MODE
 - ND_HN : UNDRIVEN_PIN_NET
 - ND_TS : TRI_STATE
 - ND_FTA : FORCE_TA
 - ND_FNS : FORCE_NO_SCAN
4. UT fault: A fault which cannot be detected because it has a non-X simulation value. (Note that faults with nonX sim value in PG mode itself are classified as TI faults.)
5. UU fault: An unused fault. Fault at a terminal becomes UU if that terminal cannot be reached from the output ports. They are ignored because they cannot affect system operation. By default, latches / blackboxes, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking is set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.
6. SyRd fault: Faults at terminals which are Sy[= synthesis]Rd[= redundant]. These are terminals which are likely to be removed by

an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis. These are of the following types:

- a) Pins which are tied high/low,
- b) Pins blocked in power ground mode, and
- c) Unconnected combinational logic - all logic combinationaly connected to nodes that do not fanout to primary port. SyRd faults are reported iff fault analysis is done for an RTL design. If design is a netlist, synthesis is already done and the question of synthesis redundancy is considered invalid.

7. LR fault: These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.

8. PT fault: Faults in which difference between the good circuit result and bad circuit result for a particular fault produces any of the following are called Potential

Detected Faults:

Good Circuit Result	----	Bad Circuit Result
0		z
0		x
1		z
1		x

9. TI/BL fault: This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets. If a net is tied to 0, S@0 is TIED. S@1 may or may not be detectable. Similarly, a net tied to 1 will make S@1 TIED. An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked. For example, an

input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

$$\text{Fault-Coverage \%} = \frac{\text{Detectable Fault Count} + ((\text{PT}) * \text{POSDETECT_credit})}{(2 * \text{terminalCount}) - \text{SyRd_faultCount}} \times 100$$

$$\text{Test-Coverage \%} = \frac{\text{Detectable Fault Count} + ((\text{PT}) * \text{POSDETECT_credit})}{(2 * \text{terminalCount}) - \text{SyRd_faultCount} - \text{TI} - \text{BL} - \text{UUCount} - (\text{ATPG_credit} * \text{UTCCount}) - \text{LR}} \times 100$$

<>For top module the summary has been reported in three columns as follows:

- Column 1: Fault and Test coverage data due to top-level ports and internal design.
- Column 2: Fault and Test coverage data due to top-level ports only.
- Column 3: Fault and Test coverage data due to internal design only.

```
#####
# Design information :
#   Top module           : 'top1'
#   Flip-flop count      : 0
#   Testclock count      : 0
#####
```

<>TOP MODULE SUMMARY for 'top1'

Fault Heads	Total	Ports	Internal
Total faults	166	12	154
SyRd faults	0	0	0
Faults Considered	166	12	154
Un-Used (UU)	6	2	4
Tied (TI)	0	0	0
Blocked (BL)	0	0	0
Logical Redundant faults (LR)	0	0	0
Un-Testable (UT)	0	0	0
Detectable (DT)	40	6	34
Un-Detectable (!DT)	120	4	116
Potential Detectable (PT)	0	0	0
ATPG_credit	0.00	0.00	0.00
POSDTECT_credit	0.00	0.00	0.00
Fault-coverage(in %)	24.1	50.0	22.1
Test-coverage(in %)	25.0	60.0	22.7

fault type	status	cell-type	hier/pin/name
-----	-----	-----	-----

<>Listing 'all' faults for all ports

s/0	DT	primary_input	in1
s/1	DT	primary_input	in1
s/0	DT	primary_input	in2
.			
.			

<>Listing 'all' faults for all instantiation of module 'level2'

<>Listing 'all' faults for instance 'top1.lv11b.lv12b' of module 'level2'

s/0	DT	buffer	top1.lv11b.lv12b.rtlc_I1.A
s/1	DT	buffer	top1.lv11b.lv12b.rtlc_I1.A
.			

```

.

<>Listing 'all' faults for instance 'top1.lv11b.lv12a' of
module 'level2'
s/0          DT          buffer  top1.lv11b.lv12a.rtlc_I1.A
s/1          DT          buffer  top1.lv11b.lv12a.rtlc_I1.A
.
.

<>Listing 'all' faults for instance 'top1.lv11a.lv12b.lv13b.lv13leafa'
s/0          ND_BB      blackbox
top1.lv11a.lv12b.lv13b.lv13leafa.in
s/1          ND_BB      blackbox
top1.lv11a.lv12b.lv13b.lv13leafa.in
The stuck_at_faults.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

```

stuck_at_coverage

The stuck_at_coverage.rpt file is generated by the [Info_coverage](#) rule. This report contains the summary and details of stuck at fault numbers.

To generate the *stuck_at_coverage* report, set the value of the [dftGenerateStuckAtFaultReport](#) parameter to a value other than none.

A sample format of stuck_at_coverage report is shown below:

```

#####
# Format :
# The definition of various kinds of faults is given.
# This is followed by a tabular information of the
# count of faults associated with each module in the
# design.
#####
<>ABBREVIATIONS & DEFINITIONS OF TERMS :
1. hierarchical instance : an instance which has other
                           hierarchical instances and/or
                           basic cells within it.
2. DT fault              : Faults which are testable AND detectable.

```

- It is possible that a fault is testable, but not detectable.
3. !DT fault : A fault which was deemed testable but found to be undetectable. (Note that faults classified as TI or BL are not included here.) (Please see Info_coverage / Info_undetectedCause violations).
 4. UT fault : A fault which cannot be detected because it has a non-X simulation value. (Note that faults with nonX sim value in PG mode itself are classified as TI faults.)
 5. UU fault : An unused fault. Fault at a terminal becomes UU if that terminal cannot be reached from the output ports. They are ignored because they cannot affect system operation. By default, latches / blackboxes, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.
 6. SyRd fault : Faults at terminals which are Sy[=synthesis]Rd[= redundant]. These are terminals which are likely to be removed by an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis. These are of the following types:
 - a) Pins which are tied high/low,
 - b) Pins blocked in power ground mode, and
 - c) Unconnected combinational logic - all logic combinationaly connected to nodes that do not fanout to primary port.SyRd faults are reported iff fault analysis is done for an RTL design. If design is a netlist, synthesis is already done and the question of

- synthesis redundancy is considered invalid.
7. LR fault : These are the faults that are logically redundant. That means whatever simulation value the user applies the output of that part of circuit will always be a fixed value.
8. PT fault : Faults in which difference between the good circuit result and bad circuit result for a particular fault produces any of the following are called Potential Detected Faults:
- | Good Circuit Result | -- | Bad Circuit Result |
|---------------------|----|--------------------|
| 0 | | z |
| 0 | | x |
| 1 | | z |
| 1 | | x |
9. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets.
 If a net is tied to 0, S@0 is TIED. S@1 may or may not be detectable. Similarly, a net tied to 1 will make S@1 TIED.
 An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked.
 For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAES USED :

$$\text{Fault-Coverage \%} = \frac{\text{Detectable Fault Count} + ((\text{PT}) * \text{POSDetect_credit})}{(2 * \text{terminalCount}) - \text{SyRd_faultCount}} \times 100$$

$$\text{Test-Coverage \%} = \frac{\text{Detectable Fault Count} + ((\text{PT}) * \text{POSDetect_credit})}{(2 * \text{terminalCount}) - \text{SyRd_faultCount} - \text{TI} - \text{BL} - \text{UUCount} - (\text{ATPG_credit} * \text{UTCCount}) - \text{LR}} \times 100$$

<>For top module the summary has been reported in three columns as follows:

Column 1: fault and coverage data due to top-level ports and internal design.

Column 2: fault and coverage data due to top-level ports only.

Column 3: fault and coverage data due to internal design only.

<>For each module and hierarchical instance, eleven numbers are reported. These are (in order)

1. count of terminals (IT) within the module (or instance)
2. count of synth-redundant faults (SR) within the module (or instance)
3. count of faults considered (FC) within the module (or instance)
4. count of unused (UU) faults within the module (or instance)
5. count of un-testable (UT) faults within this module (or instance)
6. count of non-detectable (!DT) faults within this module (or instance)
7. count of detectable (DT) faults within this module (or instance)
8. count of logical redundant fault (LR) within this module (or instance)
9. count of potential detected (PT) faults within the module (or instance)
10. count of tied (TI) faults within the module (or instance)
11. count of blocked (BL) faults within the module (or instance)

instance)

<> Alongside each module (and hierarchical instance) there is a search-key, of the form

```
Module_[moduleIndex] | Inst_[moduleIndex]_[instIndex]
```

When opened in a text-editor, this key can be used to jump to the next (and only other) reference to the module (or instance) in the file. If an instance does NOT have a key then it means that this hierarchical instance has no hierarchical instances within it.

PS : The instance names have been wrapped around to the next line in order to maintain the clarity of the Table.

----- END -----

<>TOP MODULE SUMMARY for 'top'

Fault Heads	Total	Ports	Internal
Total faults	72	12	60
SyRd faults	0	0	0
Faults Considered	72	12	60
Un-Used (UU)	2	2	0
Un-Testable (UT)	0	0	0
Un-Detectable (!DT)	54	9	45
Detectable (DT)	16	1	15
Tied (TI)	0	0	0
Blocked (BL)	0	0	0
Logical Redundant faults (LR)	0	0	0
Potential Detectable (PT)	0	0	0
ATPG_credit	0.00	0.00	0.00
POSDETECT_credit	0.00	0.00	0.00
Fault-coverage(in %)	22.2	8.3	25.0
Test-coverage(in %)	22.9	10.0	25.0

Coverage Summary for each Instance:

```

-----
Columns          IT  SR  FC  UU  UT  !DT  DT  LR  PT  TI  BL
-----
<>TOP MODULE 'top'

  Child-Instances
  -----
  Inferred_LOGIC.....30  0  60  0  0  45  15  0  0  0  0
  
```

The stuck_at_coverage.rpt file is generated in the
 <current-working-directory>/spyglass_reports/dft/> directory.

stuck_at_coverage_audit

The stuck_at_coverage_audit.rpt file is generated by the *Coverage_audit* rule. This is basically a coverage audit report.

A sample format of the stuck_at_coverage_audit report is shown below:

```
#####
# Format :
# Section 1:
# The first section of the report indicates the current
# coverage for the design and the steps, in tabular format,
# that you need to follow to improve the coverage. Each
# step has:
#   - The description of the root cause
#   - The improved fault and test coverage number, if the
#     reported issues for the step are fixed.
#   - The diagnostic rules to debug the respective steps
#
# Section 2:
# The second section of the report covers rules that
# detect conditions which, if not fixed, can prevent ATPG
# tests from operating as expected. As a result, actual
# coverage may be less than the result predicted by ATPG
# tools or tests may fail even when a chip is operating
# correctly.
#
# The rules in this section of the report are sorted by
# the number of flip-flops affected by each rule
#####

# Section 1:
#
<>Reasons for Test Coverage to be less than 100 percent:
0. Current Coverage: This step reports fault and test
   coverage of design in its current state.
   Please note that coverage number may be
   100% even when there are Async_07 and /
   or Clock_11 violations. This is because
```

- violating flip-flops have been declared forced scan using 'scan' sgdc command. Such flip-flops are detected using Info_forcedScan
1. PIs and POs : Set dft_treat_primary_inputs_as_x_source to 'off' to make primary inputs controllable. Set dft_treat_primary_outputs_as_unobservable to 'off' to make primary outputs observable
 2. Non-Scannable Flops : Non-scannable flip-flops reduce circuit controllability and observability for combinational ATPG. Detected using Async_07 and Clock_11 rules in DFT/dft_scan_ready goal
 3. Non-Scan-wrapped black box : I/P terminals of non-scan-wrapped BB's are unobservable and o/p terminals are uncontrollable. Detected using any DFT rule Info_scanwrap or DFT/dft_scan_ready goal
 4. Non-Transparent latches : Non-transparent latches require sequential ATPG. Detected using Latch_08 rule in DFT/dft_latches goal
 5. Combinational Loops : Combinational loops may not be controllable. This may reduce test coverage of the circuit. Such Loops are detected by Topology_01. Additionally, loops with non-X value may also reduce coverage
 6. Testmode values : Testmode values in capture in Capture mode restrict ATPG. This step includes the undetectable faults due to pins tied high or low. Detected using Info_untestable and Info_pwrGndSim
 7. Hanging Nets : Undriven nets and pins are uncontrollable. Detected using DFT/dft_test_points goal (TA_09 rule)
 8. Tristates : Enable pins of tristate devices are

- 9. 'force_ta' and 'test_point' : unobservable. Detected using DFT /dft_test_points goal (TA_09 rule).
 : Pins and nets having 'force_ta' or 'test_point' constraint pins/nets will have their testability governed by constraint. These will hinder test coverage
- 10. 'no_scan' Flip-flops : Flops which are forced no_scan by user or inferred no_scan will not generate Async_07 or Clock_11 violations, but will hinder coverage

For each top module 2 numbers are reported against the step taken. These are (in order):

- a) Fault Coverage in percentage
- b) Test Coverage in percentage

```

----- END -----
<> For Top Module 'izar'
    Base stuck-at fault coverage :20.8
    Base stuck-at test coverage :20.8
  
```

Expected stuck-at coverage data after each step.

	Expected stuck-at coverage		
	FC	TC	Action
0. Original Design	20.8	20.8	No DFT changes are made. It assumes 'scan' declared flip-flops scannable. Use Info_forcedScan to detect such flip-flops
1. PIs and POs made controllable and observable and off	70.8	70.8	Set dft_treat_primary_inputs_as_x_source and dft_treat_primary_outputs_as_unobservable to off

2. Flip-flops made scannable	70.8	70.8	No action required
3. Scan-wrap black boxes	70.8	70.8	No action required
4. Latches made Transparent	70.8	70.8	No action required
5. Combinational Loops made controllable	70.8	70.8	No action required
6. Testmode/Tied pins made controllable	70.8	70.8	Info_synthRedundant, Info_untestable and Info_pwrGndSim. Add -scanshift switch to test_mode constraint, if appropriate
7. Hanging nets made controllable	70.8	70.8	No action required
8. Tristate enables made observable	70.8	70.8	No action required
9. 'force_ta' and 'test_point' pins made testable	70.8	70.8	No action required
10. 'no_scan' flip-flops made scannable	100.0	100.0	Use Info_noScan and Info_inferredNoScan to view all no_scan flip-flops

```

-----
#####
#
# Section 2:
#
<> For Top Module 'izar'
    Percentage of flip-flops which have at least one reason
    for non-robustness: 0.0% (0/3)
    Percentage of flip-flops tagged by each rule are listed
    below:

```

```

-----
Rule          |          Percentage          |          Ratio(tagged/total)

```

```
-----  
Async_02_capture      0.0          0/3  
Async_11              0.0          0/3  
Clock_04              0.0          0/3  
Clock_08              0.0          0/3  
Clock_16              0.0          0/3  
Clock_17              0.0          0/3  
Clock_21              0.0          0/3  
Clock_27              0.0          0/3  
Clock_28              0.0          0/3  
Scan_07               0.0          0/3  
Scan_22               0.0          0/3  
Topology_03           0.0          0/3  
Topology_13           0.0          0/3  
Tristate_06           0.0          0/3  
Tristate_07_shift     0.0          0/3  
-----
```

```
#####
```

The stuck_at_coverage_audit.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

stuck_at_coverage_collapsed

The *stuck_at_coverage_collapsed.rpt* report contains the stuck-at coverage summary with reference to fault collapsing. This report is generated by the [Info_coverage](#) rule when the value of the [dft_collapse_equivalent_faults](#) parameter is set to on.

Accessing the Report

To view the report, refer to the following directory:

```
<current-working-directory>/spyglass_reports/dft/>
```

Alternatively, to view the report from GUI, click the **Report** Icon in the Results pane under the **Analyze Results** tab, and select the required report from the **DFT** menu.

Sample Report

A sample *stuck_at_coverage_collapsed* report follows:

```
#####
# Format :
# The definition of various kinds of faults is given.
# This is followed by a tabular information of the
# count of faults associated with each module in the
# design.
#####
<>ABBREVIATIONS & DEFINITIONS OF TERMS :
  1. DT fault      : Faults which are testable AND detectable.
                    It is possible that a fault is testable,
                    but not detectable.
  2. !DT/ND fault : A fault which was deemed testable but
                    found to be undetectable. (Note that
                    faults classified as TI or BL are not
                    included here.) (Please see Info_coverage
                    / Info_undetectedCause violations).
  3. UT fault      : A fault which cannot be detected because
                    it has a non-X simulation value. (Note
                    that faults with nonX sim value in PG
                    mode itself are classified as TI faults.)
  4. UU fault      : An unused fault. Fault at a terminal
                    becomes UU if that terminal cannot be
```

reached from the output ports. They are ignored because they cannot affect system operation. By default, latches / blackboxes, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.

5. SyRd fault : Faults at terminals which are Sy[=synthesis]Rd[= redundant]. These are terminals which are likely to be removed by an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis. These are of the following types:
- a) Pins which are tied high/low,
 - b) Pins blocked in power ground mode, and
 - c) Unconnected combinational logic - all logic combinationaly connected to nodes that do not fanout to primary port.
- SyRd faults are reported iff fault analysis is done for an RTL design. If design is a netlist, synthesis is already done and the question of synthesis redundancy is considered invalid.
6. LR fault : These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.
7. PT fault : Faults in which difference between the good circuit result and bad circuit result for a particular fault produces any of the following are called Potential Detected Faults:
- Good Circuit Result -- Bad Circuit Result

```

0          z
0          x
1          z
1          x

```

8. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets.
 If a net is tied to 0, S@0 is TIED. S@1 may or may not be detectable. Similarly, a net tied to 1 will make S@1 TIED.
 An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked.
 For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

$$\text{Fault-Coverage \%} = \frac{\text{Detectable Fault Count} + ((\text{PT}) * \text{POSDTECT_credit})}{\text{Total Collapsed Fault Count} - \text{SyRd_faultCount}} \times 100$$

$$\text{Test-Coverage \%} = \frac{\text{Detectable Fault Count} + ((\text{PT}) * \text{POSDTECT_credit})}{\text{Total Collapsed Fault Count} - \text{SyRd_faultCount} - \text{TI} - \text{BL} - \text{UUCount} - (\text{ATPG_credit} * \text{UTCCount}) - \text{LR}} \times 100$$

<>TOP MODULE SUMMARY for 'top'

Total Faults	54
SyRd faults	0
Faults Considered	54
Un-Used (UU)	10
Tied (TI)	6

Blocked (BL)	2
Logical Redundant faults (LR)	3
Un-Testable (UT)	1
Detectable (DT)	32
Un-Detectable (!DT)	0
Potential Detectable (PT)	0
ATPG_credit	0.00
POSDTECT_credit	0.00
Fault-coverage(in %)	59.3
Test-coverage(in %)	97.0

test_points_selected

The test_points_selected.rpt file is generated by the [TA_06](#) rule. This report contains a list of suggested test points.

A sample format of test_points_selected report is shown below:

```
#####
# Module :- izar
#####

#####
# SUMMARY :
#       Test points considered :- 60.
#       Total faults considered :- 331894.
#       Total fault pins :- 176178.
#       Synthesis redundant faults :- 20462.
#       Unused faults :- 3232.
#
#       Detected Fault count before all testpoints
#       inserted :- 325530.
#       Detected Fault count after all testpoints
#       inserted :- 326357.
#
#       Potential Detected Fault count after all
#       testpoints inserted :- 2309.
#       Potential Detected credit :- 0.0.
#
#       Fault Coverage before any testpoints are
#       considered :- 98.1.
#       Test Coverage before any testpoints are
#       considered :- 99.0.
#
#       Fault Coverage after all testpoints considered
#       :- 98.3.
#       Test Coverage after all testpoints considered
#       :- 99.3.
#
#       Fault Coverage after all testpoints and all
```

```
        potentially detectable faults are considered
        :- 98.3.
#       Test Coverage after all testpoints and all
        potentially detectable faults are
        considered:- 99.3.
#
# PS.
# 1)   The Fault/Test coverage(after alltestpoints
#       considered) and the Fault/Test
#       coverage(after all testpoints and all
#       potentially detectable faults are
#       considered) will be same if parameter
#       dftPOSDTECT_credit is set to zero or
#       there are no potentially detectable faults in
#       the design.
```

The `test_points_selected.rpt` file is generated in the
<*current-working-directory*>/spyglass_reports/dft/> directory.

test_points_selected_2

The test_points_selected_2.rpt file is generated by the [TA_09](#) rule. This report contains a list of suggested test points.

A sample format of test_points_selected_2 report is shown below:

```
#####
# Purpose :
#   This file contains final fault and test coverage
#   if all the SpyGlass-DFT suggested testpoints are used.
#####
#####

# Format :
#   A tabular report is generated.
#   The test points are dealt according to the weight
#   assigned to them.
#   The nature of the test point(Obs/Cont) is given.
#   The increase in test coverage and test point name is
#   tabulated.
#
#   A '(c)' sign at the beginning of the test point name
#   indicates that this
#   is possibly a clock-net, in which case test point
#   insertion may not be very useful. You might want to
#   ignore such nets in subsequent runs by
#   providing the 'dont_touch' constraint.
#
#   A '(d)' sign at the beginning of the test point name
#   indicates that this net is possibly inside a
#   'dont_touch' module.
#
#   A '(c|d)' sign indicates a net which is both of the
#   above.
#
```

```

# Note : In case -dftAutoFix switch is specified for
# TA_09, autofix id will be attached with every
# testpoint wherein
# Auto_Fix_ID -2 indicates no autofix due to donttouch
# constraint on the module.
# Auto_Fix_ID -1 indicates no autofix due to software
# limitation.
# Auto_Fix_ID [num >= 0] indicates autofix is applied.
#####

#####
#Module :- test
#####

#####

#SUMMARY :
# Test points considered :- 2.
# Total faults considered :- 52.
# Total fault pins :- 26.
# Synthesis redundant faults :- 0.
# Unused faults :- 4.
# Tied faults :- 5.
# Blocked faults :- 2.
#
# Detected Fault count before all testpoints inserted
# :- 31.
# Detected Fault count after all testpoints inserted
# :- 39.
#
# Potential Detected Fault count after all testpoints
# inserted :- 11.
# Potential Detected credit :- 0.0.
#
# Fault Coverage before any testpoints are considered
# :- 59.6.
# Test Coverage before any testpoints are considered
# :- 75.6.

```



```

#
#   Fault Coverage after all testpoints considered
#   :- 75.0.
#   Test Coverage after all testpoints considered
#   :- 95.1.
#
#   Fault Coverage after all testpoints and all
#   potentially detectable faults are considered
#   :- 75.0.
#   Test Coverage after all testpoints and all
#   potentially detectable faults are considered:- 95.1.
#
#PS.
# 1) The Fault/Test coverage(after alltestpoints
#   considered) and the Fault/Test coverage(after all
#   testpoints and all potentially detectable faults are
#   considered) will be same if parameter dftPOSDETECT_credit
#   is set to zero or there are no potentially detectable
#   faults in the design.
#####

```

INDEX	TYPE	FAULT- COVERAGE	TEST- COVERAGE	DETECTABLE_ FAULTS	TESTPOINT_ NAME
-------	------	--------------------	-------------------	-----------------------	--------------------

1	Control	67.3	85.4	35	test.out
2	Observe	75.0	95.1	39	test.outi

The test_points_selected_2.rpt file is generated in the
<current-working-directory>/spyglass_reports/dft/> directory.

undetected_faults

The undetected_faults.rpt file is generated by the [Info_undetectedCause](#) rule. This report contains a list of undetected faults and their causes.

To generate this report, set the value of the *dftGenerateTextReport* parameter to the rule name.

A sample format of `undetected_faults` report is shown below:

```
#####
```

```
# SUMMARY
#
#     MODULE NAME           : izar
#     UNDETECTED FAULT COUNT : 1845
#     UNDETECTABLE FAULT COVERAGE: 0.0
#####
```

```
Instance :
izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.
\q_reg[0] [Flip-flop]
File      : rtl_files/wrap_headtail.v
Line      : 70 ; Parent Module : kernel_router
```

```
      D      input      s@01      nnn      1
```

```
Instance :
izar.kernel_test.kernel_router.ipp_ind_refclk_izar_tail_reg.
\q_reg[1] [Flip-flop]
File      : rtl_files/wrap_headtail.v
Line      : 70 ; Parent Module : kernel_router
```

```
      D      input      s@01      nnn      1
```

The `undetected_faults.rpt` file is generated in the `<current-working-directory>/spyglass_reports/dft/>` directory.

atspeed_03_rpt

The `atspeed_03_rpt.rpt` file is generated by the *Atspeed_03* rule. This report lists all cross-domain paths between flip-flops that are asynchronous in functional domain and synchronous in at-speed testmode domain.

A sample format of atspeed_03_rpt report is shown below:

```
#####
# Format :
#   Syntax of this file:
#   Source      Functional Clock: <src fclk>
#   Destination Functional Clock: <dst fclk>
#   [S.No.] <src flip-flop name> [Src Atspeed Clock:
#   <src aclk>] -> <dst flip-flop name> [Dst Atspeed
#   Clock: <dst aclk>]: <no-of-gates-in-cross-path>
#####

#####

# Begin data for design unit: top_Atspeed_04
#####

#####

# End data for design unit: top_Atspeed_04
#####

#####

# Begin data for design unit: top_Atspeed_03
#####

Source Functional Clock: top_Atspeed_03.u_clk_gen_a.ck1
Destination Functional Clock:top_Atspeed_03.u_clk_gen_a.ck2
  [1] top_Atspeed_03.ff1 [Src Atspeed Clock:
top_Atspeed_03.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_t1.
clkout] -> top_Atspeed_03.ff3 [Dst Atspeed Clock:
top_Atspeed_03.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_t1.
clkout]: 2
```

```
[2] top_Atsspeed_03.ff1 [Src Atsspeed Clock:
top_Atsspeed_03.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_t1.
clkout] -> top_Atsspeed_03.ff2 [Dst Atsspeed Clock:
top_Atsspeed_03.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_t1.
clkout]: 1
```

```
#####
```

```
# End data for design unit: top_Atsspeed_03
```

```
#####
```

The atsspeed_03.rpt file is generated in the <cwd>/spyglass_reports/dft_dsm/
directory.

atspeed_04_rpt

The atspeed_04_rpt.rpt file is generated by the [Atspeed_04](#) rule. This report lists all cross-domain paths between flip-flops that are synchronous in functional domain and asynchronous in at-speed testmode domain.

A sample format of atspeed_04_rpt report is shown below:

```
#####

# Format :
#   Syntax of this file:
#   Source           Functional Clock: <src fclk>
#   Destination Functional Clock: <dst fclk>
#   [S.No.] <src flip-flop name> [Src Atspeed Clock: <src
#   aclk>] -> <dst flip-flop name> [Dst Atspeed Clock:
#   <dst aclk>]: <no-of-gates-in-cross-path>
#####

#####

# Begin data for design unit: top_Atspeed_04
#####

Source Functional Clock: top_Atspeed_04.u_clk_gen_a.ck3
Destination Functional Clock:top_Atspeed_04.u_clk_gen_a.ck3
  [1] top_Atspeed_04.ff1 [Src Atspeed Clock:
top_Atspeed_04.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_f1.
clkout] -> top_Atspeed_04.ff3 [Dst Atspeed Clock:
top_Atspeed_04.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_f2.
clkout]: 2
  [2] top_Atspeed_04.ff1 [Src Atspeed Clock:
top_Atspeed_04.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_f1.
clkout] -> top_Atspeed_04.ff2 [Dst Atspeed Clock:
top_Atspeed_04.u_clk_gen_a.u_clk_gen_b.u_clk_gen_c.u_pll_f2.
clkout]: 1

#####
```

```
# End data for design unit: top_Atsspeed_04
#####

#####

# Begin data for design unit: top_Atsspeed_03

#####

#####

# End data for design unit: top_Atsspeed_03

#####
```

The atsspeed_04.rpt file is generated in the <cwd>/spysglass_reports/dft_dsm/ directory.

atsspeed_07_rpt

The atsspeed_07_rpt.rpt file is generated by the *Atsspeed_07* rule. This report lists all at-speed test clocks that are gated by the same set of logic.

A sample format of atsspeed_07_rpt report is shown below:

```
#####
The following test Clocks are gated by same
net - top.clkGate :
    top.tclk1
    top.tclk2
The following test Clocks are gated by same
net - top.q :
    top.tclk1
    top.tclk2
#####
```

The atsspeed_07.rpt file is generated in the <cwd>/spysglass_reports/dft_dsm/

directory.

atspeed_08_rpt

The `atspeed_08_rpt.rpt` file is generated by the [Atspeed_08](#) rule. This report lists all at-speed clocks that have different gates along the clock paths and should be balanced.

A sample format of `atspeed_08_rpt` report is shown below:

```
#####
# Purpose :
#   Atspeed_08 Report.
#   Lists the at-speed clocks which need to be balanced
#   and which have different gates along the clock
#   paths.
#####

#####
top.y
top.x
#####
```

The `atspeed_08.rpt` file is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

atspeed_clock_synchronization

The `atspeed_clock_synchronization.rpt` file is generated by the [Info_atspeedClockSynchronization](#) rule in the current working directory. This report shows an ordered list of the domain pairs.

If `dsmAtspeedClockSynchronizationRange` is set to 0, all domain pairs are considered valid. Domain pairs in which one or both domains lack a numeric-frequency-value are reported in a separate section.

A sample format of the atspeed_clock_synchronization.rpt file is shown below:

```
#####
# Purpose :
#   Atspeed Clock Synchronization Report:
#   For each top block
#   - Reports an ordered list of atspeed clock domain pairs that will
#     benefit most from synchronization
#   - Maximum difference in frequency of clock domain pair : 'no-range'
#   - Reports an ordered list of non-interacting-non-intersecting
#     atspeed clock domain groups
#   - Reports an ordered list of non-interacting-intersecting
#     atspeed clock domain groups
#####

#####

# Format :
# For each top block, for each item in the list the following
# information is shown:
#
#   Section - I:
#   Interacting atspeed domain pairs (both having numeric frequency):
#
#   rank  freq_domA  freq_domB  # of domA ffs  # of domB ffs  domA  domB
#                                     driven by domB  driven by domA
#
#
#   Section - II:
#   Interacting atspeed domain pairs (one or both having no/non_numeric
#   frequency):
#
#   rank  freq_domA  freq_domB  # of domA ffs  # of domB ffs  domA  domB
#                                     driven by domB  driven by domA
#
#
#   Section - III:
```


Non-interacting and non-intersecting atspeed domain groups:

```
#
# rank      # of flip-flops      # of domains      domains
#           in group           in group          in group
#
#
```

Section - IV:

Non-interacting and intersecting atspeed domain groups:

```
#
# rank      # of flip-flops      # of domains      domains
#           in group           in group          in group
#
#
```

#####

#####

Start "top"

#####

Section - I:

Interacting atspeed domain pairs (both having numeric frequency (n_n_freq)):

rank	freq_domA	freq_domB	# of domA ffs driven by domB	# of domB ffs driven by domA	domA	domB
1	700.0	300.0	2	2	top.clk3	top.clk4
2	100.0	700.0	0	2	top.clk2	top.clk3

Section - II:

Interacting atspeed domain pairs (one or both having no/
non_numeric_frequency (n_n_freq)):

```
rank  freq_domA  freq_domB  # of domA  # of domB  domA  domB
      ffs driven  ffs driven
```

		by domB		by domA	
1	n_n_freq	100.0	0	3	top.clk1 top.clk2
2	n_n_freq	700.0	2	0	top.clk1 top.clk3

Section - III:

Non-interacting and non-intersecting atspeed domain groups

rank	# of flip-flops in group	# of domains in group	domains in group
1	8 (61.54%)	3	top.clk1, top.clk4, top.clk5
2	3 (23.08%)	1	top.clk2
3	2 (15.38%)	1	top.clk3

Section - IV:

Non-interacting and intersecting atspeed domain groups

rank	# of flip-flops in group	# of domains in group	domains in group
1	8 (61.54%)	3	top.clk1, top.clk4, top.clk5
2	8 (61.54%)	3	top.clk2, top.clk4, top.clk5
3	5 (38.46%)	2	top.clk3, top.clk5

End "top"
#####

Start "top2"
#####

Section - I:

Interacting atspeed domain pairs (both having numeric frequency):

rank	freq_domA	freq_domB	# of domA	# of domB	domA	domB
------	-----------	-----------	-----------	-----------	------	------

		ffs driven by domB		ffs driven by domA	
1	7000.0	3000.0	2	2	top2.clk3 top2.clk4
2	1000.0	7000.0	0	2	top2.clk2 top2.clk3

Section - II:

Interacting atspeed domain pairs (one or both having no/
non_numeric_frequency (n_n_freq)):

rank	freq_domA	freq_domB	# of domA ffs driven by domB	# of domB ffs driven by domA	domA	domB
1	n_n_freq	1000.0	0	3	top2.clk1	top2.clk2
2	n_n_freq	7000.0	2	0	top2.clk1	top2.clk3

Section - III:

Non-interacting and non-intersecting atspeed domain groups

rank	# of flip-flops in group	# of domains in group	domains in group
1	8 (61.54%)	3	top2.clk1, top2.clk4, top2.clk5
2	3 (23.08%)	1	top2.clk2
3	2 (15.38%)	1	top2.clk3

Section - IV:

Non-interacting and intersecting atspeed domain groups

rank	# of flip-flops in group	# of domains in group	domains in group
1	8 (61.54%)	3	top2.clk1, top2.clk4,

```

                top2.clk5
2             8 (61.54%)           3             top2.clk2, top2.clk4,
                top2.clk5
3             5 (38.46%)           2             top2.clk3, top2.clk5
    
```

```

#####
# End "top2"
#####
    
```

The `atspeed_clock_synchronization` report is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

dft_dsm_clock_frequency

The `dft_dsm_clock_frequency.rpt` file is generated by the [Info_freqAssignTable](#) rule. This report lists the frequency assignment data, as specified in the constraint files. The data is displayed in a tabular form with testclock name, enable pins, and frequency symbols as three columns.

A sample format of `dft_dsm_clock_frequency` report is shown below:

```

Test Clock      Enable pins      Frequency
-----
en0      en1
-----
                1      0      f0
                0      1      f1
    
```

The `dft_dsm_clock_frequency.rpt` file is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

dft_dsm_clock_gating_cell

The `dft_dsm_clock_gating_cell.rpt` file is generated by the [CG_generateReport](#) rule. This report lists each clock gating cell (CGC) with its Name, ModuleName (name of master module), Type, Test-enable pin Clock source

(shift mode), number of flip-flops connected, and CG rule violations (if any). CGCs are grouped by their source clocks. Each source clock has a different section that contains all the CGCs present in its clock tree. CGCs that are not driven by any clock are specified under the **no-clock** section. Also, the total number of CGCs and clocks are reported under the **Summary** section.

For more information on identifying CGCs, see *Identifying Clock Gating Cells* section in the *DFT Rules Reference Guide*.

A sample format of `dft_dsm_clock_gating_cell` report is shown below:

```
#####
# Format :
# Section I:
#   gating_cell -name moduleName -clkinTerm ckin -clkout ckoutTerm
#   -clkEdgeType edgeType
#       instanceCount - affected flip-flops 'X'
#       inst1 - flopCount1 inst2 - flopCount2
#
#Section II:
#   Name  moduleName Type Test-En  Clock(Shift_Mode) Flops  Violations
#
# SUMMARY
#       Top Module Name : 'designName'
#       Distinct Clk Count : XX
#       Total CGC Count : XX
#       Inferred CGC Count : XX
#       Potential CGC Count : XX
#####

#####
# Design : "top"
#####

current_design "top"
# Section I:

gating_cell -name "CGC_6_N" -clkinTerm "ckin" -clkoutTerm "ckout"
-cgcEdgeType "negative" -enTerm "e" -enValue 1 -testenTerm "t"
```

```
-testenValue 1
# gating_cell -name "CGC_6_N" -clkinTerm "ckin" -clkoutTerm "ckout"
  -cgcEdgeType "negative" -enTerm "t" -enValue 1 -testenTerm "e"
  -testenValue 1
#cgc_instance_count : 1      ,potential affected flip-flops : 2
#cgc_instance_count : 1      ,potential affected flip-flops : 2
  #top.z_cgc_b1_1 - 2

#Inferred
# gating_cell -name "CGC_5_P" -clkinTerm "ckin" -clkoutTerm "ckout"
  -cgcEdgeType "positive" -enTerm "se" -enValue 1 -testenTerm "te"
  -testenValue 1
#cgc_instance_count : 1      ,potential affected flip-flops : 2
  #top.a_cgc_b2 - 2

#Inferred
# gating_cell -name "CGC_4_N" -clkinTerm "ckin" -clkoutTerm "ckout"
  -cgcEdgeType "negative" -enTerm "te" -enValue 1
#cgc_instance_count : 2      ,potential affected flip-flops : 2
  #top.u_cgc_b1_1 - 2
  #top.u_cgc_n - 0

#Inferred
# gating_cell -name "CGC_1_P" -clkinTerm "ckin" -clkoutTerm "ckout"
  -cgcEdgeType "positive" -enTerm "se" -enValue 1 -testenTerm "te"
  -testenValue 1
#cgc_instance_count : 4      ,potential affected flip-flops : 3
  #top.u_cgc_b2 - 2
  #top.v_cgc_b2 - 0
  #top.w_cgc_b2 - 0
  #top.u_cgc_p - 1

# Section II:

#'1' CGC(s) are present on the 'no-clock' tree
top.u_cgc_p; CGC_1_P; rtl(inferred); unconstrained; no clock; 1; None
#'9' CGC(s) are present on the 'top.clk' tree
top.u_cgc_n; CGC_4_N; rtl(inferred); unconstrained; top.clk; 2; None
```

```
top.w_cgc_b2; CGC_1_P; rtl(inferred); unconstrained; top.clk; 2; None
top.v_cgc_b2; CGC_1_P; rtl(inferred); unconstrained; top.clk; 2; None
top.u_cgc_b2; CGC_1_P; rtl(inferred); unconstrained; top.clk; 2; None
top.c_cgc_b2; CGC_2_P; rtl(constraint); unconstrained; top.clk; 2; None
top.b_cgc_b2; CGC_2_P; rtl(constraint); unconstrained; top.clk; 2; None
top.a_cgc_b2; CGC_5_P; rtl(inferred); unconstrained; top.clk; 2; None
top.u_cgc_b1_1; CGC_4_N; rtl(inferred); unconstrained; top.clk; 2; None
#####
# Summary :
#       Top Module Name : 'top'
#       Distinct Clk Count : 2
#       Total CGC Count : 9
#       Inferred CGC Count : 7
#       Potential CGC Count : 1
#####
```

The dft_dsm_clock_gating_cell.rpt file is generated in the <cwd>/spyglass_reports/
dft_dsm/ directory.

dft_dsm_constr-err-file

The `dft_dsm_constr-err-file.rpt` file is generated by the [dftDsmConstraintCheck_01](#) rule. This report contains all the violations reported by this rule. A sample format of the `dft_dsm_constr-err-file.rpt` is as shown below:

```
#####
# Purpose :
# The message generated below corresponds to errors found in
# the corresponding constraint file
#####

#####
atspeed_clock_frequency constraint errors
#####
FILE :test.sgdc LINE:3
Mismatch in number of freqs and values in clk

#####
clock_shaper constraint errors
#####
```

dft_dsm_enabled_flipflops

The `dft_dsm_enabled_flipflops.rpt` file is generated by the [Info_enabledFlops](#) rule. This report lists each clock domain in the following format:

- Flip-flops: Flip-flop count
- Domain/Category: Domain Name
- Clock(s): Name of clock(s)
- Serial No: Name of the Flip-flop

The report contains either all or only the scannable flip-flops, sorted by their respective clock domains. The clock domains are sorted by the number of flip-flops and the flip-flops within a domain are sorted by their hierarchy.

The following statement is appended at the end of the report for the inferred clock domains:

```
i inferred at clock_shaper output
```

The following is an example of such statement in the report:

```
#      Domain   : top.u_cs_3.clk_out (inferred at clock_shaper
output)
#      Clock(s) : top.clk_in
```

A sample format of the dft_dsm_enabled_flipflops report is shown below:

```
#####
# Purpose :
#      Report Type : Enabled Flip-Flop, Memory & Macros
instances
#      Contains all the Flip-Flop, Memory & Macros instances
in the design.
#####

#####
# Format :
#      Domain/Category      : Domain/Category Name
#      Clock(s)             : Name of clock(s)
#      Single Bit Flip-flops : Flip-flop terminal count
#          Serial No| Name of the terminal path
#      Multi-Bit Flip-flops  : Flip-flop terminal count
#          Serial No| Name of the terminal path
#      Memory                : Memory terminal count
#          Serial No| Name of the terminal path
#      Macros                : Macros terminal count
#          Serial No| Name of the terminal path
```

```
#####

#####

# Design : top
# Parameter 'dsmReportEnabledFlops' : all
#####

#+++++
# of Single Bit # of Multi-Bit # of Multi-Bit # of Memory
# of Macros Domain/Category
      Flip-flop      Flip-flop      Flip-flop      Clock Pins
Clock Pins
      Clock Pins      Clock Pins      Storage Bits
      0              0              0              0
0  Atspeed_Clock_Blocked
      0              1              2              1
0  Clock_Tied_Low
      0              1              1              0
1  Clock_Tied_High
      0              0              0              0
0  No_Atspeed_Clock_in_Fanin_Cone
      0              0              0              0
0  Async_Pin_set_Tied_Active
      0              0              0              0
0  Async_Pin_reset_Tied_Active

      2              1              2              3
2  first
      1              0              0              2
1  second
```

```

#####
# Category : Atspeed_Clock_Blocked
# Single Bit Flip-flops : 0
# Multiple Bit Flip-flops : 0
# Memory : 0
# Macros : 0
#####
# Category : Clock_Tied_Low
# Single Bit Flip-flops : 0
# Multiple Bit Flip-flops : 1
#           1| top.u_ffb01.CLK
# Memory : 1
#           1| top.m4.c
# Macros : 0
#####
# Category : Clock_Tied_High
# Single Bit Flip-flops : 0
# Multiple Bit Flip-flops : 1
#           1| top.u_cs.clk
# Memory : 0
# Macros : 1
#           1| top.t3.CLK1
#####
# Category :
No_Atspeed_Clock_in_Fanin_Cone
# Single Bit Flip-flops : 0

```

```

#      Multiple Bit Flip-flops      : 0
#      Memory                       : 0
#      Macros                       : 0
#+++++

#      Category                     : Async_Pin_set_Tied_Active
#      Single Bit Flip-flops        : 0
#      Multiple Bit Flip-flops      : 0
#      Memory                       : 0
#      Macros                       : 0
#+++++

#      Category                     : Async_Pin_reset_Tied_Active
#      Single Bit Flip-flops        : 0
#      Multiple Bit Flip-flops      : 0
#      Memory                       : 0
#      Macros                       : 0
#+++++

#      Domain                       : first
#      Clock(s)                    : top.clk3, top.clk2,
top.clk1
#      Single Bit Flip-flops        : 2
#          1| top.ff2.ff_reg.CP
#          2| top.ff1.ff_reg.CP
#      Multiple Bit Flip-flops      : 1
#          1| top.u_ffb02.CLK
#      Memory                       : 3
#          1| top.m3.clk1

```

```

                2| top.m2.c
                3| top.m1.c
#      Macros                                     : 2
                1| top.t1.CLK1
                2| top.t1.CLK2
#+++++
#      Domain                                     : second
#      Clock(s)                                  : top.clk5, top.clk4
#      Single Bit Flip-flops                     : 1
                1| top.ff3.ff_reg.CP
#      Multiple Bit Flip-flops                   : 0
#      Memory                                     : 2
                1| top.m5.c
                2| top.m3.clk2
#      Macros                                     : 1
                1| top.t3.CLK2
#####
# End Design : top
#####

```

The `dft_dsm_enabled_flip_flops` report is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

dft_dsm_ip_report

The `dft_dsm_ip_report.rpt` file is generated by the [Info_IP_Report](#) rule and provides the following information on a per Instance basis:

- For Mode: value <-> pin-name (value: 0 | 1)
- For Control: value <-> pin-name (value: nnn | nny | ... | yyy)

- For Observe: value <-> pin-name (value: n | y)
- For Clock: pin_name ; direction ; clock_edge (rtz | rto);
- Flip-flop_count ; root_clock ; root_clock_frequency ;
- Multiplier (frequency multiplier with respect to root_clock)

A sample format of dft_dsm_ip_report is shown below:

```
#####
# Purpose:
# This rule generates boundary report (in 'RPT' format) # on
# the IPs specified using ip_block SGDC command.
# Report may contain one or more of following
# information:
# 1. clock (shift, capture, @speed capture)
# 2. mode (shift, capture, @speed capture)
# 3.controllability of the input pins at static
# capture mode
# 4.observability of the output pins at static capture #
mode
#####

#####
# Format :
# The Information is reported (per Instance basis) as
# follows:
# For Mode --> value <-> pin-name (value: 0 | 1)
# For Control --> value <-> pin-name (value: nnn | nny |
... | yyy)
# For Observe --> value <-> pin-name (value: n | y)
# For Clock --> pin_name ; direction ; clock_edge (rtz |
rto);
# flip-flop_count ; root_clock ; root_clock_frequency ;
# multiplier (frequency multiplier with respect to
root_clock)
#####
```

```

*****      Design Name is :CGC      *****
Instance Name:top1.u2
  Test_Mode: SHIFT_MODE
    Value      PinName
      --      --
  Test_Mode: CAPTURE_MODE
    Value      PinName
      --      --
  Test_Mode: ATSPEED_MODE
    Value      PinName
      0      in[0] (input)
  Clock: SHIFT_MODE
  pin_name ; direction ; clock_edge ; clock_frequency ; flip-
flop_count ; root_clock ; root_clock_frequency ; multiplier
clk ; input ; rtz ; -- ; 0 ; top1.clk1 ; -- ; --
  Clock: CAPTURE_MODE
  pin_name ; direction ; clock_edge ; clock_frequency ; flip-
flop_count ; root_clock ; root_clock_frequency ; multiplier
clk ; input ; rtz ; -- ; 0 ; top1.clk1 ; -- ; --
  Clock: ATSPEED_MODE
  pin_name ; direction ; clock_edge ; clock_frequency ; flip-
flop_count ; root_clock ; root_clock_frequency ; multiplier
clk ; input ; rtz ; DSM_DEFAULT_FREQUENCY x 1.00 ; 0 ;
top1.clk1 ; DSM_DEFAULT_FREQUENCY ; 1.00
  Control: CAPTURE_MODE
    Value      PinName
      yyn      clk (input)
      nnn      gclk (output)
      yyy      in[2:0] (input)
  Observe: CAPTURE_MODE
    Value      PinName
      n      clk (input)
      n      gclk (output)
      n      in[2:0] (input)
#####
The dft_dsm_ip_report is generated in the <cwd>/spyglass_reports/dft_dsm/

```

directory.

ff_async_reconvergence

The ff_async_reconvergence.rpt file is generated by the [Atspeed_25](#) rule. This report contains a list of all combinational paths that re-convergence at async pins of flip-flops.

A sample format of ff_async_reconvergence report is shown below:

```
#####
Format :
# The name of the start net and end net are given
# followed by the details of the reconvergence path
#####

Reconvergence Detected
    Start net :-
'cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortex
m3_45_9t_hvt_inst.n_3017'
    End net   :-
'cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortex
m3_45_9t_hvt_inst.uCortexM3.uCM3MPU.AlignedDrive'
    Actual (RTL) start net :-
'cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortex
m3_45_9t_hvt_inst.uCortexM3.n123'
    Actual (RTL) end net   :-
'cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortex
m3_45_9t_hvt_inst.uCortexM3.HRESETn'
    FIRST PATH :-
        Terminal = Q
        Net =
cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortexm
3_45_9t_hvt_inst.n_3017
        Terminal = A2
        Instance =
cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortexm
3_45_9t_hvt_inst.U653
        Terminal = ZN
        Net =
cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortexm
3_45_9t_hvt_inst.n1087
```

```
Terminal = A2
Instance =
cortexm3_45_9t_hvt_ctag.cortexm3_45_9t_hvt_crux_inst.cortexm
3_45_9t_hvt_inst.U555
```

The `ff_async_reconvergence.rpt` file is generated in the
<*current-working-directory*>/spyglass_reports/dft/> directory.

ff_clock_reconvergence

The `ff_clock_reconvergence.rpt` file is generated by the [Atspeed_30](#) rule and lists all re-convergent combinational paths. These paths start from a primary input port or the output pin of a flip-flop and re-converge at the clock pin of a flip-flop.

A sample format of the `ff_clock_reconvergence` report is shown below:

```
#####
#Info :
# This report file reports generated by both rules
# Clock_28 in capture mode and Atpseed_30 in capture_atseed
# mode
#####

#####
# Purpose :
# List of all combinational path which reconvergences
# at clock pins of flip-flops.
#####

#####
# Format :
# The name of the start net and end net are given
# followed by the details of the reconvergence path
#####
Reconvergence Detected

    Start net :- 'test.clk1'
    End net   :- 'test.w3'
    Actual (RTL) start net :- 'test.clk1'
    Actual (RTL) end net   :- 'test.w3'

FIRST PATH :-
    Port = clk1
    Net = test.clk1
    Terminal = in1
```

```
Instance = test.rtlc_I9
Terminal = Z
Net = test.w2
Terminal = in2
Instance = test.rtlc_I11
Terminal = Z
Net = test.w3
Terminal = in2
Instance = test.rtlc_I4
Terminal = Z
Net = test.w5
Terminal = in2
Instance = test.rtlc_I17
Terminal = Z
Net = test.w6
Terminal = CP
```

SECOND PATH :-

```
Port = clk1
Net = test.clk1
Terminal = in1
Instance = test.rtlc_I7
Terminal = Z
Net = test.w1
Terminal = in1
Instance = test.rtlc_I11
Terminal = Z
Net = test.w3
Terminal = in2
Instance = test.rtlc_I4
Terminal = Z
Net = test.w5
Terminal = in2
Instance = test.rtlc_I17
Terminal = Z
Net = test.w6
Terminal = CP
```

Reconvergence Detected

```
Start net :- 'test.w3'  
End net   :- 'test.w6'  
Actual (RTL) start net :- 'test.w3'  
Actual (RTL) end net   :- 'test.w6'
```

FIRST PATH :-

```
Terminal = Z  
Net = test.w3  
Terminal = in2  
Instance = test.rtlc_I4  
Terminal = Z  
Net = test.w5  
Terminal = in2  
Instance = test.rtlc_I17  
Terminal = Z  
Net = test.w6  
Terminal = CP
```

SECOND PATH :-

```
Terminal = Z  
Net = test.w3  
Terminal = in2  
Instance = test.rtlc_I3  
Terminal = Z  
Net = test.w4  
Terminal = in1  
Instance = test.rtlc_I17  
Terminal = Z  
Net = test.w6  
Terminal = CP
```

```
#####
```

```
#SUMMARY :  
#design : test  
#mode   : capture_atspeed  
#       2 reconverging logic found.
```

#####

Reconvergence Detected

```
Start net :- 'test.clk1'  
End net   :- 'test.w3'  
Actual (RTL) start net :- 'test.clk1'  
Actual (RTL) end net   :- 'test.w3'
```

FIRST PATH :-

```
Port = clk1  
Net = test.clk1  
Terminal = in1  
Instance = test.rtlc_I9  
Terminal = Z  
Net = test.w2  
Terminal = in2  
Instance = test.rtlc_I11  
Terminal = Z  
Net = test.w3  
Terminal = in2  
Instance = test.rtlc_I4  
Terminal = Z  
Net = test.w5  
Terminal = in2  
Instance = test.rtlc_I17  
Terminal = Z  
Net = test.w6  
Terminal = CP
```

SECOND PATH :-

```
Port = clk1  
Net = test.clk1  
Terminal = in1  
Instance = test.rtlc_I7  
Terminal = Z  
Net = test.w1  
Terminal = in1
```

```
Instance = test.rtlc_I11
Terminal = Z
Net = test.w3
Terminal = in2
Instance = test.rtlc_I4
Terminal = Z
Net = test.w5
Terminal = in2
Instance = test.rtlc_I17
Terminal = Z
Net = test.w6
Terminal = CP
```

Reconvergence Detected

```
Start net :- 'test.w3'
End net   :- 'test.w6'
Actual (RTL) start net :- 'test.w3'
Actual (RTL) end net   :- 'test.w6'
```

FIRST PATH :-

```
Terminal = Z
Net = test.w3
Terminal = in2
Instance = test.rtlc_I4
Terminal = Z
Net = test.w5
Terminal = in2
Instance = test.rtlc_I17
Terminal = Z
Net = test.w6
Terminal = CP
```

SECOND PATH :-

```
Terminal = Z
Net = test.w3
Terminal = in2
Instance = test.rtlc_I3
```

```
Terminal = Z
Net = test.w4
Terminal = in1
Instance = test.rtlc_I17
Terminal = Z
Net = test.w6
Terminal = CP
```

```
#####
```

```
#SUMMARY :
#design : test
#mode : capture
#      2 reconverging logic found.
#####
```

The ff_clock_reconvergence.rpt file is generated in the <cwd>/spyglass_reports/dft_dsm/ directory.

no_atspeed

The *no_atspeed* report is generated by the *Info_noAtspeed* rule. This report lists all the flip-flops that are marked as *no_atspeed* using the *no_atspeed* constraint.

The following is a sample *no_atspeed* report:

```
#####
# Format :
#   Design Name : design
#   'number of no-atspeed objects' <sgdc command>
#   FF | Latch | BB: Object name
|#####

#####
Design Name : "top"

      '3' objects are marked as 'no_atspeed' due to constraint
'no_atspeed -clock_control clk01'
      FLOP: top.u_B_1.u_A_1.ff1_reg
      FLOP: top.u_B_1.u_A_1.xbist_ff_reg
      FLOP: top.u_B_1.u_A_1.BIST_reg

      '3' objects are marked as 'no_atspeed' due to constraint
'no_atspeed -reset_control rst11'
      FLOP: top.u_B_1.u_A_2.ff1_reg
      FLOP: top.u_B_1.u_A_2.xbist_ff_reg
      FLOP: top.u_B_1.u_A_2.BIST_reg

      '3' objects are marked as 'no_atspeed' due to constraint
'no_atspeed -set_control set21'
      FLOP: top.u_B_2.u_A_1.ff1_reg
      FLOP: top.u_B_2.u_A_1.xbist_ff_reg
      FLOP: top.u_B_2.u_A_1.BIST_reg

      '4' objects are marked as 'no_atspeed' due to constraint
'no_atspeed -register_suffix ff'
      FLOP: top.u_B_1.u_A_1.xbist_ff_reg
      FLOP: top.u_B_1.u_A_2.xbist_ff_reg
      FLOP: top.u_B_2.u_A_1.xbist_ff_reg
      FLOP: top.u_B_2.u_A_2.xbist_ff_reg

SUMMARY:
'13' objects are marked as 'no_atspeed' for design "top"
#####
```

FIGURE 12. The *no_atspeed* Report

random_pattern_coverage

The *random_pattern_coverage.rpt* report lists the coverage estimate associated with each module.

The `random_pattern_coverage` report is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

This report displays the following data:

- A definition of different types of faults
- Tabular information of the count of faults and coverage estimate associated with each module in the design

The following figure shows a sample report:

```
<>TOP MODULE SUMMARY for 'test'
```

Fault Heads	Total	Ports	Internal
Total faults	32	12	20
SyRd faults	0	0	0
Faults Considered	32	12	20
Un-Used (UU)	0	0	0
Tied (TI)	0	0	0
Blocked (BL)	0	0	0
Logical Redundant faults (LR)	0	0	0
Un-Testable (UT)	0	0	0
Un-Detectable (!DT)	14	6	8
Potential Detectable (PT)	14	6	8
ATPG_credit	0.00	0.00	0.00
POSDetect_credit	0.00	0.00	0.00
Fault coverage estimate(in %)	56.2	50.0	60.0
Test coverage estimate(in %)	56.2	50.0	60.0

random_pattern_dc_testpoints

The `random_pattern_dc_testpoints.rpt` is generated by the `Info_random_resistance` rule. This report lists the test points for improving random pattern fault coverage which can be used by the Design Compiler as input. The `random_pattern_testpoints` report is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

The report is not generated when the `dft_rrf_tp_generate_dc_report` parameter is set to off.

The following figure shows a sample report:

```
# test points for target "untestable_logic"
# start time : 00:10:10

1  observe      g2f_mem_U1/GroupA_assembly_u0/S8KX16_U1/
PORT0_MEM_ADD_SEG0[9]  # -ut_logic TC gain: 0.13, Obs gain:
1088, TC: 98.44
...
72  force_01    g2f_mem_U1/ram_rd_data[30]  # -ut_logic TC
gain: 0.00, Cnt gain: 38, Obs gain: 2, TC: 99.88
...
93  observe      g2f_mem_U1/GroupA_assembly_u0/S8KX16_U1/
PORT0_MEM_ADD_SEG0[10] # -ut_logic TC gain: 0.00, Obs gain:
34, TC: 99.97

94  observe      g2f_mem_U1/GroupA_assembly_u0/S8KX16_U2/
TO_MEM_CS_SEG0_BIT0_MUXED_xgu  # -ut_logic TC gain: 0.00,
Obs gain: 24, TC: 99.98
# runtime : 0.5065
# end time : 00:10:10

# test points for target "random_resistant"
# start time : 00:10:20
# Design : "tc5"
# Initial Random Pattern Test Coverage : 86.80093
# Stuck At Test Coverage : 98.30295
# Random Pattern Count : 6000
# Effort Level : low
```

```

# Requested Test Points : 500
# Threads requested : 8
# Threads created : 7 (excluding main thread)
95 control-0 ccd_U1/n15363 #Gain : 1.01147 Cov : 87.81239
S@TC : 98.3
...
591 observe g2f_mem_U1/g2f_queue_u3/g2f_queue_fsm_u0/n22
#Gain : 0.00292 Cov : 97.09572 S@TC : 98.3
592 observe g2f_mem_U1/g2f_queue_u1/g2f_queue_fsm_u0/n22
#Gain : 0.00292 Cov : 97.09863 S@TC : 98.3
593 observe g2f_mem_U1/g2f_queue_u4/g2f_queue_fsm_u0/n22
#Gain : 0.00292 Cov : 97.10155 S@TC : 98.3
594 observe g2f_mem_U1/g2f_queue_u5/g2f_queue_fsm_u0/n22
#Gain : 0.00291 Cov : 97.10446 S@TC : 98.3
# Test point search completed : 500 (dft_rrf_tp_count(3% -
500)) test points identified.
# runtime : 49.7882
# end time : 00:11:10

# test points for target "x_blocking"
# start time : 00:11:15
595 force_01 g2f_mem_U1/ram_rd_data[0] # -xsource -
cell_name g2f_mem_U1/GroupA_assembly_u0/S8KX16_U1/S8KX16_U1
...
625 force_01 g2f_mem_U1/ram_rd_data[24] # -xsource -
cell_name g2f_mem_U1/GroupA_assembly_u0/S8KX16_U2/S8KX16_U1
626 force_01 g2f_mem_U1/ram_rd_data[25] # -xsource -
cell_name g2f_mem_U1/GroupA_assembly_u0/S8KX16_U2/S8KX16_U1
# runtime : 0.1302
# end time : 00:11:15

#+----- TARGET-WISE TEST POINT SUMMARY -----
# target : random_resistant
# requested test point count : 500
# reported test point count : 500
# termination reason : 500 (dft_rrf_tp_count(3% -
500)) test points identified.
# runtime : 49.7882

```

```
# target : untestable_logic
#   requested test point count : 500
#   reported test point count  : 94
#   runtime                    : 0.5065
# target : x_blocking
#   reported test point count  : 32
#   runtime                    : 0.1302
# target : isolation
#   reported test point count  : 0
#   runtime                    : 0.0000
# target : multicycle_paths
#   reported test point count  : 0
#   runtime                    : 0.0000
#+-----+
#+----- UNIFIED TEST POINT SUMMARY -----+
# total test points   : 626
# total runtime      : 50.4249
#+-----
```

If the [Info_self_gating_logic](#) rule is run and the value of the [dft_rrf_tp_generate_dc_report](#) parameter is set to off, the following additional data is generated in the [random_pattern_dc_testpoints](#) report:

```

# Self-Gating Logic Test-Point Selection
# Index Test_Point_Type Test_Point Test_Point_Driver Test_Point_Group_ID
Comment
1 control-1 I_REG_FILE.n191 - I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN
-root_clock __no_clock_found__ -clock __no_clock_found__ #control-1 test-
point
2 observe I_REG_FILE.n191 - I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN
-root_clock Clk -clock I_REG_FILE.Reg_Array_reg_0__4__._CLK #observe test-
point
3 control-0 I_REG_FILE.n208 I_REG_FILE.Reg_Array_1__15__
I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN -root_clock __no_clock_found__
-clock __no_clock_found__ #control-0 test-point
4 control-0 I_REG_FILE.n209 I_REG_FILE.Reg_Array_1__0__
I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN -root_clock __no_clock_found__
-clock __no_clock_found__ #control-0 test-point
5 control-0 I_REG_FILE.n201 I_REG_FILE.Reg_Array_1__1__
I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN -root_clock __no_clock_found__
-clock __no_clock_found__ #control-0 test-point
6 control-0 I_REG_FILE.n203 I_REG_FILE.Reg_Array_1__2__
I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN -root_clock __no_clock_found__
-clock __no_clock_found__ #control-0 test-point
7 control-0 I_REG_FILE.n204 I_REG_FILE.Reg_Array_1__3__
I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN -root_clock __no_clock_found__
-clock __no_clock_found__ #control-0 test-point
8 control-0 I_REG_FILE.n202 I_REG_FILE.Reg_Array_1__4__
I_REG_FILE.self_gate_Reg_Array_reg_1__0.EN -root_clock __no_clock_found__
-clock __no_clock_found__ #control-0 test-point

```

This testpoint information is used as an input by the Design Compiler. However, this information is not reported when the value of the

random_pattern_node_distribution

The *random_pattern_node_distribution.rpt* report lists all the thresholds, for all three probabilities, and the node distribution for each bucket.

The *random_pattern_node_distribution* report is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

The following figure shows a sample report:

```
#####
# Format :
# Column 1: bucket_id
# Column 2: Node count for which 'detect' probability [min(pdt0, pdt1)] is
#           falling in the bucket detect probability range
# Column 3: Node count for which 'control_0' probability is falling in the
#           bucket control probability range
# Column 4: Node count for which 'control_1' probability is falling in the
#           bucket control probability range
# Column 5: Node count for which 'observe' probability is falling in the
#           bucket observe probability range
#####

Node Distribution Summary : sample_block
-----
```

bucket id	pdt node count	cnt_0 node count	cnt_1 node count	obs node count
0	361	1	107	888
1	10	0	573	2826
2	259	0	2	6870
3	847	0	0	52
4	0	0	321	94
5	18	0	12	99
6	0	0	2	29
7	12	0	2	50

FIGURE 13. The random_pattern_node_distribution report

Also, when you set the value of the `dft_rrf_generate_fault_report` parameter to on to generate the *Node List for Probability Distribution* section as shown in Figure 10:

Node List for probability distribution

```

-----
pdt-ID Cnt0-ID Cnt1-ID Obs-ID  cell-type                pin/port-name
-----
  0      100      0      82    or                sample_block.rtlc_I429.in1
  0      100      0      100   RTL_XOR           sample_block.rtlc_I432.in2
  0      100      0      100   RTL_XOR           sample_block.rtlc_I431.Z
  0      100      0      100   RTL_XOR           sample_block.rtlc_I431.in2
  0      100      0      100   RTL_XOR           sample_block.rtlc_I407.in1
  0      100      0      100   RTL_XOR           sample_block.rtlc_I407.in2
  0      100      0      100   RTL_XOR           sample_block.rtlc_I407.Z
  0      100      0      0     and               sample_block.rtlc_I408.in1
  0      100      0      82    or                sample_block.rtlc_I428.in3
  0      100      0      0     and               sample_block.rtlc_I408.in2
  0      100      0      100   RTL_XOR           sample_block.rtlc_I465.in1
  
```

FIGURE 14. Node List for Probability Distribution

random_pattern_testpoints

The *random_pattern_testpoints.rpt* is generated by the [Info_random_resistance](#) rule. This report lists the testpoints for improving random pattern fault coverage.

The `random_pattern_testpoints` report is generated in the `<cwd>/spyglass_reports/dft_dsm/` directory.

```

#####
# Purpose :
#   Random Pattern Testpoints Report.
#   Lists testpoints for improving random pattern fault
coverage.
#####

#####
# Parameter values :
#   'dft_rrf_tp_count'           : 15
#   'dft_rrf_tp_type'           : both
#   'dft_rrf_tp_effort_level'   : medium
#   'dft_rrf_tp_ignore_generated_nets' : on
#####

#####
# Design information :
#   Top module : 'horizontal_mux'
#####

current_design "horizontal_mux"

force_probability -name "horizontal_mux.n18869" -control_one
0.625000 -control_zero 0.375000 # gain : 1.0, tc : 97.2, fc :
94.3, s@tc : 100.0
force_probability -name "horizontal_mux.n13345" -observe 1.000000
# gain : 0.0, tc : 99.2, fc : 96.3, s@tc : 100.0
.
.
force_probability -name "horizontal_mux.n13527" -observe 1.000000
# gain : 0.0, tc : 99.2, fc : 96.4, s@tc : 100.0
force_probability -name "horizontal_mux.n13436" -observe 1.000000
# gain : 0.0, tc : 99.2, fc : 96.4, s@tc : 100.0
# Test point search completed : 15 (dft_rrf_tp_count(15 - 15))
test points identified.

```

FIGURE 15. Random Pattern Testpoints Report

transition_coverage

The transition_coverage.rpt file is generated by the [Info_transitionCoverage](#) rule. This report contains the summary and details of transition fault numbers.

A sample format of transition_coverage report is shown below:

```
#####
# Format :
#   The definition of various kinds of transition
#   faults is given. This is followed by a tabular
#   information of the count of transition faults
#   associated with each module in the design.
#####
<>ABBREVIATIONS & DEFINITIONS OF TERMS :
  1. hierarchical instance : an instance which has other
hierarchical instances and/or basic cells within it.
  2. DT fault : Faults which are either 1->0 transition
detectable or 0->1 detectable.
  3. !DT/ND fault : Faults which are neither 1->0 transition
detectable or 0->1 detectable.
  4. UT fault : Faults which are
      1. Tied to constant value
      2. Driving only asynchronous pins of scan
flip-flops
      3. Acting as clock enable logic.
  5.UU fault : An unused fault. Fault at a terminal
becomes UU if that terminal cannot be reached from the
output ports. They are ignored because they cannot affect
system operation. By default, latches / blackboxes, whose
outputs fanout to SyRd nodes only, and their fanin cone
can be UU. If parameter dftUUMarking is set to 'on' then
flip-flops, whose outputs fanout to SyRd nodes only, and
their fanin cone can be UU.
  6. SyRd fault : Faults at terminals which are
Sy[= synthesis]Rd[= redundant]. These are terminals which
are likely to be removed by an optimizing synthesizer,
and hence should not be considered for fault/test coverage
analysis. These are of the following types:
```

- a) Pins which are tied high/low,
 - b) Pins blocked in power ground mode, and
 - c) Unconnected combinational logic - all logic combinationally connected to nodes that do not fanout to primary port.
7. LR fault : These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.
 8. FP fault : These are the faults which lie on false/multi-cycle path False paths can be specified through SDC file or through 'false_path' sgdc command.
 9. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets. If a net is either tied to 0 or 1, both t/01 and t/10 faults on the corresponding fault-node are TIED. An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked. For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

$$\text{Fault-Coverage(FC) \%} = \frac{\text{Transition Detectable Fault Count}}{(2 * \text{terminalCount}) - \text{SyRd_faultCount} - \text{FP Count}} \times 100$$

$$\text{Test-Coverage(TC) \%} = \frac{\text{Transition Detectable Fault Count}}{(2 * \text{terminalCount}) - \text{SyRd_faultCount} - \text{TICount} - \text{BLCount} - \text{UUCount} - \text{FPCount} - \text{LRCOUNT}} \times 100$$

<>For top module the summary has been reported in three columns as follows:

Column 1: fault and coverage data due to top-level ports and internal design.

Column 2: fault and coverage data due to top-level ports only.

Column 3: fault and coverage data due to internal design only.

<>For each module and hierarchical instance, eleven numbers are reported. These are (in order)

1. Count of terminals (IT) within the module (or instance)
2. Count of synth-redundant faults (SR) within the module (or instance)
3. Count of faults considered (FC) within the module (or instance)
4. Count of unused (UU) faults within the module (or instance)
5. Count of tied faults (TI) faults within this module (or instance)
6. Count of blocked faults (BL) faults within this module (or instance)
7. Count of false path faults (FP) faults within this module (or instance)
8. Count of logical redundant fault (LR) within this module (or instance)
9. Count of unstable faults (UT) faults within this module (or instance)
10. Count of detectable (DT) faults within this module (or instance)
11. Count of undetectable (!DT) faults within this module (or instance)

<> Alongside each module (and hierarchical instance) there is a search-key, of the form

Module_[moduleIndex] | Inst_[moduleIndex]_[instIndex]

When opened in a text-editor, this key can be used to jump to the next (and only other) reference to the module (or instance) in the file. If an instance does NOT have a key

then it means that this hierarchical instance has no hierarchical instances within it.

PS:The instance names have been wrapped around to the next line in order to maintain the clarity of the Table.

```

----- END -----
<>TOP MODULE SUMMARY for 'top'|
ATSPEED LAUNCH METHOD : 'LAUNCH_ON_CAPTURE'
Fault Heads          |   Total   |   Ports   | Internal |
Total faults        |         66 |         18 |         48|
SyRd faults         |          0 |          0 |          0|
Faults Considered   |         66 |         18 |         48|
Un-Used (UU)        |          4 |          0 |          4|
Tied (TI)           |          0 |          0 |          0|
Blocked (BL)        |          0 |          0 |          0|
False path faults (FP)|         0 |          0 |          0|
Logical Redundant
faults (LR)         |          0 |          0 |          0|
Un-Testable (UT)    |          0 |          0 |          0|
Detectable (DT)     |          6 |          2 |          4|
Un-Detectable (!DT) |         56 |         16 |         40|
Fault-coverage(in %)|         9.1|        11.1|         8.3|
Test-coverage(in %)|         9.7|        11.1|         9.1|

```

Coverage Summary for each Instance:

```

-----
-----
Columns  IT   SR   FC  UU  TI  BL   FP   LR   UT  DT   !DT
-----
-----

```

<>TOP MODULE 'top'

Child-Instances

I2.....5	0	10	2	0	0	0	0	0	2	6
I3.....5	0	10	2	0	0	0	0	0	2	6
I0.....3	0	6	0	0	0	0	0	0	0	6
I1.....2	0	4	0	0	0	0	0	0	0	4
I4.....3	0	6	0	0	0	0	0	0	0	6
I5.....3	0	6	0	0	0	0	0	0	0	6
I6.....3	0	6	0	0	0	0	0	0	0	6
Inferred_										
LOGIC.....0	0	0	0	0	0	0	0	0	0	0

The transition_coverage.rpt file is generated in the
 <cwd>/spyglass_reports/dft_dsm/ directory.

transition_coverage_audit

The transition_coverage_audit.rpt file is generated by the [Info_transitionCoverage_audit](#) rule. This report contains the predicted improvement in transition coverage number after each major step.

A sample format of transition_coverage_audit report is shown below:

```
#####

# Format :
# The fault and test coverage are listed. Then a series of
# steps that identify causes for < 100 percentage are given.
# If any of these steps does not apply, it will be marked as
# "No action needed".
#####
Explanation of each step in the order that they appear in
the report:|
1. Non scan flip-flops      : Non scan flip-flops reduce
                             circuit controllability and
                             observability. Use Async_07/
                             Clock_11 to debug
2. Black box                : Terminals of non-scan-wrapped
                             BB's are unobservable and
                             uncontrollable
3. 'no_scan' flip-flops    : 'force_no_scan' and inferred no
                             scan flip-flops cannot be used
                             for atspeed testing
4. At speed domains of
   scan flip-flops         : Scannable flip-flops must be
                             clocked by atspeed testclocks
                             Use Atspeed_11
5. Use PIs and POs        : Set dsmUsePIForAtSpeed to 'on'
                             to use primary inputs for launch
                             Set dsmUsePOForAtSpeed to 'on'
                             to use primary outputs for capture
6. Uncontrollable scan
   flip-flop d-pins        : Uncontrollable scan flip-flop
                             data pins prevent launch of
                             transition edges. Use Atspeed_09
```


- 7. Clock domain crossing : Faults near domain crossings are not detectable
- 8. 'no_atspeed' faults : User-specified 'no_atspeed' faults are not considered detectable
- 9. Atspeed testmode value : Testmode values in atspeed mode restrict ATPG
- 10. Tristate enable : Enable pins of tristate devices are unobservable
- 11. 'force_ta' constraints: forca_ta reduces testability of a design
- 12. Hanging terminals : Hanging terminals reduces controllability
- 13. Feedback flip-flops : Flip-flop q to d feedback may restrict launch transitions
- 14. False and multicycle paths : Faults on false paths and multicycle paths are not detectable. Use parameter `dsm_count_false_path_transition_faults` to control their impact on transition fault and test coverage
- 15. Undetected faults due : Power ground values restrict ATPG to PG nets

Transition fault coverage and test coverage in percentage are reported for each top module.

#####

For top module 'topnodes'
 Atspeed launch method 'LAUNCH_ON_CAPTURE'

Base transition fault coverage :60.9
 Base transition test coverage :71.0

Expected transition coverage data after each step	Expected transition coverage -----	Action
---	---------------------------------------	--------

	FC	TC	
1. Flip-flops made scannable	68.2	79.5	No action required
2. Black-box made scanwrapped	69.2	80.7	No action required
3. 'no_scan' flip-flops made scannable	73.8	86.1	No action required
4. Atspeed test clocks	75.5	88.0	No action required
5. PI and PO used for launch and capture	75.5	88.0	No action required
6. Controllable flip-flop d-pins	75.5	88.0	No action required
7. Clock domain crossing	76.5	89.2	No action required
8. 'no_at-speed' faults removed	78.5	91.5	No action required
9. Atspeed test mode	80.0	93.2	Info_testmode && Info_untestatble
10. Tristate enables	81.0	94.4	TA_09
11. Undetected faults due to 'force_ta'	83.3	97.1	TA_09
12. Hanging terminals	84.6	98.6	TA_09
13. Flip-flops with q to d feedback	84.6	98.6	No action required
14. False path faults	84.6	98.6	No action required
15. Undetected faults due to PG nets	85.8	100.0	Info_pwrGndSim, Info_syntRedundant & Atspeed_09

#####

<> For Top Module 'topnodes'

Percentage of flip-flops which have at least one reason
for non-robustness :0.0% (0/29)

Percentage of flip-flops tagged by each rule are listed
below.

Rule	Percentage	Ratio(tagged/total)
Atspeed_03	0.0	0/29
Atspeed_04	0.0	0/29
Atspeed_05	0.0	0/29
Atspeed_14	0.0	0/29
Atspeed_22	0.0	0/29
Atspeed_25	0.0	0/29
Atspeed_30	0.0	0/29
CG_07	0.0	0/29

The transition_coverage_audit.rpt file is generated in the
<cwd>/spyglass_reports/dft_dsm/ directory.

transition_coverage_clockdomain

The *transition_coverage_clockdomain.rpt* file is generated by the [Info_transitionCoverage](#) rule. It lists number of different type of faults in a clock domain along with the fault coverage and test coverage within the domain. The report also shows a quick summary of the data below the detailed information.

A sample format of *transition_coverage_clockdomain* report is as shown below:

```
#####

# Purpose :
#   Report Type : Coverage Report
#   Fault Coverage report Grouped by Clock Domain
#####

#####

# Format :
#   Lists all the types of Faults in the left column
#   Lists the corresponding fault count in the right column
#####

<>ABBREVIATIONS & DEFINITIONS OF TERMS :
  1. hierarchical instance : an instance which has other
     hierarchical instances and/or basic cells within it.
  2. DT fault      :   Faults which are either 1->0 transition
     detectable or 0->1 detectable.
  3. !DT/ND fault :   Faults which are neither 1->0
     transition detectable or 0->1 detectable.
  4. UT fault      :   Faults which are
     1. Tied to constant value
     2. Driving only asynchronous pins of
        scan flip-flops
     3. Acting as clock enable logic.
```

5. UU fault : An unused fault. Fault at a terminal becomes UU if that terminal cannot be reached from the output ports. They are ignored because they cannot affect system operation. By default, latches / blackboxes, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking is set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.

6. SyRd fault : Faults at terminals which are Sy[= synthesis]Rd[= redundant]. These are terminals which are likely to be removed by an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis.

These are of the following types:

- a) Pins which are tied high/low,
- b) Pins blocked in power ground mode, and
- c) Unconnected combinational logic - all logic combinationally connected to nodes that do not fanout to primary port.

7. LR fault : These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.

8. FP fault : These are the faults which lie on false/multi-cycle path False paths can be specified through SDC file or through 'false_path' sgdc command.

9. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets. If a net is either tied to 0 or 1, both t/01 and t/10 faults on the corresponding fault-node are TIED. An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked. For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

$$\text{Fault-Coverage \%} = \frac{\text{Transition Detectable Fault Count}}{\text{-----}} \times 100$$

(2*terminalCount)-FPCount-
SyRd_faultCount

Transition Detectable Fault Count
 Test-Coverage % = -----x 100
 (2*terminalCount)-SyRd_faultCount-
 TICount-BLCount-UUCount-FPCount-LRCount

<>For each module and hierarchical instance, eleven numbers are reported. These are (in order)

1. Count of FlipFlop (FF) within the domain
2. Count of Total Possible fault (TF) within the domain
3. Count of Syrd faults(SR) within the domain
4. Count of unused (UU) faults within the domain
5. Count of tied faults (TI) faults within this module (or instance)
6. Count of blocked faults (BL) faults within this module (or instance)
7. Count of false path faults (FP) faults within the domain
8. Count of Logical redundant (LR) faults within the domain
9. Count of untestable faults (UT) faults within the domain
10. Count of Detectable faults (DT) within the domain
11. Count of undetectable (!DT) faults within the domain

----- END -----

TOP-MODULE : top
 ATSPEED LAUNCH METHOD : LAUNCH_ON_CAPTURE

 Start of coverage report of Domain id : 1

DomainId : 1
 DomainName : top.clk
 ClockName : top.clk

Fault Types		Total Count
-------------	--	-------------

Flip flop(FF)		2 (100.00 % of total)
Total possible Faults(TF)		4
Syrd Fault(SR)		0
UnUsed Fault(UU)		0
Tied Fault(TI)		0
Blocked Fault(BL)		0
False Path Fault(FP)		0
Logical Redundant		
faults (LR)		0
Untestable Fault(UT)		0
Detectable Faults(DT)		4(100.00 % of total)
Undetectable Fault(!DT)		0
Fault-coverage(in %)		100.00
Test-coverage(in %)		100.00

End of coverage report of Domain id : 1

#####

```
# Summary :
# Total domains : 1
# FC/TC for domain id 1 are : 100.00/100.00
```

#####

--End of Top-Module--

The *transition_coverage_clockdomain.rpt* file is generated in the
 <cwd>/spyglass_reports/dft_dsm/ directory.

transition_coverage_collapsed

The *transition_coverage_collapsed.rpt* report contains the transition coverage summary with fault collapsing. This report is generated by the [Info_transitionCoverage](#) rule when the value of the *dft_collapse_equivalent_faults* parameter is set to on.

Accessing the Report

To generate the *transition_coverage_collapsed* report, set the value of the parameter *dft_collapse_equivalent_faults* to on.

To view the report, refer to the following directory:

```
<current-working-directory>/spyglass_reports/dft_dsm/>
```

Alternatively, to view the report from GUI, click the **Report** Icon in the Results pane under the **Analyze Results** tab, and select the required report from the **DFT** menu.

Sample Report

A sample *transition_coverage_collapsed* report follows:

```
#####
# Format :
# The definition of various kinds of transition faults is
# given. This is followed by a tabular information of the
# count of transition faults in the design.
#####
<>ABBREVIATIONS & DEFINITIONS OF TERMS :
  1. DT fault : Faults which are either 1->0 transition
              detectable or 0->1 detectable.
  2. !DT/ND fault:Faults which are neither 1->0 transition
              detectable or 0->1 detectable.
  3. UT fault : Faults which are
              1. Tied to constant value
              2. Driving only asynchronous pins of scan
                 flip-flops
              3. Acting as clock enable logic.
  4.UU fault   : An unused fault. Fault at a terminal
              becomes UU if that terminal cannot be reached from the
              output ports. They are ignored because they cannot affect
              system operation. By default, latches / blackboxes, whose
```


outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking is set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.

5. SyRd fault : Faults at terminals which are Sy[= synthesis]Rd[= redundant]. These are terminals which are likely to be removed by an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis. These are of the following types:
 - a) Pins which are tied high/low,
 - b) Pins blocked in power ground mode, and
 - c) Unconnected combinational logic - all logic combinationally connected to nodes that do not fanout to primary port.
7. LR fault : These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.
8. FP fault : These are the faults which lie on false/multi-cycle path False paths can be specified through SDC file or through 'false_path' sgdc command.
9. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets. If a net is either tied to 0 or 1, both t/01 and t/10 faults on the corresponding fault-node are TIED. An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked. For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

$$\text{Fault-Coverage(FC) \%} = \frac{\text{Transition Detectable Fault Count}}{\text{-----x 100}}$$

```

Total Collapsed Fault Count -SyRd_faultCount

Transition Detectable Fault Count
Test-Coverage(TC)% =-----x 100
                    Total Collapsed Fault Count -
                    SyRd_faultCount - TICount - BLCount -
                    UUCount - LRCount
    
```

```

<>TOP MODULE SUMMARY for 'top'
  ATSPPEED LAUNCH METHOD : 'LAUNCH_ON_CAPTURE'
    
```

Total faults	64
SyRd faults	0
Faults Considered	64
Un-Used (UU)	10
Tied (TI)	16
Blocked (BL)	4
False path faults (FP)	4
Logical Redundant faults (LR)	0
Un-Testable (UT)	0
Detectable (DT)	30
Un-Detectable (!DT)	0
Fault-coverage(in %)	46.9
Test-coverage(in %)	88.2

transition_faults

The transition_faults.rpt file is generated by the [Info_transitionCoverage](#) rule. This is a pin-based fault report for transition faults.

NOTE: Ensure that the [dsmGenerateTransitionFaultReport](#) parameter is set to a value other than none to enable the generation of the transition_faults report.

A sample format of the transition_faults report is shown below, which is generated by default, is shown below:

```

#####

# Purpose :
```

```
# Report type          : Transition coverage
# Transition faults at ports and terminals.
#####
```

<>ABBREVIATIONS & DEFINITIONS OF TERMS :

1. hierarchical instance : an instance which has other hierarchical instances and/or basic cells within it.
2. DT fault : Faults which are either 1->0 transition detectable or 0->1 detectable.
3. !DT/ND fault : Faults which are neither 1->0 transition detectable or 0->1 detectable.
ND faults can be further categorised based on their cause.

Abbreviation	Sub Category
-----	-----
ND_CLF	: CONTROL_LINE_FAULTS
ND_NSF	: NON_SCAN_FF
ND_BB	: BLACK_BOX
ND_NSC	: NO_SCAN_CONST
ND_AC	: NO_ATSPEED_CLOCK
ND_PI	: PI_PO
ND_BD	: BAD_D_PIN
ND_DC	: DOMAIN_CROSSING
ND_NAF	: NO_ATSPEED_FAULT
ND_ATM	: ATSPEED_TESTMODE
ND_FP	: FALSE_PATH_SDC
ND_TE	: TRISTATE_ENABLE
ND_FTA	: FORCE_TA
ND_HT	: HANGING_TERM
ND_QD	: Q_TO_D_FEEDBACK
ND_PG	: PG_NETS

4. UT fault : Faults which are
 1. Tied to constant value
 2. Driving only asynchronous pins of scan flip-flops
 3. Acting as clock enable logic.
5. UU fault : An unused fault. Fault at a terminal becomes UU if that terminal cannot be reached from the output

ports. They are ignored because they cannot affect system operation. By default, latches / blackboxes, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking is set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.

6. SyRd fault : Faults at terminals which are Sy[= synthesis]Rd[= redundant]. These are terminals which are likely to be removed by an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis. These are of the following types:=
 - a) Pins which are tied high/low,
 - b) Pins blocked in power ground mode, and
 - c) Unconnected combinational logic - all logic combinationally connected to nodes that do not fanout to primary port.
7. LR fault : These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.
8. FP fault : These are the faults which lie on false/multi-cycle path False paths can be specified through SDC file or through 'false_path' sgdc command.
9. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets. If a net is either tied to 0 or 1, both t/01 and t/10 faults on the corresponding fault-node are TIED. An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked. For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

Transition Detectable
Fault Count

Fault-Coverage(FC) % = -----x 100
 (2*terminalCount) - SyRd_faultCount

Transition Detectable
 Fault Count

Test-Coverage(TC) % = ----- x 100
 (2*terminalCount) - SyRd_faultCount
 - TICount - BLCount - UUCount
 - FPCount - LRCount

<>For top module the summary has been reported in three columns as follows:
 Column 1: fault and coverage data due to top-level ports and internal design.
 Column 2: fault and coverage data due to top-level ports only.
 Column 3: fault and coverage data due to internal design only.

```
#####
# Design information :
#   Top module           : 'top'
#   Flip-flop count      : 2
#   Atspeed testclock count: 1
#   dsmUsePIForAtSpeed   : on
#   dsmUsePOForAtSpeed   : on
#####
```

<>TOP MODULE SUMMARY for 'top'

Fault Heads	Total	Ports	Internal
Total faults	36	8	28
SyRd faults	0	0	0
Faults Considered	36	8	28
Un-Used (UU)	0	0	0
Tied (TI)	0	0	0
Blocked (BL)	0	0	0

False path			
faults (FP)	0	0	0
Logical Redundant			
faults (LR)	0	0	0
Un-Testable (UT)	0	0	0
Detectable (DT)	20	4	16
Un-Detectable (!DT)	16	4	12
Fault-coverage(in %)	55.6	50.0	57.1
Test-coverage(in %)	55.6	50.0	57.1

fault type	status	cell-type	hier/pin/name
-----	-----	-----	-----

<>Listing 'all' faults for all ports

t/01	DT	primary_input	sysclk1
t/10	DT	primary_input	sysclk1
t/01	ND_BB	primary_input	in1
t/10	ND_BB	primary_input	in1
.			
.			

<>Listing 'all' faults for instance 'top.g1'

t/01	ND_BB	and	top.g1.rtlc_I2.in1
t/10	ND_BB	and	top.g1.rtlc_I2.in1
t/01	ND_BB	and	top.g1.rtlc_I2.Z
t/10	ND_BB	and	top.g1.rtlc_I2.Z
.			
.			

<>Listing 'all' faults for instance 'top.p2'

t/01	DT	clock shaper cell	top.p2.clkin
t/10	DT	clock shaper cell	top.p2.clkin
.			
.			

However, when you set the value of the [dsm_report_domain_for_faults](#) parameter to yes, the transition_faults report also reports the launch and

capture atspeed clock domains. A sample format of the `transition_faults` report is shown below:

<>ABBREVIATIONS & DEFINITIONS OF TERMS :

1. hierarchical instance : an instance which has other hierarchical instances and/or basic cells within it.
2. DT fault : Faults which are either 1->0 transition detectable or 0->1 detectable.
3. !DT/ND fault : Faults which are neither 1->0 transition detectable or 0->1 detectable.
4. UT fault : Faults which are
 1. Tied to constant value
 2. Driving only asynchronous pins of scan flip-flops
 3. Acting as clock enable logic.
5. UU fault : An unused fault. Fault at a terminal becomes UU if that terminal cannot be reached from the output ports. They are ignored because they cannot affect system operation. By default, latches / blackboxes, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU. If parameter dftUUMarking is set to 'on' then flip-flops, whose outputs fanout to SyRd nodes only, and their fanin cone can be UU.
6. SyRd fault : Faults at terminals which are Sy[= synthesis]Rd[= redundant]. These are terminals which are likely to be removed by an optimizing synthesizer, and hence should not be considered for fault/test coverage analysis. These are of the following types:
 - a) Pins which are tied high/low,
 - b) Pins blocked in power ground mode, and
 - c) Unconnected combinational logic - all logic combinationally connected to nodes that do not fanout to primary port.
7. LR fault : These are logically redundant faults. Whatever be the state of circuit, output on such nodes will always be a fixed value or are blocked by such fixed value nodes.
8. FP fault : These are the faults which lie on false/multi-cycle path. False paths can be specified through SDC file or through 'false_path' sgdc command.
9. TI/BL fault : This fault class comprises of faults which arise because of nets being tied to 0 (Low) or 1(High), and the logic blocked by such nets. If a net is either tied to 0 or 1, both t/01 and t/10 faults on the corresponding fault-node are TIED. An input of a combinational gate tied to an appropriate value will make the faults in the fanin-cone of the other input as blocked. For example, an input of an AND gate tied to 0, will make faults in fanin-cone of the other input BLOCKED.

FORMULAE USED :

$$\text{Fault-Coverage(FC) \%} = \frac{\text{Transition Detectable Fault Count}}{(2 * \text{terminalCount}) - \text{SyRd_faultCount}} \times 100$$

$$\text{Test-Coverage(TC) \%} = \frac{\text{Transition Detectable Fault Count}}{(2 * \text{terminalCount}) - \text{SyRd_faultCount} - \text{TICount} - \text{BLCount} - \text{UUCount} - \text{FPCount} - \text{LRCCount}} \times 100$$

<>For top module the summary has been reported in three columns as follows:
 Column 1: fault and coverage data due to top-level ports and internal design.
 Column 2: fault and coverage data due to top-level ports only.
 Column 3: fault and coverage data due to internal design only.

#####

```
# Design information :
#   Top module       : 'izar'
#   Flip-flop count  : 6430
#   Atspeed testclock count: 4
#   dsmUsePIForAtSpeed : on
#   dsmUsePOForAtSpeed : on
```

#####

<>TOP MODULE SUMMARY for 'izar'

Fault Heads	Total	Ports	Internal
Total fault pins	175518	250	175268
SyRd faults	20434	0	20434
Faults Considered	330602	500	330102
Un-Used (UU)	2	2	0
Tied (TI)	0	0	0
Blocked (BL)	0	0	0
False path faults (FP)	0	0	0
Logical Redundant faults (LR)	0	0	0
Un-Testable (UT)	3084	196	2888
Detectable (DT)	323440	302	323138
Un-Detectable (!DT)	4076	0	4076
Fault-coverage(in %)	97.8	60.4	97.9
Test-coverage(in %)	97.8	60.6	97.9

fault type	status	cell-type	hier/pin/name (launch domain::capture domain)
<>Listing 'all' faults for design unit 'izar'			
t/01	DT	primary_input	ipg_hard_async_reset_b (L: - :: C: izar.ipg_clk izar.ipp_ind_refclk)
t/10	DT	primary_input	ipg_hard_async_reset_b (L: - :: C: izar.ipg_clk izar.ipp_ind_refclk)
t/01	DT	primary_input	ipg_clk (L: - :: C: izar.ipg_clk)
t/10	DT	primary_input	ipg_clk (L: - :: C: izar.ipg_clk)
t/01	DT	primary_input	ips_module_en[0] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/10	DT	primary_input	ips_module_en[0] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/01	DT	primary_input	ips_module_en[1] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/10	DT	primary_input	ips_module_en[1] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/01	DT	primary_input	ips_module_en[2] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/10	DT	primary_input	ips_module_en[2] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/01	DT	primary_input	ips_module_en[3] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/10	DT	primary_input	ips_module_en[3] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/01	DT	primary_input	ips_module_en[4] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])
t/10	DT	primary_input	ips_module_en[4] (L: - :: C: izar.ipg_clk izar.ipt_test_wrapper_clk_in[1])

The transition_faults.rpt file is generated in the <cwd>/spyglass_reports/dft_dsm/ directory.

Customizing the SpyGlass DFT ADV Product

Common SpyGlass DFT ADV Rule Parameters

This section lists the parameters used by all the rules of the SpyGlass DFT ADV product.

You can set these parameters by using the following command in SpyGlass Explorer and Tcl Shell Interface:

```
set_parameter <parameter_name> <parameter_value>
```

For more information on setting parameters, refer to the *SpyGlass Tcl Interface User Guide* and *SpyGlass Explorer User Guide*.

The following are the common parameters used by all the rules of the SpyGlass DFT ADV product:

- ***dft_all_scan_chains_defined***: The default value of the parameter is yes or on. Set the value of the parameter to no or off to disable the scannability criteria when the *dftDesignState* parameter is set to post_scan_stitched.
- ***dft_allow_inactive_resets_from_scan_ffs***: The default value is on. Set the value of the parameter to off to report all inactive set/resets, which are drive by scan flip-flops.

- *dft_check_path_name_for_instance_suffix*: The default value of the parameter is off. Set the value of the parameter to on to modify the behavior of `-instance_suffix` field of the *no_fault* or *add_fault* sgdc commands.
- *dft_check_path_name_for_register_suffix*: The default value of the parameter is off. Set the value of the parameter to on to modify the behavior of `-register_suffix` field of the *force_scan*, *force_no_scan*, *no_fault*, or *add_fault* sgdc commands.
- *dft_detect_shadow_latches*: The default value of the parameter is on. Set the value of the parameter to off to disable shadow latch detection.
- *dft_maximum_number_of_messages_with_spreadsheet*: The default value is 1000. Set the value of the parameter to any non-negative integer to set the number of messages generated, per rule, which will have the spreadsheet report.
- *dft_infer_sequential_propagation_as_no_scan*: The default value of the parameter is off. Set the value of the parameter to on to enable sequential propagation of test_mode through scannable flip-flops.
- *dft_ignore_inout_ports_as_driver*: The default value is on. Set the value of the parameter to off to ignore the port's impact on the constant propagation.
- *dft_scannable_latches*: The default value of the parameter is off. Set the value of the parameter to on to mark non-transparent latches as scannable, when the enable pin of the latch has an inferred test clock.
- *dft_set_scan_wrap_on_memory*: The default value of the parameter is on. Set the value of the parameter to off to prevent the application of scan_wrap constraint on memories.
- *dft_show_spreadsheet_path_in_message*: The default value of this parameter is on. Set the value of the parameter to off to remove the spreadsheet path from the violation message.
- *dft_store_tcl_query_data*: The default value of the parameter is sg_shell. Set the value of the parameter to on to preserve the design query data after completion of the goal run.
- *dft_treat_primary_inputs_as_x_source*: The default value of the parameter is off. Set the value of the parameter to on to consider primary input ports as fully uncontrollable for DFT analysis.

- *dft_treat_primary_outputs_as_unobservable*: The default value of the parameter is off. Set the value of the parameter to on to treat the primary output ports as unobservable.
- *dftDesignState*: The default value of the parameter is `pre_scan_stitched`. Set the value of the parameter to `post_scan_stitched` to replace the regular flip-flops by the scan flip-flops and form a scan chain between them.
- *dft_max_files_in_a_directory*: The default value of the parameter is 2000. Set the value of the parameter to any natural number to specify the maximum number of files in a single directory.
- *dft_treat_suffix_as_pattern*: The default value of the parameter is off. Set the value of the parameter to on to use the value of the `-register_suffix` field of the `force_scan`, or `force_no_scan` SGDC commands, as a pattern to be matched with the register name.
- *dft_use_specified_and_inferred_clocks*: The default value of the parameter is off. Set the value of the parameter to on to use both user-specified and inferred clocks in the design.
- *dft3BitClock*: The default value of the parameter is off. Set the value of the parameter to on to restrict the number of clock simulation cycles to 3.
- *dft_infer_clock_gating_cell*: The default value of the parameter is on. Set the value of the parameter to off to disable the automatic inference of the CGCs.
- *dftSetAllBBScanwrapped*: The default value of the parameter is off. Set the value of the parameter to on to make all the black boxes scan wrapped.
- *dftSetAllLatchT*: The default value of the parameter is off. Set the value of the parameter to on to consider all latches in the design as transparent.
- *dftSetAllTriEnableObs*: The default value of the parameter is off. Set the value of the parameter to on to set all tri-state enable pins as observable.
- *dftSetLatchFedByTCIkAsT*: The default value of this parameter is on. Set the value of the parameter to off to consider the latches, which get testclock pulse on their enable pins, when all testclocks are simulated simultaneously, to be considered as transparent. These latches are then considered for further analysis.

- *dftTestclockCycles*: The default value of the parameter is 30. Set the value of the parameter to any natural number to limit the number of pulse application to the specified value.
- *dftTMPropThruFF*: The default value of the parameter is off. Set the value of the parameter to on to allow non-X values to propagate through the flip-flops.
- *monitorFlow*: The default value of the parameter is off. Set the value of the parameter to on to enable monitoring of the time and memory usage at critical points of the SpyGlass DFT ADV run.
- *useFirstSource*: The default value of the parameter is off. Set the value of the parameter to on to set the definition of clock source as the direct source of a flip-flop clock pin for all the SpyGlass DFT ADV product rules.
- *dft_check_path_name_for_register_suffix*: The default value of the parameter is off. Set the value of the parameter to on to match the value of the `-register_suffix` field of the *force_scan*, *force_no_scan*, *no_fault*, *add_fault*, or *no_atspeed* constraints with the register name along with the path in which the register is present.
- *dft_detect_shadow_latches*: The default value of the `dft_detect_shadow_latches` parameter is on. Set the value of the parameter to off to treat all the latches normally.
- *dft_infer_clock_gating_cell*: The default value of the parameter is on. Set the value of the parameter to off to disable the CGC inference behavior.
- *dft_maximum_number_of_messages_with_spreadsheet*: The default value is off. Set the value of the parameter to on to ignore forced `no_scan` latches from rule checking.
- *dft_show_spreadsheet_path_in_message*: The default value of this parameter is on. Set the value of the parameter to on to remove the spreadsheet path from the violation message.
- *dftUseOffStateOfClockInClockPropagation*: The default value of this parameter is off. Set the value of this parameter to on so that all clocks, except the propagated clock, are set to their off state during clock propagation.

SpyGlass DFT ADV Rule Parameters

This section explains all the parameters that are used by the rules of the SpyGlass DFT ADV product.

You can set these parameters by using the following command in SpyGlass Explorer and Tcl Shell Interface:

```
set_parameter <parameter_name> <parameter_value>
```

For more information on setting the parameters, refer to the SpyGlass *Tcl Interface User Guide* and *SpyGlass Explorer User Guide*.

ATPG_credit

This parameter is used to set the fraction of untestable faults that will be considered by the [Info_untestable](#) and [Info_undetectedCause](#) rules.

The value of ATPG_credit rule parameter can be any floating-point value between 0 and 1. A value of 1 implies that all untestable faults are ignored. A value of 0 (default) implies that all untestable faults are considered. Intermediate values have an effect between these two extremes.

Used by	Info_untestable , Info_undetectedCause
Options	Floating-point values between 0 and 1 (inclusive)
Default value	0
Example	
<i>Console/Tcl-based usage</i>	set_parameter ATPG_credit 0.5
<i>Usage in Goal/source files</i>	-ATPG_credit=0.5

busWidthHigh

The busWidthHigh rule parameter specifies the maximum limit of number of tristate drivers of a tristated-bus. Buses where the number of tristate drives is greater than the value of the busWidthHigh rule parameter are not processed.

Used by	Tristate_06 , Tristate_10 , Tristate_11
Options	<positive integer value>
Default value	-1 (negative number implies no upper bound)
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter busWidthHigh 4</code>
<i>Usage in goal/source files</i>	<code>-busWidthHigh=4</code>

busWidthLow

The `busWidthLow` rule parameter specifies the minimum limit of number of tristate drivers of a tristated-bus. Buses where the number of tristate drives is less than the value of the `busWidthLow` rule parameter are not processed.

Used by	Tristate_06 , Tristate_10 , Tristate_11
Options	<positive integer value>
Default value	2
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter busWidthLow 3</code>
<i>Usage in goal/source files</i>	<code>-busWidthLow=3</code>

captureClockCount

The `captureClockCount` rule parameter specifies the number of capture clocks for atspeed testing and various bus tests.

Used by	Tristate_10
Options	<positive integer value>
Default value	1
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter captureClockCount 3</code>
<i>Usage in goal/source files</i>	<code>-captureClockCount=3</code>

captureClockEnd

The `captureClockEnd` rule parameter specifies the last capture clock on which capture rules (`Tristate_10`) should report messages.

Used by	Tristate_10
Options	<positive integer value>
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter captureClockEnd 3</code>
<i>Usage in goal/source files</i>	<code>-captureClockEnd=3</code>

captureClockStart

The `captureClockStart` rule parameter specifies the first capture clock on which capture rules (the [Tristate_10](#) rule) should report messages.

Used by	Tristate_10
Options	<positive integer value>
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter captureClockStart 3</code>
<i>Usage in goal/source files</i>	<code>-captureClockStart=3</code>

checkPLLSourceForAllNonFFCellClockPins

Enables the [Atspeed_23](#) rule to check for all clock pins of Non FF cells that are identified from the "clock : true" entry in the .lib library, whether

they are memories or not.

Used by	Atspeed_23
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter checkPLLSourceForAllNonFFCellClockPins on</code>
<i>Usage in goal/source files</i>	<code>-checkPLLSourceForAllNonFFCellClockPins=on</code>

debugInst

Use the `debugInst` rule parameter to identify instances for which a SpyGlass Constraint file should be automatically generated by the [Diagnose_testmode](#) and [CreateDebugSGDC](#) rules.

A file named, `dft_debug_inst.sgdc`, is created in which `sgdc` commands for all the specified instances are listed. The `dft_debug_inst.sgdc` file contains the following information:

- the instance as the `current_design` specification
- a [test_mode](#) constraint for each instance pin with a non-x value due to `testmode`
- a [clock](#) constraint with `-testclock` argument for each instance pin with a `testclock` pulse
- a [force_ta](#) constraint for each instance pin to assign controllability and observability.

The purpose is to recreate the same environment that this instance has in the current run for a standalone run with this instance as the top-level design. This “divide and conquer” approach may provide a more efficient means to debug SpyGlass DFT ADV messages in large designs.

Used by	Diagnose_testmode , CreateDebugSGDC
Options	A space-separated string
Default value	A non-empty string with text 'nothing'
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter debugInst {top.inst1.inst2 top2.inst4.inst5}</code>
<i>Usage in goal/source files</i>	<code>-debugInst="top.inst1.inst2 top2.inst4.inst5"</code>

dftmax_si_so_pin_number

Use this parameter to specify number of top-level SI-SO pin count that can be used for DFTMAX Ultra planning.

Used by	Info_dftmax_configuration
Options	Any positive integer greater than 1
Default value	2
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftmax_si_so_pin_number 20</code>
<i>Usage in goal/source files</i>	<code>-dftmax_si_so_pin_number=20</code>

dftmax_is_occ_present

Use this parameter to enable the on-chip clock controller (OCC).

By default, the value of the parameter is set to off. In this case, the number of top-level pins available for DFTMAX Ultra planning is equal to the value of `dftmax_si_so_pin_number` parameter.

To allow OCC, where two additional I/O pins (common for all blocks) are used, set the value of the `dftmax_is_occ_present` parameter to on. In this case, the number of top-level pins available for DFTMAX Ultra planning is two less than the value of `dftmax_si_so_pin_number` parameter.

Used by	Info_dftmax_configuration
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftmax_is_occ_present on</code>
<i>Usage in goal/source files</i>	<code>-dftmax_is_occ_present=on</code>

dftmax_allow_asymmetric_pin_allocation

Use this parameter to turn on/off the asymmetric input/output allocation of pins among blocks for DFTMAX Ultra planning.

By default, value of the parameter is set to off. Set the value of the parameter to on to allow asymmetric allocation.

NOTE: *The asymmetric allocation is currently uncommon in the field and has much less production mileage than the symmetric allocation.*

Used by	Info_dftmax_configuration
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftmax_allow_asymmetric_pin_allocation on
<i>Usage in goal/source files</i>	-dftmax_allow_asymmetric_pin_allocation=on

dftmax_config_lookup_file

Use this parameter to provide the absolute path of configuration estimator lookup file (in .csv format) for DFTMAX Ultra planning.

By default, value of the parameter is set to off. In this case, tool picks the configuration estimator lookup from <SPYGLASS_HOME>/policies/dft_dsm/dftMaxUltra/dftMaxUltra_config_lookup.csv file.

Set the value of the parameter to absolute path of configuration estimator lookup file (in .csv format) for DFTMAX Ultra planning.

Used by	Info_dftmax_configuration
Options	off, <file_path>
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftmax_config_lookup_file dummy.csv
<i>Usage in goal/source files</i>	-dftmax_config_lookup_file=dummy.csv

dft_conn_check_allow_trace_through_async

Use this parameter to allow combinational traversal through the asynchronous pins of a flip-flop while performing the checks for require path ([require_path](#) constraint) or illegal path ([illegal_path](#) constraint).

Used By	Conn_02 , Soc_02, Soc_02_Info, Conn_08 , Conn_09 , Soc_08, Soc_09
Options	on, off
Default Value	off
Example	
Console/Tcl-based Usage	set_parameter dft_conn_check_allow_trace_through_async on
Usage in goal/ source files	-dft_conn_check_allow_trace_through_async=on

dft_conn_check_allow_trace_through_clock_shaper

Set this parameter to allow combinational traversal through a clock shaper using clock_in / clock_out pins

Used By	Conn_02 , Soc_02, Soc_02_Info, Conn_08 , Conn_09 , Soc_08, Soc_09
Options	on, off
Default Value	off
Example	
Console/Tcl-based Usage	set_parameter dft_conn_check_allow_trace_through_clock_shaper on
Usage in goal/ source files	- dft_conn_check_allow_trace_through_clock_shaper =on

dft_allow_clock_merge_at_mux_via_data

By default, the value of the parameter is set to off. In this case, the [Clock_08](#)/[Atspeed_22](#) rules report violation, if clock is merging at a mux through its data-pins, irrespective of involvement of a control pin.

Set the value of the parameter to on to enable the [Clock_08](#) and [Atspeed_22](#) rule to ignore the violation if a clock is merging at a mux through its data-pins.

Consider [Figure 1](#) where clock are merging with mux through its data pin:

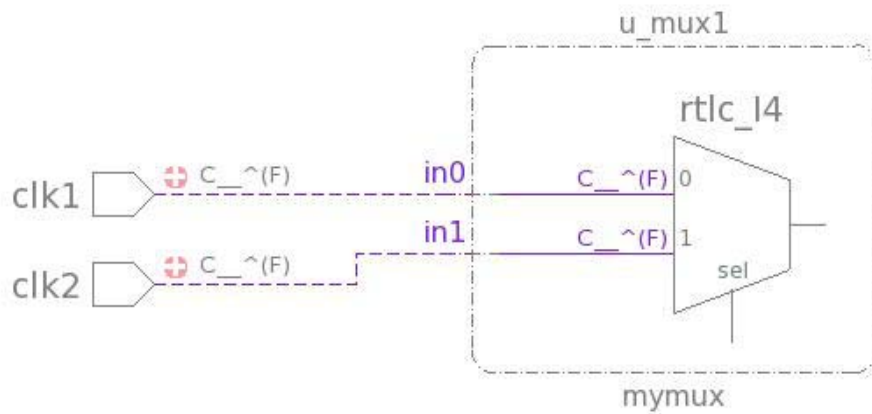


FIGURE 1. Clocks merging with mux through data pin

In the above example, clocks, `clk1` and `clk2` are merging with the mux, `u_mux1`, through its data pin, `rtlc_I4`. By default, when the value of the `dft_allow_clock_merge_at_mux_via_data` parameter is `off`, the `Clock_08/Atspeed_22` rules will report violation for `clk1` and `clk2`. To ignore violation for this case, set the value of the `dft_allow_clock_merge_at_mux_via_data` parameter to `on`.

Similarly, consider [Figure 2](#):

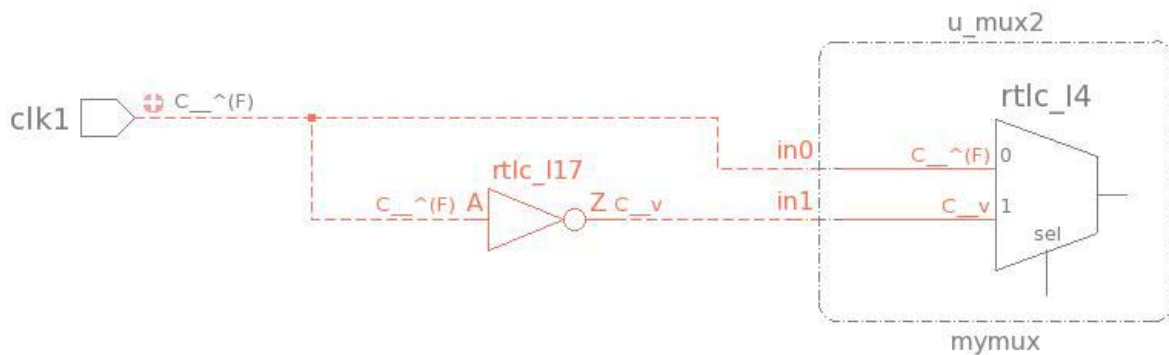


FIGURE 2. Clock merging with mux through its data pin

In the above figure, clock, `clk1`, is merging with the mux, `u_mux2`, through its data pin, `rtlc_14`. By default, when the value of the `dft_allow_clock_merge_at_mux_via_data` parameter is `off`, the `Clock_08/Atspeed_22` rules will report violation for `clk1`. To ignore violation for this case, set the value of the `dft_allow_clock_merge_at_mux_via_data` parameter to `on`.

Used by	Clock_08 , Atspeed_22
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_allow_clock_merge_at_mux_via_data on</code>
<i>Usage in goal/source files</i>	<code>-dft_allow_clock_merge_at_mux_via_data = on</code>

dft_allow_inactive_resets_from_scan_ffs

Use this parameter to report all the inactive set/resets, which are driven by scan flip-flops.

SpyGlass DFT ADV Rule Parameters

By default, the value of this parameter is off. In this case, the inactive set/resets, which are driven by scan flip-flops, are not reported.

Set the value of the of the parameter to on to report violation for such cases.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_allow_inactive_resets_from_scan_ffs off
<i>Usage in goal/source files</i>	-dft_allow_inactive_resets_from_scan_ffs = off

dft_allow_sequential_propagation_during_clock_simulation

Enables flip-flops and non-transparent latches to allow constant to propagate through them when pulse sequence is simulated on the clock line.

By default, the value of the parameter is off. In this case, flip-flops and non-transparent latches do not allow constant to propagate through them when pulse sequence is simulated on the clock line. If needed, an explicit test_mode pulse-sequence is put on the clock line followed by X to release the constant from the clock line.

Set the value of the parameter to on enable the flip-flops and non-transparent latches to allow constant to propagate through them when pulse sequence is simulated on the clock line.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	off
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_allow_sequential_propagation_during_clock_simulation on
<i>Usage in goal/source files</i>	- dft_allow_sequential_propagation_during_clock_simulation = on

dft_allow_path_from_enable_to_cgc_clkout

Use this parameter to allow a connectivity path from enable (data and test) to CGC clock-out pin.

Used by	Conn_02 , Conn_08 , Conn_09
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_allow_path_from_enable_to_cgc_clkout on
<i>Usage in goal/source files</i>	-dft_allow_path_from_enable_to_cgc_clkout=on

dft_allow_single_flipflop_driver_in_Async_02_capture

Use this parameter to allow a connectivity path from enable (data and test) pin to CGC clock-out pin.

By default, the value of this parameter is set to off.

You can set the value of the parameter to on, if you want to focus on the glitch-prone circuit. That is, allow a single flip-flop as driver of an asynchronous sources in Async_02_capture rule as this may not cause a glitch on the async-line.

Used by	Async_02_capture
Options	on, off
Default value	off
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_allow_single_flipflop_driver_in_Async_02 _capture on
<i>Usage in goal/source files</i>	- dft_allow_single_flipflop_driver_in_Async_02 _capture=on

dft_all_scan_chains_defined

This parameter sets the scannability criteria for a flip-flop when the value of the dftDesignState parameter is set to post_scan_stitched. In case the dftDesignState parameter is set to pre_scan_stitched, the dft_all_scan_chains_defined parameter has no impact.

By default, the value of the dft_all_scan_chains_defined parameter is set to yes or on. It implies that all scan chains are defined in sgdc using the scan_chain constraint. Therefore, a flip-flop will only be scannable if it has good clock / async pin control and it is part of a valid scan chain. The scannability of flip-flops not connected to a scan chain is determined by the Async_07 and Clock_11 violations. Any flip-flop without such violations and without a no_scan constraint will be set to scannable

You can set the value of the dft_all_scan_chains_defined parameter to no or off to disable the scannability criteria when the dftDesignState parameter is set to post_scan_stitched. This implies that not all scan chains are defined in sgdc using the scan_chain constraint.

Used by	All DFT Rules
Options	yes on, no off
Default value	yes on

Example

<i>Console/Tcl-based usage</i>	set_parameter dft_all_scan_chains_defined no
<i>Usage in goal/source files</i>	-dft_all_scan_chains_defined=no

dft_agnostic_cgc_flipflop_deadlock_check

Use this parameter to consider/ignore the DFT related phrase during the

CG_08 rule checking. However, during simulation, the test_mode constraint, if defined, is considered.

Used by	CG_08
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_agnostic_cgc_flipflop_deadlock_check on
<i>Usage in goal/source files</i>	-dft_agnostic_cgc_flipflop_deadlock_check=on

dft_atpg_conflict_tp_count

Use this parameter for specifying the upper limit on the number of test_points to be suggested for reducing conflicts during test pattern generation. You can specify either an absolute number or percentage of flip-flops.

Used by	Info_atpg_conflict
Options	any number or percentage
Default value	5%
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_atpg_conflict_tp_count 500
<i>Usage in goal/source files</i>	-dft_atpg_conflict_tp_count=500

dft_autofix_testmode_signal

Use this parameter to specify testmode signal for autofix and RTL testpoint insertion.

Used by	Rules supporting autofix and RTL Testpoint Insertion
Options	Testmode signal name

SpyGlass DFT ADV Rule Parameters

Default value	atrenta_generated_port_tm
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_autofix_testmode_signal testmode_signal_name
<i>Usage in goal/source files</i>	-dft_autofix_testmode_signal=testmode_signal_name

dft_autofix_testclock_signal

Use this parameter to specify testclock signal for autofix and RTL testpoint insertion.

If the value of the [dft_infer_tp_clock](#) parameter is off, then the clock specified using the `dft_autofix_testclock_signal` parameter is used for inserting testpoints.

If the value of the [dft_infer_tp_clock](#) parameter is on, then the clock specified using the `dft_autofix_testclock_signal` is only used for testpoints for which testclock can not be inferred.

Used by	Rules supporting autofix feature and RTL testpoint insertion
Options	Testclock signal name
Default value	atrenta_generated_port_tclk
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_autofix_testclock_signal testclock_signal_name
<i>Usage in goal/source files</i>	- dft_autofix_testclock_signal=testclock_signal_name

dft_block_unstable_value_trace_on_no_clock

Use this parameter to specify if the trace, for finding sources of unstable values, be stopped at a flip-flop / latch, if it does not gets a clock.

By default, the value of this parameter is off.

In case, latch has an active value on its enable pin, clock-check is ignored

even when this parameter is on. Trace continues through such transparent latches.

Used by	Conn_14
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_block_unstable_value_trace_on_no_clock on</code>
<i>Usage in goal/source files</i>	<code>-dft_block_unstable_value_trace_on_no_clock=on</code>

dft_cgc_pre_scan_stitched_treatment_only_to_tied_test_enable

Specifies whether both the tied-test-enable pin and missing test enable pin of a CGC should be treated as if it is connected with global scan enable and is active in scan shift mode.

By default, the value of the parameter is off. In this case, both the tied-test-enable pin and the missing test enable pin of a CGC are treated as if its test enable pin is connected with global scan enable and is active in the scan shift mode.

Set the value of the parameter to on to treat only the tied-test-enable pin of a CGC as if it is connected with global scan enable and is active in scan shift mode. Also, If the test enable pin is missing then it is not considered to be connected with global scan enable and active in scan shift mode.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	off
Example	

SpyGlass DFT ADV Rule Parameters

<i>Console/Tcl-based usage</i>	set_parameter dft_cgc_pre_scan_stitched_treatment_only_to_tied_test_enable on
<i>Usage in goal/source files</i>	- dft_cgc_pre_scan_stitched_treatment_only_to_tied_test_enable=on

dft_check_clock_merge_in_shift_as_well

Determines whether or not the [Clock_08](#) rule reports for clock merging in capture mode or scanshift mode.

By default, the value of the *dft_check_clock_merge_in_shift_as_well* parameter is set to `off`. Therefore, the *Clock_08* rule reports for clock merging only in the capture mode.

Set the value of the parameter to `on` to enable the *Clock_08* rule to report for clock merging in the scanshift mode and in the capture mode.

Used by	Clock_08
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_check_clock_merge_in_shift_as_well yes
<i>Usage in goal/source files</i>	-dft_check_clock_merge_in_shift_as_well=yes

dft_check_latch_transparency_in_shift

Set the value of this parameter to enable the [Latch_08](#) rule to check for latch transparency in all the modes, including the scanshift mode.

Used by	Latch_08
Options	on, off
Default value	off
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_check_latch_transparency_in_shift on
<i>Usage in goal/source files</i>	-dft_check_latch_transparency_in_shift=on

dft_check_path_name_for_instance_suffix

Use this parameter to modify the behavior of the `-instance_suffix` field of the `no_fault` or `add_fault` sgdc commands.

By default the value of this parameter is off. In this case, the value of the `-instance_suffix` field is used to match with the instance name, excluding the path.

Set the value of the parameter to on to match the value of the `-instance_suffix` field with the instance name along with the path in which the register is present.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	off

Example

<i>Console/Tcl-based usage</i>	set_parameter dft_check_path_name_for_instance_suffix on
<i>Usage in goal/source files</i>	-dft_check_path_name_for_instance_suffix=on

dft_check_path_name_for_register_suffix

Use this parameter to modify the behavior of the `-register_suffix` field of the `force_scan`, `force_no_scan`, `no_fault`, or `add_fault` sgdc commands.

By default the value of this parameter is off. In this case, the value of the `-register_suffix` field is used to match with the register name, excluding the path.

Set the value of the parameter to on to match the value of the `-register_suffix` field with the register name along with the path in which the register is present.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_check_path_name_for_register_suffix on</code>
<i>Usage in goal/source files</i>	<code>-dft_check_path_name_for_register_suffix=on</code>

dft_check_test_mode_conflicts

When a `test_mode` is specified on a signal that has an internal driver other than primary port, there is a possibility that the internal driver may not be driving the signal to the specified `test_mode` value. The internal driver may either be driving the signal to a conflicting value (1 instead of 0 or 0 instead of 1) or to an unknown value ('X' or 'U').

The following list describes the possible values for the `dft_check_test_mode_conflicts` parameter and the corresponding rule behavior:

- **allow_match_with_X:** A conflict is considered, if the simulation value is non-X and does not match the specified `test_mode` value. If the simulation value has 'X', 'Z', or 'U', then it is not considered a conflict.

For more information on the [Info_uncontrollable](#) rule behavior, when `allow_match_with_X` is assigned to the `dft_check_test_mode_conflicts` parameter, see the *Example Code and/or Schematic* section of the [Info_uncontrollable](#) rule.

- **allow_exact_match:** A conflict is considered, if the simulation value is non-X and does not match the specified `test_mode` value or unknown value ('X' or 'U'). However, no conflict is considered in case of either exact match or when the internal driver is driving the signal to a high-impedance state ('Z').

For more information on the [Info_uncontrollable](#) rule behavior, when `allow_exact_match` is assigned to the `dft_check_test_mode_conflicts` parameter, see the *Example Code and/or Schematic* section of the [Info_uncontrollable](#) rule.

Used by	Info_uncontrollable
Options	allow_match_with_X allow_exact_match
Default value	allow_match_with_X
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_check_test_mode_conflicts "allow_exact_match"
<i>Usage in goal/source files</i>	- dft_check_test_mode_conflicts="allow_exact_match"

dft_clock_end_point_types_for_Clock_02

Specifies the type of instances that are checked for clock edge usage by the [Clock_02](#) rule. The type of instances can be either flip-flops, or all flip-flops, memories, and hard macros.

Used by	Clock_02
Options	flipflops, flipflops_memories_hardmacros
Default value	flipflops_memories_hardmacros
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_clock_end_point_types_for_Clock_02 flipflops
<i>Usage in goal/source files</i>	-dft_clock_end_point_types_for_Clock_02= flipflops

dft_collapse_equivalent_faults

The `dft_collapse_equivalent_faults` parameter is used by *SpyGlass DFT ADV* to generate collapsed stuck-at fault coverage.

Fault Collapsing allows you to debug only the unique faults. You can generate the stuck-at fault and test coverage with fault collapsing. This improves the correlation with ATPG in cases where ATPG is also run with the fault collapsing capability turned on.

By default, the value of the parameter is set to `off`.

Set the value of the parameter to on to generate stuck-at fault coverage with fault collapsing in the *stuck_at_coverage_collapsed.rpt* report.

Used by	<i>Info_coverage</i>
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_collapse_equivalent_faults on</code>
<i>Usage in goal/source files</i>	<code>-dft_collapse_equivalent_faults=on</code>

dft_conn_check_allow_non_x_value_on_sensitizable_path

Allows non-x value (0 or 1) on a sensitizable path.

By default, the value of the parameter is on. Therefore, the [Conn_02](#) rule allow non-x value on the path while performing checks on the sensitizable paths.

Set the value of the parameter to off to ignore non-X value on the path while performing checks on the sensitizable paths.

Used by	Conn_02
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<pre>set_parameter dft_conn_check_allow_non_x_value_on_sensitizable_path off</pre>
<i>Usage in goal/source files</i>	<pre>- dft_conn_check_allow_non_x_value_on_sensitizable_path=off</pre>

dft_conn_check_handle_rtl_negedge

Considers the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.

By default the value of the parameter is off. In this case, the CP/CLR/PRE pin of the flip-flop remains the start/end point of the connectivity check.

Set the value of the parameter to yes to consider the input to the inverter, in front of the CP/CLR/PRE pin, as the start/end point of the connectivity check.

Used by	DFT: Conn_01 , Conn_02 , Conn_08 , Conn_09 , Conn_10 Connectivity Verify: Soc_01 Soc_01_Info Soc_02 Soc_02_Info Soc_08 Soc_09 Soc_10
Options	on, off
Default value	off

Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_conn_check_handle_rtl_negedge off
<i>Usage in goal/source files</i>	-dft_conn_check_handle_rtl_negedge = off

dft_conn_treat_pass_as_strict_pass

Use this parameter to treat all PASS modifiers as PASS_AND_NO_FAIL, that is, PASS is same as PASS_AND_NO_FAIL.

The default value of the parameter is on for backward compatibility. This implies that the meaning of PASS is same as 'strict pass'.

See [The constraint_message_tag Modifiers](#) for more information.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_conn_treat_pass_as_strict_pass off
<i>Usage in goal/source files</i>	-dft_conn_treat_pass_as_strict_pass=off

dft_conn_treat_fail_as_strict_fail

Use the parameter to treat all FAIL modifiers as FAIL_AND_NO_PASS, that is, FAIL is same as FAIL_AND_NO_PASS. The default value of the parameter is off for backward compatibility. This means that meaning of FAIL is fail.

See [The constraint_message_tag Modifiers](#) for more information.

Used by	All SpyGlass DFT ADV Rules
Options	off, on
Default value	off

Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_conn_treat_fail_as_strict_fail on
<i>Usage in goal/source files</i>	-dft_conn_treat_fail_as_strict_fail on

dft_conn_check_treat_endpoint_as_stoppoint

Considers endpoints as stop points while checking for paths between nodes. By default, the value of this parameter is `rp_off_rsp_on`. Therefore, in this case:

- the [Conn_02](#) rule doesn't treat endpoint as stop point while performing checks.
- the [Conn_08](#) rule does treat endpoint as stop point while performing checks.

Set the value of this parameter to `on` to treat endpoint as stop point in both [Conn_02](#) and [Conn_08](#) rules while performing checks for path between nodes.

Used by	Conn_02 , Conn_08
Options	all, any positive integer
Default value	0
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_conn_check_treat_endpoint_as_stoppoint on
<i>Usage in goal/source files</i>	-dft_conn_check_treat_endpoint_as_stoppoint=on

dft_conn_check_rp_rsp_unified_flow

Considers the field `-from_one_of` and `-from_one_of_type` of [require_strict_path](#) constraint as deprecated. Also, performs the `path_type` dependent checking and incorporate named, positional or instance association, if specified with [require_strict_path](#) constraint, while performing checks.

SpyGlass DFT ADV Rule Parameters

Used by	Conn_08
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_conn_check_rp_rsp_unified_flow off</code>
<i>Usage in goal/source files</i>	<code>-dft_conn_check_rp_rsp_unified_flow=off</code>

dft_coverage_report_depth

Controls the hierarchical depth up to which coverage summary is reported in the coverage reports.

Used by	Info_coverage
Options	all, any positive integer
Default value	0
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_coverage_report_depth all</code>
<i>Usage in goal/source files</i>	<code>-dft_coverage_report_depth all</code>

dft_detect_shadow_latches

Controls the treatment of the shadow latches.

A latch is considered as a shadow latch when all of the following conditions hold true:

- Its D-pin is driven by exactly one scan flip-flop
- Its EN-pin is clocked by same clock and in same phase as that of the scan flip-flop which is driving its D-pin

In this case, the latch output controllability is the same as the controllability of the scan flip-flop which is driving its D-pin. The shadow latch analysis is done in the shift mode.

By default, the value of the *dft_detect_shadow_latches* parameter is set to on. Therefore, shadow latches are identified and treated accordingly. Set the value of the parameter to off to treat all the latches normally, that is, to disable the detection of shadow latches.

The following figure shows the effect on the design when the value of the *dft_detect_shadow_latches* parameter is set to on:



In the above example, SpyGlass treats the ld1_reg as the shadow latch.

SpyGlass DFT ADV Rule Parameters

The following figure shows the effect on the design when the `dft_detect_shadow_latches` parameter is set to off:



In the above example, SpyGlass does not treat the `ld1_reg` as a shadow latch.

Used by	All DFT Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_detect_shadow_latches off</code>
<i>Usage in goal/source files</i>	<code>-dft_detect_shadow_latches=off</code>

dft_enable_checks_for_all_cgc

Enables rule checking for all Clock Gating Rules.

Used by	Clock Gating Rules
Options are	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_enable_checks_for_all_cgc on
<i>Usage in goal/source files</i>	-dft_enable_checks_for_all_cgc=on

dft_ignore_constant_latches_in_rule_checking

Use this parameter to ignore latches from the Latch_08 / Atspeed_19 rule checking, when one of the following conditions is true:

- D-pin has a constant value and EN-pin gets a test clock
- Q-pin has a constant value

By default, the value of the parameter is on. Therefore, the latches which do not contribute to coverage loss are ignored from the rule checking.

Set the value of the parameter to off to consider such latches for rule checking.

Used by	Latch_08, Atspeed_19
Options are	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_ignore_constant_latches_in_rule_checking off
<i>Usage in goal/source files</i>	- dft_ignore_constant_latches_in_rule_checking =off

dft_ignore_no_scan_latches_in_rule_checking

Enables you to ignore forced *no_scan* latches.

By default, the value of the `dft_ignore_no_scan_latches_in_rule_checking` parameter is `off`. In this case, forced *no_scan* latches are considered for rule checking.

Set the value of the parameter to `on` to ignore forced *no_scan* latches from rule checking.

Used by	Latch_08 , Atspeed_19
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_no_scan_latches_in_rule_checking on</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_no_scan_latches_in_rule_checking=on</code>

dft_ignore_inout_ports_as_driver

Use this parameter to ignore port's impact on the constant propagation.

By default, the value of this parameter is `on`. Therefore, all inout ports are ignored from the driver list while simulating constants.

Set the value of the parameter to `off` to explicitly specify Z if the port's impact on the constant propagation has to be ignored.

Used by	All DFT Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_inout_ports_as_driver off</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_inout_ports_as_driver=off</code>

Used by : All DFT Rules

Options are : on | off

Default value : on

Example : `set_parameter dft_ignore_inout_ports_as_driver off`

dft_identify_equivalent_faults

Enables equivalent fault identification to be identified as EQ in the fault reports. This parameter has no effect on the coverage calculation.

For more information on equivalent faults, see [Equivalent \(EQ\)](#).

Used by	Info_coverage
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_identify_equivalent_faults on</code>
<i>Usage in goal/source files</i>	<code>-dft_identify_equivalent_faults=on</code>

dft_identify_lockup_latch_only_through_buffers

Default value is off. Set the value of the parameter to on to infer lockup latches only through buffers.

Used by	All DFT Rules
Options	on, off
Default value	off
Example	

SpyGlass DFT ADV Rule Parameters

<i>Console/Tcl-based usage</i>	set_parameter dft_identify_lockup_latch_only_through_buffers on
<i>Usage in goal/source files</i>	- dft_identify_lockup_latch_only_through_buffers=on

dft_infer_flop_enable

Specifies nomenclature used for enable pin of the flip-flops present in the design.

The enable pin of the flip-flops are identified using the name.

Used by	All DFT Rules
Options	space-separated list of enable pins
Default value	"E EN LH ENABLE"
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_infer_flop_enable "my_en"
<i>Usage in goal/source files</i>	-dft_infer_flop_enable="my_en"

dft_infer_sequential_propagation_as_no_scan

Controls whether or not testmode propagation of testmode data is allowed through scannable flip-flops.

By default, the value of the *dft_infer_sequential_propagation_as_no_scan* parameter is set to `off`. Therefore, the scannable flip-flops do not allow sequential propagation of test_mode. The output of such flip-flops remain at X.

Set the value of the *dft_infer_sequential_propagation_as_no_scan* parameter to `on` to enable sequential propagation of test_mode through scannable flip-flops. The output of such flip-flops may become non-X, that is, 0 or 1, based on the sequential test_mode values making them inferred no scan.

Used by	All DFT Rules
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_infer_sequential_propagation_as_no_scan on</code>
<i>Usage in goal/source files</i>	<code>- dft_infer_sequential_propagation_as_no_scan=on</code>

dft_ignore_x_sources

Use this parameter to specify types of x-sources, which needs to be ignored.

By default, the value of this parameter is none.

Set the value of the parameter to one of the following values to specify the types of x-sources, which needs to be ignored.

Set the value of the parameter to one of the following values to specify the types of x-sources, which needs to be ignored:

- Uncontrollable Input/Outputs, UIO
- Black box instance, BB
- Multi-cycle path, MCP

SpyGlass DFT ADV Rule Parameters

- False path, FP
- Non-scan flip-flop, NSF
- Non-scan latches, NSL
- Combinational loop, CL
- Floating net, HN
- Multiple-driven net/pin, MD
- Non-transparent latch, NTL

Used by	All SpyGlass DFT ADV Rules
Options	Underscore ('_') separated combination of UIO, BB, MCP, FP, NSF, NSL, CL, HN, MD, and NTL
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_x_sources BB_FP</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_x_sources = BB_FP</code>

dft_ignore_x_sources_for_bist

Use this parameter to specify types of x-sources, which needs to be ignored.

By default, the value of this parameter is none.

Set the value of the parameter to one of the following values to specify the types of x-sources, which needs to be ignored:

- Black box instance, BB
- Scan Flip Flops in different clock domain, DDSF
- Non-scan flip-flop, NSF
- Combinational loop, CL
- Floating net, HN
- Primary input/inout port, PI
- Non-transparent latch, NTL

Used by	BIST_04
Options	'_' seperated combination of BB, DDSF, NSF, CL, HN, PI, and NTL
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_x_sources_for_bist BB_DDSF</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_x_sources_for_bist=BB_DDSF</code>

dft_ignore_state_holding_flipflop_driver_in_Async_02_capture

Use this parameter to ignore flip-flops, which can retain their states in capture mode, as drivers of asynchronous sources for the [Async_02_capture](#) rule.

By default, the value of this parameter is `on`. You can set the value of the parameter to `off` to consider the flip-flops, which can retain their states in capture mode, as drivers of asynchronous sources for the [Async_02_capture](#) rule.

Used by	Async_02_capture
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<pre>set_parameter dft_ignore_state_holding_flipflop_driver_in_Async_02_capture off</pre>
<i>Usage in goal/source files</i>	<pre>- dft_ignore_state_holding_flipflop_driver_in_Async_02_capture=off</pre>

dft_ignore_unate_reconvergence

Enables you to report/ignore rule checking for unate re-convergence.

By default, the value of the parameter is `on`. This implies that unate re-convergence is ignored.

Set the value of the parameter to `on` to check for unate re-convergence.

Used by	Topology_13 , Atspeed_25 , Atspeed_32
Options	on, off
Default value	off
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_infer_sequential_propagation_as_no_scan on
<i>Usage in goal/source files</i>	- dft_infer_sequential_propagation_as_no_scan=on

dft_insert_ta_tp

Use this parameter to insert testpoints reported by the [TA_10](#) rule.

NOTE: To insert the testpoints, ensure that the value of the `dft_insert_ta_tp` parameter is on and the value of the `rme_active` parameter is set to 1.

Used by	TA_10
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_insert_ta_tp on
<i>Usage in goal/source files</i>	-dft_insert_ta_tp=on

dft_infer_tp_clock

Use this parameter to on to infer clock connection for testpoints inserted by the [Latch Rules](#) and [TA_10](#) rules.

Used by	TA_10 , Latch Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_infer_tp_clock off
<i>Usage in goal/source files</i>	-dft_infer_tp_clock=off

dft_insert_rrf_tp

SpyGlass DFT ADV Rule Parameters

Use this parameter to insert testpoints reported by the [Latch Rules](#) rule.

NOTE: To insert the testpoints, ensure that the value of the `dft_insert_rrf_tp` parameter is on and the value of the `rme_active` parameter is set to 1.

Used by	Latch Rules
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_insert_rrf_tp off</code>
<i>Usage in goal/source files</i>	<code>-dft_insert_rrf_tp=off</code>

dft_internal_testmode_nodes_stability_check

Controls stability flow for internal nodes, which are constrained to a fixed value.

By default, the value of the parameter is off. In this case, if you run the stability check for node and an internal signal, constrained to a fixed value through testmode constraint, is encountered, the stability check is skipped for the signal. This is because the rule considers the signal as stable.

Set the value of the parameter to on to run the stability check for such signals.

Used by	Conn_14
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_internal_testmode_nodes_stability_check on</code>
<i>Usage in goal/source files</i>	<code>- dft_internal_testmode_nodes_stability_check=on</code>

dft_list_latches_of_type

Enables you to specify the type of latches to be listed by the Info_latch

rule.

Used by	<i>Info_Latch</i>
Options	off, all, transparent, non_transparent, scannable, non_scannable, shadow, non_shadow, non_transparent_non_scannable_non_shadow
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_list_latches_of_type all</code>
<i>Usage in goal/source files</i>	<code>-dft_list_latches_of_type=all</code>

dft_maximum_number_of_messages_with_spreadsheet

Controls the number of messages generated, per rule, which will have the spreadsheet report. Using this parameter limits the creation of spreadsheet report, which otherwise wastes the disk space.

By default, 1000 messages with attached spreadsheet report are generated for a rule. You can set the value of the parameter to any non-negative integer value.

However, if the parameter reaches the specified threshold, the spreadsheet report does not get attached to the violation message. Also, the violation message states the reason for the same. However, it does not impact the violation messages generated.

For example, consider that the value of the *dft_maximum_number_of_messages_with_spreadsheet* parameter is set to 1. Also, consider the following violation messages:

```
Latch enable source 'top.root_src_1' is driving latch enables on both levels in Capture mode (drives 4 Latch(es) out of which 4 Latch(es) are non-transparent/controlable transparent), SPREADSHEET_PATH: 'spyglass_reports/dft/Latch_08/000/Latch_08_violation001.csv'
```

```
Latch enable source 'top.root_src_2' is driving latch enables on both levels in Capture mode (drives 4 Latch(es) out of which 4 Latch(es) are non-transparent/controlable transparent), SPREADSHEET_PATH: No spreadsheet attached. Parameter
```

SpyGlass DFT ADV Rule Parameters

'dft_maximum_number_of_messages_with_spreadsheet' is set to '1'
 The spreadsheet report for the second violation message is not generated because the value of the *dft_maximum_number_of_messages_with_spreadsheet* parameter is set to 1.

Used by	All DFT Rules
Options	all, any non-negative integer
Default value	1000
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_maximum_number_of_messages_with_spreadsheet 100
<i>Usage in goal/source files</i>	- dft_maximum_number_of_messages_with_spreadsheet=100

dft_max_flops_in_diagnose_scan_chain_violation_schematic

Controls the maximum number of flip-flops shown in schematic for the violation messages reported by the *Diagnose_ScanChain* rule.

Used by	<i>Diagnose_ScanChain</i>
Options	all, any non-negative integer
Default value	3
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_max_flops_in_diagnose_scan_chain_violation_schematic all
<i>Usage in goal/source files</i>	- dft_max_flops_in_diagnose_scan_chain_violation_schematic=5

dft_min_scannability_ratio_for_test_points

Use this parameter to specify the minimum scannability ratio, the design must be at, before test_point analysis (TA_09/TA_10) starts and test points (control/observe) are identified.

Used by	TA_09 , TA_10
Options	Any number between 0 and 1
Default value	0.7
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_min_scannability_ratio_for_test_points 0.8
<i>Usage in goal/source files</i>	- dft_min_scannability_ratio_for_test_points=0.8

dft_report_all_paths_between_reconvergence_start_and_end

Use this parameter to report all paths between re-convergence start and end points.

By default, the value of this parameter is set to off. Therefore, SpyGlass does not report all paths between re-convergence start and end points.

Set the value of the parameter to on to report all paths between re-convergence start and end points.

Used by	Clock_28 , Topology_13 , Atspeed_25 , Atspeed_30
Options	off, on
Default Value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_report_all_paths_between_reconvergence_start_and_end on
<i>Usage in goal/source files</i>	- dft_report_all_paths_between_reconvergence_start_and_end=on

dft_report_cgc_centric_deadlock

Use this parameter to make CG_08 checks CGC centric and more accurate, by considering path polarity and impact of other controllable sources.

Used by	CG_08
Options	on, off
Default Value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_report_cgc_centric_deadlock on</code>
<i>Usage in goal/source files</i>	<code>-dft_report_cgc_centric_deadlock=on</code>

dft_require_path_fail_limit

Selects the number of violations reported by the [Conn_02](#) and [Conn_08](#) rules for the [require_path](#) and [require_strict_path](#) constraints failure when either the `-from_one_of` or `-to_one_of` arguments of the constraints are not specified.

Used by	Conn_02 , Conn_08
Options	<any natural number>
Default Value	10
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_require_path_fail_limit -1</code>
<i>Usage in goal/source files</i>	<code>-dft_require_path_fail_limit=-1</code>

dft_require_path_invalid_limit

Limits the number of invalid path violations reported by the Soc_08 rule for the [require_strict_path](#) constraint failure.

Used by	Conn_08
Options	<any natural number>
Default Value	10
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_require_path_invalid_limit -1</code>
<i>Usage in goal/source files</i>	<code>--dft_require_path_invalid_limit=-1</code>

dft_require_path_pass_limit

Limits the number of violations reported by the [Conn_02](#) and [Conn_08](#) rules for the [require_path](#) and [require_strict_path](#) constraints failure when either the `-from_one_of` or `-to_one_of` arguments of the constraints are specified.

Used by	Conn_02 , Conn_08
Options	<any natural number>
Default Value	-1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_require_path_pass_limit 1</code>
<i>Usage in goal/source files</i>	<code>-dft_require_path_pass_limit=1</code>

dft_rrf_display_limit

Specifies the probability detection bucket count for faults, which need to be highlighted by the violation message generated by the [Latch Rules](#) rule.

Following table lists the bucket count for the various values available for the `dft_rrf_display_limit` parameter:

Value	Bucket Count
low	lowest 4 non-empty bucket IDs
medium	lowest 8 non-empty bucket IDs
high	lowest 12 non-empty bucket IDs

Additionally, you can also specify an integer between 1-99 (both inclusive) along with already supported ENUMS (low, medium, high).

Used by	Latch Rules
Options	low, medium, high
Default value	low
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_display_limit medium</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_display_limit=medium</code>

dft_rrf_generate_fault_report

Generates the *Node List for Probability Distribution* section in the [random_pattern_node_distribution](#) report. This section lists probability of detection, controllability, and observability bucket IDs.

Used by	Latch Rules
Options are	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_generate_fault_report on</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_generate_fault_report=on</code>

dft_rrf_tp_count

Specifies the required number of test points to be suggested for improving random pattern fault coverage.

NOTE: *To disable the generation of test points, set the value of the `dft_rrf_tp_count` parameter to 0.*

Used by	Latch Rules
Options	Any number or percentage
Default value	5%
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_count 500</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_count= 500</code>

dft_rrf_tp_count_for_cutoff_incremental_gain

Specifies the number of test points that needs to be considered for cutoff cumulative incremental gain.

Used by	Latch Rules
Options	Any positive number
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_count_for_cutoff_incremental_gain 15</code>
<i>Usage in goal/source files</i>	<code>- dft_rrf_tp_count_for_cutoff_incremental_gain = 15</code>

dft_rrf_tp_cutoff_incremental_gain

Specifies the minimum desired gain (as percentage) for a test point to consider it in the design.

Used by	Latch Rules
Options	decimal number between 0 and 100
Default value	0.00
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_cutoff_incremental_gain 0.01</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_cutoff_incremental_gain=0.01</code>

dft_rrf_tp_effort_level

Specifies the effort level for finding test points for reducing random resistance of the design.

Used by	Latch Rules
Options	low, medium, high
Default value	medium
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_effort_level low</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_effort_level=low</code>

dft_rrf_tp_generate_dc_report

Enables you to generate the [random_pattern_dc_testpoints](#) report.

Used by	Latch Rules
Options	on, off
Default value	on

Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_generate_dc_report off</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_generate_dc_report=off</code>

dft_rrf_tp_ignore_generated_nets

Use this parameter to ascertain whether internal nets should be ignored while suggesting testpoints to reduce the random resistance of faults.

The default value of the *dft_rrf_tp_ignore_generated_nets* parameter is on. Therefore, in this case, the testpoints are not suggested on the generated nets.

Used by	Latch Rules
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_ignore_generated_nets off</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_ignore_generated_nets=off</code>

dft_rrf_tp_thread_count

Use this parameter to specify number of threads to be used by the RRF Test-point analysis.

SpyGlass allows 8 threads per license. Licenses are released as soon as additional threads are closed. Use the following environment variable to enable incremental release of licenses:

```
setenv SPYGLASS_INCRCHECKIN_LICENSE_SUPPORT 1
```

Used by	Latch Rules
Options	Any positive integer value between 4 and 128

SpyGlass DFT ADV Rule Parameters

Default value	8
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_thread_count 5</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_thread_count=5</code>

dft_rrf_tp_target_test_coverage

Defines target random pattern test coverage for RRF test point identification.

Used by	Latch Rules
Options	0-100
Default value	99.99
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_target_test_coverage 50</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_target_test_coverage = 50</code>

dft_rrf_tp_type

Specifies type of test points that should be inserted for improving random pattern fault coverage.

Used by	Latch Rules
Options	control, observe, both
Default value	both
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_rrf_tp_type control</code>
<i>Usage in goal/source files</i>	<code>-dft_rrf_tp_type = control</code>

dft_rrf_tp_use_multiple_threads

Specifies whether multiple cpu-cores should be used while doing RRF test point analysis.

Used by	Latch Rules
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<pre>set_parameter dft_rrf_tp_use_multiple_threads off</pre>
<i>Usage in goal/source files</i>	<pre>-dft_rrf_tp_use_multiple_threads = off</pre>

dft_scan_chain_report_redundant_lockup_latch

Use this parameter to turn on/off the reporting of redundant lockup latches present on a scan chain.

By default, the value of this parameter is set to off. Therefore, no redundancy check is performed on the lockup latch present on a scan chain.

Set the value of the parameter to on to report a warning, if:

- A lockup latch is found on a scan chain
- No clock domain crossing identified between the source and the destination flip-flop of the lockup latch.

Used by	Scan_22
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<pre>set_parameter dft_scan_chain_report_redundant_lockup_latch on</pre>
<i>Usage in goal/source files</i>	<pre>- dft_scan_chain_report_redundant_lockup_latch = on</pre>

dft_set_scan_wrap_on_memory

Enables the application of the [scan_wrap](#) constraint on memories. When the value of this parameter is set to on, all memories are considered as scan wrapped.

Used by	All DFT Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_set_scan_wrap_on_memory off</code>
<i>Usage in goal/source files</i>	<code>-dft_set_scan_wrap_on_memory=off</code>

dft_share_atpg_conflict_testpoint_at_branch

Enables or disables sharing the testpoints at the branch:

Used by	Info_atpg_conflict
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_share_atpg_conflict_testpoint_at_branch off</code>
<i>Usage in goal/source files</i>	<code>- dft_share_atpg_conflict_testpoint_at_branch=of f</code>

dft_show_rule_name_in_audit

By default, the value of the parameter is set to on. In this case, the rule name is also printed for the steps in the `stuck_at_coverage_audit` report, for which no action is required.

For example, consider the following sample from the

stuck_at_coverage_audit report when the value of the `dft_show_rule_name_in_audit` parameter is set to on:

```
4. Latches made Transparent      10.0      10.0      Latch_08
& Info_testmode(capture) (No action required)
```

In the above example, `Latch_08` and `Info_testmode`, is also printed in the report.

Set the value of the parameter to `off` to disable printing the rule name for the steps in the `stuck_at_coverage_audit` report, for which no action is required.

Used by	Coverage_audit
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_show_rule_name_in_audit on</code>
<i>Usage in goal/source files</i>	<code>-dft_show_rule_name_in_audit=off</code>

dft_show_scan_clock_tree

Use this parameter to enable or disable the generation of the clock propagation data in the [Clock Browser window](#) and the [Scan Clock Tree Report](#), which lists all clocks in different modes along with the flip-flop, latch, black box, hard macro, or memory as leaf nodes with clock polarity. The blocked nodes are displayed as leaf nodes and are marked with the keyword, `BLKD`.

By default, the value of the parameter is 2.

You can set the value of the parameter to one of the following values:

- **off**: Disable the generation of the clock propagation data in the Clock Browser window and the Scan Clock Tree report.
- **integer value, <n>**: Lists only the first <n> leaf nodes per hierarchy in the *Clock Tree* report.

Used by	Info_testclock
Options	on, off, any positive integer value

SpyGlass DFT ADV Rule Parameters

Default value	2
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_show_scan_clock_tree 5</code>
<i>Usage in goal/source files</i>	<code>-dft_show_scan_clock_tree=5</code>

dft_show_schematic_info_in_coverage_audit

Prints the control (CNT), observe (OBS), and stuck-at (STUCK_AT) related information in the schematic for the [Coverage_audit](#) rule.

Used by	Coverage_audit
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_show_schematic_info_in_coverage_audit on
<i>Usage in goal/source files</i>	-dft_show_schematic_info_in_coverage_audit=on

dft_show_spreadsheet_path_in_message

Use this parameter to control the presence of spreadsheet path in the violation message.

By default, the value of this parameter to set to on.

Set the value of the parameter to off to remove the spreadsheet path from the violation message.

For example, the *Clock_11* rule reports the following violation message, by default, when the value of the *dft_show_spreadsheet_path_in_message* parameter is on:

```
Clock domain 'test.bbout' [in 'test'] is not controlled by any
testclock in testmode (1 flipflop(s) affected)[Clock source is
'test.inst_BB'], SPREADSHEET_PATH: 'spyglass_reports/dft/
Clock_11/000/Clock_11_violation_001.csv'
```

However, when you set the value of the parameter to off, the *Clock_11* rule reports the following violation message:

```
Clock domain 'test.bbout' [in 'test'] is not controlled by any
testclock in testmode (1 flipflop(s) affected)[Clock source is
'test.inst_BB']
```

SpyGlass DFT ADV Rule Parameters

Used by	All SpyGlass DFT ADV Rules
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_show_spreadsheet_path_in_message off
<i>Usage in goal/source files</i>	-dft_show_spreadsheet_path_in_message=off

dft_show_unused_define_tag

Enables viewing propagation of unused define_tag through [Info_define_tag](#) and [Soc_04](#) rules.

By default, the value of the parameter is off.

Set the value of the parameter to on to enable viewing propagation of unused define_tag through [Info_define_tag/Soc_04](#) rule.

Used by	Info_define_tag , Soc_04
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_show_unused_define_tag on
<i>Usage in goal/source files</i>	-dft_show_unused_define_tag=on

dft_soc_strict_boundary_validation

Controls the severity of a violation message as either Fatal or Error when there is a mismatch in the values of clock and simulation modes at the IP boundary.

The default value of the `dft_soc_strict_boundary_validation` parameter is off and the severity of the reported violation message is Fatal.

Used by	Soc_05
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_soc_strict_boundary_validation on
<i>Usage in goal/source files</i>	-dft_soc_strict_boundary_validation=on

dft_soc_unstable_value_sources

Specifies unstable value sources, other than scannable flip-flops and latches, that needs to be reported by the Conn_14 rule.

Used by	Conn_14
Options	none, all, blackbox, hanging_net, port
Default Value	all
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_soc_unstable_value_sources none</code>
<i>Usage in goal/source files</i>	<code>-dft_soc_unstable_value_sources=none</code>

dft_stable_value_pessimistic_check

Control the flow of stability check namely:

- accurate (default)
- pessimistic

The following table describes the cases in which the dft_stable_value_pessimistic_check parameter is used:

TABLE 1 Permissible Parameter Values

Case	Rule Behavior	
	Parameter Value= accurate	Parameter Value= pessimistic
Corruption of mux select	No corruption is reported if all inputs have the same 0 or 1 value	Corruption is reported if mux select signal is corrupted or changed from <init_value> (independent on the values on the mux data inputs)
If clock of a no scan flip-flop is corrupted and async reset/set is inactive/not corrupted	No corruption is reported if current values of all sync inputs (D, EN, RST, etc) don't affect the current flop state.	Report corruption without checking of the all sync inputs
Async reset/set of no scan flip-flop	No corruption if async set reset/set becomes active/corrupted during scan shift/capture but it cannot change state of the flip-flop (it is in the matching state)	Report corruption if async set/reset changes from inactive (<init_value>) to active or corrupted state, independently on current flop state
Used by	Conn_14	
Options	accurate, pessimistic	
Default value	accurate	
Example		

SpyGlass DFT ADV Rule Parameters

<i>Console/Tcl-based usage</i>	set_parameter dft_stable_value_pessimistic_check pessimistic
<i>Usage in goal/source files</i>	- dft_stable_value_pessimistic_check=pessimistic

dft_state_holding_ff_identification_effort_level

Use this parameter to change the effort level for identifying the state holding flip-flops.

Used by	Async_02_capture , Atspeed_34 , Latch Rules , Topology_16
Options	Any natural number
Default value	4
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_state_holding_ff_identification_effort_level 20
<i>Usage in goal/source files</i>	- dft_state_holding_ff_identification_effort_level=20

dft_store_tcl_query_data

Use this parameter to specify whether to preserve the design query data after completion of the goal run.

Used by	All SpyGlass DFT ADV Product Rules
Options	off, on, sg_shell
Default value	sg_shell
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_store_tcl_query_data off
<i>Usage in goal/source files</i>	-dft_store_tcl_query_data=off

dft_stop_simulation_at_mux_with_unknown_select

Use this parameter to block the simulation value propagation through muxes when mux select line is not fully known.

The following table describes the various values available for this parameter:

TABLE 2 Permissible Values for the dft_stop_simulation_at_mux_with_unknown_select parameter

Value	Examples	Description
off (default)	Example 1	mux may allow constant (not clock) to propagate through it, even when select line is not fully known. This is feasible, if all the possible data line candidates are driving the same value. It impacts constant propagation through muxes.
on	Example 2	mux does not allow constant or clock to propagate through it when select line is not fully known, even if all the possible data line candidates are driving the same value.
off_even_for_clock	Example 3	mux may allow clock and constant to propagate through it even when select line is not fully known, if all the possible data line candidates are driving the same value. It impacts clock-propagation.

Examples

Example 1

Consider the following example where the value of the *dft_stop_simulation_at_mux_with_unknown_select* is set to off:

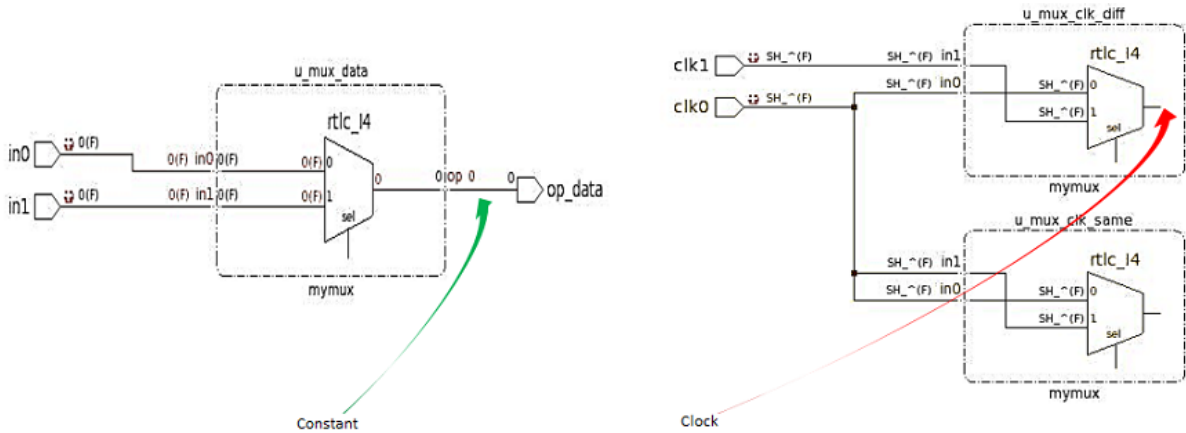


FIGURE 3. *dft_stop_simulation_at_mux_with_unknown_select* = off

In the above figure, only constant propagates through the mux even when the select pin is undefined and all the drivers are driving the same value.

Example 2

Consider the following example where the value of the `dft_stop_simulation_at_mux_with_unknown_select` is set to on:

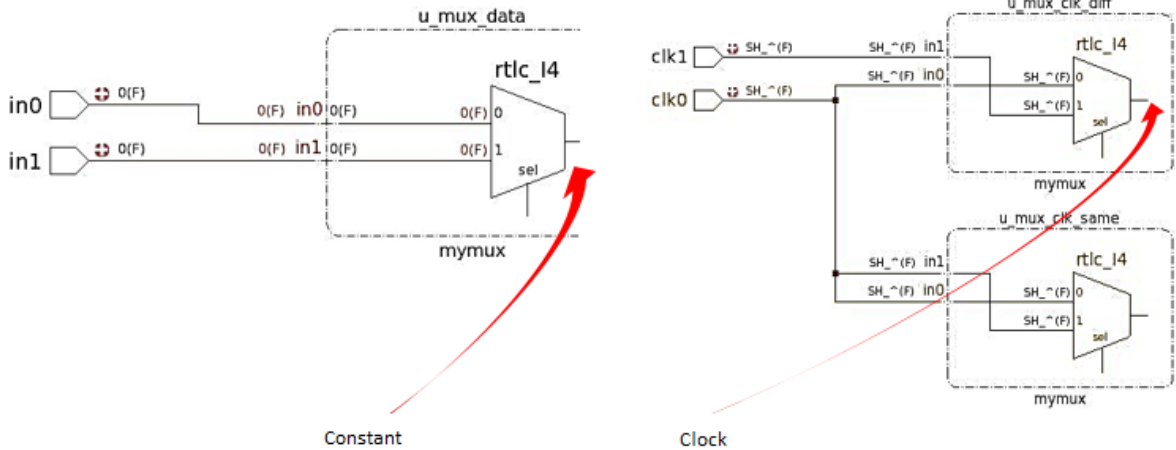


FIGURE 4. `dft_stop_simulation_at_mux_with_unknown_select = on`

In the above figure, neither constant nor clock is able to propagate through the mux when the select pin is undefined even when all the possible drivers are driving the same value.

Example 3

Consider the following example where the value of the `dft_stop_simulation_at_mux_with_unknown_select` is set to `off_even_for_clock`:

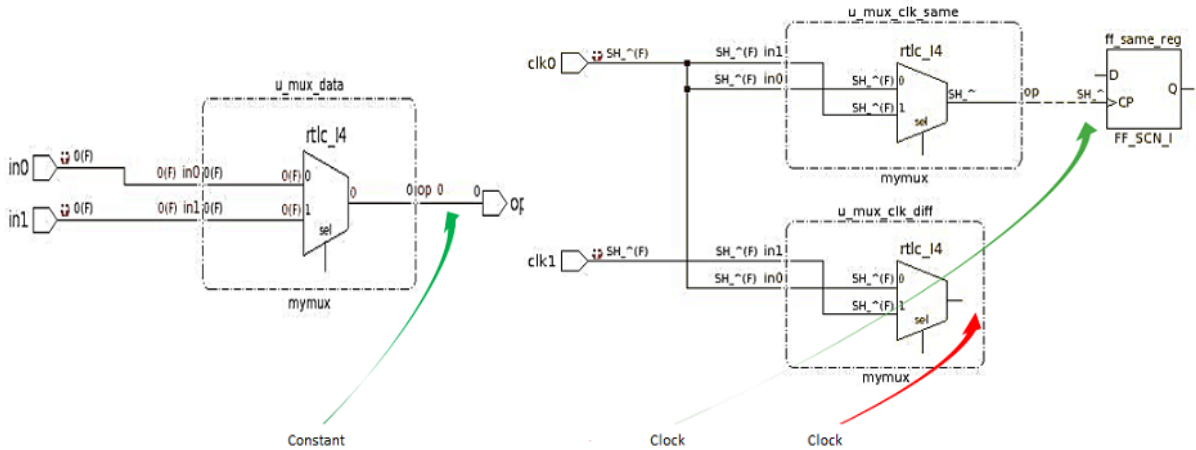


FIGURE 5. `dft_stop_simulation_at_mux_with_unknown_select = off`

In the above figure, both constant and clock are able to propagate through the mux even when select pin is undefined and all the drivers are driving the same value.

However, `u_mux_clk_diff`, does not propagate the clock because both the inputs are driven by different clocks.

Used by	All SpyGlass DFT ADV Product Rules
Options	off, on, off_even_for_clock
Default value	off

Example

Console/Tcl-based usage `set_parameter`
`dft_stop_simulation_at_mux_with_unknown_sele`
`ct off_even_for_clock`

Usage in goal/source files -
`dft_stop_simulation_at_mux_with_unknown_sele`
`ct=off_even_for_clock`

dft_support_flip_flop_bank

Use this parameter to enable flip-flop bank support.

The term, flip-flop bank, refers to multi-bit flip-flops and are identified by presence of `ff_bank` keyword in their `.lib` cell description.

For more information on `ff_bank`, see [Support for Multi-Bit Flip-Flop Cells](#).

Used by	<i>Info_coverage, Info_scanchain, Diagnose_ScanChain, Scan_24, Info_noScan, Info_forcedScan, Clock_11, Clock_11_capture, Coverage_audit, Info_testmode, Diagnose_testmode, Info_testclock, Diagnose_testclock, Info_DftDebugData, Async_07, Async_08, Async_09, Async_13, Info_inferredNoScan, Topology_01, Scan_35, Scan_36, Scan_38, Async_02_capture, Async_02_shift, Async_03, Async_05, BIST_01, Clock_02, Clock_04, Clock_08, Clock_09, Clock_21, Clock_25, Info_synthRedundant, Info_pwrGndSim, Info_untestable, Info_scanwrap, Info_noFault, Info_unused, Latch_08, Latch_17, Latch_18, Scan_31, Conn_09, Topology_02, Topology_03, Topology_10, Topology_11, Topology_13, Topology_14</i>
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_support_flip_flop_bank off</code>
<i>Usage in goal/source files</i>	<code>-dft_support_flip_flop_bank=off</code>

dft_ta_tp_for_port

Use this parameter to consider only the I/O ports as testpoint candidates.

By default, the value of this parameter is set to off. Therefore, all uncontrollable or unobservable nets are considered as testpoint candidates.

Set the value of the parameter to on to consider only the nets connected to

the I/O ports as testpoint candidates.

Used by	TA_10
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ta_tp_for_port on</code>
<i>Usage in goal/source files</i>	<code>-dft_ta_tp_for_port=on</code>

dft_ta_tp_count

Use this parameter to specify the desired number of test points to be suggested for improving the stuck at fault coverage. You can specify either an absolute number or percentage of flip-flops.

Used by	TA_10
Options	any number or percentage
Default value	5%
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ta_tp_count 500</code>
<i>Usage in goal/source files</i>	<code>-dft_ta_tp_count=500</code>

dft_ta_tp_type

Use this parameter to specify the type of test points that should be suggested for improving the fault coverage.

Used by	TA_10
Options	control, observe, both
Default value	both
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ta_tp_type control</code>
--------------------------------	---

<i>Usage in goal/source files</i>	<code>-dft_ta_tp_type=control</code>
-----------------------------------	--------------------------------------

dft_ta_tp_criteria

Use this parameter to specify the testpoint selection criteria.

By default, the value of this parameter is set to `cnt_obs_gain`. Therefore, controllability/observability gain is used as selection criteria for the testpoint selection.

Set the value of this parameter to `fault_gain` to use fault gain as the testpoint selection criteria.

Used by	TA_10
Options	<code>cnt_obs_gain</code> , <code>fault_gain</code>
Default value	<code>cnt_obs_gain</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ta_tp_criteria fault_gain</code>
<i>Usage in goal/source files</i>	<code>-dft_ta_tp_criteria=fault_gain</code>

dft_target_random_pattern_fault_coverage

Use this parameter to define target random pattern fault coverage.

Used by	Latch Rules
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_target_random_pattern_fault_coverage 20.7</code>
<i>Usage in goal/source files</i>	<code>-dft_target_random_pattern_fault_coverage =20.7</code>

dft_target_random_pattern_test_coverage

Use this parameter to define target random pattern test coverage.

Used by	Latch Rules
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_target_random_pattern_test_coverage 20.7</code>
<i>Usage in goal/source files</i>	<code>-dft_target_random_pattern_test_coverage =20.7</code>

dft_target_stuck_at_fault_coverage

Use this parameter to define target stuck at fault coverage.

Used by	Info_coverage
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_target_stuck_at_fault_coverage 20.7</code>
<i>Usage in goal/source files</i>	<code>-dft_target_stuck_at_fault_coverage=20.7</code>

dft_target_stuck_at_test_coverage

Specifies the target test coverage for the Info_coverage rule.

Used by	Info_coverage
Options	real values between 0 and 100
Default value	none
Example	

SpyGlass DFT ADV Rule Parameters

<i>Console/Tcl-based usage</i>	set_parameter dft_target_stuck_at_test_coverage 20.7
<i>Usage in goal/source files</i>	-dft_target_stuck_at_test_coverage=20.7

dft_target_transition_fault_coverage

Use this parameter to define target transition fault coverage.

Used by	Latch Rules
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_target_transition_fault_coverage 20.7
<i>Usage in goal/source files</i>	-dft_target_transition_fault_coverage=20.7

dft_target_transition_test_coverage

Use this parameter to define target transition test coverage.

Used by	Latch Rules
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_target_transition_test_coverage 20.7
<i>Usage in goal/source files</i>	-dft_target_transition_test_coverage=20.7

dft_treat_latches_with_X_on_enable_as_combinational_for_soc_path_checks

Defines the treatment of latches, that is, whether to consider them as combinational or sequential, where enable pin does not get either 0 or 1, when running Soc path check rules.

Used by	Conn_02 , Conn_08 , Conn_09
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<pre>set_parameter dft_treat_latches_with_X_on_enable_as_combin ational_for_soc_path_checks on</pre>
<i>Usage in goal/source files</i>	<pre>- dft_treat_latches_with_X_on_enable_as_combin ational_for_soc_path_checks = on</pre>

dft_treat_primary_inputs_as_x_source

Enables the DFT rules to manage the controllability of the primary input ports.

By default, the value of the parameter is set to `off`. In this case, there is no impact on the test and fault coverage.

When an IP is integrated in to SoC, there is a possibility that most of the IP block pins are not controllable. When Spyglass is used at the IP level, you can use the `dft_treat_primary_inputs_as_x_source` parameter to set all PIs to uncontrollable so that you can estimate coverage at the SoC level.

Set the value of the parameter to `on` to consider primary input ports (PIs) as X-source and report violations for the same. These primary input ports are fully uncontrollable. In this case, parameter will not affect the observability of the PIs. Also, the [stuck_at_coverage_audit](#) report shows how the test coverage and fault coverage are affected.

Exceptions

A primary input is not made uncontrollable when the input port:

- is defined as a clock using the clock constraint
- is given a value using testmode constraint

SpyGlass DFT ADV Rule Parameters

Used by	All DFT Rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_treat_primary_inputs_as_x_source on
<i>Usage in goal/source files</i>	-dft_treat_primary_inputs_as_x_source=on

dft_treat_primary_outputs_as_unobservable

Use this parameter to treat primary output ports as unobservable.

By default, the value of the parameter is set to off and the primary output ports are considered observable. In this case, capture is allowed at primary outputs.

Set the value of the parameter to on when the primary output ports as unobservable. In this case, capture will not be allowed at primary outputs.

Used by	All DFT Rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_treat_primary_outputs_as_unobservable on
<i>Usage in goal/source files</i>	-dft_treat_primary_outputs_as_unobservable =on

dft_treat_primary_port_as_valid_clock_point_for_cgc_check

Considers ports as valid end point in order to report violations for a CGC.

Used by	Clock Gating Rules
Options	on, off

Default value	off
Example	
<i>Console/Tcl-based usage</i>	<pre>set_parameter dft_treat_primary_port_as_valid_clock_point_ for_cgc_check off</pre>
<i>Usage in goal/source files</i>	<pre>- dft_treat_primary_port_as_valid_clock_point_ for_cgc_check =off</pre>

dft_use_n_cycle_capture

Use this parameter to specify the number of capture cycles.

This parameter overrides the value specified using the [sequentialDepth](#) parameter.

Example

Consider the following example showing controllability/observability for 4 capture cycles:

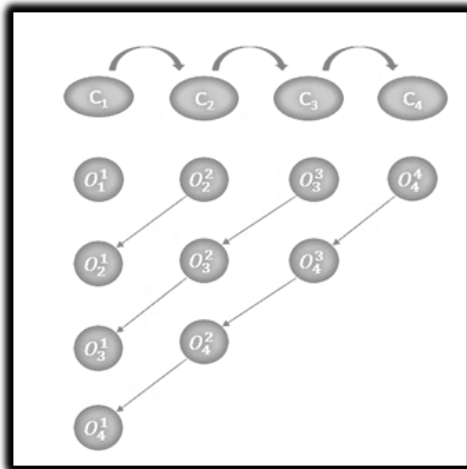


FIGURE 6.

In the above example:

SpyGlass DFT ADV Rule Parameters

- C_n is the controllability in n^{th} capture cycle
- o_x^y is the observability evaluated for x^{th} capture cycle using the controllability in the y^{th} capture cycle

Used by	Info_coverage , Scan_17
Options	<positive integer value>
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_use_n_cycle_capture 3</code>
<i>Usage in goal/source files</i>	<code>-dft_use_n_cycle_capture=3</code>

dft_use_old_Topology_10

Use this parameter to run the old implementation of the [Topology_10](#) rule.

Used by	Topology_10
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_use_old_Topology_10 on</code>
<i>Usage in goal/source files</i>	<code>-dft_use_old_Topology_10=on</code>

dft_use_source_centric_Async_02_capture

Use this parameter to switch to the old behavior of the [Async_02_capture](#) rule.

Used by	Async_02_capture
Options	off, on
Default value	on
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_use_source_centric_Async_02_capture off
<i>Usage in goal/source files</i>	-dft_use_source_centric_Async_02_capture=off

dft_use_stable_simulation_conditions

Enables the stable value mode analysis to existing rule.

By default, the value of this parameter is off. In this case, all non-scan flip-flops retain their non-X value, during test_mode (scanshift/capture/atspeed) simulation.

When you set the value of the parameter to on, only those non-scan flip-flops, which are not corrupted by scan shift operation retain their non-X value, during test_mode (scanshift/capture/atspeed) simulation.

To see the analysis data, add the [Info_unstable_testmode_registers](#) rule in the SpyGlass run.

Used by	Info_unstable_testmode_registers
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_use_stable_simulation_conditions on
<i>Usage in goal/source files</i>	-dft_use_stable_simulation_conditions=on

dft_use_specified_and_inferred_clocks

Use this parameter to set the type of clock to be used in the design.

The default value of this parameter is set to off. In this case, SpyGlass uses inferred clocks if the [gating_cell_enable](#) constraint is set. Else, SpyGlass uses the user-specified clocks.

You can set the value of the parameter to on to use both user-specified and inferred clocks.

SpyGlass DFT ADV Rule Parameters

Used by	All SpyGlass DFT ADV Rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_use_specified_and_inferred_clocks on
<i>Usage in goal/source files</i>	-dft_use_specified_and_inferred_clocks=on

dft_use_new_Atsspeed_19

Use this parameter to enable/disable the source-centric message reporting by the [Atsspeed_19](#) rule.

By default, the value of this parameter is set to on. In this case, the Atsspeed_19 rule will report source-centric violation messages as described in the [Atsspeed_19](#) rule description.

Set the value of the parameter to off to report latch-based violation messages.

Used by	Atsspeed_19
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_use_new_Atsspeed_19 off
<i>Usage in goal/source files</i>	-dft_use_new_Atsspeed_19=off

dft_x_sources_display_limit

Signifies the volume of x-sources to be highlighted in the violation message by the [Info_x_sources](#).

Used by	Info_x_sources
Options	low, medium, high

Default value	low
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_x_display_limit medium</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_x_display_limit = high</code>

dft3BitClock

Use this parameter to limit the number of clock simulation cycles to 3. By default, SpyGlass performs clock simulation for 5 cycles.

Used by	All SpyGlass DFT ADV Rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft3BitClock on</code>
<i>Usage in goal/source files</i>	<code>-dft3BitClock=on</code>

dft_use_cumulative_fault_status

Use this parameter when more than one run is used to determine stuck-at coverage. If this is the case, set the value of the `dft_use_cumulative_fault_status` parameter to start for the first run of multi-mode analysis. For subsequent runs, specify the `cumulative_stuck_at_fault_status.rpt` file generated in a previous run, as an input to this parameter.

Used by	Info_coverage
Options	off, start, cumulative_stuck_at_fault_status.rpt
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_use_cumulative_fault_status start</code>
<i>Usage in goal/source files</i>	<code>-dft_use_cumulative_fault_status= "cumulative_stuck_at_fault_status.rpt"</code>

dftAllowMergeOfSameClock

Enables the [Clock_08](#) rule to report for clock merging only if the merged node is coming from the different user specified test clocks and not same test clock.

When the value of the parameter is set to off, all cases of clock merging are reported.

Used by	Clock_08
Options	on, off
Default	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftAllowMergeOfSameClock on</code>
<i>Usage in goal/source files</i>	<code>-dftAllowMergeOfSameClock=on</code>

dftAllowNonXValueAtStartOfSensitizedPathInSoc_02

The *dftAllowNonXValueAtStartOfSensitizedPathInSoc_02* parameter is deprecated.

The non-x nodes are now always allowed as start point for the sensitized *require_path* check. Therefore, the *dftAllowNonXValueAtStartOfSensitizedPathInSoc_02* parameter is ignored.

dftAllowedScanChainLengthDeviation

Specifies the maximum allowed deviation from the expected length for scan chains. The *Scan_36* rule flags scan chains, where the expected scan chain length differs from the associated number of scan flip-flops by more than the allowed deviation (specified with *dftAllowedScanChainLengthDeviation* parameter).

Used by	<i>Scan_36</i>
Options	Integer
Default value	20
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftAllowedScanChainLengthDeviation 10</code>
<i>Usage in goal/source files</i>	<code>-dftAllowedScanChainLengthDeviation=10</code>

dftAutoFix

Causes the [Async_07](#), [Clock_11](#), [Clock_11_capture](#), [Latch_08](#), [Topology_01](#), [TA_06](#), and [TA_09](#) rules to generate modified RTL source files that have the suggested changes to fix the rule-violations of these rules. By default, the `dftAutoFix` rule parameter is not set.

The `dftAutoFix` parameter generates the [dft_summary.rpt](#) report that contains the details of autofix performed by the SpyGlass DFT ADV product. This report is generated in the `<current_working_directory>/spyglass_reports/rme/` directory.

NOTE: Please note the following points:

- 📖 *SpyGlass honors the `dftAutoFix` rule parameter only if the [rme_active](#) parameter is also set.*
- 📖 *Use of the `dftAutoFix` rule parameter may not result in suggested fixes for all rule violations of these rules.*
- 📖 *The `dftAutoFix` rule parameter can use already existing test mode and test clock ports for Verilog modules.*
- 📖 *SpyGlass DFT ADV now ignores the value of test mode port name (`TMPORT`) and test clock port name (`TCLKPORT`), if specified using this parameter. You can specify test mode signal and test clock signal using the `dft_autofix_testmode_signal` and `dft_autofix_testclock_signal` parameters, respectively.*

Used by	Async_07 , Clock_11 , Clock_11_capture , Latch_08 , Topology_01 , TA_06 , TA_09
Options	<Rules-list> <option-files-list>
Default value	RULES[nothing]
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftAutoFix " {+RULES\[Clock_11,Clock_11_capture\]} "</code>
<i>Usage in goal/source files</i>	<code>-dftAutoFix " {+RULES\[Clock_11,Clock_11_capture]} "</code>

dftDebugData

Use this parameter to enable the back annotation (SpyGlass DFT ADV

related information in the Atrenta Console) feature for better debugging information.

Used by	<i>Info_DftDebugData</i>
Options	off, on, bp_off, cell_properties_off, bp_and_cell_properties_off
Default value	bp_off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftDebugData bp_off</code>
<i>Usage in goal/source files</i>	<code>-dftDebugData=bp_off</code>

To view the SpyGlass DFT ADV debugging information in the Atrenta Console, set the value of the `dftDebugData` rule parameter to `on`.

When you run the SpyGlass DFT ADV product and enable the `dftDebugData` rule parameter, you can view the following SpyGlass DFT ADV debugging information in the schematics (Hierarchical Schematic and Incremental Schematic Windows in the Atrenta Console) at both the instance and terminal level:

- Simulation Value
- Clock reachability attributes
- Blocked Path attributes
- Cell / Instance attributes

The following table explains the available parameter values:

TABLE 3 dftDebugData Parameter Values

Value	Description
off	All the debugging information is off.
on	All the debugging information is switched on.
bp_off	All the debugging information is switched on except blocked path information.

TABLE 3 dftDebugData Parameter Values

Value	Description
cell_properties_off	All the debugging information is switched on except cell or instance properties.
bp_and_cell_properties_off	Only simulation and clock attributes are switched on. The blocked path attributes and cell properties are not displayed.

To view the examples of the behavior of the *Info_dftDebugData* rule, when the *dftDebugData* parameter is set to the above values, see Example Code and/or Schematic section of the [Info_DftDebugData](#) rule.

For more information on how to view the SpyGlass DFT ADV debugging information in the Atrenta Console, refer to the *Atrenta Console Reference Guide*.

dftCaptureCriteria

Use this parameter to select the criteria used to mark the flip-flops as capture-ready.

If the value of the parameter is set to controllability, flip-flops are treated capture-ready if:

- Their set/reset pins are controllable to their inactive state
- The states of scan flip-flops allow test-clock to reach the clock-pin.

When you set `dftCaptureCriteria` to `simulation`, flip-flops are treated as capture-ready if their set/reset pins are disabled and test clocks are reaching their clock-pins.

Used by	Clock_11_capture, Info_coverage
Options	controllability, simulation
Default value	simulation
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter dftCaptureCriteria controllability</code>
<i>Usage in goal/source files</i>	<code>-dftCaptureCriteria=controllability</code>

dftClockEdgeFaultAnalysis

Use this parameter to direct the [Info_untestable](#) and [Info_undetectedCause](#) rules to identify faults that could not be detected because the capture happens at opposite edge of testclock used for these faults.

By default, these faults are not listed as detected in the fault Coverage calculations.

NOTE: *Using the `dftClockEdgeFaultAnalysis` rule parameter may result in increased runtime.*

Used by	Info_untestable , Info_undetectedCause
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftClockEdgeFaultAnalysis on</code>
<i>Usage in goal/source files</i>	<code>-dftClockEdgeFaultAnalysis=on</code>

dftClockEdgeType

Use the `dftClockEdgeType` rule parameter to specify the clock edge conditions for a [Clock_23](#) rule violation.

By default, all flip-flops triggered by the `Clock_23` rule-violating clock are highlighted.

Similarly, when the `dftClockEdgeType` rule parameter is set to `negedge`, then the `Clock_23` rule flags all the positive edged scan flip-flops. Otherwise, when the rule parameter is set to `posedge` or `none`, then the `Clock_23` rule flags all the negative edged scan flip-flops.

SpyGlass DFT ADV Rule Parameters

Used by	Clock_23
Options	none, posedge, negedge
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftClockEdgeType posedge</code>
<i>Usage in goal/source files</i>	<code>-dftClockEdgeType=posedge</code>

dftCPMechanism

Use the `dftCPMechanism` parameter to simulate the clocks either in parallel or one-by-one separately for the [Clock_10](#), [Clock_11](#), and [Info_testclock](#) rules.

Used by	Clock_10 , Clock_11 , Info_testclock
Options	sequential, parallel
Default value	sequential
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftCPMechanism parallel</code>
<i>Usage in goal/source files</i>	<code>-dftCPMechanism=parallel</code>

dftDesignState

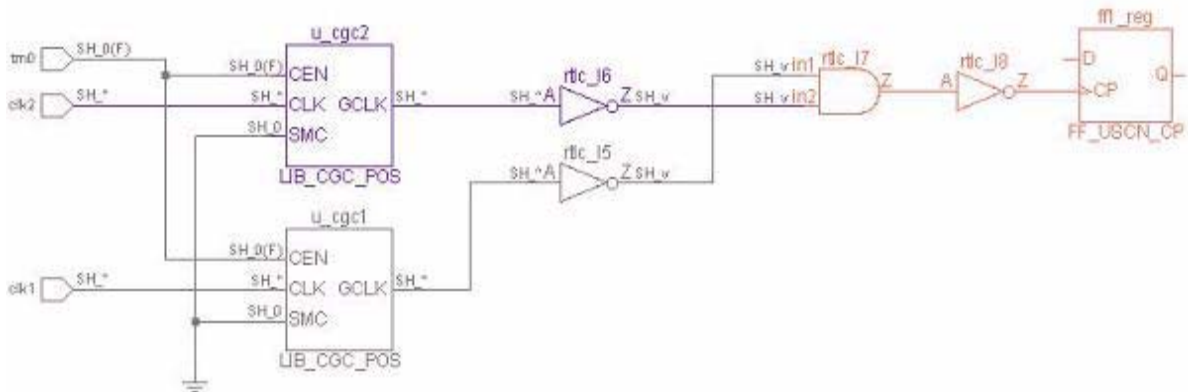
Use this parameter to specify the design state as either `pre_scan_stitched` state or `post_scan_stitched`.

If the Design State is Pre-Scan Stitched

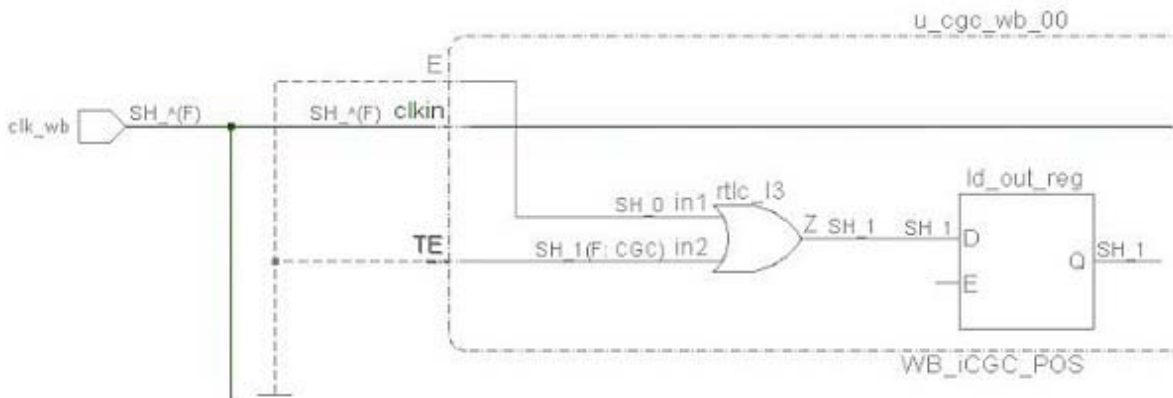
If the design state is `pre_scan_stitched`, a CGC that has a `test_enable` tied to ground (VSS), allows the test clock to propagate through it during `scanshift` mode. In this case it is assumed that after scan stitching, that is, when the value of the `dftDesignState` parameter is `post_scan_stitched`, the grounded `test_enable` pin of the CGC will be properly connected with the `SCAN_MODE` signal.

For example, consider the following figure where both the CGC outputs

have test clock when respective clock-in is simulated:



Use the [dft_stitching_exception](#) constraint to override the CGC behavior in the `pre_scan_stitched` state. However, if the [dft_stitching_exception](#) constraint is not used, then it is assumed that the grounded test enable pin of CGC would be later connected to scan enable when scan stitch is done for design. To handle such a scenario, the test enable is considered as if it is connected to 1 in the shift mode. Also, this scenario is annotated in schematic on Test Enable (TE) pin as `SH_1(F:CGC)` as shown in the following figure:

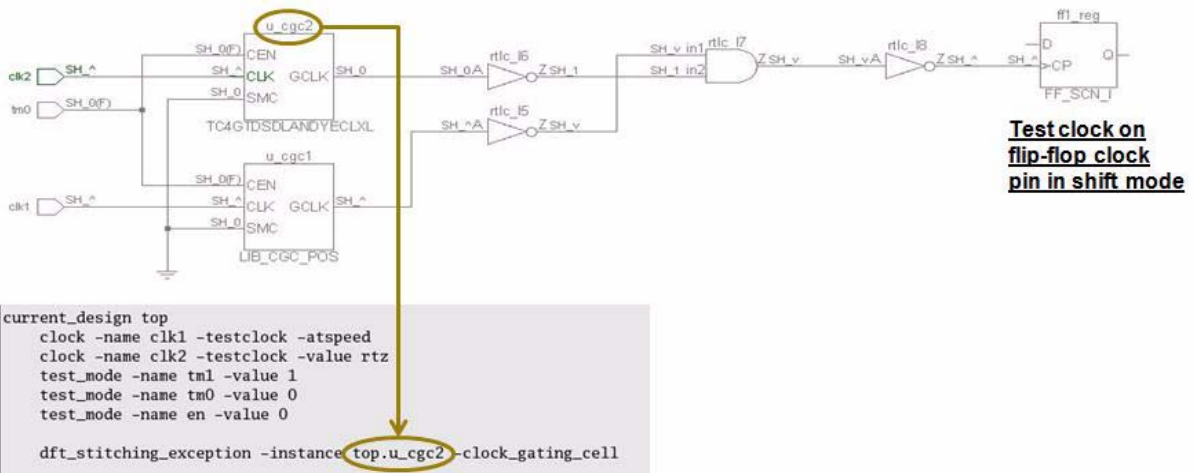


In the SOC designs, where some IP blocks are `pre_scan_stitched` and

SpyGlass DFT ADV Rule Parameters

others are `post_scan_stitched`, it is recommended to set the value of the `dftDesignState` parameter to `pre_scan_stitched` and use the `dft_stitching_exception` constraint to designate the areas of the design that should be treated as `post_scan_stitched`. For these exceptions, the CGC will be treated as a regular CGC and if its `test_enable_pin` is tied to ground, the test clock will not propagate through it during the scanshift mode.

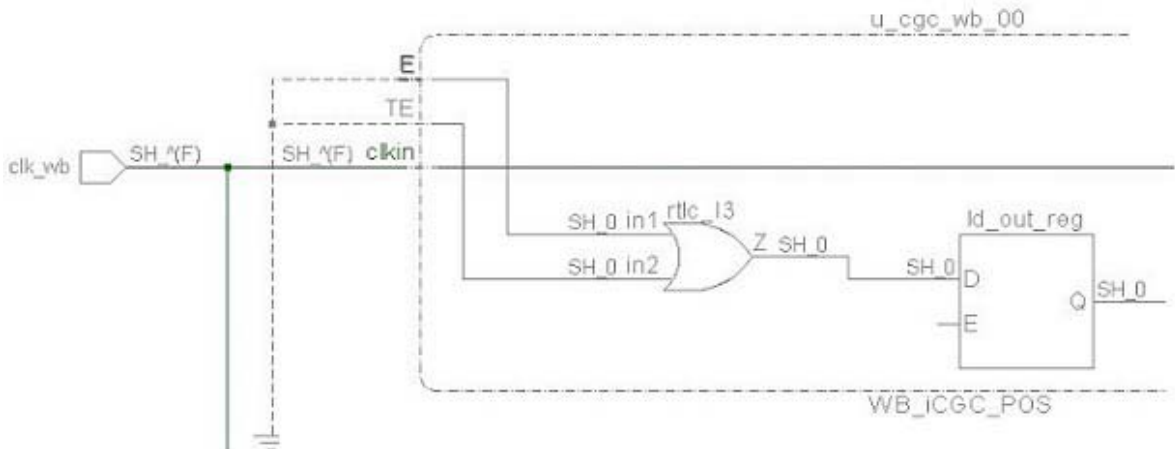
For example, one CGC allows the clock to propagate but other CGC is used to drive a constant value as shown in the following figure:



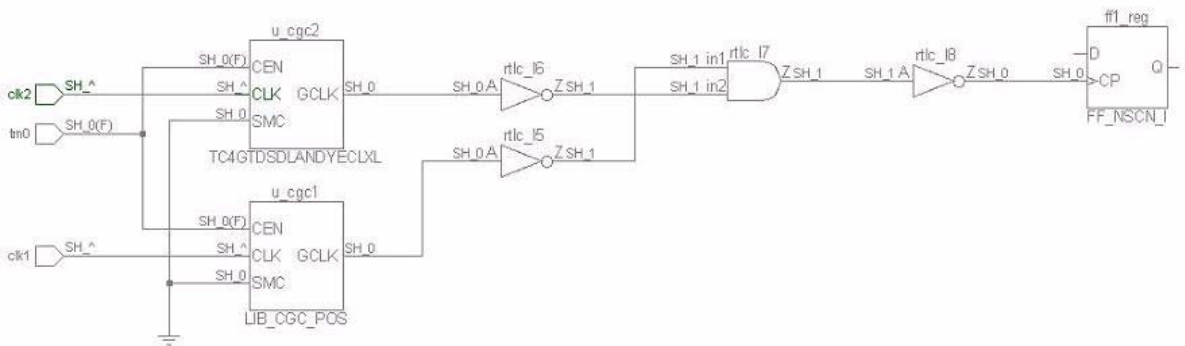
If the Design State is Post-Scan Stitched

if design is `post_scan_stitched`, regular flip-flops are replaced by the scan flip-flops, a scan chain is formed between them and the clock propagation behavior remains unchanged in shift, capture and atspeed mode. Simulation of `test_mode` values and clock propagation is based on the actual values of the CGC pin. Also, the clock propagation depends upon the enable pin (system enable and test enable) condition.

For `post_scan_stitched` designs, if the `dft_stitching_exception` SGDC constraint is used, then it means that design is stitched and the CGC is treated as regular with test enable pin to tied to ground as shown in the following figure:



For example, consider the following figure where both the CGCs have a constant value of 0 during scanshift mode.



By default, the value of the dftDesignState is set to pre_scan_stitched.

Used by	All DFT rules
Options	pre_scan_stitched, post_scan_stitched
Default value	pre_scan_stitched
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter dftDesignState post_scan_stitched</code>
<i>Usage in goal/source files</i>	<code>-dftDesignState=post_scan_stitched</code>

dftDoInferredNoScanAnalysis

Causes the applicable SpyGlass DFT ADV rules to infer those flip-flops as unscannable that are found in the fan-in cone of a set/reset/clock pin of any other flip-flop.

Sets depth limit for inferring flip-flops as no scan. This parameter considers only those flip-flops that are found in the fanin-cone of a set/reset/clock pin of another flip-flop.

By default, the `dftDoInferredNoScanAnalysis` rule parameter is set to 1 and SpyGlass DFT ADV infers only the directly connected flip-flop as no scan.

Set the `dftDoInferredNoScanAnalysis` rule parameter to another positive integer to have SpyGlass DFT ADV check for flip-flops in the specified depth of the fan-in. For example, if you set the `dftDoInferredNoScanAnalysis` rule parameter to 2, SpyGlass DFT ADV infers flip-flops as no scan that are directly connected and the flip-flops that are one level removed.

Set the `dftDoInferredNoScanAnalysis` rule parameter to 0 to disable the feature.

Used by	All applicable SpyGlass DFT ADV rules
Options	0 or a positive integer number
Default value	1

Example

<i>Console/Tcl-based usage</i>	<code>set_parameter dftDoInferredNoScanAnalysis 2</code>
<i>Usage in goal/source files</i>	<code>-dftDoInferredNoScanAnalysis=2</code>

dftDoLogicalRedundancyCheck

Controls the *Info_untestable* and *Info_logicalRedundant* rules with respect to the detection and processing of the logically-redundant pins.

A pin is a logically-redundant pin when it has a constant value during simulation regardless of the values applied its driving signals.

By default, the `dftDoLogicalRedundancyCheck` parameter is set to off and the `Info_coverage` and `Info_logicalRedundant` rules do not check for logically-redundant pins.

When the `dftDoLogicalRedundancyCheck` parameter is set to on, the `Info_coverage` and `Info_logicalRedundant` rules detect and process logically-redundant pins, which can impact fault coverage and test coverage.

When this parameter is set to `distinct_pg_capture`, then the `Info_logicalRedundant` rule reports separate violation messages for the power, ground, and capture modes.

NOTE: *Use of the `dftDoLogicalRedundancyCheck` rule parameter increases runtime.*

Used by	<i>Info_untestable</i> , <i>Info_logicalRedundant</i>
Options	off, on, <code>distinct_pg_capture</code>
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftDoLogicalRedundancyCheck on</code>
<i>Usage in goal/source files</i>	<code>-dftDoLogicalRedundancyCheck=on</code>

dftDsmConvergingPathsLimit

Specifies the number of converging paths to be shown in the violation of the *Atspeed_27* rule.

When the value of the parameter is set to 0, the *Atspeed_27* rule reports all converging paths in the violation.

Used by	<i>Atspeed_27</i>
Options	Any positive integer, 0

SpyGlass DFT ADV Rule Parameters

Default value	5
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftDsmConvergingPathsLimit 50</code>
<i>Usage in goal/source files</i>	<code>-dftDsmConvergingPathsLimit=50</code>

dftEnablePinNameForCGC

Sets the enable pin name of the clock gating cell for the [TA_08](#) rule.

Used by	TA_08
Options	string
Default value	non empty string with the text 'EN'
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftEnablePinNameForCGC EN11</code>
<i>Usage in goal/source files</i>	<code>-dftEnablePinNameForCGC=EN11</code>

dftExpectedScanChainLength

Specifies the expected average length of scan chains. The rule [Scan_36](#) flags scan chains, where the expected scan chain length (specified with `dftExpectedScanChainLength` parameter) differs from the actual number of scan flip-flops by more than the value set using the [dftAllowedScanChainLengthDeviation](#) parameter.

Used by	Scan_36
Options	Integer
Default value	100
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftExpectedScanChainLength 250</code>
<i>Usage in goal/source files</i>	<code>-dftExpectedScanChainLength=250</code>

dftFindCombLoopThruSetResetToQ

This parameter controls tracing combinational loops through set and reset pins of the flip-flop by the `Topology_01` rule.

By default, the value of this parameter is on and the `Topology_01` rule traces combinational loops through set and reset pin of the flip-flop.

When the value of the parameter is set to off, the `Topology_01` rule does not consider the path between the reset pin and the Q pin of the flip-flop as combinational.

Used by	<code>Topology_01</code>
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftFindCombLoopThruSetResetToQ off</code>
<i>Usage in goal/source files</i>	<code>-dftFindCombLoopThruSetResetToQ=off</code>

dftGenerateStuckAtFaultReport

This parameter is used to generate pin based fault reports for stuck-at faults. This parameter is used by `Info_coverage` rule.

Used by	<code>Info_coverage</code>
Options are	none, all, all_except_detected, detected, undetected, synthesis_redundant, unused, untestable, tied, blocked, logical_redundant
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftGenerateStuckAtFaultReport all</code>
<i>Usage in goal/source files</i>	<code>-dftGenerateStuckAtFaultReport=all_except_detected</code>

dftFunctionalClockPropagation

This parameter controls the technique used for populating the functional clock trees. A functional clock tree for a particular clock source is defined as the set of all flip-flops that get a pulse when the clock source is pulsed for some circuit state.

By default, the value of this parameter is set to `simulation`. In this case, each clock pulse (one functional clock at a time) is simulated over functional mode (this sets the circuit state) and all design nodes that get a clock pulse (in phase or inverted) are marked as belonging to the tree of the source clock. This ensures that a flip-flop, gets one functional clock or no functional clock.

When the value of this parameter is set to `traversal`, a sensitizable fanin traversal, starting at a flip-flop clock pin, is performed. All functional clocks that reach this fanin traversal are associated with the flip-flop. In this approach, a flip-flop may get more than one functional clock.

Used by	Clock Rules
Options are	<code>simulation</code> <code>traversal</code>
Default value	<code>simulation</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftFunctionalClockPropagation traversal</code>
<i>Usage in goal/source files</i>	<code>-dftFunctionalClockPropagation=traversal</code>

dftGenerateTextReport

Specifies whether the corresponding rules generate a text report.

NOTE: *Currently, the `dftGenerateTextReport` rule parameter support is available for a limited set of rules only. In future releases, this support will be available for all rules that allow optional text report generation.*

Used by	Async_02_shift , Async_02_capture , Info_inferredNoScan , Info_undetectedCause , Info_potDetectable
Options	Space-separated rule name list enclosed in curly brackets
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftGenerateTextReport {Async_02_shift}</code>
<i>Usage in goal/source files</i>	<code>-dftGenerateTextReport={Async_02_shift}</code>

dftIdentifyTestPoints

Specifies type of control points selected in the TA_09 rule.

Used by	TA_09
Options	<control_only, observe_only, both_control_observe>
Default value	both_control_observe
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftIdentifyTestPoints control_only</code>
<i>Usage in goal/source files</i>	<code>-dftIdentifyTestPoints=control_only</code>

dftIgnoreConstantOrUnusedFlipFlops

Controls whether or not the [Clock_10](#), [Clock_11](#), [Clock_11_capture](#), [Scan_08](#), and [Scan_16](#) rules would report violation for the following conditions:

- A constant value is specified to the D pin of a flip-flop.
- A constant value is specified to the Q pin of a flip-flop.
- The value at the Q pin of a flip-flop is not captured at a scan point.

SpyGlass DFT ADV Rule Parameters

Used by	Clock_10 , Clock_11 , Clock_11_capture , Scan_08 , Scan_16
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftIgnoreConstantOrUnusedFlipFlops on</code>
<i>Usage in goal/source files</i>	<code>-dftIgnoreConstantOrUnusedFlipFlops=on</code>

dft_generate_no_fault_and_add_fault_report

Use this parameter to generate reports showing faults marked as `no_fault` and `add_fault`.

By default, the value of the parameter is set to on. Set the value of the parameter to off to disable report generation for the faults that are marked as `no_fault` and `add_fault`.

Used by	Info_addFault , Info_noFault
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_generate_no_fault_and_add_fault_report off</code>
<i>Usage in goal/source files</i>	<code>-dft_generate_no_fault_and_add_fault_report =off</code>

dft_generate_robustness_audit_report

Enables robustness audit generation.

When set to on, causes the [Coverage_audit](#) rule to generate the robustness information in the [stuck_at_coverage_audit](#) report.

Used by	<i>Coverage_audit</i>
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_generate_robustness_audit_report off
<i>Usage in goal/source files</i>	-dft_generate_robustness_audit_report=off

dft_max_files_in_a_directory

Specifies maximum number of csv files in a single directory.

Used by	All DFT Rules
Options	<any natural number>
Default value	2000
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_max_files_in_a_directory 5000
<i>Usage in goal/source files</i>	-dft_max_files_in_a_directory=5000

dft_maximum_number_of_rows_with_schematic

Specifies the number of rows in the spreadsheet that contain the schematic data. The spreadsheets are generated by the rules listed in the table below. For example, assume that the value for the *dft_maximum_number_of_rows_with_schematic* parameter is set to 4. In this case, even if the spreadsheet for a rule contains more than 4 rows, spyglass will show schematic for only first 4 rows given in the spreadsheet. Subsequent rows will not have any schematic back-reference.

Double-click on a row in the spreadsheet to display the violating instances for the affected rules. These instances can either be flip-flops or memory cells. For such instances, the source in the design, which has caused the rule violation, is not displayed

SpyGlass DFT ADV Rule Parameters

However, if you open the Incremental Schematic for a rule violation using Atrenta Console, the Incremental Schematic window displays the source, which has caused the violation. It also displays the affected instances.

Used By	Async_07 Async_08 Async_09 Async_11 Async_13 Async_15 Clock_08 Clock_11 Clock_11_capture Clock_17 Clock_29 Info_DftDebugData
Options	Any non-negative integer
Default Value	10
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_maximum_number_of_rows_with_schematic 15</code>
<i>Usage in goal/source files</i>	<code>-dft_maximum_number_of_rows_with_schematic 15</code>

dft_min_scannability_ratio_for_test_points

Specifies the minimum scannability ratio (defined as # of scannable flip-flops/ total # of flip-flops) the design must be at, before test_point analysis (TA_09) starts and test points (control /observe) are identified.

You can also set the value of the parameter to 0 turn off this parameter.

Used by	TA_09
Options	0, any value between 0 and 1
Default value	0.7
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_min_scannability_ratio_for_test_points 0.8
<i>Usage in goal/source files</i>	-dft_min_scannability_ratio_for_test_points=0.8

dft_pattern_count

Specifies the number of test patterns used for coverage estimation by the [Latch Rules](#) rule.

Used by	Latch Rules
Options	A positive non-zero integer
Default value	64000
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_pattern_count 10
<i>Usage in goal/source files</i>	-dft_pattern_count=10

dft_ignore_bb_inout_term_for_topology_05

Use this parameter to ignore inout terminals of black boxes to be considered as a driver by the [Topology_05](#) rule.

By default, the value of the *dft_ignore_bb_inout_term_for_topology_05* parameter is set to off. Therefore, the [Topology_05](#) rule considers the inout terminals of the black boxes as a driver.

Set the value of the parameter to on to ignore the inout terminals of the black boxes.

Used by	Topology_05
Options	off, on
Default value	off
Example	

<i>Console/Tcl-based usage</i>	set_parameter dft_ignore_bb_inout_term_for_topology_05 on
<i>Usage in goal/source files</i>	-dft_ignore_bb_inout_term_for_topology_05=on

dft_ignore_constant_supply_flip_flops

This parameter ignores synthesis redundant (SR) flip-flops from rule checking for the following rules:

- [Clock_10](#)
- [Clock_11](#)
- [Clock_11_capture](#)
- [Scan_08](#)
- [Scan_16](#)

A flip-flop is inferred synthesis redundant (SR) when both the following conditions are true:

- The flip-flop is inferred from behavioral RTL code and not an instantiation of the .lib cell
- The dft_ignore_constant_supply_flip_flops parameter is on and when at least one of the following conditions hold true:
 - D-pin of a flip-flop is tied to a constant value in PG_MODE and async pins have no effect on the state of the flip-flop
 - Q-pin of flip-flop is tied to a constant value in PG_MODE

It is recommended to use this parameter with the `set_option enable_const_prop_thru_seq` command. If you do not specify the `enable_const_prop_thru_seq` command, constant supply is propagated only through first level of flip-flops. Therefore, cascaded flip-flops are not impacted.

Used by	Clock_10 , Clock_11 , Clock_11_capture , Scan_08 , Scan_16
Options	on, off
Default value	off

Used by	Clock_10 , Clock_11 , Clock_11_capture , Scan_08 , Scan_16
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_constant_supply_flip_flops on</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_constant_supply_flip_flops=on</code>

dft_ignore_no_scan_driver_in_Async_02_capture

Use this parameter to ignore no_scan flip-flops in the [Async_02_capture](#) rule.

Used by	Async_02_capture
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_ignore_no_scan_driver_in_Async_02_capture on</code>
<i>Usage in goal/source files</i>	<code>-dft_ignore_no_scan_driver_in_Async_02_capture=on</code>

dft_infer_clock_gating_cell

Selects the behavior of automatic clock gating cell (CGC) inferencing.

By default, the value of this parameter is set to `on` and the cells similar to Clock Gating Cells (CGCs) are treated as CGCs.

Set the value of the parameter to `off` to turn off CGC inferencing.

See [Identifying Clock Gating Cells](#) section for more information on the ways to infer CGCs.

For more information on identifying CGCs, see [Identifying Clock Gating Cells](#).

SpyGlass DFT ADV Rule Parameters

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_infer_clock_gating_cell off</code>
<i>Usage in goal/source files</i>	<code>-dft_infer_clock_gating_cell=off</code>

dft_infer_clock_gating_cell_test_enable

Use this parameter to specify the nomenclature for the test enable (TE) pin of the Clock Gating Cells (CGCs) using which they are identified in the design.

Once the TE pin is identified based on the specified input, the other enable pins are inferred as system/data enable pins.

Used by	All SpyGlass DFT ADV Rules
Options	space-separated list of TE names
Default value	"TEST_ENABLE SCAN ENABLE TEN TE T SE"
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_infer_clock_gating_cell_test_enable "my_te"</code>
<i>Usage in goal/source files</i>	<code>- dft_infer_clock_gating_cell_test_enable="my_ te"</code>

dft_scannable_latches

Use this parameter to mark non-transparent latches as scannable.

By default, the value of this parameter is set to off. In this case, latches are not considered for scannability.

Set the value of the parameter to on to mark non-transparent latches as scannable when at least one of the following conditions holds true:

- Enable pin of the latch has an inferred test clock
- Enable pin of the latch has a user specified test clock when [dft_use_specified_and_inferred_clocks](#) parameter is set to on

For more information on scannable latches, see [Scannable](#) latches.

Used by	All DFT rules
Options	on, off
Default Value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_scannable_latches on
<i>Usage in goal/source files</i>	-dft_scannable_latches=on

dft_treat_suffix_as_pattern

Modifies the behavior of the `-register_suffix` field of the of the [force_scan](#), [force_no_scan](#), or [no_atspeed](#) SGDC commands.

When the value of the parameter is set to on, the `-register_suffix` value is used as a pattern to be matched with the register name. The pattern may be present anywhere in the register name, excluding the path. See the [Examples](#) section of the [force_no_scan](#) constraint for more information.

When the value of the parameter is set to off, the `-register_suffix` value is used to match only with register name suffix.

Used by	All SpyGlass DFT ADV Rules
Options	on, off
Default Value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dft_treat_suffix_as_pattern on
<i>Usage in goal/source files</i>	-dft_treat_suffix_as_pattern=on

dft_use_simulation_based_unblocked_path_check_for_scan_chain_tracing

Modifies blocking-check during scan chain tracing.

By default, the value of the parameter is set to `off`. In this case, only structural blockage check is performed on instance pins and scan chain tracing stops at point where multiple unblocked structural paths are found.

Set the value of the parameter to `on` to perform simulation based analysis to get unique driver when multiple unblocked structural paths are found during scan chain tracing.

Used by	All SpyGlass DFT ADV Rules
Options	<code>on</code> , <code>off</code>
Default Value	<code>off</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dft_use_simulation_based_unblocked_path_check_for_scan_chain_tracing on</code>
<i>Usage in goal/source files</i>	<code>-dft_use_simulation_based_unblocked_path_check_for_scan_chain_tracing=on</code>

dftInferredDriverControl

Affects clock propagation through clock gating cell (CGC) during capture (static and atspeed) as this affects system-enable controllability.

By default the value of this parameter is `optimistic` and the presence of a single scan source (scan flip-flop or primary port) in the unblocked fan-in cone makes the node controllable.

You can set the value of this parameter to one of the following values:

- `pessimistic`: Specifies that the presence of a single non-scan source (black box, hanging net, noscan) in the unblocked fan-in cone will make the node uncontrollable. This impacts clock propagation through clock gating cell (CGC) during capture (static and atspeed) because this will affect system-enable controllability.

- `optimistic_through_cgc_n_shaper`: Specifies that the presence of a single scan source (scan flip-flop or primary port) in the unblocked fanin cone, including cgc and clock shapers, makes the node controllable.

Used by	Clock_11_capture , Info_testclock , Diagnose_testclock
Options	optimistic, pessimistic, <code>optimistic_through_cgc_n_shaper</code>
Default	optimistic
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftInferredDriverControl pessimistic</code>
<i>Usage in goal/source files</i>	<code>-dftInferredDriverControl=pessimistic</code>

dftMaxScanChainLength

Specifies the maximum length for scan chains. The [Scan_35](#) and [Scan_36](#) rules report scan chains, where the number of scan flip-flops exceeds this limit.

Used by	Scan_35 , Scan_36
Options	Integer
Default value	600
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftMaxScanChainLength 250</code>
<i>Usage in goal/source files</i>	<code>-dftMaxScanChainLength=250</code>

dftPOSDETECT_credit

Use the `dftPOSDETECT_credit` rule parameter to regulate the effect of potentially detectable faults on the test coverage and fault coverage evaluation by the [Info_untestable](#), [Info_undetectedCause](#), [TA_06](#), [TA_09](#) rules.

By default, the value of the `dftPOSDETECT_credit` parameter is set to

SpyGlass DFT ADV Rule Parameters

0, which indicates that all potentially detectable faults would not be detected.

You can set the `dftPOSDETECT_credit` rule parameter to any floating-point value between 0 and 1. A value of 1 indicates that all potentially detectable faults would be detected. A value of 0 (default) indicates all potentially detectable faults would not be detected. Other values indicate that corresponding percentage of potentially detectable faults are to be considered. For example, a value of 0.5 indicates that 50% of potentially detectable faults would be detectable by the ATE tool. Hence, the test-coverage and fault-coverage figures calculated by these rules are accordingly updated.

When this parameter is set to any floating-point value between 0 and 1 then that fraction of all potentially detectable faults would be considered as detected. For example, a value of 0.5 indicates that 50% of potentially detectable faults would be considered as detectable by the ATE tool. Hence, the test-coverage and fault-coverage figures calculated by these rules are accordingly updated.

Used by	Info_untestable , Info_undetectedCause , TA_06 , TA_09
Options	a floating-point number between 0 and 1 (both inclusive)
Default value	0
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftPOSDETECT_credit 0.25</code>
<i>Usage in goal/source files</i>	<code>-dftPOSDETECT_credit=0.25</code>

dftReportIfUsedAsClock

Enables the [Clock_08](#) rule to report a violation for clock merging only if the merged node is driving CP-pin of a flip-flop or EN-pin of a latch.

Used by	Clock_08
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftReportIfUsedAsClock off</code>
<i>Usage in goal/source files</i>	<code>-dftReportIfUsedAsClock=off</code>

dftReportOnlyBBForBIST

When this parameter is set to on, [BIST_04](#) ignores all black boxes except for the case when a black box is driving an observable point. When set to off, black boxes are treated as X-sources unless scan wrap is specified.

Used by	BIST_04
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftReportOnlyBBForBIST off</code>
<i>Usage in goal/source files</i>	<code>-dftReportOnlyBBForBIST=off</code>

dftReportWDriversInTriBus

Causes the [Tristate_06](#) rule to flag if all strong drivers are off and only the weak drivers (pullup, pulldown, keeper, bus holder) are connected to the bus. When the value of the parameter is set to off, then the presence of weak drivers on a floating node is ignored.

SpyGlass DFT ADV Rule Parameters

Used by	Tristate_06
Options	on,off
Default	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftReportWDriversInTriBus on</code>
<i>Usage in goal/source files</i>	<code>-dftReportWDriversInTriBus=on</code>

dftSkipLogicForTestPoints

Use this parameter to select the conditions for ignoring nodes as candidates for test points.

Use this parameter to set an improvement threshold accepting candidates for test point insertion. If the number of nodes for which controllability / observability improves is less than this threshold for a candidate node, then that candidate is ignored. The improvement count depends on the basis on which TA messages are sorted. See [dftSortTAMsgOnFaultCount](#) for more details.

Used by	TA_09
Options	<DFT_UU DFT_SR DFT_NonX DFT_UU_SR DFT_SR_NonX DFT_UU_SR_NonX>
Default value	DFT_UU_SR_NonX
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSkipLogicForTestPoints DFT_UU</code>
<i>Usage in goal/source files</i>	<code>-dftSkipLogicForTestPoints=DFT_UU</code>

dftSkipTestPointsOnImprovementLessThan

Used to skip those nets, which result in improvement count less than the specified value. The improvement count is dependent on the basis on which TA messages are sorted. See '[dftSortTAMsgOnFaultCount](#)' for more details.

Used by	TA_09
Options	<positive integer value>
Default value	0
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSkipTestPointsOnImprovementLessThan 5</code>
<i>Usage in goal/source files</i>	<code>-dftSkipTestPointsOnImprovementLessThan=5</code>

dftTagAsync07Clock11Scan08

This parameter simplifies debugging Scan_08 violations. When the value of this parameter is set to on, a unique tag is added to asynchronous sources (set, reset, or clock) for the violations reported by the Async_07, Clock_11, and Scan_08 rules. This enables the correlation of Async_07 and Clock_11 rules with the Scan_08 rule. The tag is displayed only when Scan_08 rule is run along with at least one of Async_07 or Clock_11 rules.

When the value of this parameter is set to off, it becomes difficult to determine the cause of the violations reported by the Scan_08 rule.

Used by	Async_07 , Clock_11 , Scan_08 , Scan_16 , and Clock_11_capture
Options	on,off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTagAsync07Clock11Scan08 on</code>
<i>Usage in goal/source files</i>	<code>-dftTagAsync07Clock11Scan08=on</code>

dftScanDataConnectivityCheck

Specifies the type of connectivity checking to be performed on the scan data input pin. The value is selected based on the current stage of the RTL in the design flow (pre/post scan stitched design).

SpyGlass DFT ADV Rule Parameters

Used by	Scan_27
Options	TIED_0, TIED_1, FEEDBACK_DIRECT, FEEDBACK_THRU_BUF_INV, STICHED_CHAIN
Default value	TIED_0
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftScanDataConnectivityCheck FEEDBACK_DIRECT</code>
<i>Usage in goal/source files</i>	<code>- dftScanDataConnectivityCheck=FEEDBACK_DIRECT</code>

NOTE: *In the pre stitch stage, the scan input pin of a scan flip-flop is either tied (ground or power) or it is driven by a feedback loop from the flip-flop's output. In the post stitch stage the scan input pin of a flip-flop is driven by the primary inputs, Lockup latch, or other scan flip-flop outputs.*

dftScanEnablePinValue

This parameter specifies the active value scan cell enable pins. When set to active, then Scan_28 rule reports a violation, if the pin is inactive.

When set to inactive, then Scan_28 rule reports a violation, if the pin is active.

Used by	Scan_28
Options	active, inactive
Default value	inactive
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftScanEnablePinValue active</code>
<i>Usage in goal/source files</i>	<code>-dftScanEnablePinValue=active</code>

dftSENames

Defines the pin names for the scanenable ports of scan modules used by the Scan_34 rule.

Used by	Scan_34
Options	space separated list of names
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSENames se</code>
<i>Usage in goal/source files</i>	<code>-dftSENames=se</code>

dftSetAllBBScanwrapped

Use this parameter to declare all black box as scanwrapped.

By default, a black box instance is assumed to be scanwrapped only if it is specified with the [scan_wrap](#) constraint.

This parameter is for making all the black boxes scanwrapped.

Used by	All SpyGlass DFT ADV rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSetAllBBScanwrapped on</code>
<i>Usage in goal/source files</i>	<code>-dftSetAllBBScanwrapped=on</code>

dftSetAllLatchT

Use this parameter to declare all latches as transparent.

By default, a latch is assumed to be transparent only when its enable pin is active in the scanshift mode.

Used by	All SpyGlass DFT ADV rules
Options	off, on
Default value	off
Example	

<i>Console/Tcl-based usage</i>	set_parameter dftSetAllLatchT on
<i>Usage in goal/source files</i>	-dftSetAllLatchT=on

dftSetAllTriEnableObs

Use this parameter to declare all tristate enable pins as observable.

By default, a tristate enable pin is inferred as observable based on the Observability algorithm.

Used by	All SpyGlass DFT ADV rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftSetAllTriEnableObs on
<i>Usage in goal/source files</i>	-dftSetAllTriEnableObs=on

dftSetCGCEnablePinsValue

Use this parameter to define the undesired value for the test enable/enable pins of the clock gating cells for the [TA_08](#) rule.

Used by	TA_08
Options	active, inactive
Default value	active
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftSetCGCEnablePinsValue inactive
<i>Usage in goal/source files</i>	-dftSetCGCEnablePinsValue=inactive

NOTE: The value active corresponds to a 1 and the value inactive corresponds to 0.

dftSetEffortLevel

Use this parameter to list the rules and specify the number of times a path is analyzed to determine overlapping loops by that rule.

You need to specify each rule name followed by an integer between 0 to infinity as the effort level. If no effort level is set for a rule, it is assumed to be 1.

NOTE: *Currently, this feature is enabled only for [Topology_01](#) rule.*

Used by	Topology_01
Options	Rule list -> Integer (between 0 and infinity)
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSetEffortLevel {Topology_01->10}</code>
<i>Usage in goal/source files</i>	<code>-dftSetEffortLevel={Topology_01->10}</code>

dftSetLatchFedByTClkAsT

Causes all latches that get testclock pulse on their enable pins, when all testclocks are simulated simultaneously, to be considered as transparent for all further analysis.

NOTE: *Do not use the `dftSetLatchFedByTClkAsT` parameter along with the `dft_scannable_latches` parameter as scannability takes precedence over transparency.*

Used by	All SpyGlass DFT ADV Rules
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSetLatchFedByTClkAsT off</code>
<i>Usage in goal/source files</i>	<code>-dftSetLatchFedByTClkAsT=off</code>

dftSetTMTClkNodesAsUnObs

Causes all nodes declared as testmode nodes or testclock nodes (in the user-specified SpyGlass Design Constraints files) to be considered as unobservable for testability analysis.

Used by	Information Rules , Testability Analysis Rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSetTMTClkNodesAsUnObs on</code>
<i>Usage in goal/source files</i>	<code>-dftSetTMTClkNodesAsUnObs=on</code>

dftShowDataInversionInScanChain

Specifies whether or not the [Scan_22](#) rule reports the first flip-flop from the scanin point that gets inverted data.

Used by	Scan_22
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftShowDataInversionInScanChain on</code>
<i>Usage in goal/source files</i>	<code>-dftShowDataInversionInScanChain=on</code>

dftShowForcedValues

Use the `dftShowForcedValues` rule parameter to differentiate between the values enforced by the user and values that have been propagated automatically by the [Info_testmode](#) rule.

The `dftShowForcedValues` rule parameter controls the display of signals in the schematic view. When the `dftShowForcedValues` is set,

signals forced onto specific nodes by the `test_mode` constraints appear as 0(F) or 1(F). Signals that result or are implied from these forced signals appear as 0 or 1 (without the (F) suffix).

This provides clear differentiation between causal signals and result signals. For example, in a design where the output of an AND gate is set to 1 by a `test_mode` constraint and where other constraints on nodes in the fan-in cone for AND cause a 0 on one of the AND gate inputs, the output of the AND will retain its forced value. Therefore, the inconsistency of a 0 on an AND input and a 1 on the AND output may be resolved.

When the `dftShowForcedValues` is not set, the (F) suffix does not appear.

Used by	Info_testmode
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftShowForcedValues off</code>
<i>Usage in goal/source files</i>	<code>-dftShowForcedValues=off</code>

dftShowSourceCentricViolation

Use this parameter to report instance-specific violations and source-specific violations for [Clock_17](#) and [Info_blackboxDriver](#) rules.

If the option is not specified for a rule, the rule reports source-specific violation.

Used by	Clock_17 , Info_blackboxDriver
Options	A space-separated list of rule names followed by the string <code>on</code> or <code>off</code> in the following format: <pre>set_parameter dftShowSourceCentricViolation {<rule>-><off/on> <rule>-><on/off> ...}</pre> Where, <i><rule></i> is a rule name.
Default value	none

Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftShowSourceCentricViolation {Clock_17->on}</code>
<i>Usage in goal/source files</i>	<code>-dftShowSourceCentricViolation={Clock_17->on}</code>

dftShowWaveForm

Use this parameter to select whether or not to display the waveform information for the [Info_testmode](#) and [Diagnose_testmode](#) rules.

Used by	Info_testmode , Diagnose_testmode
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftShowWaveForm on</code>
<i>Usage in goal/source files</i>	<code>-dftShowWaveForm=on</code>

dftSINames

Defines the standard naming convention of scan-in ports of scan modules.

Used by	Scan_34
Options	A space-separated list of names
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSINames {si}</code>
<i>Usage in goal/source files</i>	<code>-dftSINames={si}</code>

dftSkipNonXValueNetsForTA

Use this parameter to select whether or not the [TA_01](#) and [TA_06](#) rules should ignore nets with a non-X value.

By default, these rules process nets with a non-X value.

Used by	TA_01 , TA_06
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSkipNonXValueNetsForTA off</code>
<i>Usage in goal/source files</i>	<code>-dftSkipNonXValueNetsForTA=off</code>

dftSkipPwrGndNetsForTA

Use this parameter to select whether or not the [TA_01](#) rule will report nets that are not controllable due to power/ground nets and the [TA_02](#) rule will report those nets that are not observable due to power/ground nets.

Used by	TA_01 , TA_02
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSkipPwrGndNetsForTA off</code>
<i>Usage in goal/source files</i>	<code>-dftSkipPwrGndNetsForTA=off</code>

dftSkipTAMsgCount

Use this parameter to set the minimum improvement count for the [TA_01](#), [TA_02](#), [TA_06](#), and [TA_09](#) rules. These rule will ignore those nets that result in improvement count less than this minimum value specified. See the [dftSortTAMsgOnFaultCount](#) rule parameter for details.

Used by	TA_01 , TA_02 , TA_06 , TA_09
Options	any positive integer number
Default value	0
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter dftSkipTAMsgCount 5</code>
<i>Usage in goal/source files</i>	<code>-dftSkipTAMsgCount=5</code>

dftSkipUnObservableNetsForTA

Causes the [TA_01](#) rule to ignore unobservable nets.

Used by	TA_01
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSkipUnObservableNetsForTA on</code>
<i>Usage in goal/source files</i>	<code>-dftSkipUnObservableNetsForTA=on</code>

dftSONames

Defines the naming convention of scan-out ports of scan modules.

Used by	Scan_34
Options	A space-separated list of names
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSONames {so}</code>
<i>Usage in goal/source files</i>	<code>-dftSONames={so}</code>

dftSortTAMsgOnFaultCount

Specifies whether or not the [TA_01](#) and [TA_02](#) rules, will sort rule messages by fault improvement count when the uncontrollable nets (for TA_01) and unobservable nets (for TA_02) are made controllable and observable respectively.

Used by	TA_01 , TA_02
Options are	off, on
Default	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftSortTAMsgOnFaultCount on</code>
<i>Usage in goal/source files</i>	<code>-dftSortTAMsgOnFaultCount=on</code>

dftStrictTestClkDomains

Specifies whether or not the clock domains use the clock edge information. If the `dftStrictTestClkDomains` rule parameter is set to `no`, the different edges of the same test clock are not considered for domain crossing check in a scan chain.

If the `dftStrictTestClkDomains` rule parameter is set to `yes`, the different edges of the same test clock are considered for domain crossing check in a scan chain. A lockup latch is required when a positive edge triggered flip-flop drives a negative edge triggered flip-flop on a scan chain.

Used by	Scan_22
Options	no, yes
Default value	no
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftStrictTestClkDomains yes</code>
<i>Usage in goal/source files</i>	<code>-dftStrictTestClkDomains=yes</code>

dftTargetFaultCoverage

Specifies the target fault coverage for the `Info_coverage` rule.

NOTE: *The `dftTargetFaultCoverage` parameter has been renamed to [dft_target_stuck_at_fault_coverage](#).*

SpyGlass DFT ADV Rule Parameters

Used by	Info_coverage
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTargetFaultCoverage 20.7</code>
<i>Usage in goal/source files</i>	<code>-dftTargetFaultCoverage=20.7</code>

dftTargetTestCoverage

Specifies the target test coverage for the [Info_coverage](#) rule.

NOTE: *The `dftTargetTestCoverage` parameter has been renamed to `dft_target_stuck_at_test_coverage`.*

Used by	Info_coverage
Options	real values between 0 and 100
Default value	none
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTargetTestCoverage 20.7</code>
<i>Usage in goal/source files</i>	<code>-dftTargetTestCoverage=20.7</code>

dftTestclockCycles

Use this parameter to apply clock pulses on the testclock nodes.

Constant propagation through flip-flop may enable testclock line leading to more constant propagation through flip-flops. You can use the `dftTestclockCycles` parameter to put a limit to number of pulse application. However, note that the software may not apply all the pulses if they are not required.

The `dftTestclockCycles` parameter is relevant only when the value of the [dftTMPPropThruFF](#) parameter is set to on.

Used by	All Rules in SpyGlass DFT ADV Product
Options	<any natural number>
Default value	30
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTestclockCycles 10</code>
<i>Usage in goal/source files</i>	<code>-dftTestclockCycles=10</code>

dftTestEnablePinNameForCGC

Defines the test enable pin name of the clock gating cell for the [TA_08](#) rule.

Used by	TA_08
Options	string
Default value	a non empty string with the text 'TE'
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTestEnablePinNameForCGC TE11</code>
<i>Usage in goal/source files</i>	<code>-dftTestEnablePinNameForCGC=TE11</code>

dftTreatControllableLatchTransparentForLoop

Specifies whether or not controllable latches are considered as transparent latches.

By default, the value of this parameter is `off`, and the latches with an active value on the EN pin in the power, ground, or `test_mode` mode are considered as transparent latches.

When the value of this parameter is `on`, latches that do not have an active value on the EN pin, but can be controlled to an active value, are considered as transparent latches. In addition, combinational loops are detected through such latches.

SpyGlass DFT ADV Rule Parameters

Used by	Coverage_audit , Latch_02 , Topology_01
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftTreatControllableLatchTransparentForLoop on
<i>Usage in goal/source files</i>	- dftTreatControllableLatchTransparentForLoop= on

dftTreatFlipFlopsAsScan

Specifies whether or not all flip-flops are treated as scannable for TA_01, TA_02, TA_06, Info_uncontrollable, Info_unobservable, and TA_09 rules.

Used by	TA_01 , TA_02 , TA_06 , Info_uncontrollable , Info_unobservable , TA_09
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftTreatFlipFlopsAsScan on
<i>Usage in goal/source files</i>	-dftTreatFlipFlopsAsScan=on

dftTreatLatchesAsTransparent

Use the `dftTreatLatchesAsTransparent` rule parameter to treat all latches as transparent for rules TA_01, TA_02, TA_06, Info_uncontrollable, Info_unobservable, TA_09.

Used by	TA_01 , TA_02 , TA_06 , Info_uncontrollable , Info_unobservable , TA_09
Options	off, on
Default value	off

Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTreatLatchesAsTransparent on</code>
<i>Usage in goal/source files</i>	<code>-dftTreatLatchesAsTransparent=on</code>

dftUseOffStateOfClockInClockPropagation

Sets clocks, except the clock which is currently being propagated, to the X (don't care) state or their off state during clock propagation.

By default, the `dftUseOffStateOfClockInClockPropagation` parameter is set to `off` and all clocks, except the propagated clock, are set to the X state during clock propagation.

Used by	All rules in the SpyGlass DFT ADV product
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftUseOffStateOfClockInClockPropagation on</code>
<i>Usage in goal/source files</i>	<code>-dftUseOffStateOfClockInClockPropagation=on</code>

dftTreatBBoxAsScanwrapped

Use the `dftTreatBBoxAsScanwrapped` rule parameter to treat all black boxes as scan-wrapped for rules `TA_01`, `TA_02`, `TA_06`, `Info_uncontrollable`, `Info_unobservable`, `TA_09`.

Used by	<i>TA_01</i> , <i>TA_02</i> , <i>TA_06</i> , <i>Info_uncontrollable</i> , <i>Info_unobservable</i> , <i>TA_09</i>
Options	off, on
Default value	off

SpyGlass DFT ADV Rule Parameters

Default Value in GuideWare2.0	on (for <i>dft_abstract</i> , <i>dft_abstract_validate</i> , <i>dft_dsm_clocks</i> , and <i>dft_dsm_clocks_soc</i> goals only in GuideWare2.0)
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTreatBBoxAsScanwrapped on</code>
<i>Usage in goal/source files</i>	<code>-dftTreatBBoxAsScanwrapped=on</code>

dftTMPPropThruFF

Use this parameter to allow non-X values to propagate through the flip-flops even when the explicit test_mode values, that is, 0101... are not applied on the clock line.

In this case, flip-flop will allow the constant value at D-pin to propagate to its output, only if it gets a test clock, and other asynchronous pins are at non-conflicting values. A flip-flop that gets the non-X value at its output through this process is inferred as no scan.

Used by	All Rules in SpyGlass DFT ADV Product
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftTMPPropThruFF on</code>
<i>Usage in goal/source files</i>	<code>-dftTMPPropThruFF=on</code>

dftUseOffStateOfClockInClockPropagation

Specifies the treatment of the off state of the clocks during shift, capture, or atspeed mode.

By default, the `dftUseOffStateOfClockInClockPropagation` parameter is set to `on` to simulate the off state of the test clock, that is, 0 for `rtz` and 1 for `rto`. It also keeps functional clocks to unknown state during the shift mode.

Set this parameter to `off` to keep all clocks, including functional and test

clocks to unknown state. See the table below to view the list of other possible values.

The following table lists the clock state and the corresponding event applied on it during scanshift mode simulation:

Clock State	Event Applied
Unknown	No event is applied
off	0X (rtz) or 1X (rto)
free running	0X1X

An 'X' is always simulated at the end is that clock line is free during clock propagation. You can use this parameter when ICGs are used to drive constant value.

Used by	All rules in the SpyGlass DFT ADV product
Options	off, on, fclk_unknown_n_tclk_unknown, fclk_unknown_n_tclk_off_state, fclk_unknown_n_tclk_free_running, fclk_off_state_n_tclk_off_state, fclk_off_state_n_tclk_free_running, fclk_free_running_n_tclk_off_state, fclk_free_running_n_tclk_free_running
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dftUseOffStateOfClockInClockPropagation off
<i>Usage in goal/source files</i>	-dftUseOffStateOfClockInClockPropagation= fclk_free_running_n_tclk_off_state

dftUUMarking

The `dftUUMarking` parameter has been deprecated. Use the [dftIgnoreConstantOrUnusedFlipFlops](#) parameter instead.

Used by	Info_untestable , Info_unused , Info_unobservable , TA_01 , TA_02 , TA_06 , Async_08 , TA_09
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftUUMarking on</code>
<i>Usage in goal/source files</i>	<code>-dftUUMarking=on</code>

dsm_apply_migrated_constraints

Enables the usage of constraints successfully migrated by the [Latch Rules](#) to migrate the remaining constraints.

The default value for this parameter is on. In this case, the constraints successfully migrated by the `Info_Top_SGDC_Report` rule are used in the same run to migrate the remaining constraints.

When you set the value of the parameter to off, the successfully migrated constraints are not used for migration of remaining constraints.

Used by	Latch Rules
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_apply_migrated_constraints off</code>
<i>Usage in goal/source files</i>	<code>-dsm_apply_migrated_constraints = off</code>

dsm_check_unblocked_mcp

Use this parameter to enable reporting of Multi-Cycle Paths (MCPs) and/or False Paths (FPs), within same and different domains.

By default, the value of the parameter is off. In this case, the `Atspeed_05` rule checks for the presence of both MCPs and FPs.

You can set the value of the parameter to one of the following options:

- **on**: Checks for MCPs, in same as well as different domains
- **same_domain**: Checks for MCPs in the same domain

Used by	Atspeed_05
Options	off, on, same_domain
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_check_unblocked_mcp same_domain</code>
<i>Usage in goal/source files</i>	<code>-dsm_check_unblocked_mcp=same_domain</code>

dsm_count_false_path_transition_faults

The `dsm_count_false_path_transition_faults` parameter is used to include/exclude faults on false paths while computing transition fault and test coverage.

The following list explains the different input values that you can specify for the `dsm_count_false_path_transition_faults` parameter and the respective design impact:

- **off**: This is the default value. In this case, faults on false paths are not considered while computing transition fault and test coverage.
- **on**: The faults on false paths are considered while computing transition fault and test coverage.
- **fault_coverage**: The faults on false paths are considered while computing only transition fault coverage but are not considered while computing transition test coverage.

SpyGlass DFT ADV Rule Parameters

Used by	<i>Latch Rules, Info_transitionCoverage_audit</i>
Options	off, on, fault_coverage
Default value	off
Example	
<i>Console/Tcl-based usage</i>	set_parameter dsm_count_false_path_transition_faults on
<i>Usage in goal/source files</i>	-dsm_count_false_path_transition_faults = on

dsm_gen_hiersgdc

Runs the [Latch Rules](#) rule.

By default, the value of this parameter is set to `off` and the *Info_Top_SGDC_Report* rule is not run. Set the value of the parameter to `on` to run the *Info_Top_SGDC_Report* rule.

Used by	Latch Rules
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_gen_hiersgdc on</code>
<i>Usage in goal/source files</i>	<code>-dsm_gen_hiersgdc=on</code>

dsm_infer_clock_domain_at_clock_shaper_boundary

Specifies whether a separate clock domain is should be inferred at a `clock_shaper` boundary while generating enabled flip-flops report in the *Info_enabledFlops* rule.

The default value of the parameter is `off`. It implies that the clock domain of the input clock propagates to the fanout cone.

Set the value of the parameter to `on` to infer a separate clock domain at the `clock_shaper` output pin through which the `atspeed` clock is passing.

Used by	Info_enabledFlops
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_infer_clock_domain_at_clock_shaper_bound ary on</code>
<i>Usage in goal/source files</i>	<code>- dsm_infer_clock_domain_at_clock_shaper_bound ary=on</code>

dsm_launch_method

Specifies the transition test method as launch on shift or launch on capture.

Used by	Info_transitionCoverage_audit and Latch Rules
Options	LOC, LOS
Default value	LOC
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_launch_method "LOS"</code>
<i>Usage in goal/source files</i>	<code>-dsm_launch_method=LOS</code>

dsm_max_fanin_paths_for_Atsspeed_12

Limits the fan-in tracing of the clock/frequency paths from a given clock source in the [Atsspeed_12](#) rule.

By default, the value of the parameter is 8.

Set the value of the parameter to any natural number to specify the limit for the fan-in tracing of the clock/frequency paths from a given clock-source in the [Atsspeed_12](#) rule.

Used by	Atsspeed_12
Options	A natural number
Default value	8
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_max_fanin_paths_for_Atsspeed_12 10</code>
<i>Usage in goal/source files</i>	<code>-dsm_max_fanin_paths_for_Atsspeed_12=10</code>

dsm_memory_clock_check

Specifies the type of check needed, on memory clock-pin, using the [Atsspeed_23](#) rule.

The following table describes the supported values for the `dsm_memory_clock_check` parameter.

TABLE 4 Supported Parameter Values

Value	Description
<code>atspeed_clock</code>	Specifies that a memory-clock-pin should get an atspeed clock in the atspeed mode. A violation is reported if a memory-clock-pin does not get an atspeed clock.
<code>atspeed_clock_via_pll</code>	Specifies that a memory-clock-pin should get an atspeed clock in the atspeed mode through a pll. No violation is reported if a memory-clock-pin does not receive an atspeed clock. However, a violation is reported if it gets an atspeed clock but not through a pll.
<code>both</code>	Specifies that a violation is reported if either a memory-clock-pin does not get an atspeed clock or it gets an atspeed clock but not through a pll
Used by	Atspeed_23
Options	<code>atspeed_clock</code> , <code>atspeed_clock_via_pll</code> , <code>both</code>
Default value	<code>both</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_memory_clock_check atspeed_clock</code>
<i>Usage in goal/source files</i>	<code>-dsm_memory_clock_check=atspeed_clock</code>

dsm_report_domain_for_faults

Reports launch and capture atspeed clock domains in the [transition_faults](#) report.

By default, the value of the `dsm_report_domain_for_faults` parameter is set to `off`. Therefore, the `Info_transitionCoverage` rule does not report the launch and capture atspeed clock domains in the [transition_faults](#) report.

Set the value of the parameter to `on` to report launch and capture atspeed clock domains in the [transition_faults](#) report.

Used by	Latch Rules
Options	<code>on</code> , <code>off</code>
Default value	<code>off</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_report_domain_for_faults on</code>
<i>Usage in goal/source files</i>	<code>-dsm_report_domain_for_faults=on</code>

dsm_use_cumulative_fault_status

Enables multi-mode analysis of transition coverage. For first run of multi-mode analysis, set the value of the `dsm_use_cumulative_fault_status` parameter to `start`. For subsequent runs, specify the `cumulative_transition_fault_status.rpt` file generated in a previous run, as an input to this parameter.

Used by	Latch Rules
Options	<code>off</code> , <code>start</code> , <code>cumulative_stuck_at_fault_status.rpt</code>
Default value	<code>off</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsm_use_cumulative_fault_status start</code>
<i>Usage in goal/source files</i>	<code>-dsm_use_cumulative_fault_status="cumulative_stuck_at_fault_status.rpt"</code>

dsmAtspeedClockSynchronizationRange

This parameter is used for specifying the maximum acceptable difference in the frequencies of two atspeed clock domains that are considered as candidates for synchronization. The parameter is specified as a percentage.

The default value of the parameter is 0. This indicates that there is no cutoff for frequency difference and all atspeed clock domains may be considered as candidates for synchronization.

Used by	Info_atspeedClockSynchronization
Options	0 to infinite
Default value	0
Example	
<i>Console/Tcl-based usage</i>	set_parameter dsmAtspeedClockSynchronizationRange 20
<i>Usage in goal/source files</i>	-dsmAtspeedClockSynchronizationRange=20

dsmGenerateTransitionFaultReport

The `dsmGenerateTransitionFaultReport` parameter is used to generate pin based fault report for transition faults.

Set the value of the `dsmGenerateTransitionFaultReport` parameter to a value other than none to generate the [transition_faults](#) report.

Used by	Latch Rules
Options	none, all, all_except_detected, detected, undetected, synthesis_redundant, unused, untestable, tied, blocked, logical_redundant, false_path
Default value	none
Example	
<i>Console/Tcl-based usage</i>	set_parameter dsmGenerateTransitionFaultReport all
<i>Usage in goal/source files</i>	-dsmGenerateTransitionFaultReport= all_except_detected

dsmIgnoreControlLineFaults

When the `dsmIgnoreControlLineFaults` parameter is set to `on`, transition faults on control signals, that is, clock, scan enable, set, and reset pins are excluded from Transition fault space.

Used by	Info_transitionCoverage_audit and Latch Rules
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsmIgnoreControlLineFaults on</code>
<i>Usage in goal/source files</i>	<code>-dsmIgnoreControlLineFaults=on</code>

dsmLimitMigrationReportFanin

Provides a limit to the number of partially migrated fan-in report.

Used by	Latch Rules
Options	any integer value
Default value	5
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsmLimitMigrationReportFanin 10</code>
<i>Usage in goal/source files</i>	<code>-dsmLimitMigrationReportFanin=10</code>

dsmReportAllCGCCEFaninFlops

Specifies whether the [CG_06](#) rule reports all flip-flops hit in the fan-in cone of the CGC Enable pins.

The default value of the parameter is set to `on` and the [CG_06](#) rule reports all flip-flops hit in the fan-in cone of the CGC Enable pins.

You can set the value of the parameter to `off` to enable the [CG_06](#) rule to report only those flip-flops whose testclock domain drives multiple

functional domain.

Used by	CG_06
Options	on, off
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter dsmReportAllCGCCEFaninFlops off
<i>Usage in goal/source files</i>	-dsmReportAllCGCCEFaninFlops=off

dsmReportEnabledFlops

Specifies which flip-flops are reported by the [Latch Rules](#) rule.

The default value of the parameter is set to `all` and the `Info_enabledFlops` rule reports all flip-flops.

You can set the value of the parameter to `scannable` to enable the `Info_enabledFlops` rule to report only the scannable flip-flops.

Used by	Latch Rules
Options	<code>all</code> , <code>scannable</code>
Default value	<code>all</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsmReportEnabledFlops scannable</code>
<i>Usage in goal/source files</i>	<code>-dsmReportEnabledFlops=scannable</code>

dsmUsePIForAtSpeed

When the `dsmUsePIForAtSpeed` parameter is set to `on`, primary inputs are used for launching transition fault tests. Also see section [Transition Delay Fault Launch Point](#)

Used by	Info_transitionCoverage_audit and Latch Rules
Options	<code>on</code> , <code>off</code>
Default value	<code>on</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsmUsePIForAtSpeed off</code>
<i>Usage in goal/source files</i>	<code>-dsmUsePIForAtSpeed=off</code>

dsmUsePOForAtSpeed

When the `dsmUsePOForAtSpeed` parameter is set to `on`, primary outputs are used as capturing points for transition fault tests.

Used by	Info_transitionCoverage_audit , Latch Rules
Options are	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dsmUsePOForAtSpeed off</code>
<i>Usage in goal/source files</i>	<code>-dsmUsePOForAtSpeed=off</code>

dftXPropForScanFF

Specifies whether the capture mode simulation is dependent on the scanshift mode simulation or not.

When the value of the `dftXPropForScanFF` parameter is set to on, all scan flip-flops have state X on their output nets in the scanshift mode. This scanshift mode state of circuit is used as initial state for the capture mode simulation.

When the value of the `dftXPropForScanFF` parameter is set to off, then both the scanshift mode simulation and the capture mode simulation are independent of each other.

Used by	Information Rules , Scan Rules , Testability Analysis Rules
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter dftXPropForScanFF off</code>
<i>Usage in goal/source files</i>	<code>-dftXPropForScanFF=off</code>

faultAnnotation

Specifies whether the [Info_untestable](#) rule generates the schematic back-annotation data for faults in addition to the text report (default).

SpyGlass DFT ADV Rule Parameters

When 'on', the schematic back-annotation data is generated. When 'off', (default), the schematic back-annotation data is not generated. However, the `stuck_at_coverage` report is always be generated.

NOTE: *Using the `faultAnnotation` rule parameter increases the memory requirement.*

Used by	Info_untestable
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter faultAnnotation on</code>
<i>Usage in goal/source files</i>	<code>-faultAnnotation=on</code>

flopInFaninCount

Specifies the limit on the number of flip-flops and black boxes allowed in the data pin fan-in cone of any flip-flop as checked by the [BIST_01](#) rule.

A message is reported if the actual number exceeds the value of the `flopInFaninCount` rule parameter.

Used by	BIST_01
Options	<any natural number>
Default value	150
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter flopInFaninCount 20</code>
<i>Usage in goal/source files</i>	<code>-flopInFaninCount=20</code>

gateInputCount

Specifies the limit on the number of inputs of a primitive combinational logic element as checked by the [BIST_02](#) rule.

A message is reported if the actual number exceeds the value of the `gateInputCount` rule parameter.

Used by	BIST_02
Options	<any natural number>
Default value	8
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter gateInputCount 10</code>
<i>Usage in goal/source files</i>	<code>-gateInputCount=10</code>

head_pipeline_clock_phase

Defines the active shift clock for head registers.

SpyGlass DFT ADV Rule Parameters

Used by	TC_02 , TC_03
Options are	<"falling" or "rising">
Default value	"falling"
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter head_pipeline_clock_phase "rising"</code>
<i>Usage in goal/source files</i>	<code>-head_pipeline_clock_phase="rising"</code>

head_pipeline_stages

Defines the sequential depth between a root level scan in port and the decompressor data-in ports.

Used by	TC_01 , TC_02
Options are	Any positive integer.
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter head_pipeline_stages 5</code>
<i>Usage in goal/source files</i>	<code>-head_pipeline_stages=5</code>

ignoreBlackBoxDuringSimulation

When set to `on`, causes the SpyGlass-Logic-Evaluator engine to ignore the effect of black boxes on the design (as if they were not there).

For example, the simulation value of a net driven by an AND gate and one or more black boxes will be determined by the AND gate output alone when this rule parameter is set to `on`. The black boxes will have no effect on the resultant net value. If this rule parameter is set to `off` (the default value), the net value, for this example, will be X.

Used by	All SpyGlass DFT ADV rules that require simulation
Options	<code>off</code> , <code>on</code>
Default value	<code>off</code>
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ignoreBlackBoxDuringSimulation on</code>
<i>Usage in goal/source files</i>	<code>-ignoreBlackBoxDuringSimulation=on</code>

ignoreBlackBoxDuringTestability

When set to `on`, causes the SpyGlass-Testability engine to ignore the effect of black boxes on the design (as if they were not there).

For example, the controllability value of a net driven by an AND gate and one or more black boxes will be determined by the controllability of the AND gate output alone when this rule parameter is set to `on`. The black boxes will have no effect on the resultant net controllability. If this rule parameter is set to `off` (the default value), the net in this example will be uncontrollable regardless of the controllability of the AND gate.

Used by	All SpyGlass DFT ADV rules that require simulation
Options	<code>off</code> , <code>on</code>
Default value	<code>off</code>
Example	

<i>Console/Tcl-based usage</i>	set_parameter ignoreBlackBoxDuringTestability on
<i>Usage in goal/source files</i>	-ignoreBlackBoxDuringTestability=on

incrementalTA

Specifies that the [TA_01](#) and [TA_02](#) rules should perform incremental fault analysis.

When the `incrementalTA` parameter is set to `on`, if the reported net is made controllable (TA_01) or observable (TA_02), then the additional number of faults that become detectable in the fan-out (TA_01) or fan-in (TA_02) are reported. When set to `off`, the additional faults in the fan-out (TA_01) or in the fan-in (TA_02) are not considered detectable.

NOTE: *Please note that only one of controllability or observability is changed at a time; not both together. Also, please be prepared for increased run-times when using the `incrementalTA` rule parameter.*

Used by	TA_01 , TA_02
Options	off, on
Default value	on
Example	
<i>Console/Tcl-based usage</i>	set_parameter incrementalTA off
<i>Usage in goal/source files</i>	-incrementalTA=off

limitFaninPorts

Use this parameter to set the limit on the number of objects (ports, latches, flip-flops, and black box instances) in the fan-in cone of a net.

NOTE: *The `Async_06` rule ignores those flip-flops where the object count in the fan-in cone of the reset pin plus the object count in the fan-in cone of the set pin exceeds the value set by the `limitFaninPorts` rule parameter.*

If the limit specified with the `limitFaninPorts` rule parameter is exceeded, then the net is ignored, that is, simulation value (for all possible

test vectors on the fan-in cone objects) is not evaluated.

Used by	Async_06 , Tristate_06 , Tristate_09
Options	<any natural number>
Default value	12
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter limitFaninPorts 10</code>
<i>Usage in goal/source files</i>	<code>-limitFaninPorts=10</code>

mergeN

Use this parameter to limit the number of bit-slices that should be merged and hence reported together (assuming that they have been inferred at the same line and file).

For example, if signals `q[0:4]` and `q[7:15]` are violating and the `mergeN` rule parameter is set to 2, then the rule message would report that signal `q[0:4, 7:15]` is violating. This compaction would reduce the total message count.

By default, the `mergeN` rule parameter is set to -1 and all bit-slices are merged together and reported.

Used by	Async_01 , Clock_04 , Clock_14 , TA_01 , TA_02
Options	any positive integer number
Default value	-1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter mergeN 2</code>
<i>Usage in goal/source files</i>	<code>-mergeN=2</code>

monitorFlow

NOTE: *The `monitorFlow` parameter is for internal debugging purposes only.*

Enables monitoring of the time and memory usage at critical points of the

SpyGlass DFT ADV Rule Parameters

SpyGlass DFT ADV run. When a performance issue is reported, you are ordinarily asked to send the screen output when the analysis is repeated with the `set_option w <true | false>` and `set_option DEBUG <stage>` commands added. At such times, it is recommended that you also add the `monitorFlow` parameter. Use of the `monitorFlow` parameter provides useful information to the SpyGlass DFT ADV development team over and above that provided by the `set_option w <true | false>` and `set_option DEBUG <stage>` commands.

Used by	All SpyGlass DFT ADV rules
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter monitorFlow on</code>
<i>Usage in goal/source files</i>	<code>-monitorFlow=on</code>

noBlackBoxReporting

Use this parameter to suppress reporting rule messages for nets made uncontrollable or unobservable by black boxes. By default, these rule messages are reported.

Used by	Async_02_shift , Async_02_capture , BIST_01 , RAM_06 , Scan_17 , TA_01 , TA_02 , Topology_05 , TA_09
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter noBlackBoxReporting on</code>
<i>Usage in goal/source files</i>	<code>-noBlackBoxReporting=on</code>

pathDepth

Use this parameter to limit the minimum number of combinational logic levels between two registers.

A message is reported if the actual number exceeds this limit.

Used by	Topology_10
Options	<any natural number>
Default value	500
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter pathDepth 10</code>
<i>Usage in goal/source files</i>	<code>-pathDepth=10</code>

reconvergenceMinDepth

Use this parameter to limit the minimum number of logic levels that a re-convergence must have to report it as a Topology_09 violation.

Re-convergence where all paths have less than this number will not be reported.

Used by	Topology_09
Options	<any natural number>
Default value	0
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter reconvergenceMinDepth 10</code>
<i>Usage in goal/source files</i>	<code>-reconvergenceMinDepth=10</code>

ser_control_sequential_depth

Use this parameter to specify number of sequential time frames for which forward traversal of the design is done when computing probabilities of logic value controllability in the design.

SpyGlass DFT ADV Rule Parameters

Used by	Info_soft_error_propagation
Options	any non-negative integer
Default value	5
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_control_sequential_depth 6</code>
<i>Usage in goal/source files</i>	<code>-ser_control_sequential_depth=6</code>

ser_ignore_safety_mechanism_register_lreg

Use this parameter to ignore lreg values for safety mechanism registers.

By default, the value of the parameter is set to off. In this case, SpyGlass DFT ADV considers lreg values for safety mechanism registers.

Set the value of the parameter to 0 to enable the [Info_soft_error_propagation](#) rule to ignore lreg values for safety mechanism registers.

Used by	Info_soft_error_propagation
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_ignore_safety_mechanism_register_lreg on</code>
<i>Usage in goal/source files</i>	<code>-ser_ignore_safety_mechanism_register_lreg = on</code>

ser_load_enable_n_cycle_persistence_adjustment

Allows the [Info_soft_error_propagation](#) to adjust flip-flop probabilities to capture effect of retained error value due to load enable.

By default, the value of the parameter is off.

Set the value of the parameter to on to adjust flip-flop probabilities to capture effect of retained error value due to load enable.

Used by	Info_soft_error_propagation
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_load_enable_n_cycle_persistence_adjustme nt on</code>
<i>Usage in goal/source files</i>	<code>- ser_load_enable_n_cycle_persistence_adjustme nt = on</code>

ser_observe_sequential_depth

Use this parameter to specify number of sequential time frames for which backward traversal of the design is done when computing the probabilities of logic value observability in the design.

Used by	Info_soft_error_propagation
Options	any non-negative integer
Default value	5
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_observe_sequential_depth 6</code>
<i>Usage in goal/source files</i>	<code>-ser_observe_sequential_depth = 6</code>

ser_observe_nsr_initial_value

Use this parameter to specify the initial observability value assigned to non-safety-related registers before backward traversal of the design is done to compute observability probabilities.

Used by	Info_soft_error_propagation
Options	any value between 0 and 1 (including 0 and 1)
Default value	0.5

Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_observe_nsr_initial_value 0.5</code>
<i>Usage in goal/source files</i>	<code>-ser_observe_nsr_initial_value = 0.5</code>

ser_blackbox_observe_probability

Use this parameter to specify observability probability value for blackbox/memory inputs.

Used by	Info_soft_error_propagation
Options	any real value between 0 and 1
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_blackbox_observe_probability 0.8</code>
<i>Usage in goal/source files</i>	<code>-ser_blackbox_observe_probability = 0.8</code>

ser_blackbox_control_0_probability

Use this parameter to specify the control-to-0 probability value for blackbox/memory outputs. The control-to-1 probability is calculated by subtracting control-to-0 probability from 1.

Used by	Info_soft_error_propagation
Options	any real value between 0 and 1
Default value	0.5
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter ser_blackbox_control_0_probability 0.8</code>
<i>Usage in goal/source files</i>	<code>-ser_blackbox_control_0_probability = 0.8</code>

ser_propagation_difference_threshold

Use this parameter to specify the minimum change in controllability or observability value evaluations for which traversal continues. Changes below that value are not propagated to save on computing runtime.

Used by	Info_soft_error_propagation
Options	any real value between 0 and 100
Default value	5
Example	
<i>Console/Tcl-based usage</i>	set_parameter ser_propagation_difference_threshold 5
<i>Usage in goal/source files</i>	-ser_propagation_difference_threshold = 5

ser_register_report_top_contributors

Use this parameter to specify the percentage of top contributor registers listed in the soft_error_registers_spfm.rpt report.

Used by	Info_soft_error_propagation
Options	any real value between 0 and 100
Default value	15
Example	
<i>Console/Tcl-based usage</i>	set_parameter ser_register_report_top_contributors 15
<i>Usage in goal/source files</i>	-ser_register_report_top_contributors = 15

schematicForAllBits

Use this parameter to highlight information for all bits of the rule violating bus.

When the `schematicForAllBits` rule parameter is set to `off`, the highlight information for only one representative bit of the rule-violating bus is generated.

SpyGlass DFT ADV Rule Parameters

Used by	Async_01 , Clock_04 , Clock_14 , TA_01 , TA_02
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter schematicForAllBits on</code>
<i>Usage in goal/source files</i>	<code>-schematicForAllBits=on</code>

sequentialDepth

Use this parameter to limit the number of non-scannable flip-flops allowed in a path from a primary input, a black box, or a scannable flip-flop and another black box, scannable flip-flop, or a primary output as reported by the [Scan_17](#) rule. A message is reported if the actual number exceeds the specified limit.

Used by	Scan_17
Options	<any natural number>
Default value	5
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter sequentialDepth 7</code>
<i>Goal/Source file usage (In goal/source files)</i>	<code>-sequentialDepth=7</code>

showAllReconvergence

When this parameter is off (default value), only one re-convergence per path is reported. When set to on, all re-convergence are reported. This may increase the run time.

Used by	Topology_09
Options	off, on
Default value	off
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter showAllReconvergence on</code>
<i>Usage in goal/source files</i>	<code>-showAllReconvergence=on</code>

showPath

Use this parameter to control the schematic display for the Bist_01 and Scan_17 rules.

When the value of the parameter is set to on, the *BIST_01* and *Scan_17* rules highlight the complete path.

By default, only the end points are highlighted.

Used by	<i>BIST_01, Scan_17</i>
Options	on, off
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter showPath on</code>
<i>Usage in goal/source files</i>	<code>-showPath=on</code>

showPowerGroundValue

Use this parameter to control the schematic display for power/ground simulation.

When the value of the parameter is set to on, the simulation value of a net due to power/ground is displayed. To hide the power/ground simulation values of a net, set this switch to 'off'.

Used by	<i>Info_testmode, Info_synthRedundant, Diagnose_testmode</i>
Options	on, off
Default value	on
Example	

<i>Console/Tcl-based usage</i>	<code>set_parameter showPowerGroundValue off</code>
<i>Usage in goal/source files</i>	<code>-showPowerGroundValue=off</code>

tail_pipeline_clock_phase

Defines the active shift clock for tail registers.

Used by	TC_05
Options are	<"falling" or "rising">
Default value	"falling"
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter tail_pipeline_clock_phase "rising"</code>
<i>Usage in goal/source files</i>	<code>-tail_pipeline_clock_phase="rising"</code>

tail_pipeline_stages

Defines the sequential depth between a compressor data-out pin and root level scan out port.

Used by	TC_04
Options are	Any positive integer
Default value	1
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter tail_pipeline_stages 5</code>
<i>Usage in goal/source files</i>	<code>-tail_pipeline_stages=5</code>

useFirstSource

When this parameter is on, the clock source of a flip-flop clock pin is

defined as the driver of a flip-flop clock pin. Else, clock source is defined as the first multi-input device reached by traversing back through buffers and inverters from a flip-flop clock pin.

Used by	All SpyGlass DFT ADV rules that require simulation
Options	off, on
Default value	off
Example	
<i>Console/Tcl-based usage</i>	<code>set_parameter useFirstSource on</code>
<i>Usage in goal/source files</i>	<code>-useFirstSource=on</code>

Debugging Violations in SpyGlass DFT ADV Product

The SpyGlass DFT ADV informational rules are designed to assist in determining why various SpyGlass DFT ADV rules are issuing violation messages. In most cases, double-clicking a message with the structural view open will display problem areas.

Diagnosing a Problem

SpyGlass DFT ADV problems can result from either a design issue or a constraint issue.

Diagnosing Design Issues

A design issue may be caused by not yet adding SpyGlass DFT ADV logic or improperly adding SpyGlass DFT ADV logic. In both cases the informational rules can help. Suppose, for example, that [Clock_11](#) rule identifies an internal clock that is not bypassed.

Double-clicking the message will display the logic generating the internal clock. Double-clicking an [Info_testmode](#) message will back annotate any test mode (scanshift mode or capture mode) and the selected testclock. If no test signals are visible, then zoom out to display the pins on the module containing the original message. If the ports on this module are highlighted with test mode (shift mode or capture mode) and testclock, then the problem is contained within this module. The problem is to design this module so that the original derived clock is bypassed by the testclock.

If the port display does not show testmode (scanshift mode or capture mode) and testclock signals, then traverse up the hierarchy to find the lowest level, above the original level, that does contain these signals. The intervening modules must be redesigned to pass these signals to the original module.

Diagnosing Constraint Issues

SpyGlass DFT ADV rule messages may also be caused because the test constraints listed in the SpyGlass Design Constraints file do not match the SpyGlass DFT ADV logic designed into the circuit.

First check the log file to verify that the SpyGlass Design Constraints file was read and that all signal names were found in the design.

If they were found, then follow the procedure described in [Diagnosing Design Issues](#) to determine where the test signals are distributed within the design and how to route them to the failing module.

Understanding Back Annotation

The Back Annotation (BA) feature of the SpyGlass DFT ADV product provides data relevant to the debugging of a violation. This data is annotated in the form of labels. The back annotation data is generated by the [Info_DftDebugData](#) rule.

The following table lists the back annotation labels for various categories used in the SpyGlass DFT ADV product.

TABLE 1 SpyGlass DFT ADV BA Labels

Category	Label	Label Description
Flip-flop scannability	FF_SCN_F	Forced scan (using force_scan SGDC command).
	FF_SCN_I	Inferred scan (Scannable: No Clock_11 / Async_07 violation)
	FF_NSCN_F	Forced no-scan (using force_no_scan SGDC command)
	FF_USCN_CP	Unscannable because test-clock is not reaching
	FF_USCN_RST	Unscannable because async pin is not inactive
	FF_USCN_CP_RST	Unscannable because test-clock is not reaching and async pin is not inactive
	FF_NSCN_I_TCLK	ff_noscan_inferred_test_clock
	FF_NSCN_I_TM	ff_noscan_inferred_test_mode
	FF_NSCN_I_TM_0X	ff_noscan_inferred_test_mode_0X
	FF_NSCN_I_TM_1X	ff_noscan_inferred_test_mode_1X
	FF_NSCN_I_SC	ff_noscan_inferred_outside_scan_chain

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
Flip-flop participation in atspeed	FF_AS_I	Inferred atspeedable as atspeed clock is reaching
	FF_NAS_F	Forced no-atspeed (using <i>no_atspeed</i> SGDC command)
	FF_UAS_CP	Unatspeedable because atspeed clock is not reaching
	FF_PNSCN_I	Participating no_scan flip-flop may be part of a scan chain
Latch transparency in capture	LD_T_F	Transparent as enable pin has non-X simulation value in capture
	LD_T_CNT	Transparent as enable pin is controllable to its active value
	LD_T_^off	Transparent when test clock if 'off'
	LD_LKUP	Lock up latch
	LD_NTL	Non transparent and non lock up latch
Latch transparency in atspeed	LD_AS_T_F	Transparent as enable pin has non-X simulation value in atspeed
	LD_AS_T_CNT	Transparent as latch is getting free running atspeed clock
	LD_AS_T_^off	Transparent when atspeed clock if 'off'
	LD_AS_LKUP	Lock up latch
	LD_AS_NTL	Non transparent and non lock up latch

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
Latch Scannability	LD_SCN_I	Latch specified as inferred
	LD_NSCN_F	Latch specified as no scan (using <i>force_no_scan</i> SGDC command)
	LD_SCN_F	Latch specified as scan (using <i>force_scan</i> SGDC command)
	LD_USCN_EN	Unscannable because enable pin did not get a test clock
	LD_USCN_RST	Unscannable because async pin is not inactive during shift
	LD_USCN_EN_RST	Unscannable because of both enable and async pin
Black box properties	BB_CGC	It's a clock gating cell (<i>gating_cell</i> SGDC command)
	BB_SHAPER	It's a clock shaper (<i>clock_shaper</i> SGDC command)
	BB_MEMORY	It's a memory (<i>memory_type</i> SGDC command)
	BB_SCANWRAP	It's a scanwrapped cell (<i>scan_wrap</i> SGDC command)
	BB_MBYPASS_on	Bypass is present and active (<i>module_bypass</i> SGDC command)
	BB_MBYPASS_off	Bypass is present but inactive (<i>module_bypass</i> SGDC command)
Simulation Value	PG_0	0 when circuit is in power-ground mode
	PG_1	1 when circuit is in power-ground mode

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
	F__0	0 when circuit is in functional mode. When the node has an SGDC specified, the BA label changes to F__0(F). Here, (F) signifies forced value coming through SGDC
	F__1	1 when circuit is in functional mode. When the node has an SGDC specified, the BA label changes to F__1(F). Here, (F) signifies forced value coming through SGDC
	F__X(MD)	X when circuit is in the functional mode and has multiple drivers.
	SH_0	0 when circuit is in scanshift mode. When the node has an SGDC specified, the BA label changes to SH_0(F). Here, (F) signifies forced value coming through SGDC
	SH_1	1 when circuit is in scanshift mode. When the node has an SGDC specified, the BA label changes to SH_1(F). Here, (F) signifies forced value coming through SGDC
	SH_1(F: CGC)	Signifies that software has forced the internal net to value 1 because of the CGC behavior
	SH_X(MD)	X when circuit is in the shift mode and has multiple drivers.
	C__0	0 when circuit is in capture mode. When the node has an SGDC specified, the BA label changes to C__0(F). Here, (F) signifies forced value coming through SGDC

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
	C__1	1 when circuit is in capture mode. When the node has an SGDC specified, the BA label changes to C__1(F). Here, (F) signifies forced value coming through SGDC
	C__X(MD)	X when circuit is in the control mode and has multiple drivers.
	AS_0	0 when circuit is in atspeed mode. When the node has an SGDC specified, the BA label changes to AS__0(F). Here, (F) signifies forced value coming through SGDC
	AS_1	1 when circuit is in atspeed mode. When the node has an SGDC specified, the BA label changes to AS__1(F). Here, (F) signifies forced value coming through SGDC
	AS_X(MD)	X when circuit is in the atspeed mode and has multiple drivers.
	DT<i>_0	0 when a define_tag having the logical name, DT<i>, is simulated. Please note that DT<i> is a unique logical name and is not user specified.
	DT<i>_1	1 when a define_tag having the logical name, DT<i>, is simulated. Please note that DT<i> is a unique logical name and is not user specified.
	F<i>_0	0 when a frequency having the logical name, F<i>, is simulated. Please note that F<i> is a unique logical name and is not user specified.

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
	F<i>_1	1 when a frequency having the logical name, F<i>, is simulated. Please note that F<i> is a unique logical name and is not user specified.
	F<i>_^	Clock reaching when frequency having logical name, F<i>, is simulated. Please note that F<i> is a unique logical name and is not user specified.
	F<i>_v	Clock inverted when frequency having logical name, F<i>, is simulated. Please note that F<i> is a unique logical name and is not user specified.
Stuck_At Zero Fault	sa0_UU	Unused fault
	sa0_SR	Synthesis Redundant fault
	sa0_TI	Tied fault
	sa0_BL	Blocked fault
	sa0_UT	Untestable fault
	sa0_LR	Logically redundant fault
	sa0_ND	Not-Detectable fault
	sa0_DT	Detectable fault
Stuck_At One Fault	sa1_UU	Unused fault
	sa1_SR	Synthesis Redundant fault
	sa1_TI	Tied fault
	sa1_BL	Blocked fault

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
	sa1_UT	Untestable fault
	sa1_LR	Logically redundant fault
	sa1_ND	Not-Detectable fault
	sa1_DT	Detectable fault
Slow-to-Rise Fault	t01_UU	Unused fault
	t01_SR	Synthesis Redundant fault
	t01_TI	Tied fault
	t01_BL	Blocked fault
	t01_UT	Untestable fault
	t01_ND	Not-Detectable fault
	t01_DT	Detectable fault
Slow-to-Fall Fault	t10_UU	Unused fault
	t10_SR	Synthesis Redundant fault
	t10_TI	Tied fault
	t10_BL	Blocked fault
	t10_UT	Untestable fault
	t10_ND	Not-Detectable fault
	t10_DT	Detectable fault
Clock Values	F__^	Rising edge of functional clock in functional mode
	F__v	Falling edge of functional clock in functional mode

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
	SH_^	Rising edge of test clock in scanshift mode. When the node has an SGDC specified, the BA label changes to SH_^ (F). Here, (F) signifies forced value coming through SGDC
	SH_v	Falling edge of test clock in scanshift mode. When the node has an SGDC specified, the BA label changes to SH_v(F). Here, (F) signifies forced value coming through SGDC
	C__^	Rising edge of test clock in capture mode. When the node has an SGDC specified, the BA label changes to C__^ (F). Here, (F) signifies forced value coming through SGDC
	C__v	Falling edge of test clock in capture mode. When the node has an SGDC specified, the BA label changes to C__v(F). Here, (F) signifies forced value coming through SGDC
	AS_^	Rising edge of atspeed clock in atspeed mode. When the node has an SGDC specified, the BA label changes to AS_^ (F). Here, (F) signifies forced value coming through SGDC
	AS_v	Falling edge of atspeed clock in atspeed mode. When the node has an SGDC specified, the BA label changes to AS_v(F). Here, (F) signifies forced value coming through SGDC

TABLE 1 SpyGlass DFT ADV BA Labels (Continued)

Category	Label	Label Description
Blocked Path Values	SH_bp	blocked in scanshift mode
	C__bp	blocked in capture mode
	BP_SHIFT	blocked path value / status in scanshift mode
	BP_CAPTURE	Blocked path value / status in capture mode
Observability value (in capture)	OBS_y	Observable
	OBS_n	Not Observable
Controllability value (in capture)	nnn	Fully uncontrollable
	nny	Only controllable to Z
	nyn	Only controllable to 1
	nyy	Only controllable to 1 and Z
	y nn	Only controllable to 0
	y ny	Only controllable to 0 and Z
	y yn	Only controllable to 1 and 0
	y yy	Controllable to all (0, 1, and Z)
Generic Text Annotation	xpt	Expected value
	s/0	stuck_at 0 fault
	s/1	stuck_at 1 fault
	t/01	slow-to-rise (0 ' 1) transition fault
	t/10	slow-to-fall (1 ' 0) transition fault
	cnt_1	Controllability to value 1
	cnt_0	Controllability to value 0

Viewing Results in Fault Browser

The Fault Browser capability allows you to view the results generated by the [Info_coverage](#) rule in a tabular format in Atrenta Console.

To invoke the Fault Browser window, double-click on the rule message in the Msg Tree tab of the Message window.

The following figure shows the Fault Browser window:

The screenshot shows the Fault Browser window titled "FaultBrowser.gen.prj". The window has a menu bar (File, View, Help) and a toolbar with a search field and a filter dropdown set to "Based on color" with a value of "0 - 50 %". A "Configure columns" button is also visible. The main area displays a table with the following columns: Instance Hierarchy, Module Name, Fault Coverage (%), Test Coverage (%), Fault Coverage Loss (%), and Test Coverage Loss (%). The table lists various modules and their corresponding coverage percentages.

Instance Hierarchy	Module Name	Fault Coverage (%)	Test Coverage (%)	Fault Coverage Loss (%)	Test Coverage Loss (%)
ipc_ip	ipc_ip	82.910	82.930	17.090	17.070
ram64x32_inst	hip7ipsdsp006...	0.330	0.330	15.000	15.010
ipc_HOSTC...	ipc_HOSTCtrl	89.660	89.660	1.080	1.080
ipc_schedul...	ipc_scheduler	92.580	92.580	0.220	0.220
mdm	mdm	94.800	94.800	0.190	0.190
ipc_once_inst	ipc_once	96.440	96.440	0.130	0.130
ipc_ram64x...	ipc_ram64x32_...	1.170	1.170	0.120	0.120
ipc_bist_en...	ipc_bist_engine	96.820	96.820	0.100	0.100
ipc_clk_ctrl...	ipc_clk_ctrl	57.660	57.660	0.090	0.090
ipc_sysbus...	ipc_sysbus	99.330	99.330	0.040	0.040
ipc_inst	ipc	99.780	99.780	0.040	0.040
dmb	dmb	99.890	99.890	0.030	0.030
ipc_DSPCtrl...	ipc_DSPCtrl	99.710	99.710	0.010	0.010
ipc_ram102...	ipc_ram1024x...	98.790	98.790	0.000	0.000
ipc_crc_inst	ipc_crc	99.990	99.990	0.000	0.000
ipc_dspdm...	ipc_dspdma	99.990	99.990	0.000	0.000
ipc_rom256...	ipc_rom256x32...	100.000	100.000	0.000	0.000
ipc_fubusif...	ipc_fubusif	100.000	100.000	0.000	0.000

FIGURE 1. Fault Browser Window

Rule-Wise Display Information

This section lists the columns displayed in the Fault Browser window by the *Info_coverage*, *Info_transitionCoverage*, and *Info_random_resistance* rules:

Info_coverage

The Fault Browser window displays the following columns for the

Viewing Results in Fault Browser

Info_coverage rule:

Instance Hierarchy	Module Name	Fault Coverage (%)	Test Coverage (%)
Fault Coverage Loss (%)	Test Coverage Loss (%)	Total Fault Number	FC
!DT	DT	SR	UU
TI	BL	LR	UT

Info_transitionCoverage

The Fault Browser window displays the following columns for the Info_transitionCoverage rule:

Instance Hierarchy	Module Name	Fault Coverage (%)	Test Coverage (%)
Fault Coverage Loss (%)	Test Coverage Loss (%)	Total Fault Number	FC
!DT	DT	SR	UU
TI	BL	LR	UT
FP			

Info_random_resistance

The Fault Browser window displays the following columns for the Info_random_resistance rule:

Instance Hierarchy	Module Name	Fault Coverage (%)	Test Coverage (%)
RRF Coverage Loss (/%)	Total Fault Number		

NOTE: *Fault Browser displays the Is Abstract column to display abstracted blocks in the design, only when at least one abstract block is present in the design. Abstract block is shown as @blk in Is Abstract column and for others it is left empty.*

You can perform the following tasks using Fault Browser:

- [Viewing Undetectable Faults](#)
- [Organizing Report Columns](#)

- [Configuring Legend](#)
- [Searching an Instance](#)
- [Setting Search Preferences](#)

Viewing Undetectable Faults

You can view the undetectable faults for the following using Fault Browser:

- [Immediate Instances Inside the Selected Hierarchy](#) (Probe This Instance)
- [All the Instances of the Selected Hierarchy](#) (Probe Full Hierarchy)

NOTE: The **Probe This Instance** and **Probe Full Hierarchy** options are available only when you run a goal from the GUI. These options are unavailable, when you run a goal in batch or through `sg_shell` and then load the GUI only for debug. This is because, when GUI is re-loaded, all the policy data is lost and tcl-based call-back (infrastructure used for **This Instance** and **Probe Full Hierarchy**) does not work

Immediate Instances Inside the Selected Hierarchy

To view the undetectable faults, which are immediately under the selected hierarchy, perform one of the following steps:

- Double-click on any hierarchy in the fault browser ([Figure 2](#))
- Right-click on any hierarchy and click the **Probe This Instance** option

The **Probe This Instance** option shows undetectable faults present locally in the selected instance. It does not show the undetectable faults present in the sub-hierarchies.

Then, open the Incremental Schematic to view the undetectable faults. If there are no undetectable faults, the Incremental Schematic window appears blank.

[Figure 2](#) shows the reported instances in the Fault Browser and the available right-click menu:

Viewing Results in Fault Browser

Instance Hierarchy	Module Name	Fault Coverage (%)	Test Coverage
[-] top	top	20.000	
[-] s12	sub12	25.000	
[+] s41	sub4	27.500	
[+] s31	sub3		
[+] s51	sub5		
[+] s11	sub11		
s2	sub2		

FIGURE 2. Fault Browser Right-Click Menu

Performing any of the tasks listed above displays the undetectable faults, which are immediately under the selected hierarchy as shown in [Figure 3](#):

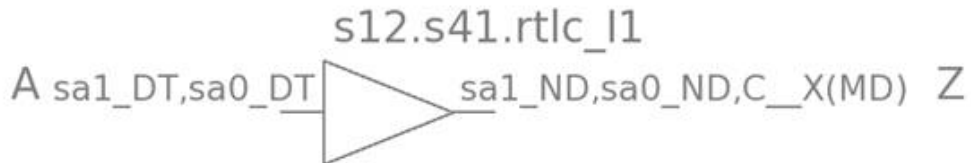


FIGURE 3. Incremental Schematic for top instances

All the Instances of the Selected Hierarchy

Right-click and select the **Probe full hierarchy** option (Figure). The Incremental Schematic window shows all the undetectable faults of the selected hierarchy, as shown in [Figure 4](#):

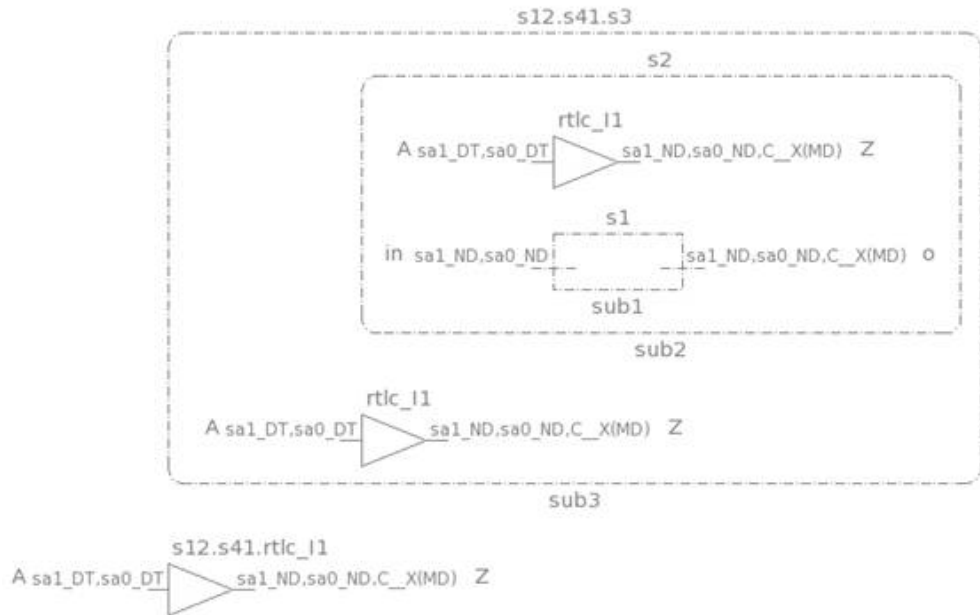


FIGURE 4. Incremental Schematic for all the instances

The **Probe full hierarchy** option shows undetectable faults present locally in the selected instance and in the sub-hierarchies.

Organizing Report Columns

You can organize the report columns in one of the following ways:

- [Sort Columns](#)
- [Show or Hide Columns](#)

Sort Columns

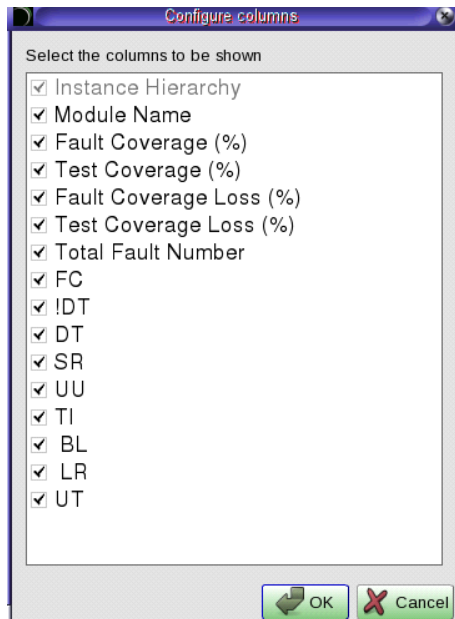
To sort a column in an ascending or descending order, click a column header.

Show or Hide Columns

To show or hide columns, perform the following steps from the Fault Browser window:

1. Select *Fault Report > Configure Columns*. Alternatively, select the *Configure Columns* option from the Right-click menu.

The *Configure Columns* window is displayed.

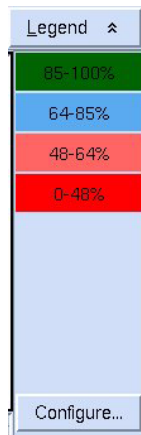


2. Select or deselect the columns you want to show or hide.
3. Click *OK*.

The Fault Report displays the information for the selected columns.

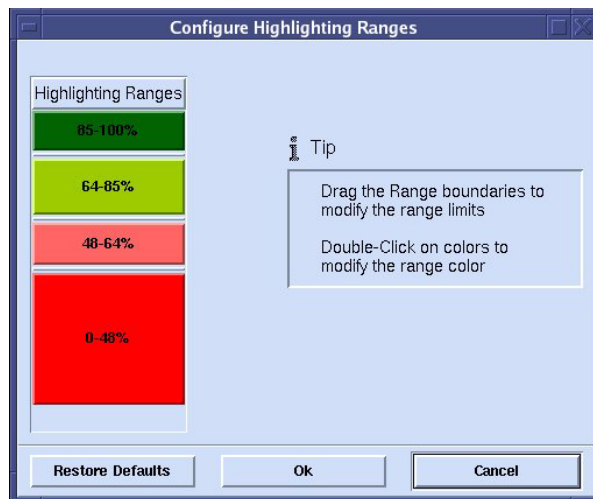
Configuring Legend

The Fault Browser window highlights the fault values by Color Range. The fault values are highlighted in different colors based on test coverage. The categories and their highlight colors are available from the Legend button.



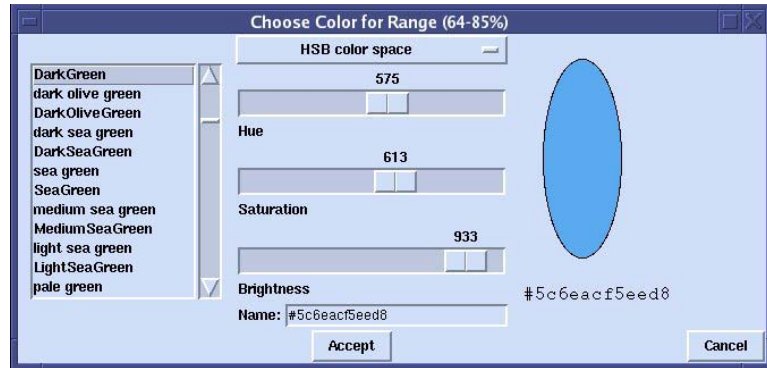
To configure the legend, perform the following steps from the Fault Browser window:

1. Click *Configure* on the right-bottom of the *Fault Browser* window.
The *Configuring Highlighting Ranges* window is displayed.



2. Double-click on the highlight range button.
The *Choose Color for Range* window is displayed.

Viewing Results in Fault Browser



3. Select the color and the related characteristics.
4. Click *Accept*.

The selected color coding is applied.

5. Drag the separator above the highlight range button up or down to increase or decrease the range to change a highlight range.

As you change a particular highlight range, the adjacent highlight range is automatically adjusted.

Searching an Instance

You can search an instance by name in the Instance Hierarchy column or a design unit by name in the Module Name column using the Find controls:



To search an instance's name in the *Find* field, select *Instance Hierarchy* in the in field, and click *Find Next* or *Find Prev* buttons to search.

To search a design unit name, enter the first few characters of the design unit's name in the *Find* field, select *Module Name* in the in field, and click *Find Next* or *Find Prev* buttons to search.

You can also search by fault value category. Select *Based on Fault* in the *Find* field, select the category in the *Range* field, and click *Find Next* or *Find Prev* buttons to search.

Setting Search Preferences

To set your search preferences, perform the following steps from the Fault Browser window:

1. Click *Fault Report > Setup*.

The *Tree Options* window is displayed.



2. Set the Find History size (number of Find strings saved), Maximum entries in a Page, and whether the Find feature searches in a case-sensitive manner and/or using regular expressions.
3. Click *OK*.

The search results are displayed based on the options selected in the *Tree Options* window.

Scanning a Clock Tree

Scanning a clock tree enables you to visualize the clock propagation and find out the signals, which are blocking a clock. You can also view the list of gating signals and the related information about the clock domain interactions.

Ensure that the value of the [dft_show_scan_clock_tree](#) parameter is on to scan a clock-tree in the shift or capture mode using the [Info_testclock](#) rule.

The rule generates violation messages, of severity INFO, for the clock propagation. Double-click on the violation message to view:

- [Clock Browser window](#)
- [Scan Clock Tree Report](#)

Clock Browser window

The *Clock Browser* window displays the tree view of a clock in the shift or capture mode. It displays the following information:

- List of clock domains interacting with this clock with count of interactions
- RxList (Receiver List), lists clock domains driven by the selected clock
- TxList (Transmitter List), lists clock domains driving this clock
- Gating signals, if any, impacting clock propagation at each branching point

A branch point can be any multi-input combinational cell, which is connected to a clock pin.

- Flip-flop, latch, black box, hard macro, or memory as leaf nodes with clock polarity.

The blocked nodes are displayed as leaf nodes and are marked with the keyword, BLKD.

[Figure 5](#) illustrates the Scan Clock Tree for `top.clk1` in the *Clock Browser* window:

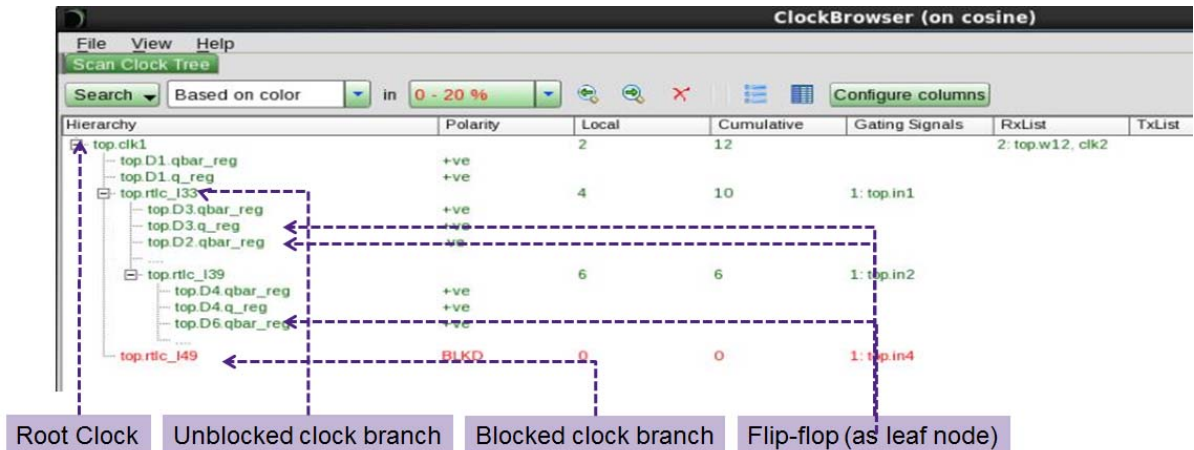


FIGURE 5. Clock Browser

As shown in [Figure 5](#), blocked clock branch is highlighted in **red**. The *Clock Browser* window also displays the root clock, unblocked clock branch and the propagated flip-flop.

Double-click on a node, except the root node, in the **Hierarchy** to display it in Incremental Schematic, as shown in [Figure 6](#):

Scanning a Clock Tree

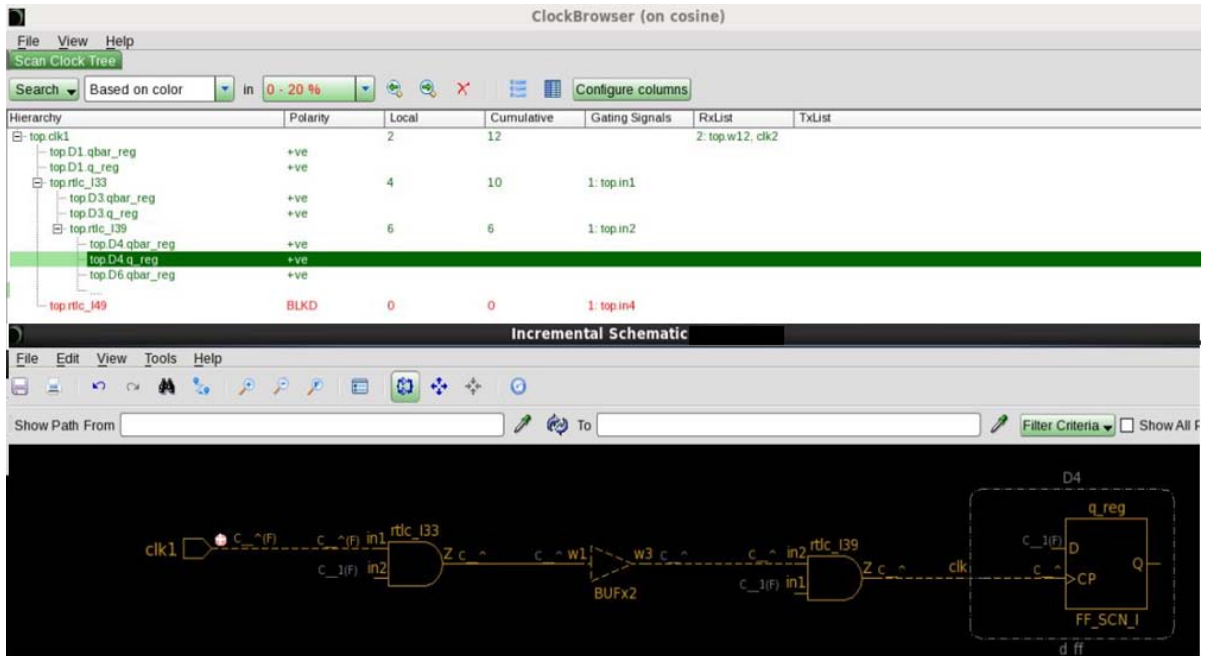


FIGURE 6. Incremental Schematic for the Selected Node

In [Figure 6](#), the Incremental Schematic window displays the path from the root-clock to the selected node, leaf, or branch.

Any subsequent click on other node, adds the selected nodes in the schematic, without refreshing it.

Scan Clock Tree Report

The Clock Tree report is generated by the [Info_testclock](#) rule. This report displays the following information:

- List of clock domains interacting with this clock with count of interactions
- RxList (Receiver List), lists clock domains driven by the selected clock
- TxList (Transmitter List), lists clock domains driving this clock
- Gating signals, if any, impacting clock propagation at each branching point

A branch point can be any multi-input combinational cell, which is connected to a clock pin.

- Flip-flop, latch, black box, hard macro, or memory as leaf nodes with clock polarity.

The blocked nodes are displayed as leaf nodes and are marked with the keyword, BLKD.

NOTE: *Latches are reported only if the [dft_scannable_latches](#) parameter is on.*

For more information on the Clock Tree report, see [Scan Clock Tree Report](#).

Performing an On-Demand Trace

The on-Demand Trace enables you to view the clocks, uncontrollable sources, or unobservable sources with a single-click.

To perform an on-demand trace, right-click on a signal in the Incremental Schematic and select one of the following options from the right-click menu:

- *Trace to Clock in Shift*
- *Trace to Clock in Capture*
- *Trace to Clock in Atspeed*
- *Trace to Uncontrollable in Atspeed (Before Launch)*
- *Trace to Uncontrollable in Atspeed (After Launch)*
- *Trace to Uncontrollable Sources in Capture*
- *Trace to Unobservable Sources in Capture*

NOTE: *The On-Demand Trace options are available in the GUI only when you have run the goal in the GUI. The options are unavailable if you have loaded a batch run in the GUI for debugging.*

Trace to Clock in Shift

When you select the **Trace to Clock in Shift** option, GUI traces the clock path to all the test clocks, which are present in the fan-in cone, in the Scan Shift mode. It allows you to get to all the root clocks along with the blockage points, if present.

Paths are color coded to distinguish the clock blockage points, if present.

If a test clock is not found in the topological fanin cone, then there is no annotation on the incremental schematic (IS).

The following violation message is displayed on the Shell Interface:

```
DFT_CLK_SRC: No topological clock is found in the fan-cone of
node <node hierarchical name>
```

Figure 7 illustrates the usage of the Trace to Clock in Shift option:

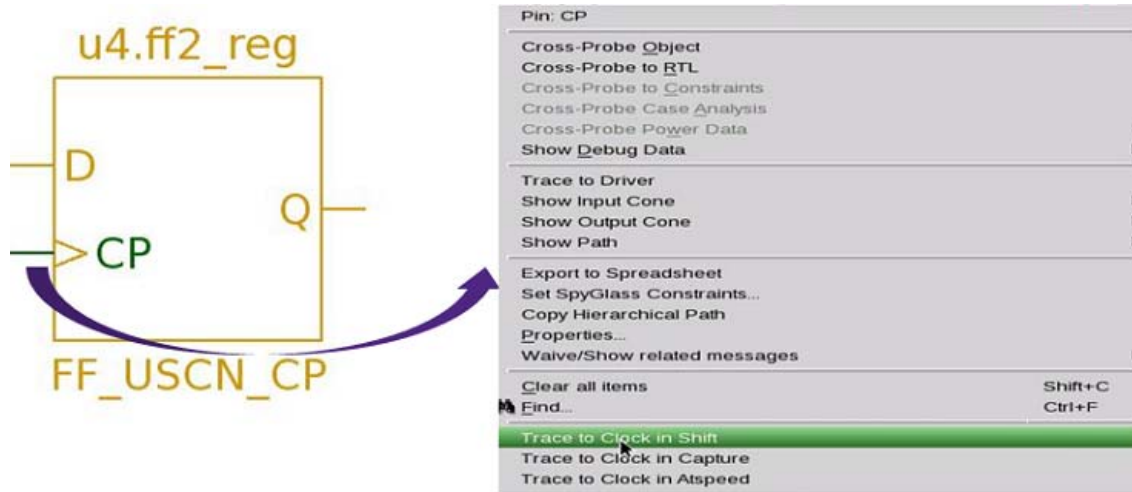


FIGURE 7. Selecting the Trace to Clock in Shift Option

When you select the **Trace to Clock in Shift** option, the Incremental Schematic is refreshed to display the following schematic (Figure 8):

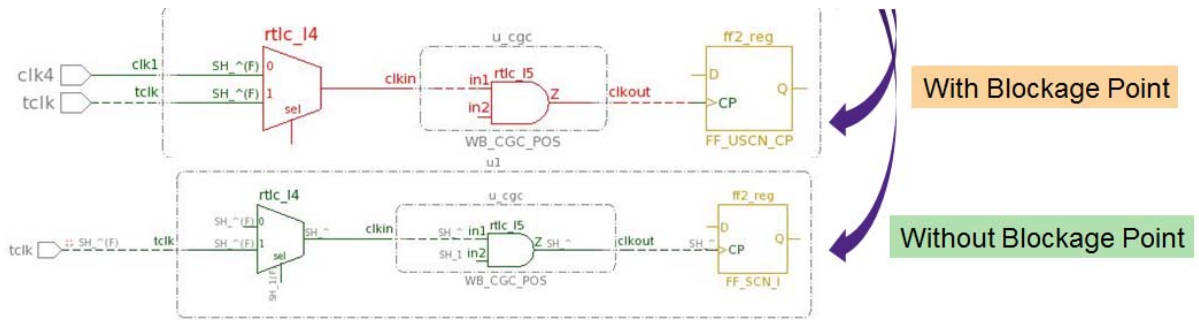


FIGURE 8. Updated Incremental Schematic

Trace to Clock in Capture

This option is same as **Trace to Clock in Shift** option. However, this option works in the *Scan Capture* mode.

For details, see [Trace to Clock in Shift](#).

Trace to Clock in Atspeed

This option is same as **Trace to Clock in Shift** option. However, this option works in the Atspeed mode.

For details, see [Trace to Clock in Shift](#).

Trace to Uncontrollable in Atspeed (Before Launch)

When you select the **Trace Uncontrollable in Atspeed (Before Launch)** option, GUI traces the root cause of uncontrollability, which are present in the fan-in cone, in the Launch Atspeed mode. It allows you to get to all the sources of the bad control.

Paths are color coded to distinguish the type of controllabilities.

If the selected node is fully controllable, then there is no annotation on the incremental schematic (IS).

The following violation message is displayed on the Shell Interface:

```
DFT_UNCNT_SRC: Node '<node hierarchical name>' is controllable to both 0 and 1
```

The following example ([Figure 9](#)) illustrates the use of the **Trace to Uncontrollable Sources in Atspeed (Before Launch)** option:

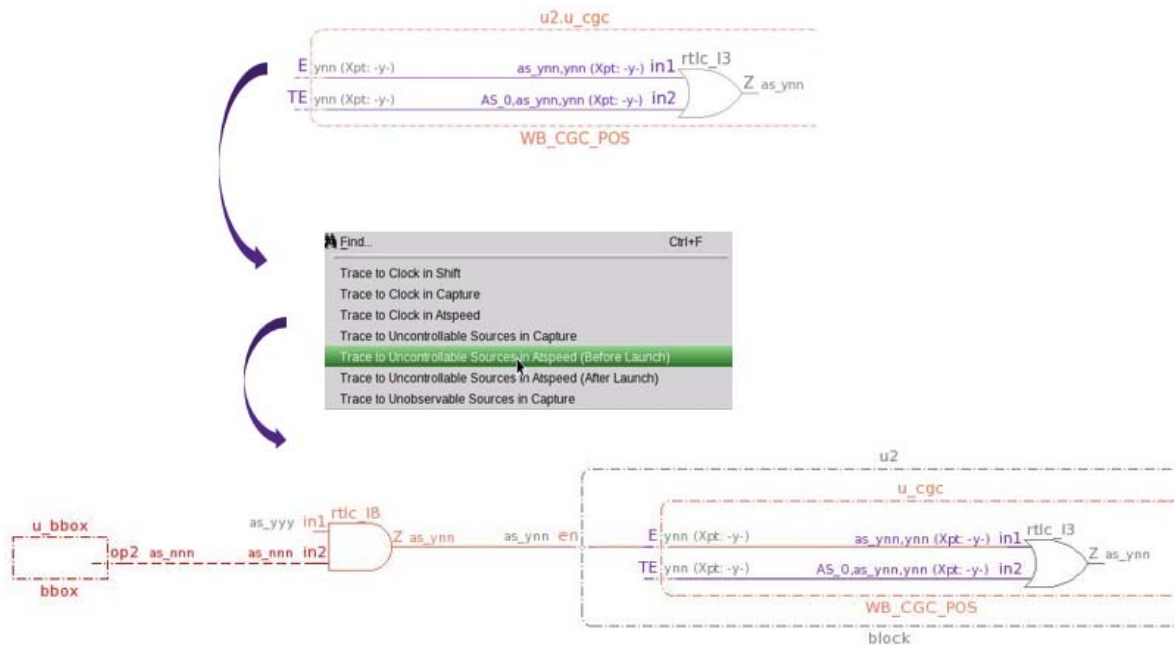


FIGURE 9. Trace to Uncontrollable Sources in Atspeed (Before Launch)

With a single click, you can view all the sources of bad control in form of flip-flop D-pins. However, on the second click (on the D-pin), you can view all the sources of its bad controllability as shown in [Figure 10](#):

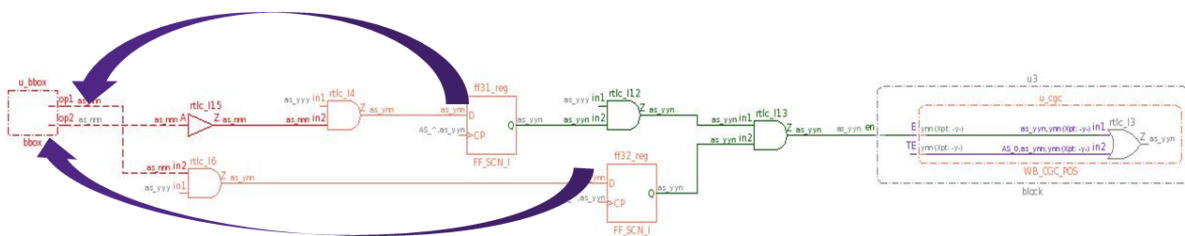


FIGURE 10. Double-clicking on D-pin

Trace to Uncontrollable in Atspeed (After Launch)

This option is same as **Trace to Uncontrollable in Atspeed (Before Launch)** option.

However, in case of after-launch, scan flip-flops may also be a valid source if its D-pin has bad control because it impacts the after-launch controllabilities.

For details, see [Trace to Uncontrollable in Atspeed \(Before Launch\)](#).

Trace to Uncontrollable Sources in Capture

This option is same as **Trace to Uncontrollable in Atspeed (Before Launch)** option. However, this option works in the *Scan Capture* mode.

For details, see [Trace to Uncontrollable in Atspeed \(Before Launch\)](#).

Trace to Unobservable Sources in Capture

When you select the **Trace to Unobservable Sources in Capture** option, GUI traces the path to root cause of unobservability, which is present in the fanout cone, in the *Scan Capture* mode. It allows you to view all the sources of the bad observe.

Paths are color coded to distinguish the type of controllabilities.

If the selected node is observable, then there is no annotation on the incremental schematic (IS).

The following violation message is displayed on the Shell Interface:

```
DFT_UNOBS_SRC: Node '<node hierarchical name>' is observable
```

The following example ([Figure 11](#)) illustrates the use of the Trace to Unobservable Sources in Capture option:

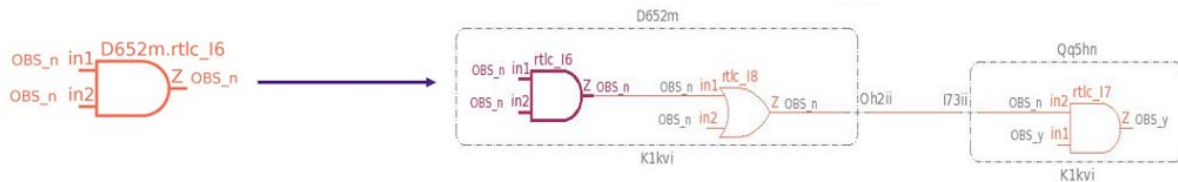


FIGURE 11. Trace to Unobservable Sources in Capture option

With a single click, you can view all the sources of bad observe. However, on the second click, you can view all the sources of its bad observability, which have blocked the signals, as shown in [Figure 12](#):

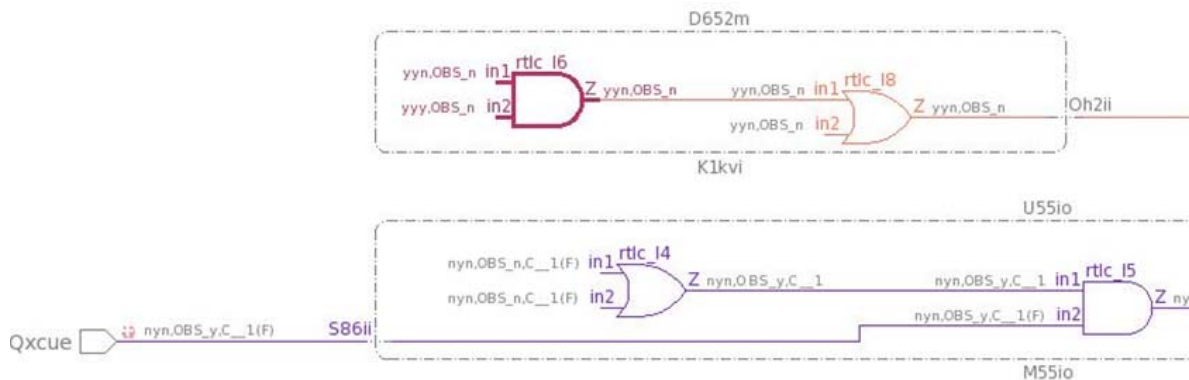


FIGURE 12. Further Traversing to the Source of Bad Observability

Viewing Flat Data Hierarchically

Some of the rules of the SpyGlass DFT ADV product report large number of nodes, usually more than 20,000 in count. For example, the [Info_untestable](#) rule shows all the untestable faults in the design. On a multi-million gate design, this number may be very high. The Incremental Schematic for such messages show huge number of nodes, which makes it difficult to debug or navigate. The hierarchical viewing of such data makes debugging and navigation easier.

To enable hierarchical viewing of a flat data, click the **Hierarchy** button on the **Incremental Schematic** tool bar as shown in below:

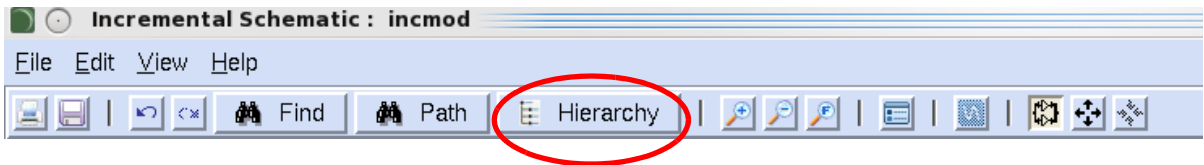


FIGURE 13. Enabling Hierarchical View

This section covers the following topics related to hierarchical viewing of flat data:

- [Understanding the structure of the hierarchy tree](#)
- [Configuring instance count threshold](#)
- [Toggling the instance view](#)

Understanding the structure of the hierarchy tree

Enabling the Hierarchy view displays the design data in the form of a hierarchical tree, which displays various data related to the design instances:

The following figure displays the hierarchical view of data in the Incremental Schematic window:

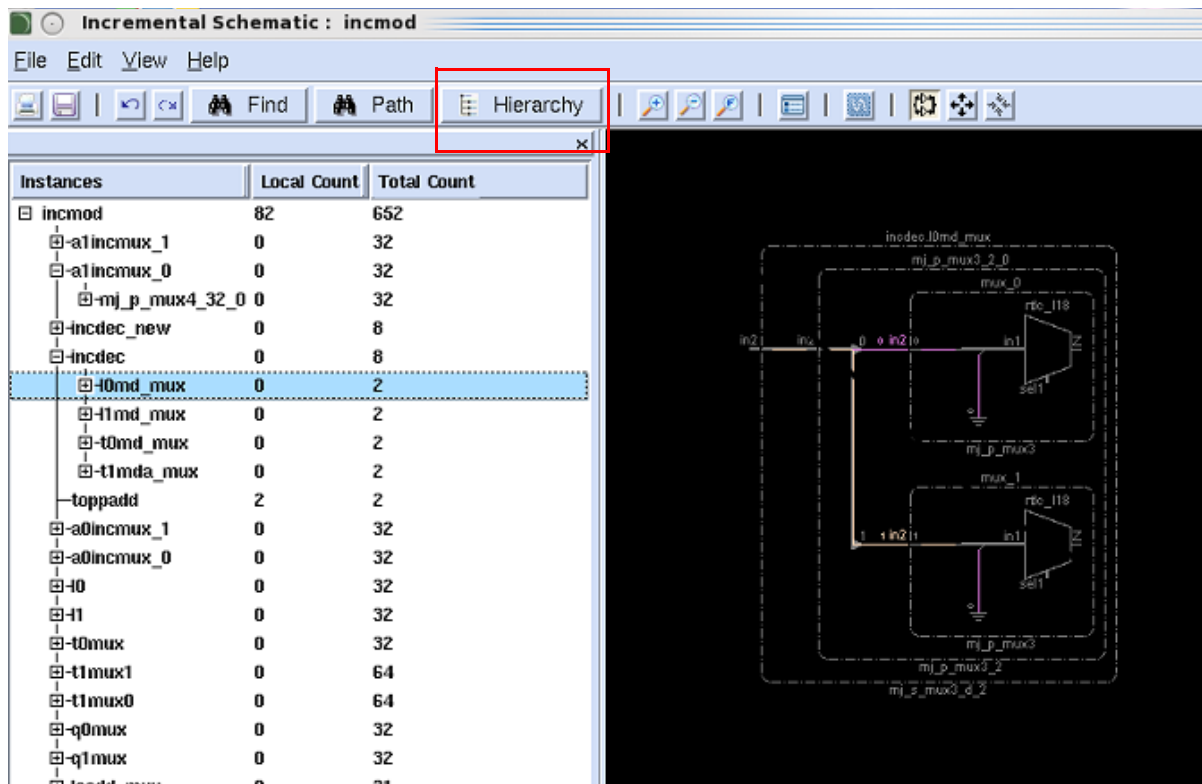


FIGURE 14. Hierarchical View of Flat Data

The hierarchy tree displays the following information:

- **Instances:** All the instances for the selected design.
- **Local Count:** Denotes the count of the highlighted nodes (instances) for that specific hierarchy.

Viewing Flat Data Hierarchically

- **Total Count:** Denotes the count of the highlighted nodes (instances) for the complete sub-hierarchy.

Configuring instance count threshold

By default, the hierarchical view is enabled when instance count is greater than 100. To change this value, perform the following steps from Atrenta Console:

1. Click **Tools -> Preferences**
The Preferences Window is displayed.
2. Click the **Schematic** option in the left window pane.
3. Go to the **Advanced** section
4. Specify any integer value greater than 0 in the **Hierarchical view enabled for instance count greater than:** field as shown below:

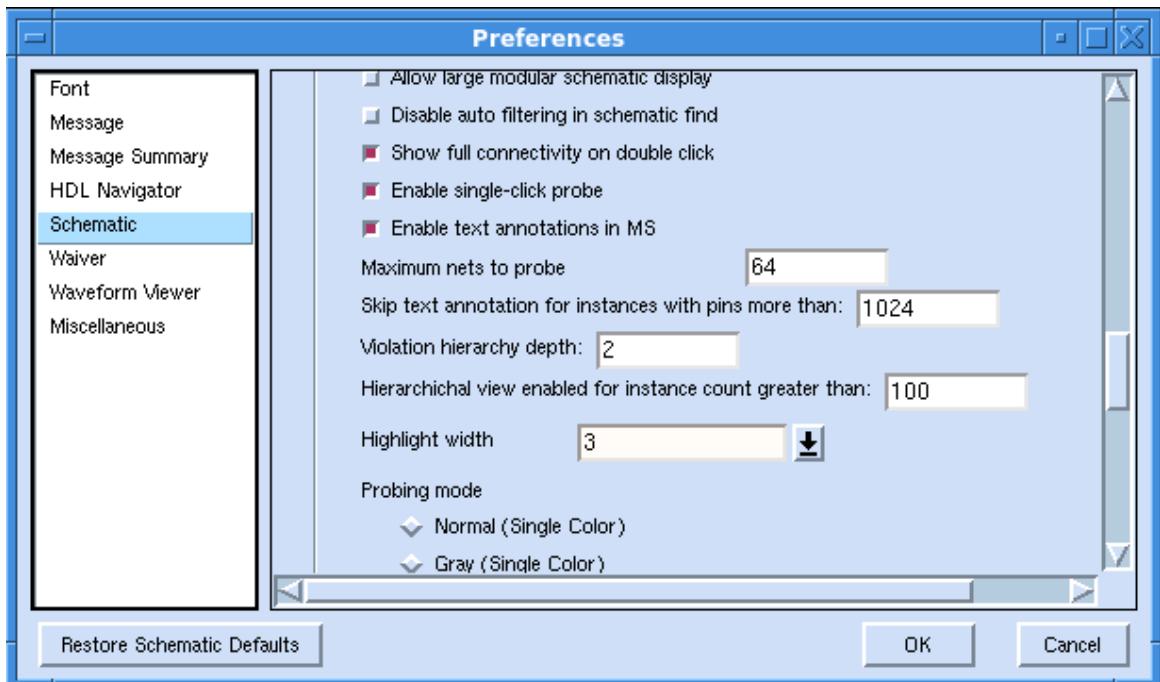


FIGURE 15. Configuring the threshold

5. Click **Ok**.

The data is now displayed hierarchically in the Incremental Schematic window when the instance count reaches the specified value.

Toggling the instance view

You can toggle between viewing the selected hierarchy or displaying the complete sub-hierarchy, by using the **Double-Click Behavior** button. This button is displayed at the bottom of the hierarchy view pane as shown below:

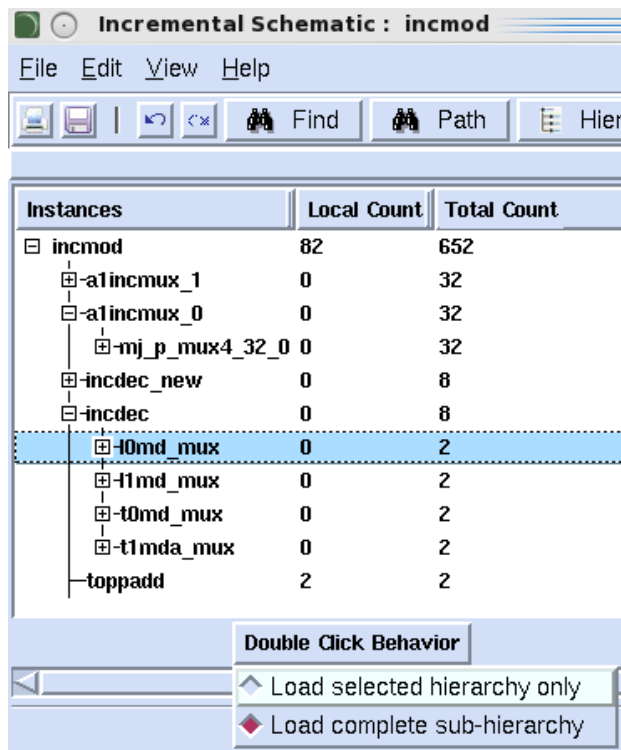


FIGURE 16. Toggle View

Viewing Flat Data Hierarchically

Clicking the **Double-Click Behavior** button displays the following options:

- *Load selected hierarchy only*
- *Load complete sub-hierarchy*

You can select any one of the option to toggle the hierarchy view.

Load selected hierarchy only

The Load selected hierarchy only option displays the local instances only, as shown below:

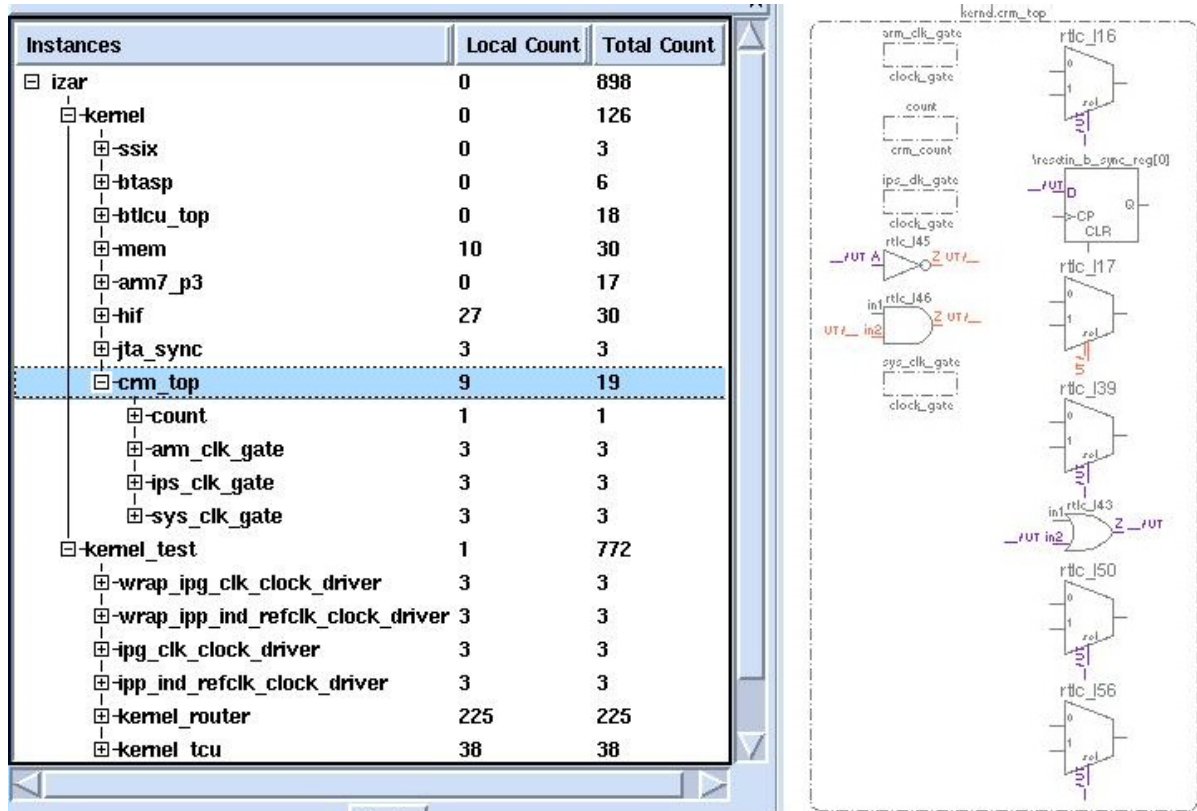


FIGURE 17. Load selected hierarchy

Load complete sub-hierarchy

Displays all the instances in the sub-hierarchy as shown in the figure:

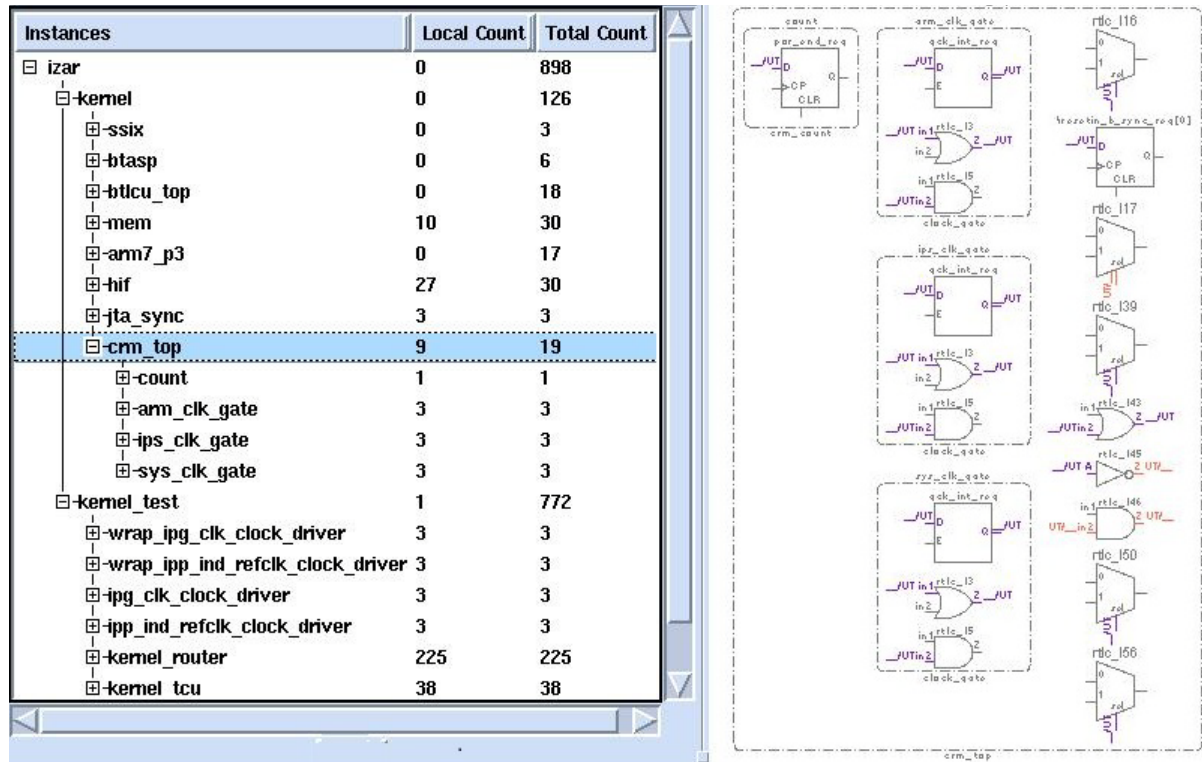


FIGURE 18. Load complete sub-hierarchy

When you double-click on any node in the hierarchy, the design specific to that node is loaded in the right pane of the Incremental Schematic window.

Unblocked Path

An unblocked path from the output Y of a device to an input IN on that device exists if the following conditions are satisfied:

Viewing Flat Data Hierarchically

- The other inputs must not have values that force or control Y,
- The other inputs must have values such that changes on IN cause changes on Y.

A path from point A to point B is unblocked if there is a topological path from A to B and all devices along that path satisfy the unblocked definition above.

SpyGlass DFT ADV determines blocking and unblocking by means of simulation of power, ground and any available test mode conditions. As a result, rule behavior automatically adapts to test mode data.

SpyGlass IEEE1500 Wrapper Debug Flow

Introduction

The IEEE1500 Wrapper debug flow is designed to debug the IEEE1500 Wrapper in an SoC design. This section describes the [Related TCL Commands](#) used to set and run the IEEE1500 Wrapper debug flow. It also provides an [Example Flow](#), which helps you understand the IEEE1500 Flow.

NOTE: *The IEEE1500 Wrapper Debug Flow uses SpyGlass TclShellInterface. Refer to SpyGlass TclShell Interface User Guide for information on sg_shell commands.*

Related TCL Commands

The IEEE1500 Wrapper debug flow requires information on IEEE1500 pins and instructions for activating and running the IEEE1500 wrapper embedded in the design. This section describes the Tcl commands related to IEEE1500 pins and wrapper.

dsm_assign_ieee1500_pins

Identifies IEEE1500 wrapper pins.

Description

This command allows you to identify the standard IEEE1500 Wrapper pins in your design. This command is mandatory as all the IEEE1500 instructions are passed through pins specified using this command.

Syntax

```
dsm_assign_ieee1500_pins
[ -capturewr <capturewr_pin_name> ]
  -selectwir <selectwir_pin_name>
  -shiftwr <shiftwr_pin_name>
  -updatewr <updatewr_pin_name>
  -wrck <wrck_pin_name>
  -wrst <wrst_pin_name>
  -wsi <wsi_pin_name>
  -wso <wso_pin_name>
```

Arguments

This command has following arguments:

<capturewr_pin_name>

(Optional) Specifies the capture pin on the IEEE1500 Wrapper.

<selectwir_pin_name>

Specifies the select pin on the IEEE1500 Wrapper

<shiftwr_pin_name>

Specifies the shift pin on the IEEE1500 Wrapper

<updatewr_pin_name>

Specifies the update pin on the IEEE1500 Wrapper

Introduction

<wrck_pin_name>

Specifies the clock pin on the IEEE1500 Wrapper

<wrst_pin_name>

Specifies the reset pin on the IEEE1500 Wrapper

<wsi_pin_name>

Specifies the instruction pin on the IEEE1500 Wrapper

<wso_pin_name>

Specifies the output pin on the IEEE1500 Wrapper

Example

```
dsm_assign_ieee1500_pins
-wsi wsi
-wso wso
-wrck wrck
-selectwir selectwir
-capturewr capturewr
-shiftwr shiftwr
-updatewr updatewr
-wrst wrst
```

Scope

None

Return Value

None

dsm_reset_ieee1500

Initialize and reset the wrapper.

Description

This command allows you to initialize the IEEE1500 wrapper and also resets the wrapper at any stage of debug.

When an argument is supplied, the reset value, specified through the `-reset_value` option, is applied for the specified number of cycles and then the inverted value of the `-reset_value` option is applied for one cycle. This means that the reset sequence is applied for specified number of cycles on the `wrst` pin specified using `dsm_assign_ieee1500_pins` command. The default number of cycles is one.

Syntax

```
dsm_reset_ieee1500
  -clock_cycles <clock_cycles>
  -reset_value <reset_value>
```

Arguments

This command has following arguments:

<clock_cycles>

Specifies the number of clock cycles for which the reset pin stays active.

<reset_value>

Specifies the reset value.

For example, 1 for active high reset logic and 0 for active low reset logic.

Example

```
dsm_reset_ieee1500
  -clock_cycles 5
  -reset_value 1
```

Result

For the above example, this command adds six clock cycles in the IEEE1500 sequence as shown below:

Clock pin: 010101010101

Reset pin: 111111111100

Scope

None

Return Value

This command returns 0 in case of successful execution and 1 in case of unsuccessful execution.

dsm_load_ieee1500_instruction

Load the instruction in the instruction register.

Description

This command allows you to load the instructions in the instruction register using the wsi pin of the `dsm_assign_ieee1500_pins` command. Instruction is loaded with RHS bit shift-in first.

Syntax

```
dsm_load_ieee1500_instruction <bit sequence>
```

Arguments

This command has following arguments:

<bit_sequence>

You can specify either binary or hexadecimal value.

Example

Assume following as the previous sequence and the number of clock cycles as two:

```
Wrck pin:      0101
Wrst pin:      1100
Wsi pin:       0000
Selectwir pin: 0000
Updatewr pin:  0000
Shiftwr pin:   0000
Capturewr pin: 0000
```

Also, assume the following command is executed:

```
dsm_load_ieee1500_instruction 11001
```

Result

Following is the new sequence when you run the command.

```
Wrck pin:      0101   010101010101
```

Introduction

```
Wrst pin:      1100   000000000000
Wsi pin:       0000   110000111100
Selectwir pin: 0000   111111111111
Updatewr pin:  0000   000000000011
Shiftwr pin:   0000   111111111100
Capturewr pin: 0000   000000000000
```

Here, the number of clock cycles is eight.

Scope

None

Return Value

None

dsm_load_ieee1500_data

Loads the data in the data register.

Description

This command allows you to load the data into the data register using the wsi pin of the dsm_assign_ieee1500_pins command. Instruction is loaded with RHS bit shift-in first.

Syntax

```
dsm_load_ieee1500_data <bit sequence>
```

Arguments

This command has following arguments:

<bit_sequence>

It is the bit sequence to be shifted and can be binary or hexadecimal.

Example

Assume following is the previous sequence the number of clock cycles is two:

```
Wrck pin:      0101
Reset pin:     1100
Wsi pin:       0000
Selectwir pin: 0000
Updatewr pin: 0000
Shiftwr pin:   0000
Capturewr pin: 0000
```

Also, assume that the following command is executed:

```
dsm_load_ieee1500_data 1100
```

Result

Following is the new sequence when your run the command.

```
Wrck pin:      0101      0101010101
```

Introduction

```
Reset pin:      1100      0000000000
Wsi pin:        0000      0000111100
Selectwir pin: 0000      0000000000
Updatewr pin:  0000      0000000011
Shiftwr pin:   0000      1111111100
Capturewr pin: 0000      0000000000
```

Also, the number of clock cycles is seven.

Scope

None

Return Value

None

dsm_assert_value

Defines a check that requires a logic value to be established.

Description

This command is used to check specific value on user specified design node under specific simulation condition. The simulation condition is the state of the circuit, using ieee1500 sequence, when the dsm_assert_value command is issued.

This command also supports expansion of hierarchy and port macro. This eliminates the requirement to run the Soc_06 rule to expand hierarchy and/or port marco.

Syntax

```
dsm_assert_value
  -name <node_list>
  -value <value_list>
  [-match_n_bits <size>]
  [-allow_inversion]
  [-use_shift]
  [-use_capture]
  [-use_capture_at_speed]
  [-waveform]
```

Arguments

This command has following arguments:

<node_list>

This argument can assume one of the following values:

- Top-module port name
- Internal net name
- Terminal name

You can specify more than one pin name for this argument.

<value_list>

This is a logic value string of 0, 1, X or Z. A single-bit value signifies a check at the end of complete simulation. If the value of the argument is X, it signifies do-not-compare.

-allow_inversion

This is an **optional** argument. It indicates that the inverted simulated pattern is also considered as a valid pattern.

<size>

This is an **optional** argument. It specifies the number of least significant bits to be considered. If *<size>* is greater than *<value>* (specified with -value argument), the latter is padded with X to match the former's width.

-use_shift | -use_capture | -use_capture_at_speed

These are **optional** arguments. For any of these modifiers, the *dsm_assert_value* command simulates testmode of that particular mode.

NOTE: *If more than one of the -tag, -use_shift, -use_capture, or -use_capture_at_speed arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with the dsm_assert_value command.*

-waveform

This is an **optional** argument. Generates waveform for better debugging in the GUI mode.

Example**Example 1**

```
dsm_assert_value -name top.a.b -value 1010  
-allow_inversion -match_n_bits 2
```

Example 2

Consider the following Tcl command:

```
dsm_assert_value -name "my_and::INPUT_PORTS" -value 1
```

In the above example, the *dsm_assert_value* command will check for the value, 1, on the following expanded nodes:

```
"top.u_and.in2"  
  "top.u_and.in1"  
    "top.u_block_2.u_and.in2"  
    "top.u_block_2.u_and.in1"  
    "top.u_block_1.u_and.in2"  
    "top.u_block_1.u_and.in1"
```

Example 3

Consider the following Tcl command:

```
dsm_assert_value -name "my_or::ALL_PORTS" -value 1
```

In the above example, the `dsm_assert_value` command will check for the value, 1, on the following expanded nodes:

```
"top.u_or.in2"  
"top.u_or.in1"  
"top.u_or.op"  
"top.u_block_2.u_or.in2"  
"top.u_block_2.u_or.in1"  
"top.u_block_2.u_or.op"  
"top.u_block_1.u_or.in2"  
"top.u_block_1.u_or.in1"  
"top.u_block_1.u_or.op"
```

NOTE: Use double colon (::) with the `-name` argument to generate port or macro expansion.

Example 4

Consider the following Tcl command:

```
dsm_assert_value -name "block::in1" -value 1
```

In the above example, the `dsm_assert_value` command will check for the value, 1, on the following expanded nodes:

```
"top.u_block_2.in1"  
"top.u_block_1.in1"
```

NOTE: Use single colon (.) with the macros, that is, with the `-type` argument to generate port or macro expansion.

Result

Stores the top.a.b node in memory for check when the *dsm_check_assertion* command is issued.

Scope

None

Return Value

This command returns violation messages, with the Error and Info severities, in the following cases:

- If a node does not have the required value: The following violation message, with the Error severity, is reported in this case:

```
Error : dsm_assert_value returns "Node '<node_name>' has  
value <current_sim_value> (Required: <required_sim_value>)
```

- If a node has the required value: The following violation message, with the Info severity, is reported in this case:

```
Info   : dsm_assert_value returns "Node '<node_name>' has  
required value (<required_sim_value>)
```

dsm_assert_path

Defines the connectivity check for a path.

Description

This command defines a connectivity check for a path from a pin specified with the `-from` argument to a pin specified with the `-to` argument under specific simulation condition. The simulation condition is the state of circuit, using `ieee1500` sequence, when `dsm_assert_path` command is issued.

The `dsm_assert_path` command also supports expansion of hierarchy and port macro. This eliminates the requirement to run the `Soc_06` rule to expand hierarchy and/or port marco.

The `dsm_assert_path` command supports wildcard expressions. The supported meta-characters are `*` (star) and `?` (question mark), where `*` matches any number of characters and `?` matches only one character. The wild card support is applicable for non-escaped names only. If the meta-characters appear inside an escaped name, it is treated as a literal. For example, consider the following expression:

```
"top.\mid*\bottom"
```

In the above example, `mid*` is considered as a literal and does expand to `mid1`, `mid2`, and so on. In addition, if you specify a hierarchical path using a wild card, any sub-portion of this path that contains the wild card does not cross the module boundary while searching for the expression in the design. This means that each level in the hierarchy path should be mentioned explicitly in the wild card string. For example, consider the following expression:

```
"top.mid*.bottom"
```

The above expression expands to `"top.mid1.bottom"`. It does not expand to `"top.mid2.bottom"`.

Also, always enclose the expression on which the wildcard is used. For example, `"top.mid*.bottom"`.

The wild card support is applicable for design objects only. It is not applicable for non-design objects.

NOTE: *Buffers and inverters are single-input-single-output devices. Therefore, there is no interference from external logic while tracing a net connectivity.*

Syntax

```
dsm_assert_path
  -from <from_list>
  -to <to_list>
  [-path_type <type>]
  [-invert]
  [-no_invert]
  [-parallel]
  [-undirected]
  [-use_shift]
  [-use_capture]
  [-use_capture_at_speed]
  [-waveform]
```

Arguments

This command has following arguments:

<from_list> | <to_list>

Specifies the start-point and end-point nodes, respectively, in the circuit for which a path is searched after the circuit is simulated by LE into the desired state (with the `-tag` argument specified) or a net connection is checked (without the `-tag` argument specified). These arguments can assume one of the following values: list of top-module port names, internal net names, or terminal names.

<type>

This is an **optional** argument. Accepts only the following predefined list of values: `buffered`, `sensitized`, and `sensitizable`. The default value of this qualifier is `sensitizable`.

-invert | -no_invert

These are **optional** argument. it checks for a positive polarity or negative polarity path between the two nodes. If neither of these is specified, polarity is not relevant for such paths.

-parallel

This is an **optional** argument. Specifies to search the paths from a node in the `<frompinlist>` list to a node at the same relative position in the `<topinlist>` list. This means that path is searched from the first node in the `<frompinlist>` list to the first node in the `<topinlist>` list, and so on.

When you specify the `-parallel` argument, ensure that the number of nodes is both the `<frompinlist>` and the `<topinlist>` list is the same.

-undirected

This is an **optional** argument. The undirected qualifier checks in both the directions, that is, from `-from` node to `-to` node and then from `-to` node to `-from` node.

-use_shift | -use_capture | -use_capture_at_speed

These are **optional** arguments. For any of these modifiers, the `dsm_assert_path` command simulates testmode of that particular mode.

NOTE: *If more than one of the `-tag`, `-use_shift`, `-use_capture`, or `-use_capture_at_speed` arguments is specified, an error condition occurs. Specify only one or none of these modifiers with the `require_path` constraint.*

-waveform

This is an **optional** argument. This option generates a waveform for better debugging in GUI mode.

Example

```
dsm_assert_path -from top.a[2:0] top.b  
-to top.c[3:0] -parallel
```

Result

None

Scope

None

Introduction

Return Value

Connectivity status between specified pairs of node

dsm_assign_clock_pulse

Enables or disables clock pulse for specified clock pin.

Description

This command allows you to enable or disable clock pulse for specified clock pin. Clock pattern is generated on the pin from the point when that clock pin is enabled to the point when it is disabled. If the clock pin is not disabled then the clock pulse is generated till the end of IEEE1500 sequence.

Syntax

```
dsm_assign_clock_pulse -pin <pin_name> <-enable | -disable>
```

Arguments

This command has following arguments:

<pin_name>

Clock pin for which you want to enable or disable the clock pulse.

Example

```
dsm_assign_clock_pulse -pin clk -enable  
dsm_assign_clock_pulse -pin clk -disable
```

Result

As in the above example, -enable command is specified when the IEEE1500 sequence is at two clock cycles . Also, the -disable command is specified whether the number of clock cycles is seven.

Therefore, the sequence is as shown:

```
Wrck pin: 0101 0101010101 010101  
clk pin: 0000 0101010101 000000
```

Also, the number of clock cycles is ten.

Introduction

Scope

None

Return Value

None

dsm_assign_pin_value

Specifies a value for a pin.

Description

This command allows you to set a value on a pin. This value is applied to the pin for subsequent clock cycles. When the number of clock cycles exceeds the bit sequence specified, then the last bit value of the sequence is repeated for the following clock cycles.

Syntax

```
dsm_assign_pin_value
  -pin <pin_name>
  -value <bit_sequence>
```

Arguments

This command has following arguments:

<pin_name>

Name of the pin on which you want to apply the value.

<bit_sequence>

It is the bit sequence to be shifted and can be binary or hexadecimal. This value is applied to the pin for subsequent clock cycles.

Example

```
dsm_assign_pin_value -pin in1 -value 11001
dsm_assign_pin_value -pin in1 -value 16'b0000_1010_1011_0001
dsm_assign_pin_value -pin in1 -value 16'h0a_b1
```

Result

Consider the last command in the above example. Assume that this command is specified when the IEEE1500 sequence is at clock cycle =2. Following sequence is generated in this case:

```
Wrck pin: 0101 0101010101010101010101010101010101 0101010101
in1 pin:  xxxx  00000000110011001100111100000011 1111111111
```

Also, the number of clock cycles is 23.

NOTE: *Last value is repeated from the 19th clock cycle to the 23rd clock cycle.*

Scope

None

Return Value

None

dsm_reset_assertions

Resets all the assertion commands.

Description

This command resets all the assertion commands specified before the execution of this command. No check is performed on the [dsm_assert_path](#) and [dsm_assert_value](#) commands which are specified before reset_assertions command.

NOTE: *This command does not reset the IEEE1500 sequence generated on the pins neither will reset_iee1500. If that is required use [dsm_assign_ieee1500_pins](#) command again.*

Syntax

```
dsm_reset_assertions
```

Arguments

None

Example

```
dsm_reset_assertions
```

Result

Resets all the assertion commands specified before the execution of this command.

Scope

None

Return Value

None

dsm_check_assertions

Runs SpyGlass to check all assertions specified in the flow.

Description

This command runs SpyGlass to check all assertions specified in the flow from the start of the session till the last *dsm_reset_assertions* command is given.

After the execution of the command, the result of the assertions is described in the `moresimple.rpt` file, which is generated in the `spyglass_reports` folder. Each execution of the *dsm_check_assertions* command overwrites the `moresimple.rpt` file of the previous run.

Syntax

```
dsm_check_assertions
```

Arguments

None

Example

```
dsm_check_assertions
```

Result

Checks all assertions specified in the flow and generates the `moresimple.rpt` file in the `spyglass_reports` folder.

Scope

None

Return Value

None

dsm_capture_ieee1500_data

Captures the data from the wrapper into the output data register.

Description

This command allows you to capture the data from the wrapper into the output data register. This command neither takes an input nor returns an output.

The following table shows the values at the pins for one cycle of wrck when this command is run:

Pin	Pin-Value
Selectwir	0
Updatewr	0
Shiftwr	0
Capturewr	1

Syntax

```
dsm_capture_ieee1500_data
```

Arguments

None

Example

Consider the number of clock cycles as two. In addition, consider the following sequence as a previous sequence:

```
Wrck pin      : 0101
Wrst pin      : 1100
Wsi pin       : 0000
Selectwir pin : 0000
Updatewr pin  : 0000
Shiftwr pin   : 0000
Capturewr pin : 0000
```

Now, consider that the following command is executed:

Introduction

Dsm_capture_ieee1500_data

In this case, the following result is generated:

```
Wrck pin:      0101 01
Wrst pin:      1100 00
Wsi pin:       0000 00
Selectwir pin: 0000 00
Updatewr pin:  0000 00
Shiftwr pin:   0000 00
Capturewr pin: 0000 11
```

Scope

None

Return Value

None

dsm_capture_ieee1500_instruction

Captures the instruction from the wrapper into the output instruction register.

Description

This command allows you to capture the instruction from the wrapper into the output instruction register. This command neither takes an input nor returns an output.

The following table shows the values at the pins for one cycle of wrck when this command is run:

Pin	Pin-Value
Selectwir	1
Updatewr	0
Shiftwr	0
Capturewr	1

Syntax

```
dsm_capture_ieee1500_instruction
```

Arguments

None

Example

Consider the number of clock cycles as two. In addition, consider the following sequence as a previous sequence:

```
Wrck pin      : 0101
Wrst pin      : 1100
Wsi pin       : 0000
Selectwir pin: 0000
Updatewr pin  : 0000
Shiftwr pin   : 0000
Capturewr pin: 0000
```

Now, consider that the following command is executed:

```
dsm_capture_ieee1500_instruction
```

In this case, the following result is generated:

```
Wrck pin:      0101 01
Wrst pin:      1100 00
Wsi pin:       0000 xx
Selectwir pin: 0000 11
Updatewr pin:  0000 00
Shiftwr pin:   0000 00
Capturewr pin: 0000 11
```

Scope

None

Return Value

None

dsm_read_ieee1500_data

Reads the data from output data register.

Description

This command allows you to read the data from output data register by using the wso pin. This command takes bits sequence or integer as input and returns simulation value of length N (specified in input) on the wso pin.

Syntax

```
dsm_read_ieee1500_data  
    [-input_sequences <input_bit_sequence>]  
    [-registers_size <registers_size>]
```

Arguments

This command has following arguments:

<input_bit_sequence>

Specifies the input bits sequence of length N.

When bits sequence of length N is specified in -input_sequences:

1. User-specified bits sequence will be applied at the wsi pin.
2. Circuit will be simulated for N clock cycles.
3. Simulation value of N + 1 clock cycles will be read at the wso pin.

Following Info message will be reported:

```
'info: Node 's349_wrapped.WSO' has value <value at wso pin>'
```

For -input_sequences 101100:

WSI pin : 101100

WRCK pin : 010101

WSO pin : < 8 bits sequence will be read>

<registers_size>

Specifies the size of registers of integer N.

When integer N is specified in <registers_size>:

1. X value will be applied at the wsi pin.
2. Circuit will be simulated for N - 1 clock cycles.
3. Simulation value of N clock cycles will be read at the wso pin.

Following Info message will be reported:

```
'info: Node 's349_wrapped.WSO' has value <value at wso pin>'
```

For -registers_size 4:

```
WSI pin : XXXXXX
```

```
WRCK pin : 010101
```

```
WSO pin : < 8 bits sequence will be read>
```

NOTE: *You must specify only one of the -input_sequences or -registers_size options at a time.*

Example

Example 1

Consider the number of clock cycles as three. In addition, consider the following sequence as a previous sequence:

```
Wrck pin      : 010101
Wrst pin      : 110000
Wsi pin       : 0000xx
Selectwir pin: 000000
Updatewr pin  : 000000
Shiftwr pin   : 000000
Capturewr pin: 000011
```

Now, consider that the following command is executed:

```
dsm_read_ieee1500_data -input_sequences 1010
```

In this case, the following result is generated:

```
Wrck pin:      010101  0101
Wrst pin:      110000  0000
Wsi pin:       0000xx  1010
Selectwir pin: 000000  0000
Updatewr pin:  000000  0000
```

```
Shiftwr pin: 000000 1111
Capturewr pin: 000011 0000
```

NOTE: *The sequence 1010, specified through the `-input_sequences` option, is loaded in the data register using the `wsi` pin.*

Example 2

Consider the number of clock cycles as three. In addition, consider the following sequence as a previous sequence:

```
Wrck pin      : 010101
Wrst pin      : 110000
Wsi pin       : 0000xx
Selectwir pin: 000000
Updatewr pin  : 000000
Shiftwr pin   : 000000
Capturewr pin: 000011
```

Now, consider that the following command is executed:

```
dsm_read_ieee1500_data -registers_size 3
```

In this case, the following result is generated:

```
Wrck pin:      010101 010101
Wrst pin:      110000 000000
Wsi pin:       0000xx xxxxxxx
Selectwir pin: 000000 000000
Updatewr pin:  000000 000000
Shiftwr pin:   000000 111111
Capturewr pin: 000011 000000
```

Scope

None

Return Value

Current simulation value at the `wso` pin for the required cycle

dsm_read_ieee1500_instruction

Reads the data from output instruction register.

Description

This command allows you to read the data from output instruction register by using the wso pin. This command takes bits sequence or integer as input and returns simulation value of length N (specified in input) on the wso pin.

Syntax

```
dsm_read_ieee1500_instruction  
    [-input_sequences <input_bit_sequence>]  
    [-registers_size <registers_size>]
```

Arguments

This command has following arguments:

<input_bit_sequence>

Specifies the input bits sequence of length N.

When bits sequence of length N is specified in -input_sequences:

1. User-specified bits sequence will be applied at the wsi pin.
2. Circuit will be simulated for N clock cycles.
3. Simulation value of N + 1 clock cycles will be read at the wso pin.

Following Info message will be reported:

```
'info: Node 's349_wrapped.WSO' has value <value at wso pin>'
```

For -input_sequences 101100:

WSI pin : 101100

WRCK pin : 010101

WSO pin : < 8 bits sequence will be read>

<registers_size>

Specifies the size of registers of integer N.

When integer N is specified in <registers_size>:

1. X value will be applied at the wsi pin.
2. Circuit will be simulated for N - 1 clock cycles.
3. Simulation value of N clock cycles will be read at the wso pin.

Following Info message will be reported:

```
'info: Node 's349_wrapped.WSO' has value <value at wso pin>'
```

For -registers_size 4:

WSI pin : XXXXXX

WRCK pin : 010101

WSO pin : < 8 bits sequence will be read>

NOTE: *You must specify only one of the -input_sequences or -registers_size options at a time.*

Example

Example 1

Consider the number of clock cycles as three. In addition, consider the following sequence as a previous sequence:

```
Wrck pin      : 010101
Wrst pin      : 110000
Wsi pin       : 0000xx
Selectwir pin: 000011
Updatewr pin : 000000
Shiftwr pin  : 000000
Capturewr pin: 000011
```

Now, consider that the following command is executed:

```
dsm_read_ieee1500_instruction -input_sequences 1010
```

In this case, the following result is generated:

```
Wrck pin:      010101  0101
Wrst pin:      110000  0000
Wsi pin:       0000xx  1010
Selectwir pin: 000011  1111
```

```

Updatewr pin: 000000 0000
Shiftwr pin: 000000 1111
Capturewr pin: 000011 0000

```

NOTE: *The sequence 1010, specified through the `-input_sequences` option, is loaded in the instruction register using the `wsi` pin.*

Example 2

Consider the number of clock cycles as three. In addition, consider the following sequence as a previous sequence:

```

Wrck pin      : 010101
Wrst pin      : 110000
Wsi pin       : 0000xx
Selectwir pin: 000011
Updatewr pin  : 000000
Shiftwr pin   : 000000
Capturewr pin: 000011

```

Now, consider that the following command is executed:

```
dsm_read_ieee1500_instruction -registers_size 3
```

In this case, the following result is generated:

```

Wrck pin:      010101 010101
Wrst pin:      110000 000000
Wsi pin:       0000xx xxxxxxx
Selectwir pin: 000011 111111
Updatewr pin:  000000 000000
Shiftwr pin:   000000 111111
Capturewr pin: 000011 000000

```

Scope

None

Return Value

Current simulation value at the `wso` pin for the required cycle

Example Flow

This section describes an example of IEEE1500 flow. It consists of the following topics in a sequential order:

1. [Sample Tcl Script](#)
2. [Sample SGDC File](#)
3. [Sample Waveform Viewer](#)

Sample Tcl Script

Consider the following sample Tcl script for activating and debugging the IEEE1500 wrapper in a design.

```
new_project sample_project -force
set_option top top
read_file -type hdl test.v
dsm_reset_assertions
dsm_assign_ieee1500_pins -wsi wsi -wso wso -wrck wrck
-selectwir selectwir -capturewr capturewr -shiftwr shiftwr -
updatewr updatewr -wrst wrst
dsm_reset_ieee1500 5
dsm_load_ieee1500_instruction 0110
dsm_load_ieee1500_data 1110
dsm_load_ieee1500_instruction 1
dsm_load_ieee1500_instruction 0
dsm_assert_value -name wso -value 0 -matchNBits 1 -waveform
dsm_check_assertions
gui_start
```

Sample SGDC File

The IEEE-1500 flow internally invokes Soc_01, Soc_01_Info, Soc_02 and Soc_02_Info rules of the SpyGlass Connectivity Verify product. For this example, the following SGDC file is generated in the back end to invoke these rules:

```
current_design top
//          # clock pulses: 20
```



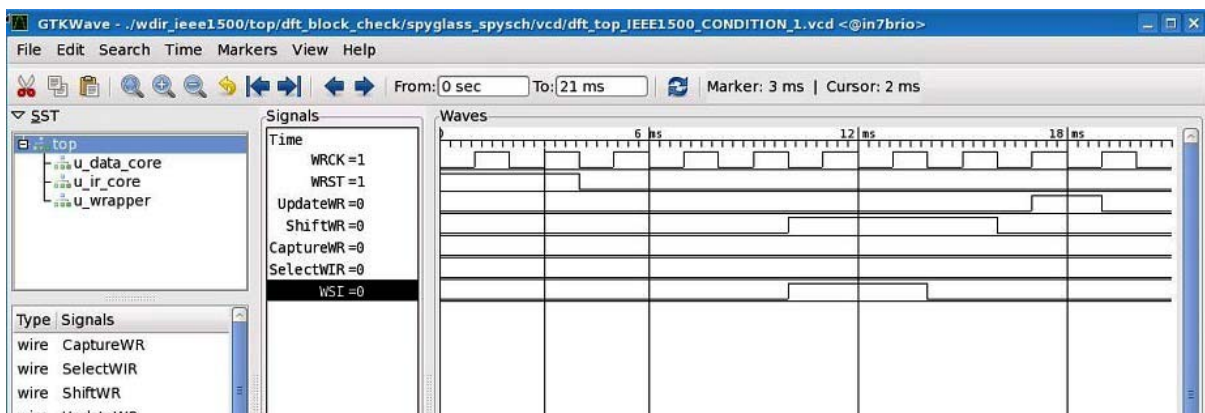
```

# bit sequence for IEEE1500 pin SHIFTWR
testmode -name shiftwr \
-value 0000000000001111111100111111100110011000
# bit sequence for IEEE1500 pin CAPTUREWR
testmode -name capturewr \
-value 0000000000000000000000000000000000000000
# bit sequence for IEEE1500 pin UPDATEWR
testmode -name updatewr \
-value 00000000000000000000110000000011001100111
# bit sequence for IEEE1500 pin WSI
testmode -name wsi \
-value 000000000000001111100000011111100110000000

```

Sample Waveform Viewer

Also, the following .vcd file is generated in the SpyGlass Waveform Viewer, if you have specified the `-waveform` option in the [dsm_assert_path](#) command:



The generated waveform allows you to validate your results against the specified input.

JTAG Based SoC DFT Debug

Introduction

The JTAG IEEE 1149.1 Test Access Port (TAP) based test access is a mature and common test technology.

Often, a SoC may integrate several IP's that require one or more test mode configurations that are controlled by TAP. Such test modes may consist of internal scan, BIST, test isolation, debug access, and IEEE 1500 wrapper test configurations. These test modes often involve over 100 different test modes that are setup via the TAP and require several test cycles to setup. These test cycles may range from 100 to 1000 cycles.






The JTAG based SoC debug is a useful feature that provides a debug environment to test such modes and related SpyGlass DFT ADV logic. You may want to setup each of these modes, and verify that the necessary test conditions are achieved, such as, specific test-mode values at internal nodes, necessary paths sensitized, required pulses propagated, and so on.

This section describes the [Related TCL commands](#) used to set and run the JTAG/TAP based SoC DFT debug. It also provides an [Example Flow](#), which helps you understand the JTAG/TAP based debug flow.

Related TCL commands

The JTAG based SoC DFT debug requires information on JTAG/TAP pins and instructions for activating and running the JTAG/TAP flow embedded in the design. This section describes the Tcl commands related to JTAG/TAP pins.

NOTE: *The JTAG based SoC DFT debug uses following commands from the IEEE1500 Wrapper debug flow:*

-  [dsm_assign_clock_pulse](#)
-  [dsm_assign_pin_value](#)
-  [dsm_assert_value](#)
-  [dsm_assert_path](#)
-  [dsm_check_assertions](#)

dsm_jtag_get_current_state

Get current state of TAP controller

Description

The *dsm_jtag_get_current_state* gets the current state of the TAP controller.

Syntax

```
dsm_jtag_get_current_state
```

Arguments

None

Example

Not Applicable

Scope

None

Return Value

None

dsm_set_current_jtag_state

Move TAP controller to specified state

Description

This command is an alias of the [*dsm_goto_jtag_state*](#) command.

dsm_get_jtag_pins

Gets the JTAG/TAP pins

Description

The *dsm_get_jtag_pins* command identifies and gets the JTAG/TAP pins in the design.

Syntax

```
dsm_get_jtag_pins
```

Arguments

None

Example

Not Applicable

Scope

None

Return Value

None

dsm_set_jtag_pins

Sets the JTAG/TAP pins

Description

The *dsm_set_jtag_pins* command identifies and sets the five JTAG/TAP pins in the design.

Syntax

```
dsm_set_jtag_pins
-tck <tck_pin_name>
-tdi <tdi_pin_name>
-tdo <tdo_pin_name>
-tms <tms_pin_name>
-trst <trst_pin_name>
```

Arguments

This command has following arguments:

<tck_pin_name>

Name of the TCK pin.

<tdi_pin_name>

Name of the TDI pin.

<tdo_pin_name>

Name of the TDO pin.

<tms_pin_name>

Name of the TMS pin.

<trst_pin_name>

Name of the TRST pin.

Example

Not Applicable

Introduction

Scope

None

Return Value

None

dsm_reset_jtag

Resets the TAP

Description

The *dsm_reset_jtag* command resets the TAP using asynchronous TRST signal. The command takes an integer as an optional argument, which specifies the number of cycles to keep the TRST signal active. TRST signal may need to be kept active for multiple cycles to ensure full chip resets without any timing issues.

Syntax

```
dsm_reset_jtag [<clock_cycles>]
```

Arguments

<clock_cycles>

This is an optional argument. You can specify an integer as an input. This argument specifies the number of cycles to keep the TRST signal active.

Example

```
dsm_reset_jtag 100
```

Scope

None

Return Value

None

dsm_load_jtag_instruction

Loads the specified bit sequence as a TAP instruction

Description

The *dsm_load_jtag_instruction* command loads the specified bit_sequence as a TAP instruction. The bit sequence can be either binary or a hexadecimal sequence.

Syntax

```
dsm_load_jtag_instruction <bit_sequence>
```

Arguments

<bit_sequence>

Specifies the bit sequence to be loaded as a TAP instruction. The bit sequence can be either binary or a hexadecimal sequence of the following format:

```
<integer>'[b|h]<bit|_>* or <[0|1|_>*
```

Here, bit is either 0 or 1 for binary, and 0,1,a,b,c,d,e,f for hex. The leading integer indicates the binary length of the sequence. The binary sequence is shifted-in with RHS bit first. The state of the TAP Controller is *exit1_ir* at the end of this sequence.

Following are the some examples of valid bit sequences:

```
4'b0010, 8'b00_11_00_11, 8'h01, 16'ha0_ff, 0010_0010
```

Please note that the bit sequence is prepended with 0 to meet the specified length.

Example

```
dsm_load_jtag_instruction 8'b0010
```

Scope

None

Return Value

None

dsm_load_jtag_data

Loads the specified bit sequence into a TAP data register

Description

The *dsm_load_jtag_data* command loads the specified bit_sequence into a TAP data register. The bit sequence can be either binary or hexadecimal sequence.

Syntax

```
dsm_load_jtag_data <bit_sequence>
```

Arguments

<bit_sequence>

Specifies the bit sequence to be loaded into a TAP data register. The bit sequence can be either binary or a hexadecimal sequence of the following format:

```
<integer>'[b|h]<bit|_>*' or <[0|1|_>*
```

Here, bit is either 0 or 1 for binary, and 0,1,a,b,c,d,e,f for hex. The leading integer indicates the binary length of the sequence. The binary sequence is shifted-in with RHS bit first. The state of the TAP Controller is *exit1_dr* at the end of this sequence.

Please note that the bit sequence is prepended with 0 to meet the specified length.

Example

```
dsm_load_jtag_data 100'b00110011
```

Scope

None

Return Value

None

dsm_goto_jtag_state

Takes the TAP controller to the specified state

Description

The *dsm_goto_jtag_state* command generates the necessary sequence to take the TAP controller to the specified state.

Syntax

```
dsm_goto_jtag_state <state>
```

Arguments

<state>

Name of the valid TAP state. Following are some of the valid states:

reset, run_test_idle, select_dr, capture_dr, shift_dr, exit1_dr, pause_dr, exit2_dr, update_dr, select_ir, capture_ir, shift_ir, exit1_ir, pause_ir, exit2_ir, update_ir

SpyGlass tracks the state of the controller and generates the necessary sequence for TRST, TDI, TMS, and TCK to take the controller to the specified state through the smallest number of state transitions, while avoiding the reset state as an intermediate state. This is necessary to avoid unintended reset of the controller when transitioning from one state to another. To reset the TAP Controller, the user can use either *dsm_reset_jtag* or *dsm_goto_jtag_state* reset.

Example

```
dsm_goto_jtag_state reset
```

Scope

None

Return Value

None

dsm_hold_jtag_state

Holds the current JTAG/TAP state for the specified number of cycles

Description

The *dsm_hold_jtag_state* command generates the necessary sequence to hold the current JTAG/TAP state for the specified number of cycles. Not that you can hold only following states, while TCK is pulsing:

`reset, run_test_idle, shift_dr, shift_ir, pause_dr, pause_ir`

Syntax

```
dsm_hold_jtag_state <clock_cycles>
```

Arguments

<clock_cycles>

Number of clock cycles to hold the current TAP state. You can specify an integer as an input to this argument.

Example

Consider the following example:

```
dsm_goto_jtag_state shift_dr;  
hold_jtag_state 10;
```

The above commands keep TAP in `shift_dr` state by holding TMS to 0.

Scope

None

Return Value

None

dsm_show_jtag_sequence

Print jtag sequence

Description

The *dsm_show_jtag_sequence* command prints the jtag sequence.

Syntax

```
dsm_show_jtag_sequence <-expanded>
```

Arguments

<-expanded>

Prints jtag sequence with each clock cycle expanded to 01.

Example

Scope

None

Return Value

None

Example Flow

This section describes an example of JTAG based SoC DFT debug flow. It consists of the following topics in a sequential order:

- [Sample Tcl Script](#)
- [Sample SGDC File](#)
- [Sample Waveform Viewer](#)

Sample Tcl Script

Consider the following sample Tcl script for activating and debugging the JTAG based SoC DFT in a design:

```
new_project PROJECT -force
set_option top top
read_file -type hdl test.v

dsm_reset_assertions
dsm_set_jtag_pins -trst trst -tms tms -tdi tdi -tdo tdo -tck\
tck

dsm_reset_jtag 5
dsm_assign_pin_value -pin a -value 0
dsm_reset_jtag 20

dsm_assign_pin_value -pin a -value 1
dsm_hold_jtag_state 10
dsm_assign_clock_pulse -pin ck -disable
dsm_load_jtag_instruction 4'b1001
dsm_assert_value -name top.exit_ir -value 1 -match_n_bits 1
dsm_assert_path -from top.mbist_tdo -to tdo -path_type\
sensitized -waveform
dsm_goto_jtag_state run_test_idle
dsm_assert_value -name top.run_test_idle -value 1 \
-match_n_bits 1
dsm_assert_value -name mbist_select -value 1 -match_n_bits 1
dsm_assert_value -name debug_select -value 0 -match_n_bits 1
```

Appendix:

CPF and UPF Commands

The SpyGlass DFT ADV solution provides support for Common Power Format (CPF) commands, Version 1.0e, and Unified Power Format (UPF) commands, Version 1.0 and 2.0.

This section describes the *CPF Commands* and *UPF Commands* supported by the SpyGlass DFT ADV solution.

CPF Commands

You can specify power specifications in a CPF file and provide it as an input. SpyGlass will read and parse the CPF commands and based on the power intent information provided with these commands, SpyGlass performs the rule checking.

The SpyGlass DFT ADV solution supports the *create_power_domain* CPF command.

Supported SGDC commands

For the CPF flow, SpyGlass honors the following SGDC commands:

- *activity*
- *always_on_buffer*
- *assume_path*
- *assertion_signal*
- *cell_hookup*
- *cell_pin_info*
- *cell_tie_class*
- *clock*
- *multivt_lib*
- *non_pd_inputcells*
- *power_data*
- *power_down*
- *power_down_sequence*
- *pg_cell*
- *pg_pins_naming*
- *ram_instance*
- *ram_switch*
- *set_case_analysis*
- *special_cell*
- *switchoff_wrapper_instance*

create_power_domain

Syntax

```
create_power_domain
  -name <power_domain>
  [ -instances instance_list ]
  [ -boundary_ports pin_list ]
  [ -default ]
  [ -shutoff_condition <expression> |
    -external_controlled_shutoff ]
  [ -default_restore_edge <expression> |
    -default_save_edge <expression> |
    -default_restore_edge <expression>
    -default_save_edge <expression> |
    -default_restore_level <expression>
    -default_save_level <expression> ]
  [ -power_up_states <high|low|random> ]
  [ -default_isolation_condition expression ]
  [ -active_state_conditions active_state_condition_list ]
  [ -secondary_domains domain_list ]
```

Description

The `create_power_domain` command creates a power domain.

Arguments

The `create_power_domain` command has the following arguments:

-name <power-domain>

Specifies the name of the voltage/power domain.

-default

(Optional) Specifies that the top domain name specified with the `set_design` command should be taken as the name of the top-level design unit belonging to the voltage/power domain being specified.

-instances <instance_list>

(Optional) Space-separated name list of instances belonging to the voltage/power domain being specified.

-boundary_ports <pin_list>

(Optional) Specifies a space-separated name list of top level ports and terminal of leaf-level design units working on a different voltage level from their parent module.

-shutoff_condition <expression>

(Optional) Specifies the condition when the power domain is shut off. If this condition is given, the domain is treated as power domain.

-external_controlled_shutoff

(Optional) Specifies that the power domain is shut off by a power switch, which is controlled by an external signal.

-default_restore_edge <expression>

(Optional) Specifies the condition when the state of the sequential elements is to be restored for state retention rules.

NOTE: *Simple expressions or complex expressions only with | and ! (OR and NOT) operators are supported currently. For example, <restore-sig-name> or !<restore-sig-name> or <restore-sig-name> | !<restore-sig-name>.*

-default_save_edge <expression>

(Optional) Specifies the condition when the state of the sequential elements is to be saved for state retention rules.

NOTE: *Simple expressions or complex expressions only with | and ! (OR and NOT) operators are supported currently. For example, <save-sig-name> or !<save-sig-name> or <save-sig-name> | !<save-sig-name>.*

The following argument of the `create_power_domain` command do not have any effect on the rule behavior.

- `-power_up_states <high | low | random>`

CPF Commands

- -default_isolation_condition <expression>
- -default_restore_level <expression>
- -default_save_level <expression>
- -active_state_conditions active_state_condition_list
- -secondary_domains domain_list

UPF Commands

You can specify power specifications in a UPF file and provide it as an input. SpyGlass will read and parse the UPF commands and based on the power intent information provided with these commands, performs the rule checking.

This section provides details of the following UPF commands supported by the SpyGlass DFT ADV solution:

- *create_power_domain*
- *create_power_switch*
- *set_isolation*
- *set_isolation_control*
- *set_retention*
- *set_retention_control*

Supported SGDC Commands

For the UPF flow, SpyGlass also honors the following SGDC commands:

- *activity*
- *always_on_buffer*
- *assume_path*
- *assertion_signal*
- *cell_hookup*
- *cell_pin_info*
- *cell_tie_class*
- *clock*
- *multivt_lib*
- *non_pd_inputcells*
- *power_data*
- *power_down*
- *power_down_sequence*
- *pg_cell*
- *pg_pins_naming*

UPF Commands

- *ram_instance*
- *ram_switch*
- *set_case_analysis*
- *special_cell*
- *switchoff_wrapper_instance*

create_power_domain

Syntax

```
create_power_domain <domain_name>
[-elements <list>]
[-include_scope]
[-scope <instance_name>]
[-supply <supply_set_handle supply_set_ref>]
[-update]
```

NOTE: *To use the arguments in blue, change the UPF Version to 2.0.*

Description

The `create_power_domain` command creates a power domain.

Arguments

The `create_power_domain` command has the following arguments:

<domain-name>

(Mandatory) Specifies the name of the voltage/power domain.

-elements <list>

(Optional) Space-separated name list of instances belonging to the voltage/power domain being specified.

-include_scope

(Optional) Specifies that the top domain name specified with the `set_design_top` command should be taken as the name of the top-level design unit belonging to the voltage/power domain being specified.

-scope <instance_name>

(Optional) Specifies that domain is to be created with the specified hierarchy.

-supply <supply_set_handle supply_set_ref>

(Optional) Specifies the supply set handle and reference of the associated supply set. The supply set should be created with the *create_supply_set* command.

For details of this command, refer to the *SpyGlass Power Verify Solution Rules Reference Guide*.

create_power_switch

Syntax

```
create_power_switch <switch_name>
-output_supply_port {<port_name> <supply_net_name>}
{-input_supply_port {<port-name> <supply-net-name>}}*
{-control_port {<port-name> <net-name>}}
[-domain <domain_name>]
```

Description

The `create_power_switch` command defines the power switch in the power domain.

Arguments

The command has the following arguments:

<switch_name>

(Mandatory) Specifies the name of the power switch.

-output_supply_port <port-name> <supply-net-name>

(Mandatory) Specifies the output supply port of the power switch and the supply net to which the output port is connected.

-input_supply_port <port-name> <supply-net-name>

(Mandatory) Specifies the input supply port of the power switch and the supply net to which the input port is connected

-control_port {<port-name> <net-name>}

(Mandatory) Specifies the enable port of the power switch and the signal name (on signal) to which this enable port is connected.

NOTE: *SpyGlass supports only one input supply port and up to two control ports.*

-domain <domain_name>

(Optional) Specifies the name of the domain containing the switch.

The following arguments of the `create_power_switch` command do not have any effect on the rule behavior:

- `-on_state { <state-name> <input-supply-port> { <boolean-function> } } *`
- `-on_state { <state-name> <input-supply-port> { <boolean-function> } } *`
- `-ack_delay { <port-name> <delay> }] *`
- `-off_state { <state-name> { <boolean-function> } }] *`
- `-error_state { <state-name> { <boolean-function> } }] *`

set_isolation

Syntax

```
set_isolation <isolation_name>
-domain <domain_name>
[-isolation_power_net <net_name>]
[-isolation_ground_net <net_name>]
[-no_isolation]
[-elements <list>]
[-clamp_value {< 0 | 1 | any | Z | latch | value>*}]
[-applies_to <inputs | outputs | both>]

[-source <source_supply> | -sink <sink_supply>]
[-isolation_supply_set <list_of_supply_sets>]
[-isolation_sense {<high | low>}]
[-location <automatic | self | other | fanout | fanin |
faninout | parent | sibling>]
[-diff_supply_only < TRUE | FALSE> ]
[-update]
```

NOTE: *To use the arguments in blue, change the UPF Version to 2.0.*

Description

The `set_isolation` command creates an isolation rule.

Arguments

The `set_isolation` command has the following arguments:

<isolation_name>

(Mandatory) Specifies the name of the isolation rule to be mapped.

-domain <domain_name>

(Mandatory) Specifies the name of the power domain.

-elements <list>

(Optional) Specifies the hierarchical names of pins to be isolated.

-source <source_supply>

(Optional) Specifies the source supply. Use this argument to filter the ports receiving a net that are driven by logic powered by the supply set.

-sink <sink_supply>

(Optional) Specifies the sink supply. Use this argument to filter the ports driving a net that fans out to logic powered by the supply set.

-applies_to <inputs | outputs | both>

(Optional) Specifies if the input ports, output port, or both types of ports of the power domain are to be isolated.

-isolation_supply_set <list_of_supply_sets>

(Optional) Defines the supply sets for the isolation logic inferred by this strategy.

NOTE: *Only the first supply set mentioned is used.*

-isolation_power_net <net_name>

(Optional) Defines the power supply net for the isolation logic inferred by this strategy.

NOTE: *An explicit supply connection specified for isolation logic using `connect_supply_net` is given higher priority than the `isolation_power_net` specified here.*

-isolation_ground_net <net_name>

(Optional) Defines the ground supply net for the isolation logic inferred by this strategy.

-no_isolation

(Optional) Specifies that isolation logic is not required for the elements specified with the `-elements` argument.

-isolation_sense <high | low>

(Optional) Specifies the state of the signal list specified with `-isolation_signal` argument. The default value is `high` for UPF version 1.0 and 2.0.

-clamp_value <0 | 1 | latch | Z>

(Optional) Specifies the expected steady-state value of power domain inputs or outputs.

NOTE: *SpyGlass does not support the value, Z, for this field.*

-location <automatic | self | fanout | parent | sibling>

(Optional) Specifies where the isolation cell is to be placed in the logical hierarchy. The default value is `automatic`.

NOTE: *The -isolation_signal, -isolation_sense, and -location options have been added in UPF Version 2.0.*

-diff_supply_only <TRUE | FALSE>

(Optional) Specifies the isolation behavior between the driver and receiver supply sets.

-update

(Optional) Signifies that the `set_isolation` command refers to an `isolation_name` that was previously defined.

set_isolation_control

Syntax

```
set_isolation_control <isolation_name>  
-domain <domain_name>  
-isolation_signal <signal_name>  
[-isolation_sense <high | low>]  
[-location <automatic | self | fanout | parent | sibling>]
```

Description

The `set_isolation_control` command specifies the control signal for an isolation rule.

Arguments

The `set_isolation_control` command has the following arguments:

<isolation_name>

(Mandatory) Specifies the name of the isolation rule.

-domain <domain_name>

(Mandatory) Specifies the name of the power domain.

-isolation_signal <signal_name>

(Mandatory) Specifies name of an isolation control signal.

-isolation_sense <high | low>

(Optional) Specifies the active level of isolation control signal.

-location <automatic | self | fanout | parent | sibling>

(Optional) Specifies where the isolation cell is to be placed in the logical hierarchy. The default value is `automatic`.

-update

(Optional) Signifies that the `domain_name` has already been defined.

set_retention

Syntax

```
set_retention <retention_name>
-domain <domain_name>
[-elements <list>]
[-save_signal {<net_name> <high | low | posedge | negedge>}]
[-restore_signal {<net_name> <high | low | posedge |
negedge>}]
[-update]

[-retention_power_net <net_name>]
[-retention_ground_net <net_name>]
[-retention_supply_set <retention_supply_name>]
```

NOTE: To use the arguments in blue, change the UPF Version to 2.0.

Description

The `set_retention` command creates an retention rule.

Arguments

The `set_retention` command has the following arguments:

<retention_name>

(Mandatory) Specifies the name of the retention rule.

-domain <domain_name>

(Mandatory) Specifies the name of the power domain.

[-elements <list>]

(Optional) Specifies space-separated hierarchical instance name list indicating the regions to be checked.

-save_signal {<net_name> <high | low | posedge | negedge>}

(Optional) Specifies the signal name and its active value, which causes the state of the sequential elements to be saved for state retention rule. The default sensitivity is high.

-restore_signal {<net_name> <high | low | posedge | negedge>}

(Optional) Specifies the signal name and its active value, which causes the state of the sequential elements to be restored for state retention rule. The default sensitivity is high.

NOTE: *The -save_signal and -restore_signal options have been added in UPF Version 2.0.*

-update

(Optional) Signifies that the set_retention command refers to a retention_name that was previously defined.

-retention_supply_set <retention_supply_name>

(Optional) Defines the supply set used to power the logic inferred by the retention_name strategy.

-retention_power_net <net_name>

(Optional) Defines the supply net used as the power for the retention logic inferred by this strategy.

-retention_ground_net <net_name>

(Optional) Defines the supply net used as the power for the retention logic inferred by this strategy.

set_retention_control

Syntax

```
set_retention_control <retention_name>  
-domain <domain_name>  
-save_signal {<net_name> <high | low | posedge | negedge>}  
-restore_signal {<net_name> <high | low | posedge | negedge>}
```

Description

The `set_retention_control` command specifies a control signal and assertion for an retention rule.

Arguments

The `set_retention_control` command has the following arguments:

<retention_name>

(Mandatory) Specifies the name of the retention rule.

-domain <domain_name>

(Mandatory) Specifies the name of the power domain.

-save_signal {<net_name> <high | low | posedge | negedge>}

(Mandatory) Specifies the signal name and its active value which causes the state of the sequential elements to be saved for state retention rule.

-restore_signal {<net_name> <high | low | posedge | negedge>}

(Mandatory) Specifies the signal name and its active value which causes the state of the sequential elements to be restored for state retention rule.

Appendix: SGDC Constraints

SGDC Concepts

SpyGlass Design Constraints (SGDC) provides additional design information that is not apparent in an RTL.

In addition, you can restrict SpyGlass analysis to certain objects in a design by specifying these objects by using SGDC commands.

SpyGlass Design Constraints

The following table lists the SGDC commands used by SpyGlass DFT ADV product:

<i>assume_path</i>	<i>balanced_clock</i>	<i>bypass</i>
<i>clock</i>	<i>clock_pin</i>	<i>clock_shaper</i>
<i>complex_cell</i>	<i>dbist</i>	<i>define_illegal_input_values</i>
<i>define_legal_input_values</i>	<i>define_tag</i>	<i>dont_touch</i>
<i>force_ta</i>	<i>gating_cell</i>	<i>illegal_path</i>
<i>initialize_for_bist</i>	<i>ip_block</i>	<i>keeper</i>
<i>memory_force</i>	<i>memory_read_pin</i>	<i>memory_tristate</i>
<i>memory_type</i>	<i>memory_write_disable</i>	<i>memory_write_pin</i>
<i>module_bypass</i>	<i>module_pin</i>	<i>no_fault</i>
<i>force_no_scan</i>	<i>pll</i>	<i>pulldown</i>
<i>pullup</i>	<i>require_path</i>	<i>require_strict_path</i>
<i>require_structure</i>	<i>require_value</i>	<i>reset_async</i>
<i>reset_pin</i>	<i>rme_config</i>	<i>force_scan</i>
<i>scan_cell</i>	<i>scan_chain</i>	<i>scan_ratio</i>
<i>scan_type</i>	<i>scan_wrap</i>	<i>scan_wrap</i>
<i>seq_atpg</i>	<i>set_pin</i>	<i>shadow_ratio</i>
<i>test_mode</i>	<i>test_point</i>	<i>tie_x</i>
<i>tristate_cell</i>	<i>abstract_port</i>	<i>atspeed_clock_frequency</i>
<i>clock</i>	<i>clock_root</i>	<i>clock_shaper</i>
<i>clockgating</i>	<i>complex_cell</i>	<i>compressor</i>
<i>decompressor</i>	<i>define_tag</i>	<i>expect_frequency</i>
<i>false_path</i>	<i>force_ta</i>	<i>gating_cell</i>
<i>gating_cell_enable</i>	<i>ip_block</i>	<i>module_bypass</i>
<i>no_atspeed</i>	<i>no_fault</i>	<i>force_no_scan</i>
<i>pll</i>	<i>pulldown</i>	<i>pullup</i>
<i>require_pulse</i>	<i>reset</i>	<i>force_scan</i>

SpyGlass Design Constraints

<i>assume_path</i>	<i>balanced_clock</i>	<i>bypass</i>
<i>scan_chain</i>	<i>scan_enable_source</i>	<i>test_mode</i>
<i>test_point</i>	<i>voltage_domain</i>	

Appendix: RME Parameters

Overview

RTL Modification Engine (RME) parameters are used by certain rules of the SpyGlass DFT ADV product.

An RME parameter is specified in the same way as any other parameter of SpyGlass. For details on how to specify a parameter, refer to the *Edit Parameters* topic in *Atrenta Console Reference Guide*.

RME Parameters

The following table lists the RME parameters used by the SpyGlass DFT ADV product:

SpyGlass DFT ADV Product	
<i>reme_active</i>	<i>reme_design_type</i>
<i>reme_modify_vhdl_configuration</i>	<i>reme_selection</i>

Glossary

A

Active clock phase

is the value on a latch enable that will cause the latch output to follow the value on its data pin.

C

Capture

Refers to the action of storing test values into a scan element for subsequent scan out.

Clock source

is defined as the first logic element found in the fan-in cone for a flip-flop clock input pin that has more than one input.

Clock domain

is all the flip-flops whose clocks have the same clock source. Note that the definition of clock source allows both rising and falling edge flip-flops to be in the same domain.

Clock line gating

are combinational gates in the clock distribution logic that control whether or not various clock domains receive clock edges/pulses.

Combinational feedback	is a set of combinational gates connected to form a closed loop in the netlist.
Combinationally connected	refers to a serially connected path (output of one gate in the path is connected an input on the next gate in the path) of combinational gates in a netlist.
Controlling value	is a value on a gate input such that when asserted, while all other inputs to the same gate have x values, the output of the gate is forced to a non-x value.
Controllability	Refers to whether or not a circuit node can be controlled to a particular value from the circuit inputs. SpyGlass controllability determines whether or not each circuit node can be controlled to logic 0, logic 1 or a high impedance state. The determination is made by first assigning all circuit root level inputs as controllability values to yyy (corresponding to 0, 1 and z states). An internal node is controllable to a value (0, 1 or z) if at least one row for the truth table defining that node to "value" has controllable inputs. For example, a two input mux with input I0 controllable to logic 1 and the selector controllable to logic 0 is sufficient to have the mux output controllable to logic 1. A flip-flop is controllable to logic "V" if its clock pin is controllable to both a 0 and a 1 and its data pin is controllable to "V." WARNING: controllability does not take signal independence into account so a 2-input AND gate fed by signal A and not A will be declared controllable to both 0 and 1 even though the 1 state cannot be statically maintained.
Gated clock	is a design style involving gates inserted into the clock distribution logic. The extra logic provides a means to selectively disable clocks to portions of a design.

I

Initializable circuits

are circuits that can be driven to a particular state when starting from an “all x” state. Implicitly initializable means that regardless of the power up state, the circuit may be driven to a particular state. Since the logic simulator embedded in SpyGlass starts simulating from an x state, implicitly initializable circuits may yield unreliable results.

Internally generated clocks

are clocks produced by internal state machines.

L

Lock-up latches

are those latches with the following connections:
The latch data pin has a single unblocked path from the output of a scan flip-flop.
Assuming cFF be the clock net driving the scan flip-flop, the latch enable pin must be controlled by \sim cFF.

N

Lock-up latches

are those latches with the following connections:
The latch data pin has a single unblocked path from the output of a scan flip-flop.
Assuming cFF be the clock net driving the scan flip-flop, the latch enable pin must be controlled by \sim cFF.

Non-controlling value

is a value on a gate input such that when asserted, while all other inputs to the same gate have x values, the output of the gate is x.

O

Observability

is a binary metric associated with each node in a circuit. Observability for all circuit primary output pins is defined to be YES. Observability of a node N is YES if and only if N is feeding an observable node M and the conditions required to sensitize M to N can be controlled according to controllability analysis.

P

Pin to pin paths

are combinational logic paths from a primary or root level input to a primary or root level output.

R

Retiming flip-flops

are defined by two properties:

- All flip-flops in the retiming flip-flop data cone and in the retiming flip-flop fan-out cone must be the same polarity (all are rising edge or all are falling edge)
- The retiming flip-flop clock must be opposite to the phase of the flip-flops in its fan-in and fan-out cones.

Retiming latch

is a latch used to hold data so as to prevent timing violations. Retiming latches are often used in data paths that cross clock domains.

Source clock domain latches are defined by four properties:

The latch data fan-in cone has one or more flip-flop q-pins

All flip-flops in the fan-in cone must have the same polarity (all are rising edge or all are falling edge) and be on the same clock domain as the latch

The latch enable active state must be low if the fan-in cone flip-flops are rising edge and high if the fan-in cone flip-flops are falling edge

All flip-flops in the latch fan-out cone must have the same polarity as the fan-in cone flip-flops. It is not necessary that the fan-out cone flip-flops have the same domain. The only requirement is on their polarity.

Destination clock domain latches are defined by four properties:

The latch data fan-out cone has one or more flip-flop q-pins

All flip-flops in the fan-out cone must have the same polarity (all are rising edge or all are falling edge) and be on the same clock domain as the latch

The latch enable active state must be high if the fan-out cone flip-flops are rising edge and low if the fan-in cone flip-flops are falling edge

All flip-flops in the latch fan-in cone must have the same polarity as the fan-out cone flip-flops. It is not necessary that the fan-in cone flip-flops have the same domain. The only requirement is on their polarity.

Return to value

is a logic low for a return-to-zero (rtz) signal and a high for a return-to-one (rto) signal.

S

Scan ratio	is the ratio of scan flip-flops to the total number of flip-flops in a circuit. A scan ratio of 100% is referred to as full scan.
Scannable equivalent	is flip-flop containing the same pin outs as a non-scan flip-flop and contains additional pins for scan enable and scan data. A circuit can be converted into a scan circuit by replacing each flip-flop with a scannable equivalent and providing the necessary interconnections to the test pins.
scanshift	is the process of shifting data into or out of a scan chain.
Shadow for the RAM	is the logic feeding a RAM or fed by a RAM that is not testable by a combinational ATPG tool.
Synchronous latches	are latches with their enable pin combinationally connected to a user designated clock pin.

T

Test clock	is a clock signal used to capture data or shift data during test mode.
Transparent latches	are latches whose enable are made active in test mode
Tri-state buses	are wired configurations of multiple tri-stateable components.
Tri-state logic	is logic that can supply a high impedance value

U

Un-blocked path

from the output Y of a device to an input IN on that device exists if the following conditions are satisfied:

- IN is currently assigned an x value,
- The other inputs must not have values that force or control Y,
- The other inputs must have values such that changes on IN cause changes on Y.

List of Topics

dft_allow_single_flipflop_driver_in_Async_02_capture.....	1776
.lib attributes.....	74
About This Book.....	31
add_fault.....	1611
Adding Test Points.....	129
All the Instances of the Selected Hierarchy.....	1927
Asynchronous Rules.....	155
atpg_conflict_testpoints.....	1613
ATPG_credit.....	1765
At-Speed (Transition Delay Fault) Coverage Definitions.....	138
At-Speed Capture Clock Discussion.....	136
Atspeed Test Rules.....	250
atspeed_03_rpt.....	1696
atspeed_04_rpt.....	1699
atspeed_07_rpt.....	1700
atspeed_08_rpt.....	1701
atspeed_clock_synchronization.....	1701
Back Annotation (BA) data for identified CGCs.....	84
BIST Rules.....	384
bist_ready_summary.....	1615
Blocked (BL).....	93
Buffered.....	104
busWidthHigh.....	1765
busWidthLow.....	1766
Capture (atspeed).....	60
Capture.....	59
captureClockCount.....	1766
captureClockEnd.....	1767
captureClockStart.....	1767
Checking Block-level Test Requirements.....	133
Checking for Presence of Valid Constraints.....	125
checkPLLSourceForAllNonFFCellClockPins.....	1767
Clock Browser window.....	1933
Clock Gating Report.....	82
Clock Gating Rules.....	542
Clock Rules.....	406

Common SpyGlass DFT ADV Rule Parameters	1761
Complying with the SpyGlass DFT ADV Best Practices	128
Configuring instance count threshold	1945
Configuring Legend	1929
Connection Rules	592
Contents of This Book	32
CPF Commands	2010
Dbist	61
debugInst	1769
Define_tag	62
Description	1429
Design Impact	107
Detectable (DT)	89
Detecting Structures Leading to Non-Robust Tests	116
Determine Coverage Percentage and Back-annotate Untestable Faults	110
Determine the Number of Undetected Faults and Back-annotate Cause Information	110
dft3BitClock	1838
dft_agnostic_cgc_flipflop_deadlock_check	1777
dft_allow_clock_merge_at_mux_via_data	1772
dftAllowedScanChainLengthDeviation	1840
dft_allow_inactive_resets_from_scan_ffs	1774
dftAllowMergeOfSameClock	1839
dftAllowNonXValueAtStartOfSensitizedPathInSoc_02	1840
dft_allow_path_from_enable_to_cgc_clkout	1776
dft_allow_sequential_propagation_during_clock_simulation	1775
dft_all_scan_chains_defined	1777
dft_atpg_conflict_tp_count	1778
dftAutoFix	1841
dft_autofix_testclock_signal	1779
dft_autofix_testmode_signal	1778
dft_block_unstable_value_trace_on_no_clock	1779
dftCaptureCriteria	1843
dft_cgc_pre_scan_stitched_treatment_only_to_tied_test_enable	1780
dft_check_clock_merge_in_shift_as_well	1781
dft_check_latch_transparency_in_shift	1781
dft_check_path_name_for_instance_suffix	1782
dft_check_path_name_for_register_suffix	1782
dft_check_test_mode_conflicts	1783
dftClockEdgeFaultAnalysis	1844
dftClockEdgeType	1844
dft_clock_end_point_types_for_Clock_02	1784

dft_collapse_equivalent_faults	1784
dft_conn_check_allow_non_x_value_on_sensitizable_path.....	1786
dft_conn_check_allow_trace_through_async	1771
dft_conn_check_allow_trace_through_clock_shaper	1772
dft_conn_check_handle_rtl_negedge	1786
dft_conn_check_rp_rsp_unified_flow.....	1788
dft_conn_check_treat_endpoint_as_stoppoint	1788
dft_connectivity_check_summary	1618
dft_conn_treat_fail_as_strict_fail.....	1787
dft_conn_treat_pass_as_strict_pass	1787
dft_coverage_report_depth.....	1789
dftCPMechanism.....	1845
dftDebugData	1841
dftDesignState	1845
dft_detect_shadow_latches.....	1789
dftDoInferredNoScanAnalysis	1849
dftDoLogicalRedundancyCheck	1849
dft_dsm_clock_frequency	1706
dft_dsm_clock_gating_cell.....	1706
dft_dsm_constr-err-file	1710
dftDsmConvergingPathsLimit	1850
dft_dsm_enabled_flipflops	1710
dft_dsm_ip_report.....	1715
dft_enable_checks_for_all_cgc.....	1791
dftEnablePinNameForCGC.....	1851
dftExpectedScanChainLength	1851
dft_ff_set_reset_active	1623
dft_ff_set_reset_sequential_in_capture	1625
dft_ff_set_reset_sequential_in_shift.....	1625
dft_ff_X_source_for_tristate_enable	1626
dftFindCombLoopThruSetResetToQ	1852
dftFunctionalClockPropagation.....	1853
dft_generate_no_fault_and_add_fault_report.....	1855
dft_generate_robustness_audit_report	1855
dftGenerateStuckAtFaultReport	1852
dftGenerateTextReport	1853
dft_identify_equivalent_faults	1794
dft_identify_lockup_latch_only_through_buffers	1794
dftIdentifyTestPoints.....	1854
dft_ignore_bb_inout_term_for_topology_05.....	1858
dft_ignore_constant_latches_in_rule_checking	1792

dftIgnoreConstantOrUnusedFlipFlops	1854
dft_ignore_constant_supply_flip_flops	1859
dft_ignore_inout_ports_as_driver	1793
dft_ignore_no_scan_driver_in_Async_02_capture	1860
dft_ignore_no_scan_latches_in_rule_checking	1793
dft_ignore_state_holding_flipflop_driver_in_Async_02_capture	1799
dft_ignore_unate_reconvergence	1799
dft_ignore_x_sources	1796
dft_ignore_x_sources_for_bist	1797
dft_infer_clock_gating_cell	1860
dft_infer_clock_gating_cell_test_enable	1861
dft_infer_flop_enable	1795
dftInferredDriverControl	1863
dft_infer_sequential_propagation_as_no_scan	1796
dft_infer_tp_clock	1800
dft_initialized_ffs	1652
dft_insert_rrf_tp	1800
dft_insert_ta_tp	1800
dft_internal_testmode_nodes_stability_check	1801
dft_latch_enable	1629
dft_list_latches_of_type	1801
dft_mandatory_sgdc	1631
dftmax_allow_asymmetric_pin_allocation	1770
dftmax_config_lookup_file	1771
dft_max_files_in_a_directory	1856
dft_max_flops_in_diagnose_scan_chain_violation_schematic	1803
dft_maximum_number_of_messages_with_spreadsheet	1802
dft_maximum_number_of_rows_with_schematic	1856
dftmax_is_occ_present	1770
dftMaxScanChainLength	1864
dftmax_si_so_pin_number	1769
dftmax_ultra_configuration	1637
dftmax_ultra_configuration_command	1639
dft_memory_report	1635
dft_min_scannability_ratio_for_test_points	1803
dft_min_scannability_ratio_for_test_points	1857
dft_multibit_flipflops	1636
dft_optional_sgdc	1647
dft_pattern_count	1858
dftPOSDetect_credit	1864
dft_report_all_paths_between_reconvergence_start_and_end	1804

dft_report_cgc_centric_deadlock	1805
dftReportIfUsedAsClock	1866
dftReportOnlyBBForBIST	1866
dftReportWDriversInTriBus	1866
dft_require_path_fail_limit	1805
dft_require_path_invalid_limit	1806
dft_require_path_pass_limit	1806
dft_rrf_display_limit	1806
dft_rrf_generate_fault_report	1807
dft_rrf_tp_count	1808
dft_rrf_tp_count_for_cutoff_incremental_gain	1808
dft_rrf_tp_cutoff_incremental_gain	1809
dft_rrf_tp_effort_level	1809
dft_rrf_tp_generate_dc_report	1809
dft_rrf_tp_ignore_generated_nets	1810
dft_rrf_tp_target_test_coverage	1811
dft_rrf_tp_thread_count	1810
dft_rrf_tp_type	1811
dft_rrf_tp_use_multiple_threads	1811
dft_scan_chain_report_redundant_lockup_latch	1812
dftScanDataConnectivityCheck	1868
dftScanEnablePinValue	1869
dft_scannable_latches	1861
dft_self_gating_ff	1641
dft_self_gating_logic_summary	1639
dft_self_gating_test_points	1643
dftSENames	1869
dftSetAllBBScanwrapped	1870
dftSetAllLatchT	1870
dftSetAllTriEnableObs	1871
dftSetCGCEnablePinsValue	1871
dftSetEffortLevel	1872
dftSetLatchFedByTCIkAsT	1872
dft_set_scan_wrap_on_memory	1813
dftSetTMTClkNodesAsUnObs	1873
dft_share_atpg_conflict_testpoint_at_branch	1813
dftShowDataInversionInScanChain	1873
dftShowForcedValues	1873
dft_show_rule_name_in_audit	1813
dft_show_scan_clock_tree	1814
dft_show_schematic_info_in_coverage_audit	1816

dftShowSourceCentricViolation	1874
dft_show_spreadsheet_path_in_message	1816
dft_show_unused_define_tag	1817
dftShowWaveForm	1875
dftSINames	1875
dftSkipLogicForTestPoints	1867
dftSkipNonXValueNetsForTA	1875
dftSkipPwrGndNetsForTA	1876
dftSkipTAMsgCount	1876
dftSkipTestPointsOnImprovementLessThan	1867
dftSkipUnObservableNetsForTA	1877
dft_soc_strict_boundary_validation	1817
dft_soc_unstable_value_sources	1819
dftSONames	1877
dftSortTAMsgOnFaultCount	1877
dft_stable_value_pessimistic_check	1820
dft_state_holding_ff_identification_effort_level	1821
dft_stop_simulation_at_mux_with_unknown_select	1822
dft_store_tcl_query_data	1821
dftStrictTestClkDomains	1878
dft_summary	1649
dft_support_flip_flop_bank	1826
dftTagAsync07Clock11Scan08	1868
dftTargetFaultCoverage	1878
dft_target_random_pattern_fault_coverage	1829
dft_target_random_pattern_test_coverage	1829
dft_target_stuck_at_fault_coverage	1830
dft_target_stuck_at_test_coverage	1830
dftTargetTestCoverage	1879
dft_target_transition_fault_coverage	1831
dft_target_transition_test_coverage	1831
dft_ta_tp_count	1827
dft_ta_tp_criteria	1829
dft_ta_tp_for_port	1826
dft_ta_tp_type	1827
dftTestclockCycles	1879
dftTestEnablePinNameForCGC	1880
dftTMPropThruFF	1883
dftTreatBBoxAsScanwrapped	1882
dftTreatControllableLatchTransparentForLoop	1880
dftTreatFlipFlopsAsScan	1881

dftTreatLatchesAsTransparent	1881
dft_treat_latches_with_X_on_enable_as_combinational_for_soc_path_checks ..	1831
dft_treat_primary_inputs_as_x_source.....	1832
dft_treat_primary_outputs_as_unobservable.....	1833
dft_treat_primary_port_as_valid_clock_point_for_cgc_check	1833
dft_treat_suffix_as_pattern	1862
dft_tristate	1651
dft_use_cumulative_fault_status	1839
dft_use_n_cycle_capture.....	1834
dft_use_new_Atsspeed_19.....	1837
dftUseOffStateOfClockInClockPropagation.....	1882
dftUseOffStateOfClockInClockPropagation.....	1883
dft_use_old_Topology_10.....	1835
dft_use_simulation_based_unblocked_path_check_for_scan_chain_tracing	1863
dft_use_source_centric_Async_O2_capture	1835
dft_use_specified_and_inferred_clocks	1836
dft_use_stable_simulation_conditions	1836
dftUUMarking.....	1885
dft_x_capture_report	1658
dftXPropForScanFF	1896
dft_x_source_report	1656
dft_x_sources_display_limit	1837
Diagnosing a Problem	1914
Diagnosing Constraint Issues	1914
Diagnosing Design Issues	1914
Diagnostic Rules.....	676
Difference Between no_scan and Unscannable Flip-Flops	66
Direct	105
dsm_apply_migrated_constraints	1885
dsmAtsspeedClockSynchronizationRange	1892
dsm_check_unblocked_mcp.....	1885
dsm_count_false_path_transition_faults.....	1886
dsmGenerateTransitionFaultReport	1892
dsm_gen_hiersgdc	1888
dsmIgnoreControlLineFaults.....	1893
dsm_infer_clock_domain_at_clock_shaper_boundary	1888
dsm_launch_method.....	1889
dsmLimitMigrationReportFanin	1893
dsm_max_fanin_paths_for_Atsspeed_12	1889
dsm_memory_clock_check	1889
dsmReportAllCGCCEFaninFlops.....	1893

dsm_report_domain_for_faults	1891
dsmReportEnabledFlops	1895
dsm_use_cumulative_fault_status	1891
dsmUsePIForAtSpeed	1895
dsmUsePOForAtSpeed	1895
Equivalent (EQ)	90
Example Flow	1986
Example Flow	2003
Example	105
Fault Coverage and Test Coverage	108
faultAnnotation	1896
Features	97
ff_async_reconvergence	1719
ff_clock_reconvergence	1721
flopInFaninCount	1898
forced_scan	1654
Functional	57
gateInputCount	1898
head_pipeline_clock_phase	1898
head_pipeline_stages	1899
Identifying Clock Gating Cells	74
Identifying Synchronizer	86
Identifying Test Points To Reduce Random Resistance	116
Identifying Testclocks	45
ignoreBlackBoxDuringSimulation	1900
ignoreBlackBoxDuringTestability	1900
Immediate Instances Inside the Selected Hierarchy	1926
Impact of Different Path Types on Fanin/Fanout Cone Traversal	102
Improvements to Fault and Test Coverage	108
incrementalTA	1901
Inference in the RTL code	75
inferred_no_scan_ffs	1655
Inferring Flat CGC	80
Information Rules	686
Inputs	97
Inserting RTL Testpoints	97
Integrity Checks	1512
initialize_for_bist	61
Introduction	1951
Introduction	1989
Key Concepts	36

Latch Rules.....	978
limitFaninPorts.....	1901
Load complete sub-hierarchy.....	1948
Load selected hierarchy only.....	1947
Logical Redundant (LR).....	95
Make Latches Transparent.....	129
Make The Design Robustness Ready.....	129
Making the RTL Ready for Atspeed Test.....	133
Making the RTL scan ready.....	125
mergeN.....	1902
monitorFlow.....	1902
no_atspeed.....	1727
noBlackBoxReporting.....	1903
no_fault.....	1646
No-Scan Flip-Flops.....	63
no_scan.....	1659
observable_ffs.....	1660
Operating Modes.....	56
Organizing Report Columns.....	1928
Output.....	98
Overview.....	1020
Overview.....	1036
Overview.....	1082
Overview.....	1209
Overview.....	1228
Overview.....	1263
Overview.....	1273
Overview.....	1296
Overview.....	1351
Overview.....	1362
Overview.....	1431
Overview.....	1512
Overview.....	155
Overview.....	2033
Overview.....	250
Overview.....	384
Overview.....	406
Overview.....	542
Overview.....	676
Overview.....	686
Overview.....	978

pathDepth	1904
Performing an On-Demand Trace	1937
Performing Conditional Connectivity Checks	143
PLL Rules	1020
Potentially Detectable (PT)	91
potentially_detected_faults	1660
Power Ground	57
Process Flow	141
RAM Rules	1036
random_pattern_coverage	1728
random_pattern_dc_testpoints	1729
random_pattern_node_distribution	1732
random_pattern_testpoints	1735
reconvergenceMinDepth	1904
Related TCL Commands	1951
Related TCL commands	1990
Reporting Missing Constraints	123
Reports Generated by the Info_dftDebugData Rule	83
Retiming or Lockup Latches	71
RME Parameters	2034
RTL Design for Test	37
Sample SGDC File	1986
Sample SGDC File	2004
Sample Tcl Script	1986
Sample Tcl Script	2003
Sample Waveform Viewer	1988
Sample Waveform Viewer	2008
Sanity Check Rules	1209
Scan Clock Tree Report	1620
Scan Clock Tree Report	1936
Scan Enable Rules	1228
Scan Power Rules	1263
Scan Rules	1082
scan_chain	1661
Scannability in Different Modes	107
Scannability	107
Scannability-Dependent Rules	108
Scannable Flip-Flops	63
Scannable	71
Scanning a Clock Tree	1933
Scan-Ready Design Conditions for Testability Rules	115

Scanshift	57
scan_wrap Report for the dumpBlackBox Rule	1667
scan_wrap Report for the Info_scanwrap Rule	1664
scan_wrap	1664
schematicForAllBits	1908
Searching an Instance	1931
Second Pass	99
Sensitizable	103
Sensitized	104
sequentialDepth	1909
ser_blackbox_control_0_probability	1907
ser_blackbox_observe_probability	1907
ser_control_sequential_depth	1904
ser_ignore_safety_mechanism_register_lreg	1905
ser_load_enable_n_cycle_persistence_adjustment	1905
ser_observe_nsr_initial_value	1906
ser_observe_sequential_depth	1906
ser_propagation_difference_threshold	1908
ser_register_report_top_contributors	1908
Setting Search Preferences	1932
SGDC Concepts	2029
SGDC-Based Flow	143
Shadow	70
Shift	57
Show or Hide Columns	1929
showAllReconvergence	1909
showPath	1910
showPowerGroundValue	1910
SoC Rules	1273
soc_06_rpt	1668
soft_error_metrics	1668
soft_error_registers_spm	1669
Sort Columns	1928
SpyGlass Design Constraints	2030
SpyGlass DFT ADV Constraints File Examples	55
SpyGlass DFT ADV Constraints Usage Features	48
SpyGlass DFT ADV Design Constraints	47
SpyGlass DFT ADV Rule Parameters	1765
stil_file	1670
stuck_at_coverage	1676
stuck_at_coverage_audit	1682

stuck_at_coverage_collapsed	1687
stuck_at_faults	1671
Suggested SpyGlass DFT ADV Operation	122
Support For Clock Shaper with Scannable Flip- Flops	95
Support for Multi-Bit Flip-Flop Cells	66
Synthesis Redundant (SR) Flip-Flops	64
Synthesis Redundant (SR)	92
tail_pipeline_clock_phase	1911
tail_pipeline_stages	1911
Tcl-Based User-Defined Macros (UDMs)	143
Test clocks	44
Test Compression Rules	1351
Test mode	37
Testability Analysis Rules	1296
Testability Analysis	111
Testmode Considerations for At-speed Testing	133
test_points_selected	1691
test_points_selected_2	1693
Tied (TI)	93
Toggling the instance view	1946
Topological	103
Topology Rules	1362
Trace to Clock in Atspeed	1939
Trace to Clock in Capture	1939
Trace to Clock in Shift	1937
Trace to Uncontrollable in Atspeed (After Launch)	1941
Trace to Uncontrollable in Atspeed (Before Launch)	1939
Trace to Uncontrollable Sources in Capture	1941
Trace to Unobservable Sources in Capture	1941
transition_coverage	1737
transition_coverage_audit	1742
transition_coverage_clockdomain	1746
transition_coverage_collapsed	1750
transition_faults	1752
Transparent	68
Tristate Rules	1431
Types of Faults	89
Types of Flip-Flops	63
Types of Latches	68
Typographical Conventions	33
Unblocked Path	1948

Understanding Back Annotation.....	1915
Understanding the structure of the hierarchy tree	1944
Un-Detectable (ND)	90
undetected_faults.....	1695
Unscannable Flip-Flops	64
Un-Testable (UT)	91
Un-Used (UU)	94
UPF Commands.....	2014
useFirstSource	1911
User-specified gating_cell SGDC command	75
Using AutoFix and Selective AutoFix.....	119
Using the Design Constraints	123
Viewing Flat Data Hierarchically.....	1943
Viewing Results in Fault Browser	1924
Viewing Undetectable Faults	1926
Working with Black Boxes.....	124
Working with Scan Chains	130
Working with SGDC Files	47

