

# **SpyGlass<sup>®</sup> Tcl Shell Interface**

## **User 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](http://www.synopsys.com)

## Boost Process

Project homepage: <http://www.highscore.de/boost/process0.5/index.html>

Project license:

Boris Schaeling

Copyright © 2006-2012 Julio M. Merino Vidal, Ilya Sokolov, Felipe Tanus, Jeff Flinn, Boris Schaeling

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE\_1\_0.txt or copy at [http://www.boost.org/LICENSE\\_1\\_0.txt](http://www.boost.org/LICENSE_1_0.txt))

### **Boost Software License - Version 1.0 - August 17th, 2003**

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## **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](mailto:spyglass_support@synopsys.com).

# Contents

---

<b>Preface</b> .....	<b>31</b>
<b>About This Book</b> .....	<b>31</b>
<b>Contents of This Book</b> .....	<b>32</b>
<b>Typographical Conventions</b> .....	<b>33</b>
<b>Using the Tcl Shell Interface</b> .....	<b>35</b>
<b>Project File</b> .....	<b>37</b>
<b>Invoking the Tcl Shell Interface</b> .....	<b>39</b>
Invoking Tcl Shell From Command-Line .....	39
Invoking Tcl Shell From SpyGlass.....	42
<b>Specifying Inputs to the sg_shell</b> .....	<b>44</b>
<b>Using sg_shell Commands</b> .....	<b>45</b>
Using Named and Positional Arguments .....	45
Properties of Tcl Command Arguments .....	46
Errors and Messages Flagged in sg_shell.....	50
Error Handling in Tcl Commands .....	51
Startup Files in sg_shell.....	52
Error Scenarios and Messages .....	53
Specifying a Tcl File as the Startup File .....	54
Specifying a Project File as the Startup File .....	56
Exit Codes Reported by sg_shell .....	58
<b>Features of sg_shell</b> .....	<b>62</b>
Using the Help Feature .....	62
Using the Tab Completion Feature .....	63
Capturing stdout and stderr .....	64
History Support in sg_shell .....	65
Command Logging in sg_shell .....	66
Screen Output Logging in sg_shell .....	67
Signal Handling in sg_shell .....	69
Using Escape Names in sg_shell .....	71
Common SDC Flow .....	73
Dual Design Read Flow .....	75
Using Key Combinations for Performing Actions .....	77
Setting SpyGlass Preferences Using Tcl Shell Interface.....	77

List of Preferences .....	77
Overriding GUI Preferences .....	81

## **SpyGlass Tcl Commands..... 83**

### **Session Commands..... 84**

<b>gui_set_obw_dialog_labels</b> : Sets OBW dialog labels.....	85
<b>gui_configure_obw_dialog</b> : Configures OBW dialog .....	86
<b>gui_save_session</b> : Saves the current SpyGlass session in a Tcl file .....	88
<b>gui_restore_session</b> : Restores the session given in a Tcl file.....	89
<b>new_project</b> : Creates a new project .....	90
<b>open_project</b> : Opens an existing project .....	92
<b>save_project</b> : Saves the specified project.....	95
<b>close_project</b> : Closes the currently active project.....	99
<b>current_project</b> : Displays data of the currently loaded project .....	101
<b>import_project</b> : Used to import existing project settings into current project .....	103
<b>exit</b> : Quits sg_shell and returns to the UNIX shell prompt .....	106
<b>set_pref</b> : Sets the specified preference variable to its specified value..	110
<b>get_pref</b> : Displays the value of the specified preference variable (s) ...	112

### **Design Setup Commands ..... 114**

<b>read_file</b> : Reads in the specified file for analysis .....	115
<b>get_file</b> : Displays the names of various types of files added .....	119
<b>remove_file</b> : Removes files of the specified type .....	122
<b>set_option</b> : Sets the specified option to the specified value.....	126
<b>get_option</b> : Get values set in the project for the specified option .....	141
<b>report_option</b> : Report options in tabular format .....	143
<b>remove_option</b> : Removes or unsets the specified option in the project scope .....	145
<b>link_design</b> : Reads the design to check design read errors .....	147
<b>compile_design</b> : Synthesizes the design to check synthesis errors ....	150
<b>read_power_data</b> : Provides the UPF files.....	154
<b>read_sdc_data</b> : Provides the SDC files.....	156
<b>read_activity_data</b> : Provides the activity data files, such as FSDB, VCD or SAIF files .....	158

### **Goal Setup or Run Commands ..... 161**

<b>current_methodology</b> : Selects a methodology .....	162
<b>addpolicy</b> : Adds policies to an existing goal.....	164

<b>current_goal</b>	: Selects a goal.....	166
<b>define_goal</b>	: Used to define custom goal in current selected methodology 172	
<b>define_regression</b>	: Used to define to define regression either by specifying a list of goals or by assigning a name to the goal list. 175	
<b>set_goal_option</b>	: Sets the specified goal option to the specified value	178
<b>get_goal_option</b>	: Get values in the goal scope for the specified option ... 186	
<b>get_messages</b>	: Returns a collection of message objects that have been reported for the goal run .....	188
<b>get_message_arg</b>	: Creates a list of message arguments for a message object(s) collection.....	190
<b>get_message_labels</b>	: Returns a string list of logical labels for message arguments of a message object(s) collection .....	195
<b>get_rules</b>	: Returns a collection of rule objects that were loaded for the goal run .....	197
<b>remove_goal_option</b>	: Removes or unsets the specified option in the goal scope .....	199
<b>get_run_option</b>	: Gets values for the current run.....	201
<b>set_run_option</b>	: Sets option/parameter to specified value.....	203
<b>set_parameter</b>	: Sets the value of the specified parameter.....	206
<b>get_parameter</b>	: Gets the value of the specified parameter set in the current goal .....	210
<b>report_parameter</b>	: Report parameters in tabular format.....	213
<b>run_goal</b>	: Runs the selected goal.....	215
<b>save_goal</b>	: Used to save design query data for the currently selected goal 229	
<b>restore_goal</b>	: Used to restore design query data for the currently selected goal .....	231
<b>ADC Setup Commands</b>	.....	<b>233</b>
ADC Commands	.....	233
SDC-Equivalent Commands	.....	240
<b>create_clock</b>	: Creates the clock for the design .....	243
<b>create_clock_attribute</b>	: Specifies the clock attributes.....	246
<b>create_generated_clock</b>	: Creates generated clocks.....	249
<b>define_sgdc_severity_class</b>	: Defines an SGDC severity class.....	251
<b>end_sgdc_severity_class</b>	: Marks the end of an SGDC severity class.	252
<b>set_annotated_transition</b>	: Sets the transition time at a given pin....	253

<b>set_case_analysis</b>	: Sets a constant logic value on a pin, port, or net.	255
<b>set_dft_signal</b>	: Specifies the DFT signal types for DRC and DFT insertion	258
<b>set_dont_touch_network</b>	: Sets the <i>dont_touch_network</i> attribute on clocks, pins, or ports in the current design to prevent cells and nets in the transitive fan-out of the <i>set_dont_touch_network</i> objects from being modified or replaced during optimization	261
<b>syn_set_dont_use</b>	: Sets the <i>dont_use</i> attribute on library cells to exclude them from the target library during optimization	263
<b>set_driving_cell</b>	: Sets attributes on input or inout ports of the current design, specifying that a library cell or output pin of a library cell drives the specified ports	265
<b>set_false_path</b>	: Removes timing constraints from particular paths	269
<b>set_ideal_network</b>	: Marks a set of ports or pins in the current design as sources of an ideal network. This disables the timing update and optimization of cells and nets in the transitive fan-out of the specified objects	272
<b>set_input_delay</b>	: Sets an input delay on the ports relative to a clock.	274
<b>set_load</b>	: Sets the load attribute to a specified value on the specified ports and nets	276
<b>set_multicycle_path</b>	: Modifies the single-cycle timing relationship of a constrained path	278
<b>set_output_delay</b>	: Sets an output delay on the ports relative to a clock..	282
<b>set_scan_group</b>	: Specifies an unordered group of cells that are not yet connected, but should be kept together within a scan chain. It also identifies the existing logic in the current design that is to be designated as a scan segment	284
<b>set_wire_load_mode</b>	: Sets the <i>wire_load_model_mode</i> attribute on the current design, specifying how wire load models are to be used to calculate the wire capacitance in nets	286
<b>set_wire_load_model</b>	: Sets the <i>wire_load_attach_name</i> attribute on designs, ports, hierarchical cells of current design, or the specified cluster of the current design, for selecting a wire load model to use in calculating the wire capacitance.	288
	Specifying Collection Objects in ADC Commands	290
<b>Utility Commands</b>		<b>291</b>
<b>get_adc</b>	: Used to get a list of ADC commands on the basis of filtering criteria, if specified.	292
<b>report_adc</b>	: Reports user-specified ADC commands	295



<b>remove_adc</b> : Used for removing constraint command .....	300
<b>save_adc</b> : Saves the active constraints .....	302
<b>convert_sgdc2adc</b> : Converts SGDC commands specified in an input file to corresponding ADC commands in the specified output file ...	303
<b>Reporting Commands</b> .....	<b>305</b>
<b>define_filter</b> : Defines a criterion to filter a set of messages from the set of all generated non-waived messages .....	306
<b>define_view</b> : Defines how the selected output should be displayed ....	311
<b>define_report</b> : Defines a new report of the specified name .....	316
<b>write_report</b> : Generates the specified report .....	318
<b>write_aggregate_report</b> : Used to generate the specified aggregate report .....	320
<b>Waiver Commands</b> .....	<b>322</b>
<b>waive</b> : Used for defining the criteria to waive a set of messages from the set of all generated messages .....	325
<b>get_waiver</b> : Used to get a list of waiver commands on the basis of filtering criteria, if specified .....	333
<b>report_waiver</b> : Reports user-specified waiver commands .....	335
<b>remove_waiver</b> : Used for removing waive commands .....	340
<b>save_waiver</b> : Saves the active waivers .....	342
<b>convert_sw12awl</b> : Converts waiver commands specified in an input file to corresponding AWL commands in the specified output file ...	343
<b>Debug Commands</b> .....	<b>345</b>
General Debug Commands .....	345
<b>gui_start</b> : Invokes Atrenta Console .....	346
<b>help</b> : Displays help for a particular command or item .....	347
<b>report_design_status</b> : Reports design status as whether design has been compiled or flattened, and whether these views are up to date . 350	
Design Query Commands .....	351
Library Commands .....	352
<b>get_libs</b> : Used to get a list of libraries currently loaded in sg_shell ....	354
<b>get_lib_cells</b> : Used to get a list of library cells currently loaded in sg_shell 357	
<b>get_lib_pins</b> : Used to get a list of library pins currently loaded in sg_shell 361	
<b>get_lib_timing_arcs</b> : Used to get a list of library timing arcs from the libraries currently loaded in sg_shell .....	364
Netlist Commands .....	369

<b>current_instance</b> : Used to select a scope (instance) for design query on hierarchically flattened netlist .....	374
<b>current_design</b> : Used to select a current design for interactive constraint and design query commands .....	377
<b>get_cells</b> : Creates a list of cells in the current design that match certain criteria .....	379
<b>get_nets</b> : Creates a list of nets in the current design that match certain criteria .....	383
<b>get_pins</b> : Creates a list of pins in the current design that match certain criteria .....	387
<b>get_ports</b> : Creates a list of ports in the current design that match certain criteria .....	391
<b>report_cell</b> : Used to display information and statistics about cells in the current instance or current design .....	395
<b>get_fanin_pins</b> : Creates a list of fan-in pins in the design that match certain criteria .....	397
<b>get_fanin_ports</b> : Creates a list of fan-in ports in the design that match certain criteria .....	401
<b>get_fanout_pins</b> : Creates a list of fan-out pins in the design that match certain criteria .....	403
<b>get_fanout_ports</b> : Creates a list of fan-out ports in the design that match certain criteria .....	407
<b>get_master</b> : Returns the master module of the specified instance.....	410
<b>get_parent</b> : Returns the parent node of the specified object .....	412
<b>get_clocks</b> : Creates a list of user-defined clocks in the current design	414
<b>get_clock_relation</b> : Returns a collection of clock objects for given names	416
<b>report_clock_relation</b> : Returns synchronous and asynchronous relationship between clocks of the design in matrix format ..	417
<b>report_clocks</b> : Reports properties of user-specified clocks in current design .....	419
<b>get_registers</b> : Used to get a list of cells driven by specified clocks/resets	422
<b>get_resets</b> : Creates a list of user defined resets in current design.....	425
<b>get_value</b> : Used to get simulation value of specified design object (port, pin, or net) in last cycle .....	427
<b>propagate_clocks</b> : Propagates the user-defined clocks .....	429
<b>propagate_resets</b> : Propagates the user-defined resets .....	430
<b>get_domains</b> : Creates a list of domains of the user-defined clocks in the	

current design .....	431
<b>report_domains</b> : Reports the list of clocks of the specified domains ..	433
<b>report_resets</b> : Reports properties of user specified resets in current design .....	435
Collection Commands .....	437
<b>add_to_collection</b> : Add objects to a base collection and form a new collection. The base collection remains unchanged .....	438
<b>append_to_collection</b> : Add objects to a collection, modifying the variable containing it .....	441
<b>compare_collections</b> : Used to compare two collections, returning 0 if they match .....	444
<b>filter_collection</b> : Used to filter a given base collection with some specific criteria .....	446
<b>foreach_in_collection</b> : Used to iterate over the objects of a collection ... 449	
<b>index_collection</b> : Used to extract a single object from a collection based on its index .....	451
<b>query_objects</b> : Used to display objects in the argument collection ....	453
<b>remove_from_collection</b> : Remove objects from a base collection and form a new collection .....	455
<b>sizeof_collection</b> : Used to determine the number of objects in a collection 458	
Attribute Commands .....	459
<b>define_user_attribute</b> : Used to define a new user-defined attribute .	461
<b>set_user_attribute</b> : Used to set a user attribute to a specified value on an object .....	463
<b>get_attribute</b> : Used to retrieve the value of an attribute on an object	465
<b>list_attributes</b> : Used to display a list of currently defined attributes...	467
<b>remove_user_attribute</b> : Used to remove attributes set with set_user_attribute command .....	471
<b>destroy_user_attribute</b> : Used to destroy an attribute .....	473
Product Commands .....	474
SpyGlass Base Commands .....	475
<b>get_combloop</b> : Creates a list of combinational loop in the current design that match certain criteria .....	476
SpyGlass Lint Turbo Commands .....	478
<b>get_lint_formal_results</b> : Gets a list of violations of lint turbo rules in the current design .....	479
<b>report_lint_formal_results</b> : Reports properties of violations of lint turbo	

rules in the current design .....	482
SpyGlass Constraints Commands .....	484
<b>autofix_sdc</b> : Generates SDC file having list of missing/incorrect constraints specified through the constraint rules .....	485
<b>get_constrained_muxes</b> : Used to get a list of MUXes where select pins are unconstrained and data pins are constrained .....	487
<b>get_sdc</b> : Used to get a list of SDC commands on the basis of filtering criteria, if specified.....	488
<b>write_sdc_node</b> : Used to print the SDC constraints for the given SDC nodes.....	490
<b>export_sdc</b> : Generates the SDC file from the user updated CSV .....	491
<b>update_crossing_file</b> : Translates the crossing file into user csv .....	493
SpyGlass CDC Commands.....	494
<b>get_cdc</b> : Creates a collection of clock domain crossings in the current design that match certain criteria .....	495
<b>get_cdc_coherency</b> : Returns the collection of Ac_conv issue based in field values .....	502
<b>get_cdc_glitch</b> : Creates collection of clock domain crossings in current design that may have glitches and match certain criteria.....	507
<b>get_cdc_sources</b> : Returns source of a crossing given by a destination name or an object returned from get_cdc collection.....	511
<b>get_conv_sync_signals</b> : Returns a collection of crossings for a given convergence object that is an element of collection returned from get_cdc_coherency .....	513
<b>get_glitch_sources</b> : Returns a collection of sources for a given glitch prone crossing or a destination .....	516
<b>get_multi_flop_sync_info</b> : Returns a collection of synchronizer flip-flops for a given crossing.....	518
<b>get_reset_sync</b> : Returns status of flip-flops with the reset synchronization issues in the current design that match certain criteria .....	520
<b>get_reset_sync_names</b> : Return the reset synchronizer information in the current design that match certain criteria .....	523
<b>get_paths</b> : Reports the complete paths between the specified start and end points.....	526
<b>report_cdc</b> : Reports clock domain crossing details .....	529
<b>report_cdc_coherency</b> : Displays the collection of coherency/convergence issues reported by get_cdc_coherency .....	532
<b>report_cdc_glitch</b> : Reports clock domain crossing with glitches .....	536
<b>report_paths</b> : Reports elements in a defined path in current design ...	538

<b>report_reset_sync</b> : Reports reset synchronization issues related information .....	540
<b>report_reset_sync_names</b> : Reports reset synchronization related information .....	543
SpyGlass DFT Commands .....	545
<b>dft_generate_coverage</b> : Generates coverage information for selected modules and instances .....	546
<b>dft_generate_fault_report</b> : Generates pin wise fault information for selected modules and instances .....	549
<b>dft_generate_scan_report</b> : Generates instance-wise scannability information of flip-flops or latches in the selected modules and scannability information for selected flip-flops or latch instances in the current design .....	552
<b>dft_generate_latch_status_report</b> : Generates instance-wise transparency information of latches in selected modules and transparency information for selected latch instances in current design .....	554
<b>cv_is_cmt_present</b> : Check if the given constraint_message_tag is present on the specified flat-object. ....	556
<b>dsm_assert_illegal_path</b> : Defines the illegal connectivity check for a path.....	558
<b>dsm_assert_illegal_value</b> : Defines a check that a logic value should not be present on a design node .....	560
<b>cv_define_user_macro</b> : Define a new user marco.....	562
<b>cv_delete_user_macro</b> : Delete a user macro. ....	563
<b>cv_reset_user_macros</b> : Delete all user macros.....	564
<b>cv_get_list_of_user_macros</b> : Get list of user macros. ....	565
<b>cv_add_element_to_user_macro</b> : Add flat-object to user macro's collection.....	566
<b>cv_get_cell_list_of_user_macro</b> : Get cell list of user macro.....	568
<b>cv_get_pin_list_of_user_macro</b> : Get pin list of user macro. ....	569
<b>cv_get_port_list_of_user_macro</b> : Get port list of user macro. ....	570
<b>cv_is_element_present_in_user_macro</b> : Check if flat-object is present in the user macro's collection. ....	571
<b>cv_remove_element_from_user_macro</b> : Remove flat-object from user macro's collection. ....	573
SpyGlass Power Verify Commands .....	575
<b>get_pwr_intent</b> : Gives a power intent node for a design element on which user can query information .....	576

<b>report_pwr_intent</b> : Displays the information of the power intent node ...	577
<b>check_pwr_intent_crossing</b> : Displays the crossing type between the two power intent nodes .....	579
<b>get_retention_info</b> : Gives a power retention node containing information related to retention strategy applied on an instance.....	581
<b>get_isolation_info</b> : Gives a power isolation node containing information related to isolation strategy applied on an instance.....	583
<b>get_power_switch_info</b> : Gives a power PSW node containing information related to create_power_switch strategy applied on an instance	585
<b>get_level_shifter_info</b> : Gives a power level shifter node containing information related to level shifter strategy applied on an instance.....	587
<b>get_supply_info</b> : Gives a power supply node containing information related to power supply corresponding to a design net .....	589
<b>report_retention_info</b> : Displays the information of the power retention node .....	590
<b>report_isolation_info</b> : Displays the information of the power isolation node .....	592
<b>report_power_switch_info</b> : Displays the information of the power PSW node .....	594
<b>report_level_shifter_info</b> : Displays the information of the power level shifter node .....	596
<b>report_supply_info</b> : Displays the information of the power supply node .	598
SpyGlass Power Estimate and Reduce Commands .....	600
<b>report_power_stats_for_cell</b> : Generates power attributes report for the given flat cells .....	601
<b>report_power_stats_for_reg</b> : Generates a report for given flat cells of register type with required attributes.....	604
Built-in Attributes .....	608
lib.....	610
lib_cell .....	612
lib_pin.....	614
lib_timing_arcs.....	617
cdc_conv_signal_node.....	618
cdc_conv_node.....	618
cdc_glitch_node.....	619
cdc_glitch_source_node.....	620

cdc_node.....	621
cdc_source_node.....	622
design.....	623
du_cell.....	625
du_pin.....	628
du_port.....	630
du_net.....	631
flat_inst.....	632
flat_cell.....	633
flat_pin.....	636
flat_port.....	640
flat_net.....	642
adc_node.....	644
sdn_node.....	645
clock.....	645
clock_domain.....	645
message.....	646
rule.....	646
reset.....	647
reset_flop_node.....	647
reset_sync_node.....	648
Product Attributes.....	649
Base Attributes.....	650
<b>is_async_sync_reset</b> : Returns the reset net used as both synchronously and asynchronously.....	652
<b>is_clock_used_as_nonclock</b> : Returns the flip-flop clock signal net which is used as non clock signal in a design.....	653
<b>is_clock_used_with_both_edges</b> : Returns the clock signal driving on both edges.....	654
<b>is_constant_pin</b> : Returns the pin of an instance at which a constant value reaches.....	655
<b>is_disabled_cell</b> : Returns the disabled gate.....	657
<b>is_internally_generated_reset</b> : Returns the internally generated reset. 658	
<b>is_latch_clock_driven_on_both_edges</b> : Returns the clock net trigger latches.....	659
<b>is_multiple_driver</b> : Returns the flattened net that has multiple drivers... 660	
<b>is_reset_used_as_nonreset</b> : Returns the asynchronous reset or preset net which is used as non asynchronous reset or preset signal ...	

	662
<b>is_reset_used_with_both_polarity</b> : Returns the reset or set net which is used as both positive and negative polarity in same design unit	663
<b>is_unregistered_port</b> : Returns the ports, which are not registered, of the module	664
CDC Attributes	666
<b>dest_type</b> : Checks the type of destination instance present in a crossing	668
<b>failure_reason</b> : Reports the reason of unsynchronization of a CDC crossing	669
<b>is_comb_conv</b> : Checks whether the converging signals are reported by the Ac_conv02 rule	670
<b>is_data</b> : Checks whether a CDC crossing is a data crossing	671
<b>is_graycoded</b> : Checks whether the converging signals are gray encoded	672
<b>is_nonconv_bus</b> : Checks whether the converging signals are reported by the Ac_conv04 rule	674
<b>is_seq_conv</b> : Checks whether the converging signals are reported by the Ac_conv01 rule	675
<b>is_synchronized</b> : Checks whether a CDC crossing is synchronized	676
<b>is_user_defined</b> : Checks whether the converging signals are reported by the Ac_conv05 rule	677
<b>num_sources</b> : Checks number of sources present in a crossing	678
<b>num_source_domains</b> : Checks number of source domains present in a crossing	679
<b>src_type</b> : Checks the type of source instance present in a crossing	680
<b>sync_method</b> : Reports the reason of synchronization of a CDC crossing	681
Constraints Attributes	682
<b>sdc_type</b> : Returns the SDC constraint type of the SDC node	683
<b>timing_state</b> : Returns the constrained status of the queried design object	684
DFT Attributes	686
<b>atspeed_sim_value</b> : Gets atspeed_capture mode simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design	689
<b>capture_sim_value</b> : Gets capture mode simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in	



current design.....	691
<b>get_at-speed_clock_n_phase</b> : Gets the at-speed mode source clock, source clock phase and phase at a user specified design node in current design.....	693
<b>get_capture_clock_n_phase</b> : Gets the capture mode source clock, source clock phase and phase at a user specified design node in the current design .....	695
<b>get_dft_functional_clock_n_phase</b> : Gets the functional source clock, source clock phase and phase at a user specified design node in current design.....	697
<b>get_latch_at-speed_status</b> : Gets the at-speed transparency status for user specified latch in current design.....	699
<b>get_latch_capture_status</b> : Gets the capture transparency status for user specified latch in current design.....	701
<b>get_latch_shift_status</b> : Gets the shift transparency status for user specified latch in current design.....	703
<b>get_scan_status</b> : Gets the scannability status for user specified flip-flop or latch in current design.....	705
<b>get_shift_clock_n_phase</b> : gets the shift mode source clock, source clock phase and phase at a user specified design node in current design	707
<b>is_scannable</b> : Checks whether a flip-flop or a latch is scannable.....	708
<b>obs_probability</b> : Gets the probability that the flat port, pin or net is observable when random test pattern is applied.....	709
<b>one_cnt_probability</b> : Gets the probability that the flat port, pin or net is at 1 when random test pattern is applied .....	711
<b>pg_sim_value</b> : Gets power_ground simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design.....	713
<b>rand_fault_cov_estimate</b> : Gets the fault coverage estimate of top module for the dft_pattern_count random test patterns .....	715
<b>rand_test_cov_estimate</b> : Gets the test coverage estimate of top module for dft_pattern_count random test patterns.....	716
<b>sa0_det_probability</b> : Gets the probability that stuck at 0 fault on flat port, pin or net is detected after dft_pattern_count random test patterns are applied .....	717
<b>sa1_det_probability</b> : Gets the probability that stuck at 1 fault on flat port, pin or net is detected after dft_pattern_count random test patterns are applied .....	718
<b>sa0_fault_detectability</b> : Gets the stuck_at zero fault detectability ...	719

<b>sa1_fault_detectability</b> : Gets the stuck_at one fault detectability.....	720
<b>shift_sim_value</b> : Gets shift mode simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design .....	721
<b>static_controllability</b> : Gets static controllability (3-bit-string (y/n): zero-control_one-control_zee-control: nnn   nny   nyn   nyy   ynn   yny   yyn   yyy) for user-specified design node (flat_net   flat_pin   flat_port) in current design .....	723
<b>static_observability</b> : Gets static observability (y (yes_observable)   n (not_observable)) for user-specified design node (flat_pin   flat_port) in current design.....	725
<b>t01_fault_detectability_los</b> : Gets zero to one transition fault detectability with launch on shift .....	727
<b>t10_fault_detectability_los</b> : Gets one to zero transition fault detectability with launch on shift .....	728
<b>t01_fault_detectability_loc</b> : Gets zero to one transition fault detectability with launch on capture.....	729
<b>t10_fault_detectability_loc</b> : Gets one to zero transition fault detectability with launch on capture.....	731
<b>zero_cnt_probability</b> : Gets the probability that the flat port, pin, or net is at 0 when random test pattern is applied .....	733
Power Attributes .....	735
<b>activity</b> : Returns the activity of a flat net .....	739
<b>blackbox_internal_power</b> : Returns the total internal power consumed by all the black box cells of a hierarchy .....	741
<b>blackbox_leakage_power</b> : Returns the total leakage power consumed by all the black box cells of a hierarchy .....	743
<b>blackbox_switching_power</b> : Returns the total switching power consumed by all the black box cells of a hierarchy .....	745
<b>capacitance_source</b> : Returns the source of the capacitance of a flat net .	747
<b>cell_size_for_power</b> : Returns the relative size of a flat cell as used for set_cell_allocation.....	749
<b>clock_internal_power</b> : Returns the total internal power consumed by all the clock cells of a hierarchy.....	750
<b>clock_leakage_power</b> : Returns the total leakage power consumed by all the clock cells of a hierarchy.....	752
<b>clock_switching_power</b> : Returns the total switching power consumed by all the clock cells of a hierarchy .....	754

<b>combinational_internal_power</b> : Returns the total internal power consumed by all the combinational cells of a hierarchy.....	756
<b>combinational_leakage_power</b> : Returns the total leakage power consumed by all the combinational cells of a hierarchy.....	758
<b>combinational_switching_power</b> : Returns the total switching power consumed by all the combinational cells of a hierarchy.....	760
<b>fanout_capacitance</b> : Returns the total pin capacitance of a given net	762
<b>internal_power</b> : Returns total internal power consumed by the given flat cell or hierarchical cell .....	764
<b>io_internal_power</b> : Returns the total internal power consumed by all the io cells of a hierarchy .....	766
<b>io_leakage_power</b> : Returns the total leakage power consumed by all the io cells of a hierarchy .....	768
<b>io_switching_power</b> : Returns the total switching power consumed by all the io cells of a hierarchy .....	770
<b>is_activity_annotated</b> : Returns a boolean value, as an annotation status of a given net.....	772
<b>is_clock_gated</b> : Returns the gating status for the given flat cell.....	774
<b>is_internal_power_defined</b> : Returns a boolean value, based on whether internal power tables are specified for a library cell or not ...	775
<b>is_instantiated</b> : Returns a boolean value as instantiation status for the given flat cell .....	776
<b>leakage_power</b> : Returns total leakage power consumed by the given flat cell or hierarchical cell .....	778
<b>leakage_power_model</b> : Returns leakage power model of a library cell... 780	
<b>megacell_internal_power</b> : Returns the total internal power consumed by all the megacell cells of a hierarchy .....	782
<b>megacell_leakage_power</b> : Returns the total leakage power consumed by all the megacell cells of a hierarchy .....	784
<b>megacell_switching_power</b> : Returns the total switching power consumed by all the megacell cells of a hierarchy.....	786
<b>memory_internal_power</b> : Returns the total internal power consumed by all the memory cells of a hierarchy.....	788
<b>memory_leakage_power</b> : Returns the total leakage power consumed by all the memory cells of a hierarchy.....	790
<b>memory_switching_power</b> : Returns the total switching power consumed by all the memory cells of a hierarchy .....	792
<b>net_frequency</b> : Returns the frequency of the flat net.....	794

<b>other_internal_power</b> : Returns the total internal power consumed by all those cells of a hierarchy that do not fall into any standard cell category .....	796
<b>other_leakage_power</b> : Returns the total leakage power consumed by all those cells of a hierarchy that do not fall into any standard cell category .....	798
<b>other_switching_power</b> : Returns the total switching power consumed by all those cells of a hierarchy that do not fall into any standard cell category .....	800
<b>power_type</b> : Returns the category of power to which this flat cell is contributing .....	802
<b>net_capacitance</b> : Returns the wire capacitance of a flat net .....	804
<b>probability</b> : Returns the probability of a flat net .....	806
<b>root_clock_for_power</b> : Returns the root clock name for the given register flat cell .....	808
<b>sequential_internal_power</b> : Returns the total internal power consumed by all the sequential cells of a hierarchy .....	809
<b>sequential_leakage_power</b> : Returns the total leakage power consumed by all the sequential cells of a hierarchy .....	811
<b>sequential_switching_power</b> : Returns the total switching power consumed by all the sequential cells of a hierarchy .....	813
<b>switching_power</b> : Returns total switching power consumed by the given flat cell or hierarchical cell .....	815
<b>virtual_buffer_info</b> : Returns virtual buffer information for the given flat net.....	817
<b>virtual_internal_power</b> : Returns total internal power consumed by all virtual buffers on a given flat net .....	819
<b>virtual_leakage_power</b> : Returns total leakage power consumed by all virtual buffers on the given flat net.....	821
<b>virtual_switching_power</b> : Returns total switching power consumed by all virtual buffers on the given flat net.....	823
<b>vt_classification</b> : Returns threshold voltage group of a library cell ....	825
Power Verify Attributes .....	827
<b>clamp_value</b> : Displays the clamp value of the isolation strategy .....	829
<b>control_port</b> : Displays the control signal of power switch.....	830
<b>input_supply_port</b> : Displays the input supply of power switch.....	831
<b>input_supply_set</b> : Displays the input supply set of level shifter .....	832
<b>isolation_ground_net</b> : Displays the isolation ground net of the isolation strategy.....	833

<b>isolation_power_net</b> : Displays the isolation power supply of the isolation strategy .....	834
<b>isolation_sense</b> : Displays the isolation sense of the isolation strategy	835
<b>isolation_signal</b> : Displays the isolation signal of the isolation strategy	836
<b>location</b> : Displays the location of the applied strategy .....	837
<b>name</b> : Displays the name of the strategy .....	839
<b>output_supply_port</b> : Displays the input supply of power switch .....	841
<b>output_supply_set</b> : Displays the input supply set of level shifter .....	842
<b>power_domain</b> : Displays the power domain name .....	843
<b>power_supply</b> : Displays the power supply name .....	844
<b>ground_supply</b> : Displays the ground supply name .....	845
<b>restore_signal</b> : Displays the restore signal of the retention strategy..	846
<b>retention_ground_supply</b> : Displays the retention ground supply of the retention strategy .....	847
<b>retention_power_supply</b> : Displays the retention power supply of the retention strategy .....	848
<b>save_signal</b> : Displays the save signal of the retention strategy .....	849
<b>sink</b> : Displays the sink supply of an applied strategy .....	850
<b>source</b> : Displays the source supply of an applied strategy .....	851
<b>supply_name</b> : Displays the supply name corresponding to a design net ..	853
<b>rule</b> : Displays the rule of level shifter strategy .....	854
<b>type</b> : Displays the type of supply net .....	855
<b>voltage_range_min</b> : Returns the minimum value to voltage range ..	856
<b>voltage_range_max</b> : Returns the maximum value to voltage range..	857
<b>Miscellaneous Commands</b> .....	<b>858</b>
<b>alias</b> : Creates an alias for a group of word(s) .....	859
<b>benchmark</b> : Monitors run-time and memory usage between two designated check points in sg_shell .....	861
<b>capture</b> : Captures output (stdout or stderr) of script to a file .....	864
<b>gui_set_preference</b> : Specifies SpyGlass preferences using Tcl Shell..	867
<b>gui_add_menu</b> : Creates a new toolbar menu item and returns the Id	869
<b>show_error</b> : Displays the last error that occurred during a particular command invocation along with its trace .....	871
<b>source</b> : <b>unalias</b> : Removes an alias set for a group of word(s) .....	874

<b>Appendix A: Deprecated Command Names and Their Corresponding New Commands .....</b>	<b>875</b>
<b>Appendix B: Application Attributes .....</b>	<b>877</b>
List of Built-in Attributes .....	878
List of Product Attributes .....	891
<b>Appendix C: SpyGlass Report Names.....</b>	<b>899</b>
General Reports .....	901
Custom Reports.....	902
Default Reports .....	903
SpyGlass area Reports.....	904
SpyGlass audits Reports .....	905
SpyGlass lint Reports .....	906
SpyGlass morelint Reports .....	907
SpyGlass OpenMore Reports.....	908
SpyGlass STARC Reports .....	909
SpyGlass STARC02 Reports .....	910
SpyGlass STARC05 Reports .....	911
SpyGlass CDC Reports .....	912
SpyGlass Constraints Reports.....	914
SpyGlass DFT Reports .....	916
SpyGlass DFT DSM Reports.....	918
SpyGlass Power Family Reports .....	920
SpyGlass Power Verify Reports .....	922
SpyGlass TXV Reports .....	924
<b>Appendix D: Preference Variables Supported by the set_pref Command.....</b>	<b>925</b>
Overview.....	925
<b>sh_command_log_file</b> : Specifies the path name for sg_shell's command log file.....	926
<b>goal_show_hidden</b> : Enables visibility of hidden goals of methodology	930
<b>goal_enforce_prerequisite</b> : Enforces prerequisite goal run before goal run .....	932

<b>dq_design_view_type</b> : Specifies the view on which design query commands are executed .....	934
<b>collection_display_limit</b> : Specifies the maximum number of objects that can be displayed by any command that returns a displayable collection .....	937

**Appendix E: CDC Application Commands .....939**

List of CDC Commands .....	940
----------------------------	-----

**Appendix:**

**SpyGlass Design Constraints .....953**

Writing Constraints in an SGDC File .....	954
Specifying SGDC File to SpyGlass .....	955
Working with SGDC Files .....	956
Handling of Duplicate Constraint Specifications .....	959
Renamed Constraints .....	960
<b>SpyGlass Design Constraints .....</b>	<b>963</b>
abstract_block_violation .....	967
abstract_file.....	969
abstract_interface_param .....	972
abstract_interface_port .....	974
abstract_port .....	976
activity.....	1000
activity_data.....	1006
add_fault.....	1010
allow_combo_logic.....	1017
allow_test_point.....	1020
always_on_buffer .....	1020
always_on_cell.....	1022
always_on_pin .....	1023
always_on_path .....	1024
antenna_cell .....	1025
aon_buffered_signals .....	1026
assertion_signal .....	1027
associate_lib .....	1029
assume_waveform.....	1030
assume_path .....	1031
atspeed_clock_frequency.....	1034

balanced_clock .....	1036
blackbox_power .....	1038
block .....	1041
blocksize .....	1044
breakpoint .....	1046
bypass .....	1047
cdc_attribute .....	1048
cdc_check_glitch .....	1053
cdc_define_transition .....	1053
reset_sense .....	1054
cdc_false_path .....	1055
cdc_filter_coherency .....	1068
cdc_filter_path .....	1072
cdc_matrix_attributes .....	1073
cell_hookup .....	1076
cell_pin_info .....	1077
cell_tie_class .....	1078
clock .....	1080
For SpyGlass CDC solution, SpyGlass Constraints solution, and SpyGlass Auto Verify solution .....	1081
For SpyGlass DFT solution, SpyGlass DFT DSM solution .....	1087
For SpyGlass Power Verify solution, SpyGlass ERC Product, and SpyGlass Power Estimation and SpyGlass Power Reduction solutions .....	1093
clock_buffer .....	1095
clock_group .....	1096
clock_path_wrapper_module .....	1100
clock_pin .....	1102
clock_root .....	1103
clock_sense .....	1104
clock_shaper .....	1105
clockgating .....	1124
complex_cell .....	1125
compressor .....	1127
dbist .....	1128
decompressor .....	1129
define_clock_tree .....	1131
define_illegal_input_values .....	1137
define_legal_input_values .....	1138
define_library_group .....	1140
define_macro .....	1141
define_reset_order .....	1146



define_tag .....	1150
For SpyGlass Auto Verify solution and SpyGlass CDC solution .....	1152
For SpyGlass DFT solution and SpyGlass DFT DSM solution .....	1154
delay_buffer .....	1162
deltacheck_ignore_instance .....	1164
deltacheck_ignore_module .....	1165
deltacheck_start .....	1166
deltacheck_stop_instance .....	1168
deltacheck_stop_module .....	1169
deltacheck_stop_signal .....	1171
design_map_info .....	1172
dftmax_partition .....	1175
dft_report_fault_list .....	1178
dft_stitching_exception .....	1179
dft_report_coverage .....	1181
disable_timing .....	1182
disallow_modification_type .....	1183
disallow_upf_command .....	1185
domain .....	1186
domain_inputs .....	1188
domain_outputs .....	1189
domain_signal .....	1191
dont_touch .....	1193
expect_frequency .....	1195
false_path .....	1198
fifo .....	1199
force_no_scan .....	1201
force_ta .....	1206
force_probability .....	1209
formal_analysis_filter .....	1211
fsm .....	1213
gating_cell .....	1216
gating_cell_enable .....	1222
generated_clock .....	1223
glitch_free_module .....	1228
gray_signals .....	1229
ignore_clock_gating .....	1230
ignore_crossing .....	1232
ignore_supply_pin .....	1234
illegal_constraint_message_tag .....	1235
illegal_path .....	1238

illegal_value.....	1248
initialize_for_bist .....	1259
initstate.....	1259
input.....	1262
input_drive_strength.....	1265
input_isocell.....	1267
instance_trace.....	1269
ip_block .....	1271
For SpyGlass CDC solution and SpyGlass Auto Verify Solution .....	1272
For SpyGlass DFT solution and SpyGlass DFT DSM solution.....	1273
isolation_cell .....	1275
isolation_wrapper .....	1278
keeper .....	1279
latched_port .....	1281
levelshifter.....	1282
lp_ignore_cells_for_erc.....	1284
make_mandatory_upf_commands_options .....	1285
mapped_pin_map .....	1286
mcp_info .....	1288
memory .....	1298
memory_force.....	1299
memory_port .....	1300
memory_inst_port .....	1306
memory_read_pin .....	1307
memory_tristate.....	1309
memory_type.....	1310
memory_write_disable .....	1312
memory_write_pin.....	1313
meta_design_hier .....	1315
meta_inst .....	1316
meta_module.....	1317
meta_monitor_options .....	1319
mode_condition.....	1321
module_bypass .....	1323
module_pin.....	1325
monitor_time .....	1327
multivt_lib .....	1329
network_allowed_cells.....	1331
no_atspeed.....	1333
no_convergence_check.....	1337
no_fault .....	1338

no_test_point .....	1344
noclockcell_start.....	1345
noclockcell_stop_instance.....	1346
noclockcell_stop_module .....	1347
noclockcell_stop_signal .....	1349
non_pd_inputcells .....	1350
num_flops .....	1351
operating_mode_set .....	1356
Example.....	1357
output.....	1358
output_not_used .....	1360
pg_cell.....	1362
pg_pins_naming.....	1364
pin_voltage.....	1365
pll .....	1369
port_time_delay .....	1370
power_data .....	1373
power_down .....	1374
power_down_sequence .....	1376
power_management_test_control_cell.....	1378
power_management_unit .....	1379
power_rail_mapping .....	1380
power_state .....	1382
power_switch.....	1384
pr_safe_clocks .....	1386
pulldown .....	1388
pullup .....	1390
qualifier .....	1391
quasi_static .....	1404
For SpyGlass TXV solution .....	1404
For SpyGlass CDC solution .....	1406
For SpyGlass Power Family.....	1410
quasi_static_style.....	1411
ram_instance .....	1414
ram_switch.....	1415
rdc_false_path .....	1418
ref_power_data .....	1421
reference_toplevel_isolation_signal .....	1423
repeater.....	1424
Purpose.....	1424
repeater_buffer .....	1425

require_constraint_message_tag .....	1430
Syntax .....	1430
require_path .....	1433
require_pulse .....	1445
require_stable_value .....	1448
require_strict_path .....	1451
require_structure .....	1458
require_value .....	1461
reset .....	1472
reset -async .....	1474
reset_filter_path .....	1476
reset_pin .....	1481
reset_synchronizer .....	1482
retention_cell .....	1484
retention_instance .....	1488
rme_config .....	1489
set_slew .....	1495
force_scan .....	1498
force_stable_value .....	1501
force_unstable_value .....	1503
scan_cell .....	1506
scan_chain .....	1507
scan_enable_source .....	1511
scan_ratio .....	1512
scan_type .....	1513
scan_wrap .....	1516
sdc_data .....	1520
For the SpyGlass Power Estimation and SpyGlass Power Reduction Solutions	
1520	
For SpyGlass Constraints solution .....	1521
For SpyGlass DFT DSM solution .....	1525
validation_filter_path .....	1525
For SpyGlass CDC Solution .....	1527
select_wireload_model .....	1527
seq_atpg .....	1531
set .....	1532
set_case_analysis .....	1533
enable_seq_propagation .....	1546
set_cell_allocation .....	1547
set_cell_name_pattern .....	1553
set_clock_gating_type .....	1559

set_fully_decoded_bus .....	1562
set_mega_cell.....	1563
set_power_info .....	1564
stil_data .....	1565
sg_multicycle_path .....	1566
syn_set_dont_use .....	1568
ignore_nets .....	1570
ser_data .....	1572
safety_related.....	1573
non_safety_related .....	1574
set_lib_timing_mode.....	1576
set_lib_name .....	1577
set_monitor_cell.....	1578
set_pin .....	1581
set_power_scaling .....	1582
set_supply_node .....	1586
sg_clock_group .....	1587
sgdc .....	1588
For SpyGlass CDC Solution.....	1589
For All Products .....	1591
shadow_ratio .....	1592
show_power_calc_details.....	1593
signal_in_domain .....	1594
signal_type.....	1597
Control Signal .....	1597
Data Signal.....	1599
simulation_data .....	1600
special_cell .....	1604
special_module .....	1605
spef_data .....	1606
supply .....	1609
For SpyGlass Power Verify solution.....	1609
For the SpyGlass Power Estimation and SpyGlass Power Reduction Solutions	
1612	
switchoff_wrapper_instance .....	1613
sync_cell .....	1615
sync_reset_style .....	1620
test_mode .....	1623
For SpyGlass DFT solution and SpyGlass DFT DSM solution .....	1623
For SpyGlass STARC Product, SpyGlass STARC02 product, SpyGlass	
STARC05 product, and SpyGlass STARCad-21 product .....	1640

test_point.....	1641
tie_x.....	1643
tristate_cell.....	1645
ungroup_cells.....	1646
use_library_group.....	1649
voltage_domain.....	1651
For SpyGlass DFT DSM Solution.....	1651
For SpyGlass Power Verify solution.....	1653
For SpyGlass Power Estimation and SpyGlass Power Reduction Solutions....	1666
vt_mix_percentage.....	1670
watchpoint.....	1674
wireload_selection.....	1676

---

# Preface

---

## About This Book

The SpyGlass® Tcl Shell Interface User Guide describes the Tcl Shell Interface and various Tcl commands.

## Contents of This Book

The SpyGlass Tcl Shell Interface User Guide consists of the following chapters:

<b>Chapter</b>	<b>Describes...</b>
<i>Using the Tcl Shell Interface</i>	Information on the usage of Tcl Shell Interface
<i>SpyGlass Tcl Commands</i>	Details of various Tcl commands
<i>Appendix A: Deprecated Command Names and Their Corresponding New Commands</i>	Deprecated command names and their corresponding new commands
<i>Appendix B: Application Attributes</i>	Detailed description of application attributes
<i>Appendix C: SpyGlass Report Names</i>	Valid report names of each product that can be used with the <report-names> argument
<i>Appendix D: Preference Variables Supported by the set_pref Command</i>	Preference variables supported by the <i>set_pref</i> command
<i>Appendix E: CDC Application Commands</i>	List of CDC application commands



## 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	<b>NOTE:</b> 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



---

# Using the Tcl Shell Interface

---

The Tool Command Language (Tcl) interface of SpyGlass<sup>®</sup> enables you to interactively control the flow of various stages during a design analysis session. Use this interface to perform a variety of operations, such as compiling a design, applying constraints, configuring tool settings, controlling flow, and/or customizing the reports. The Tcl interface provides the ability to create scripts for your CAD environment, perform "what-if" analysis, perform interactive debugging, and facilitate the organization of complex flow into smaller, logical steps for easy execution.

A typical design analysis session can be divided into the following stages:

1. *Design Setup* stage

During this stage, you set up the design by adding the design files, precompiled files, and technology files. In addition, you specify various design read options that are used during the SpyGlass run. You can read the design at this stage, fix any issues reported during design read, and identify and fix black boxes. This ensures that the basic design read is clean, before proceeding with the detailed analysis.

2. *Goal Setup* stage

During this stage, you select various goals. Next, you configure their parameter values, specify constraints, and ensure that the goal setup is complete. You can then run the selected goals and move to the subsequent stage of detailed design debug based on the information

generated at this stage.

### 3. *Analyze Results* stage

During this stage, you can perform various operations, such as debug violations, view results in a schematic, and cross-probe issues to HDL.

The Tcl interface of SpyGlass, or `sg_shell`, provides an interactive shell environment in which you can control various steps of each of the above stages by using various Tcl commands. This interface enables you to perform easy design read, goal setup, goal execution, and customized reporting to focus on intended violations.

**NOTE:** *The Tcl interface of SpyGlass, or `sg_shell`, supports Tcl 8.5. Therefore, `sg_shell` can invoke all the commands that are a part of Tcl 8.5. To view the help of Tcl 8.5 commands, use the `man <tcl-command-name>` command. For example, `man llength`.*

## Project File

As part of various stages during a design debug session within SpyGlass Tcl Shell Interface, or `sg_shell`, you work on a project. `sg_shell` requires you to define a project that captures all the design and goal setup information. As you update the design data or goal setup information, such as constraints and parameters, the project holds the latest setup information. For example, if you start a project with two design files (say, `top.v` and `mid.v`), and then add another design file (say, `lower.v`), but later remove `mid.v`, the project would finally store `top.v` and `lower.v`.

Project is a storehouse of the final setup details and allows you to quickly start your next `sg_shell` session from the point where you left it in the earlier debug session. The project file is a Tcl file with appropriate Tcl commands capturing information about design setup and goal setup till the point last saved by you.

There is already a notion of project in Atrienta Console, and you are expected to debug a session within an active project, and the same applies to `sg_shell`. There may be some commands that have changed from an earlier release, so it is recommended that you open the existing project file in `sg_shell`, and immediately save it to get the updated project file with new command names. Refer to the [Appendix A: Deprecated Command Names and Their Corresponding New Commands](#) to see the old versus new command mapping.

If there are any errors in your project file that have old command names, `sg_shell` reports the new command names in the error messages instead of the old ones.

It is recommended that you open a project in `sg_shell`, and then update the setup information, instead of directly updating it in the project file. This will help you to catch any errors in the setup in an interactive manner.

Other general information about the project file is listed below:

- It is always better to save a project file with the `.prj` extension, if manually created, to easily identify it. If the project is created inside `sg_shell`, it is automatically assigned this extension.
- There is some basic information that already exists in the project file, such as `projectcwd`, which is used internally by the software. Please ensure that you do not tweak this data while manually editing this file.

- There are Tcl commands to start a new project, open an existing project, save or close a project, or query the current project.
- If there are any environment variables used while manually editing a project file, those environment variables are lost when the project is next saved. The project file stores the final setup information. Any Tcl-specific substitution or constructs are interpreted and lost while loading the project itself. Therefore, `sg_shell` does not preserve any Tcl-specific substitutions or constructs while saving the project.

## Invoking the Tcl Shell Interface

You can invoke the Tcl Shell interface using any of the following methods:

- [Invoking Tcl Shell From Command-Line](#)
- [Invoking Tcl Shell From SpyGlass](#)

## Invoking Tcl Shell From Command-Line

You can invoke the interactive Tcl shell interface by specifying the following command at the command-line:

```
% sg_shell
```

**NOTE:** `sg_shell` is picked from the same location as `SpyGlass`, that is `$SPYGLASS_HOME/bin`.

When you specify the above command, SpyGlass displays the `sg_shell` shell prompt, as shown below:

```
+-----+
|
| SpyGlass Predictive Analyzer(R)- Version 5.6.1
|
| Last compiled on Mar 15 2016
|
| All Rights Reserved. Use, disclosure or duplication
| without prior written permission of Synopsys Inc.is
| prohibited.
| Technical support: email spyglass_support@synopsys.com.
+-----+
-----+
```

```
sg_shell>
```

Once the above shell prompt appears, you can start the required operations by using various Tcl commands. As you start typing the Tcl commands, `sg_shell` records those commands in a separate Tcl file, `sg_shell_command.log`, in the current working directory, which you can use later for regression purpose. If a Tcl file of the same name, `sg_shell_command.log`, already exists, `sg_shell` overwrites it.

You can perform the following tasks while invoking Tcl Shell Interface:

- *Specifying a Tcl File*
- *Specifying a Project File*
- *Specifying the Tcl Startup File*
- *Specifying Customized Paths*

### **Specifying a Tcl File**

You can specify a Tcl file (`.tcl`) while invoking the Tcl Shell interface in the following manner:

```
sg_shell < my.tcl
```

This Tcl file may contain the Tcl commands that are prerecorded in a previous `sg_shell` run or are written manually. When you specify a Tcl file, SpyGlass runs all the Tcl commands specified in the file, and then returns to the UNIX prompt. The Tcl file can have any name or extension, but the commands inside it should be Tcl-compatible and supported inside `sg_shell`.

The output of the invocation mentioned above is the same as when you type each of the commands mentioned in `my.tcl` directly on the `sg_shell` prompt. If there is an error in any of the commands, Tcl shell ignores that command, and the execution continues with subsequent commands.

This mode is best suited for regression purposes, where you have manually verified results with a given sequence of Tcl commands, and can then capture these commands in a playback file for regression purposes.

### **Specifying a Project File**

You can specify a project file (`.prj`) while invoking the Tcl shell interface in



the following manner:

```
sg_shell -project myProject.prj
```

The above command reads the specified project file and loads all the settings specified in this file before returning to the `sg_shell` prompt.

### **Specifying the Tcl Startup File**

You can specify a Tcl startup file while invoking the Tcl Shell interface in the following manner:

```
sg_shell -tcl mystartup.tcl
```

When you specify the above command, `sg_shell` evaluates the contents of the `mystartup.tcl` file as a Tcl script, and then the `sg_shell` prompt appears. This startup file can have some alias definitions, procedures, and predefined variables that you want to use in a given `sg_shell` session.

If there is any error in this startup file, `sg_shell` does not evaluate any further commands from the point of error. However, `sg_shell` returns to the `sg_shell` prompt so that you can query detailed error information. In such cases, apart from querying the error information, the shell is rendered useless (unless you do not worry about the contents of your startup files).

#### ***Continue-on-error mechanism for the Tcl startup file***

In this mechanism, a support for continuing on errors has been provided for the Tcl startup file. Consider a Tcl startup file, `mystartuperrors.tcl`, which contains errors. You specify the startup file while invoking the Tcl Shell interface in the following way:

```
sg_shell -tcl mystartuperrors.tcl
```

Because the Tcl startup file contains errors, you cannot ignore errors and continue sourcing of the rest of the script. To continue with the execution of the rest of the script, even after encountering errors, use the `-tcl_file_continue_on_error` command-line option.

```
sg_shell -tcl_file_continue_on_error -tcl myprocs.tcl
```

Using this option leads to ignoring errors in the Tcl startup file (here, `mystartuperrors.tcl`) on `sg_shell`'s command line.

Note the following points about this mechanism:

- The `-tcl_file_continue_on_error` option only ignores errors present in the Tcl startup file (here, `mystartuperrors.tcl`). This option does not apply to

the *source* command. Use the *-continue\_on\_error* option for the *source* command.

- This mechanism does not have any impact on the behavior of errors present in project files.
- In this mechanism, only command errors can be skipped. Parse errors in the Tcl file, such as missing close-bracket, would not get skipped.
- It is recommended that you use the *-tcl\_file\_continue\_on\_error* option only when you wish to ignore errors. This support should mainly be used when you require all the errors, such as ADC syntax errors, to be reported in one go.
- The playback mode of usage (as in `-tcl batch_run.tcl`) is not recommended. The recommended use model for using `sg_shell` in a batch run is as follows:

```
sg_shell < batch_run.tcl
```

### Specifying Customized Paths

You can configure the path for picking the product Perl files and Spyso while invoking `sg_shell` by using the *-I* option. For example, the following command picks the product data from the *<local\_path>* path, if present; otherwise, it picks it from the installation path:

```
sg_shell -project myProject.prj -64bit -I <local_path>
```

This option is useful primarily if you want to run `sg_shell` with a product patch having some fixes.

## Invoking Tcl Shell From SpyGlass

You can invoke `sg_shell` from SpyGlass by using the *-shell* switch, as specified below:

```
% spyglass -shell
```

You can perform the following additional tasks while invoking Tcl Shell from SpyGlass:

- *Loading the Project File*

### ■ *Running Multiple Tcl Commands*

#### **Loading the Project File**

You can load a project file in the `sg_shell` from SpyGlass, as specified below:

```
spyglass -shell -project <project_file_name>.prj
```

#### **Running Multiple Tcl Commands**

You can execute a sequence of Tcl commands specified in a Tcl file in the `sg_shell` using SpyGlass, as specified below:

```
% spyglass -shell -tcl <tcl-file.tcl>
```

**NOTE:** *Interactive session of SpyGlass Tcl Shell (through `sg_shell` or “`spyglass -shell`”) should be invoked in LSF (`bsub`) environment using `bsub -Is` or `bsub -Ip` options. Using simply `bsub -I` to invoke SpyGlass interactive Tcl Shell does not work properly and is not recommended by `bsub`. Refer to the `bsub` manual for more details on `-Is` and `-Ip` options for invoking interactive programs that are shell based.*

## Specifying Inputs to the sg\_shell

You can specify inputs to the `sg_shell` in any of the following ways:

### **By Using the echo Command**

You can specify a set of commands by using the `echo` command to the `sg_shell` in the following manner:

```
echo <some-commands> | sg_shell
```

### **By Using the cat Command**

You can specify a set of commands by using the `cat` command to the `sg_shell` in the following manner:

```
cat <some-commands> | sg_shell
```

### **By Specifying a Tcl File**

You can specify a Tcl file containing a set of input commands to `sg_shell` in the following manner:

```
sg_shell < <file-name>.tcl
```

## Using sg\_shell Commands

sg\_shell provides a set of Tcl commands to perform various operations. While specifying a Tcl command, ensure that you have specified all the mandatory arguments. If you do not specify the mandatory argument(s), sg\_shell flags an error message. For example, if you do not specify the project name with the `new_project` command, sg\_shell flags the following error message:

```
new_project: error: invalid usage (mandatory options missing,  
or wrong combination of options)
```

```
Try `new_project -help' for more details.
```

Similarly, if you specify the name of a Tcl command that does not exist, sg\_shell flags the following message:

```
invalid command name "<command-name>"
```

**NOTE:** You can specify unambiguous abbreviations for the sg\_shell commands to be used on the sg\_shell prompt. Therefore, if you type "new," then it will be unambiguously expanded to the `new_project` command.

## Using Named and Positional Arguments

A named argument is one that is specified in the `<argument> <value>` format, whereas a positional argument is one that is specified only in the `<value>` format, that is, it is not bound with any predefined argument name.

Consider the following `read_file` command:

```
read_file -type hdl input.v
```

In the above command, `-type` is a named argument. Here, the value, `hdl`, is bound to the `-type` argument. However, `input.v` is a positional argument, because it is specified independent of any predefined argument name.

Therefore, the following specifications are equivalent:

```
read_file -type hdl input.v
```

```
read_file input.v -type hdl
```

## Properties of Tcl Command Arguments

The following are the properties of the Tcl command arguments:

### Property 1

An argument can accept zero, one, or more values.

Consider the following example:

```
define_filter -regexp
```

In the above example, the `-regexp` argument does not accept any value.

The arguments that are usually used to turn on or turn off specific settings are considered as Boolean switches.

The following example displays different ways of specification in the [define\\_filter](#) command:

```
define_filter -name f1 -file f4.v
define_filter -name f1 -file {f1.v f2.v f3.v}
define_filter -name f1 -file "f5.v f6.v f7.v"
define_filter -name f1 -file [list f8.v f9.v]
```

In the above command, the `-file` argument can accept only one value. This value can be either a file name or a list.

If the value of the `-file` argument is a list, specify it in a particular format, as shown in the above example. To reiterate, the format is as follows:

- Enclose multiple values, which are space separated, inside curly brackets.
- Enclose multiple space-separated values inside double quotes.
- Use the `list` command with individual values as an argument to it. Since we want to execute this Tcl command, and assign its output to the `-file` argument, it should be put in square brackets `[]` as `[list <v1> <v2> ... <vN>]`.

If you do not specify a list value in the above format, `sg_shell` considers each element of the list as a separate value and, therefore, reports an

error. For example, consider the following command:

```
define_filter -file f4.v f5.v
```

For the above command, sg\_shell reports an error because the file names are not specified in a list format, by using " ", {}, or [list]. In addition, it takes f4.v as an argument to -file and f5.v becomes a positional argument, and is not considered a part of the -file argument.

Now consider another example, as shown below:

```
define_filter -file_line f1.v 25
```

In the above example, the -file\_line argument accepts two values, file name and line number. However, if you specify the value of this argument as a list containing the file name and line number, sg\_shell reports an error because this argument should be provided with two separate values. Therefore, sg\_shell reports an error in the following case:

```
define_filter -file {f1.v 25}
```

Consider another example, as shown below:

```
define_filter -file_lineblock f2.v 25 99
```

In the above example, the -file\_lineblock argument requires three values, that is, file name, start line number, and end line number. However, if you specify these three values as a list, sg\_shell would report an error since a list is considered as one value. Therefore, sg\_shell reports an error in the following case:

```
define_filter -file_lineblock "f2.v 25 99"
```

Type the following command to see the number of arguments accepted by each argument of a given Tcl command:

```
<tcl-command-name> -help
```

The above command displays a brief description for each argument with details of possible values.

Ensure that an argument taking list value versus an argument taking multiple values is appropriately specified by enclosing list values in "" {} or [list ...], while multiple values are not enclosed in any of these, but rather kept space separated.

## Property 2

Multiple specifications of an argument are either OVERWRITTEN or

APPENDED depending on the type of argument.

If an argument takes a single value (scalar type), and you specify multiple specifications of that argument, `sg_shell` retains only the last specification. For example, consider the following command in which the `-name` argument takes only a single value:

```
define_filter -name "n1" -file {f1.v f2.v} -name "n2"
```

In the above command, multiple `-name` specifications have been specified. In this case, `sg_shell` considers only the last `-name` specification, that is, `-name n2`.

If an argument takes multiple values (list type) and you have specified multiple specifications of that argument, `sg_shell` considers all the values of each argument. For example, consider the following command in which the `-file` argument takes multiple values:

```
define_filter -file {f1.v f2.v} -name n1 -file [list f3.v  
f4.v] -name n2 -file f5.v
```

In the above command, there are multiple specifications of the `-file` argument. In this case, `sg_shell` would consolidate all the values of the `-file` argument in a list and finally consider the list, `{f1.v f2.v f3.v f4.v f5.v}`, as the value of the `-file` argument.

### Property 3

Both Property 1 and Property 2 together make it possible to have arguments, which can accept a list of tuples.

For example, consider the `-file_lineblock` argument of the [define\\_filter](#) command for which the first and second properties as follows:

*-file\_lineblock (Property 1 : Accepts three values)*

*-file\_lineblock (Property 2: Tagged as list type)*

Now, consider the following command:

```
define_filter -file_lineblock f1.v 24 30 -name n1 -  
file_lineblock f2.v 9 20
```

In the above case, the `-file_lineblock` argument will have the following final value:

```
{ {f1.v 24 30} {f2.v 9 20} }
```



**NOTE:** In the above example, if the `-file_lineblock` argument was tagged as scalar type, the final value of this argument would have been `{f2.v 9 20}`.

#### Property 4

Argument values can have string property.

An argument value can have the *string* property. It means you can provide a string value in an argument. Further, a string argument can be of scalar or list type accepting single string value or multiple string values, respectively.

You should enclose the string value of an argument in brackets (`{ }`). This prevents the string value from Tcl's built-in backslash, command, or variable substitution. For example, if the value of an argument is a bit vector, `U0.data[3]`, you should specify this value as `{U0.data[3]}`. If you do not specify this value in brackets, Tcl interface will attempt a command substitution and will start looking for a command named "3".

Consider the `-msg` argument of the `define_filter` command:

```
define_filter -msg {Net 'U0.data[3]'} is not driven}
```

In this case, the `-msg` argument value is of the string type. Therefore, this argument should be specified in curly brackets. Further, if you specify the `-msg` argument multiple times, the last value is used since it is a scalar type of argument.

#### Property 5

Use the `--` operator (double dash) to specify negative values for numeric positional arguments.

By default, an argument prefixed with the `-` operator is treated as an option to a command. As a result, `sg_shell` treats a negative numerical value as an option to a command and reports an error message.

To specify a negative numerical value in scalar arguments, prefix the numerical value with the `--` operator, as shown in the following examples:

```
sg_shell> set_option sgsyn_clock_gating_threshold -- -20
```

```
sg_shell> set_parameter pe_power_saving_threshold -- -10
```

The `--` operator is a POSIX standard under command line options category for UNIX. All the arguments following a `--` specification are interpreted as non-option arguments.

## Errors and Messages Flagged in sg\_shell

sg\_shell flags errors in the following cases:

- If you specify the name of a Tcl command that does not exist, sg\_shell flags the following message:

```
invalid command name "<command-name>"
```

- If mandatory argument(s) or mandatory combination is missing, an error is flagged, as shown in the following example:

```
sg_shell> define_filter
define_filter: invalid usage (mandatory options missing,
or wrong combination of options)|
Try `define_filter -help' for more details.
```

- If the required number of values is not provided to an argument, an error is flagged as shown in the following example:

```
sg_shell> define_filter -file_lineblock f1.v 25 -name n1
define_filter: option '-file_lineblock' requires 3
argument
define_filter: -file_lineblock <file> <start_line>
<end_line>
Try `define_filter -help' for more details.
```

As a corollary, you can use the following to see all the supported options for a particular command:

```
sg_shell> define_filter -
define_filter: ambiguous option '-', matches '-name' '-
filter'
'-goal' '-du' '-ip' '-file' '-file_line' '-file_lineblock'
'-severity' '-rules' '-msg' '-except' '-weight' '-
weight_range'
'-regexp' '-invert'
```

- If the option is incorrect, an error is flagged, as shown in the following example:

```
sg_shell> new_project -dir ./output
new_project: unknown option '-dir'
Try `new_project -help' for more details.
```

- Any extra positional argument is ignored with a message, as shown in the following example:

```
sg_shell> current_methodology . p
current_methodology: extra positional argument 'p' -
ignored
Info: Current methodology: .
```

- Few options accept only a predefined set of values:

```
sg_shell> read_file -type hdle f1.v f2.v f3.v
read_file: invalid value 'hdle' for option '-type'
read_file: allowed values are 'verilog vhdl hdl gateslib
sglib
lef plib def sgdc waiver sourcelist'
```

## Error Handling in Tcl Commands

When an error occurs while using a Tcl command, you may want to capture the message, be prompted, and take appropriate action. To facilitate this feature, the following options are available for every Tcl command.

### Return Value

The return value can be NONE for few commands. The return value is decompiled on the command prompt, unless it is assigned to a Tcl variable. This is illustrated in the following example:

```
sg_shell> set goal_name [ current_goal ]
sg_shell> puts "Current Goal: $goal_name"
sg_shell> Current Goal: initial_rtl/lint/connectivity
```

### Return Status

The return status can be either pass (0 - TCL\_OK) or fail (1 - TCL\_ERROR) for each command, which you can query by using the *catch* command. This is illustrated in the following example:

```
sg_shell> catch { current_goal }
sg_shell> 0
```

### Error String

When the return status of a command is TCL\_ERROR, an error string is

reported by Tcl commands, which you can capture and report accordingly. The error string is decompiled on the command prompt, unless it is assigned to a Tcl variable by the *catch* command. You can also view the error string by using the `$.:errorInfo` Tcl variable or the *show\_error* command. This is illustrated in the following example:

```
sg_shell> new_project new
new_project: error: please save and close the current project
before opening any new project
sg_shell> show_error
Detailed Error Trace
-----
please save and close the current project before opening any
new project while executing

"new_project new "
sg_shell>
sg_shell> if { [catch { new_project new } error_string] } {
? puts "ERROR: New_Project : $error_string"
? puts "Closing current project forcefully"
? close_project -force
? puts "Creating new project..."
? new_project newProj -force
? }
ERROR: New_Project : please save and close the current
project before opening any new project
Closing current project forcefully
Creating new project...
current_methodology: info: methodology is now `./
SPYGLASS_HOME/GuideWare/New_RTL'
sg_shell>
```

## Startup Files in sg\_shell

sg\_shell provides a sample `.sg_shell.startup` file with sg\_shell installation. This file is located at `$SPYGLASS_HOME/.sg_shell.startup`. This file usually contains some *alias* definitions, some helpful procedures, and so on. Apart from this, you can keep your startup files to store your collection of procedures, aliases, or some predefined variables. The following are some

more locations where the startup file can be present:

- `$HOME/.sg_shell.startup`
- `$CWD/.sg_shell.startup`

sg\_shell searches and evaluates the startup files in the following order:

1. `$SPYGLASS_HOME/.sg_shell.startup`
2. `$HOME/.sg_shell.startup`
3. `$CWD/.sg_shell.startup`

As the CWD file is evaluated in the end, any definition in this file overrides any previous definition for the same variable, alias, or procedure. These `.sg_shell.startup` files should be Tcl files, and are evaluated as any other Tcl script. Therefore, if there is an `exit` command in any of these startup files and if the control reaches to that `exit` command, `sg_shell` exits.

You can also implicitly specify a startup file by using the `-tcl` command-line option of `sg_shell`. Therefore, when you specify `sg_shell -tcl mystartup.tcl`, `sg_shell` evaluates the contents of `mystartup.tcl` as a Tcl script, and then the `sg_shell` prompt appears. If there is any error in the startup file, no further commands from that file are evaluated. In addition, no other startup files are considered (including `-tcl <startupfile>` specification). However, `sg_shell` returns to the `sg_shell` prompt so that you can query for detailed error information. In such situations, apart from querying the error information, the shell is rendered useless (unless you do not worry about the contents of your startup files).

## Error Scenarios and Messages

Whenever there is any error in the startup file, `sg_shell` tries to publish the error trace. The trace is usually complete if there are any Tcl-specific errors related to Tcl built-in commands. However, if an error is returned from `sg_shell`-specific commands, such as `new_project`, the trace is usually an empty string. This is because the error messages coming from `sg_shell` commands are not bound to the interpreter result, and so are not available in the trace. You should not have `sg_shell` commands in startup files. Startup files are generally used to store your custom procedures to achieve some level of automation with your way of using `sg_shell` commands.

The following is an example of an error scenario:

```
[sam@chakra framework_regr]$ sg_shell -32bit
-----
----+
|
| SpyGlass Predictive Analyzer(R) - Version 5.5.0
|
| Last compiled on Apr 14 2015
|
| All Rights Reserved. Use, disclosure or duplication
| without prior written permission of Atrenta Inc.is prohibited.
|
| Technical support:emailsupport@atrenta.com or dial 1-866
| ATRENTA. |
-----
----+
|FATAL: errors found in startup file `/u/sam/
|.sg_shell.startup'
|           (missing close-brace)
|           (Use `show_error' for more details)
|
| sg_shell> show_error
| Detailed Error Trace
| -----
| missing close-brace
|   while executing
| "proc minus {a} {
|   return [expr {$a - 5}]
| proc verbose {} {
|   if {$::verbose} { set ::verbose 0 } else { set
|   ::verbose 1 }..."
|   (file "/u/sam/.sg_shell.startup" line 1)
```

## Specifying a Tcl File as the Startup File

You can specify a Tcl file as the startup file in the following manner:

```
sg_shell -tcl startup.tcl
```

A Tcl file can contain procedures, `sg_shell` commands, and so on. The `sg_shell` command enables you to reach to a certain level or stage of design analysis, running goal, and so on, and then brings you to the `sg_shell` prompt where you can proceed from where you left off in this startup file.

For example, a startup Tcl file can have your commands till the *run\_goal* command, and when you use this file with the `-tcl` option, the whole file is evaluated, and the `sg_shell` prompt appears. On the prompt, once the execution of the *run\_goal* command is complete, you can start browsing the message database and create your custom reports, and so on.

### Error Scenarios and Messages

If there is any error in the startup file, the execution is stopped at that point in the file, and the `sg_shell` prompt appears with error trace. Error trace may be empty if an error has occurred in `sg_shell` command, such as *new\_project*, *current\_project*, and so on. This is because, as of now, the error messages coming from `sg_shell` commands are not bound to interpreter result. A sample error message flagged by `sg_shell` is shown below:

```
-----+
----+
|
| SpyGlass Predictive Analyzer(R) - Version 5.5.0
|
| Last compiled on Apr 14 2015
|
|
| All Rights Reserved. Use, disclosure or duplication
|
| without prior written permission of Atrenta Inc.is prohibited.
|
| Technical support:emailsupport@atrenta.com or dial 1-866
| ATRENTA. |
-----+
----+
INFO: executing tcl startup file 'test2.tcl' ...
FATAL: errors found in tcl startup file `test2.tcl'
      (invalid command name "detect_filter")
      (Use `show_error' for more details)
```

```

sg_shell> show_error
Detailed Error Trace
-----
      invalid command name "detect_filter"
      while executing
      "detect_filter -goal [list g1 g2 g3] -goal "g4 g5" -goal
{g6
g7 g8} -regexp -file pqr.v -file test.v"
      (file "test2.tcl" line 1)
sg_shell>

```

## Specifying a Project File as the Startup File

You can specify a project file as the startup file in the following manner:

```
sg_shell -project ram.prj -projectwdir /u/prj/sam
```

Specifying a combination of `-project` and `-projectwdir` on the command-line of `sg_shell` is the same as if you are on the `sg_shell` prompt and you have specified the `open_project` `-project ram.prj -projectwdir /u/prj/sam` command.

The error trace is shown if there is any error during [open\\_project](#). The error trace might be empty if there is an error in the `sg_shell` commands.

### Error Scenario and Error Message

- If project file does not exist, `sg_shell` displays an error, as shown in the following example:

```

+-----+
+-----+
|
| SpyGlass Predictive Analyzer(R) - Version 5.5.0
|
| Last compiled on Apr 14 2015
|
|
| All Rights Reserved. Use, disclosure or duplication
|

```



## Using sg\_shell Commands

```

| without prior written permission of Atrenta Inc.is prohibited.
|
| Technical support:emailsupport@atrenta.com or dial 1-866
ATRENTA. |
+-----+
-----+
INFO: reading project file 'something' ...
Error: project file 'something.prj' does not exist
FATAL: errors found in project file `something'
      (oops, _empty_error_string!)
      (Use `show_error' for more details)
sg_shell> show_error
Detailed Error Trace
-----
while executing
  "open_project something -projectwdir ."
sg_shell>

```

- If there is some error in the project file, sg\_shell displays an error, as shown in the following example:

```

+-----+
-----+
|
| SpyGlass Predictive Analyzer(R) - Version 5.5.0
|
| Last compiled on Apr 14 2015
|
|
| All Rights Reserved. Use, disclosure or duplication
|
| without prior written permission of Atrenta Inc.is prohibited.
|
| Technical support:emailsupport@atrenta.com or dial 1-866
ATRENTA. |
+-----+
-----+
INFO: reading project file 'something' ...
open_project: error: found errors in project file

```

```
'something.prj'  
  FATAL: errors found in project file `something'  
  (oops, _empty_error_string!)  
  (Use `show_error' for more details)  
sg_shell> show_error  
Detailed Error Trace  
-----  
while executing  
"open_project something"  
sg_shell>
```

## Exit Codes Reported by sg\_shell

SpyGlass Tcl Shell Interface generates exit status codes that provide the exact status of the `sg_shell` run. You can use these status codes in a Tcl script to further debug your design.

`sg_shell` reports a return status after every execution of the [run\\_goal](#), [link\\_design](#), or [compile\\_design](#) command. Depending on the execution, the exit status codes can be classified broadly into the following categories:

- *Exit Status Codes When Goal Run is Successful*
- *Exit Status Codes When Goal Run is Not Successful*
- *Exit Status Codes When sg\_shell Exits or Terminates Abnormally*

### Exit Status Codes When Goal Run is Successful

`sg_shell` reports a status code of 0, by default, for a successful goal run and reports one of the following exit status messages:

```
0 (Rule-checking completed without errors or warnings)  
0 (Rule-checking completed with warnings)  
0 (Rule-checking completed with errors)
```

Use the `enable_pass_exit_codes` option to report different status codes (other than 0), depending on the severity of messages generated in the current run, as follows:

- When a goal run has been completed without any error or warning messages, `sg_shell` reports the following exit status message:

0 (Rule-checking completed without errors or warnings)

This message indicates that your design is clean with respect to the rules run.

- When a goal run has been completed without any error messages but with warning messages, sg\_shell reports the following exit status message:

11 (Rule-checking completed with warnings)

This message indicates that some rules of the Warning severity have been violated.

- When a goal run has been completed with error messages (and possibly warning messages), sg\_shell reports the following exit status message:

12 (Rule-checking completed with errors)

This message indicates that some rules of the Error severity have been violated.

**NOTE:** *Waived messages are not considered while deciding the exit status. Only reported messages are considered.*

### Exit Status Codes When Goal Run is Not Successful

- When a goal run is terminated due to a fatal design error (syntax errors), sg\_shell reports the following exit status message:

6 (Rule-checking terminated due to FATAL errors - design syntax error)

To rectify this issue, check the design inputs for completeness and correctness.

- When a goal run is terminated due to incorrect usage and incorrect or incomplete inputs, sg\_shell reports the following exit status message:

7 (Rule-checking terminated due to FATAL errors - usage or run error)

This message indicates that you have either missed providing some inputs that are essential for rule-checking or provided some inputs incorrectly.

To rectify this issue, check the design inputs for completeness and correctness.

- When a design is not saved because of incorrect save database directory, sg\_shell reports the following exit status message:  
8 (Design database save failure, rule checking aborted)  
To rectify this issue, check the save or restore related options in the project file.
- When a goal run is terminated due to a license failure, sg\_shell reports the following exit status message:  
4 (License failure, rule-checking aborted)  
To rectify this issue, check the license settings and make corrections as required. You can also use the LICENSEDEBUG option to retrieve more details around license setup.

### Exit Status Codes When sg\_shell Exits or Terminates Abnormally

- When sg\_shell run is terminated due to an abnormal error that could not be trapped, sg\_shell reports the following exit status message:  
1 (Abnormal termination - termination not trapped by software)  
This message indicates issues related to the operating system, such as stack overflow, have occurred. In such cases, if termination happens while running a goal, the corresponding SpyGlass log file may be incomplete because the software was not able to trap the error signal and report suitably in the log file.  
To rectify this issue, check the available machine resources and make corrections as required.
- When sg\_shell run is terminated due to an error that is trapped by the software, sg\_shell reports the following exit status message:  
3 (Abnormal termination - termination trapped by software)  
This message indicates memory-related issues, such as segmentation fault or memory corruption, have occurred but the software was able to trap the error signal and report suitably in the log file.  
To rectify this issue, check the gdb trace and stack trace printed in the SpyGlass log file and make corrections as required. If sg\_shell traps the terminating signal while running a goal, it tries to generate the moresimple report, which contains error messages and/or rule violations reported before the end of goal run. Check the generated moresimple report because it may contain useful information, which

may help to debug the issue.

- When a KILL signal is sent to sg\_shell process to terminate the session immediately, sg\_shell reports the following exit status message:

```
137 (run killed by user, or by system due to lack of
resources like memory etc.)
```

In this case, the signal is sent either by the user or by the operating system because of lack of resources, such as memory.

To rectify this issue, run sg\_shell on a machine that has higher memory.

- When a signal is sent to sg\_shell process to terminate the session if size of files generated by sg\_shell exceeds the maximum limit allowed by the operating system, sg\_shell reports the following exit status message:

```
153 (File size limit exceeded)
```

To rectify this issue, increase the maximum allowed size of a file to enable sg\_shell process to complete successfully.

**NOTE:** *For other scenarios when sg\_shell exits or terminates abnormally, sg\_shell reports exit status messages with codes greater than 128.*

## Features of sg\_shell

This section contains the following topics:

- [Using the Help Feature](#)
- [Using the Tab Completion Feature](#)
- [Capturing stdout and stderr](#)
- [History Support in sg\\_shell](#)
- [Command Logging in sg\\_shell](#)
- [Screen Output Logging in sg\\_shell](#)
- [Signal Handling in sg\\_shell](#)
- [Using Escape Names in sg\\_shell](#)
- [Common SDC Flow](#)
- [Dual Design Read Flow](#)
- [Using Key Combinations for Performing Actions](#)

## Using the Help Feature

The sg\_shell utility provides the help feature in which you can view the help of any Tcl command and its options. You can use the help feature by specifying the `-help` command-line option. For example, if you want to see the details of all the start-up options of sg\_shell, specify the following command:

```
sg_shell -help
```

The above command lists all the start-up options, such as `-project`, `-tcl`, `-64bit`, `-32bit`, and so on, available with sg\_shell and their respective help description.

Similarly, if you want to view the short help and syntax of a particular Tcl command, enter the following command on the sg\_shell prompt:

```
<Tcl-command-name> -help
```

This command also displays the short help of all the arguments of that Tcl command. In addition, there is a `help` command in sg\_shell, which provides support information for the following:

## Features of sg\_shell

- Tcl commands
- Options
- Methodology
- Goals
- Rules
- SGDC commands
- Parameters
- Reports

The details of the `help` command are provided as part of the [SpyGlass Tcl Commands](#) section.

## Using the Tab Completion Feature

Currently, Tcl shell interface of SpyGlass supports the following two types of tab completions:

- If the first word is being completed, `sg_shell` first searches for the available commands (both built-in and `sg_shell` commands). If matches are available, they are listed. Otherwise, `sg_shell` displays the matching file or directory names of the CWD.
- If the second or later word is being completed, file or directory names completion comes into effect.

**NOTE:** *The tab completion feature does not work if you leave spaces and then type something and try to do tab completion. Remove any leading spaces, and then tab completion will work as expected.*

`sg_shell` uses the `?` character whenever you specify an incomplete command as per the Tcl syntax. For example, in the following case, after `sg_shell` has encountered the closing bracket, the command is considered as complete and that command is executed:

```
sg_shell> set files { f1.v
? f2.v f3.v
? f4.v
? }
```

In addition, `sg_shell` supports tab completion when you have `?`. For example, if you want to see all the files and folders starting with the character `f`, you can specify the following:

```
? f<TAB>
f1.v f2.v f3.v f4.v files_all/
```

## Capturing stdout and stderr

The following are the two ways to capture stdout and stderr:

- By using the redirection operator, `>`, present in UNIX shells.

The redirection operator is used to redirect messages in the specified file. For example, if you specify the command, `run_goal > output.log`, `sg_shell` redirects both stdout and stderr messages in the `output.log` file. Consider another example, as shown below:

```
new_project prj1 -projectwdir ./ > outputfile
```

The command is actually `new_project prj1 -projectwdir ./`. However, the output of this command is captured in the file, `outputfile`.

Consider another command, as shown below:

```
new_project > prj1
```

In the above case, the actual command is `new_project`, and its output is captured in the file, `prj1`. The output in this case is an error message related to the invalid usage of the `new_project` command.

If the specified output file already exists, `sg_shell` truncates the existing file without any Warning or Info message. However, if the specified file does not exist, `sg_shell` creates that file.

`sg_shell` supports the `>` operator only for published `sg_shell` commands, such as `new_project`, `define_project`, and so on. It does not support this operator for Tcl's built-in commands, such as `list`, `puts`, and so on. Therefore, you cannot specify the command, `puts "help" > outputfile`.

In addition, the `>` command is not supported for user-defined procedures. Therefore, if you define a procedure, `proc hello {}`



{puts "Hello there"}, you cannot specify `sg_shell> hello > outfile` because this will not work and will give you a wrong args error.

The following are some error scenarios related to the `>` command:

- ❑ If output file name is missing, an error is flagged, as shown in the following example:

```
sg_shell> new_project prj1 -projectwdir ./ >
new_project: error: missing redirect name
```

- ❑ If the `>` operator is specified multiple times, the first one will be considered, and the remaining occurrences of the `>` operator will be treated as arguments to the `sg_shell` command. However, no error message comes for this type of specification.

- Refer to the [capture](#) command for details on an alternative method to capture `stdout` and `stderr`.

**NOTE:** *UNIX shell re-direction operator, >>, is not supported for append operation in Tcl shell. Use the `capture -append` Tcl command to implement the append operation.*

## History Support in sg\_shell

You can use the `history` command in `sg_shell`, and it will give you a list of commands that you have already used. This command is applicable only for the interactive mode. If you write ten commands in an `src.tcl` file followed by `history` command in the `src.tcl` file, and then specify the command, `sg_shell < src.tcl`, the output from `history` is going to be empty.

The `history` command is supported by the Tcl installation package itself. Other features that are usually available on the "tcsh" shell are also available in this history package. Therefore, you can have the following history substitutions also working in `sg_shell`:

- Use `!"`, to execute "n th" command from history listing.
- Use `!!`, to execute just the previous command.
- You can use modifiers `^olddd^newwww^` to replace "olddd" with "newwww".

A sample error message is shown below:

```
sg_shell> open_project urprj
Error: project file 'urprj.prj' does not exist
sg_shell> ^ur^my^
open_project myprj
Info: Current methodology:
/u/release/spyglass/SPYGLASS_HOME/GuideWare/New_RTL
Info: Starting to load goal
'initial_rtl/lint/connectivity'
Info: Finished loading goal
'initial_rtl/lint/connectivity'
sg_shell>
```

If you specify the history command, the string literal, such as "!n", will be found, as opposed to finding the actually substituted command.

If there are any leading or trailing white spaces in any command on the sg\_shell prompt, they will be retained when you use the *history* command to list them.

**NOTE:** *The leading and trailing white spaces are removed only for up-history or down-history using arrow keys or other bind keys, so that when you do up-arrow or down-arrow you get a command-line that is free of leading or trailing white spaces.*

## Command Logging in sg\_shell

If there is write permission in the current working directory, a file called sg\_shell\_command.log is created, which contains the commands typed by you in the sg\_shell.

If the sg\_shell\_command.log file already exists, its contents are truncated. However, if it does not exist, sg\_shell creates it if the CWD has write permission. Otherwise, sg\_shell silently skips creation of this file.

Only the commands used in the interactive mode are logged, and commands are logged irrespective of whether they return an error or not.

A sample log file is given below:

```
[sam@chakra framework_regr]$ cat sg_shell_command.log
## This log has been generated by sg_shell:
##   On: Fri Jul 10 14:39:20 2009
##   By: sam
```

```
## Using: Version 4.x.y (Jul 10 2009)
##-----

open_project urprj
^ur^my^
quit
```

## Screen Output Logging in sg\_shell

A new command-line scalar or string option, `-shell_log_file`, has been provided in `sg_shell` to specify the log file path and also to control the behavior of screen output logging in `sg_shell`.

`sg_shell` tries to open the log file, specified by using the `-shell_log_file` command-line option, for writing. If the file already exists and has write permissions, `sg_shell` truncates this file and starts logging the screen output of the current session. If the file does not exist, and `sg_shell` is unable to create this file for writing, `sg_shell` reports an error message and exits.

Screen logging is enabled for both interactive and non-interactive sessions of `sg_shell`. Interactive session is the session in which you are provided with the **sg\_shell>** prompt to work. Non-interactive session is usually either through a pipe or through redirection, where you do not get a prompt.

### Screen Output Logging for Interactive Sessions

Note the following points while using the `-shell_log_file` option for *interactive sessions in sg\_shell*:

- Tcl shell interface logs any activity that is visible on your current session. For example, if you open a man page of a command or you run any UNIX command, the output gets logged into the log file.
- Tcl shell interface, in a way, also logs stdin. Therefore, `sg_shell` logs anything that you type on the `sg_shell` prompt. If you press the Backspace key, the log file displays the character associated with it (usually, Ctrl+H or ^H). Therefore, some special characters might appear in your log file.
- If you open Vim or any other editor from `sg_shell`, `sg_shell` logs their output or input, mostly in terms of unprintable characters, in the log file.

- If you enter `man <command>`, `sg_shell` might not provide you the scrolling behavior for the requested man page.

### Screen Output Logging for Non-interactive Sessions

Note the following points while using the `-shell_log_file` option for *non-interactive sessions in sg\_shell*:

- If you enter the following in your UNIX shell:

```
unix_prompt> sg_shell -shell_log_file myout.log  
< my_commands.tcl
```

Where `my_commands.tcl` contains commands, such as *new\_project*, *compile\_design*, *run\_goal*, and so on.

Then, `sg_shell` logs the output from the commands in the `my_commands.tcl` file into the log file, `myout.log`. However, the commands themselves do not get logged, because the commands are not echoed back on the screen output in non-interactive sessions.

### Default Logging

If you do not specify the `-shell_log_file` option, a default log file, `.sg_shell.log`, is created in the current directory. If `sg_shell` is unable to open this default log file for writing, no error is reported, and the run proceeds as if there is no logging.

*The default logging is disabled for interactive sessions in sg\_shell* due to the following reasons:

- Enabling the default logging in interactive sessions may lead to a number of unprintable characters in the log file, because of various combinations of key presses.
- If the default logging is enabled in interactive sessions, the log file may grow to any extent if the `sg_shell>` interactive session is opened for a long duration and the same shell is used repeatedly.

*The default logging is enabled for non-interactive sessions in sg\_shell.*

Therefore, if you enter the following in your UNIX shell:

```
unix_prompt> sg_shell < my_commands.tcl
```

Then, the output from the commands in the `my_commands.tcl` file appears on the screen and is logged in the default log file, `./sg_shell.log`.

## Signal Handling in `sg_shell`

The following points illustrate signal handling in `sg_shell`:

- `CTRL + C` signal (`SIGINT`) is dealt differently in `sg_shell`. If you are using `sg_shell` interactively, that is, working on the `sg_shell` prompt, `CTRL + C` signal is ignored, and nothing will happen.

However, if the `sg_shell` is running in a non-interactive mode (say, through the command, `sg_shell < my_commands.tcl`), then using `CTRL + C` will abort the run. The generated output or data from `sg_shell` may be incomplete in such cases, and gives unpredictable results if later tried to be reused in Atrenta Console or in later sessions of `sg_shell` itself.

- Other signals have been left untouched and they will work as they are today in the normal SpyGlass flow.

`CTRL + Z`, however, needs a special mention. If you are in `sg_shell` prompt, and you open a file using `vim` as shown below:

```
sg_shell> vim output.log
```

In the above case, you are working in `vim` and if you press `CTRL + Z`, this will suspend the `vim` session as well as the `sg_shell` session, and you will return to your native shell, such as `tcsh`.

However, when you do an `"fg"` or `"%"` to resume the suspended job, then it should take you to the `vim` session (or at least to the `sg_shell` session). In this case, there is some known issue and it does not allow you to type anything in the shell, or if you are able to type in something then that command will not be honored.

It is recommended to start UNIX shell from within `sg_shell` by typing `tcsh` or `sh` etc. (depending on the shell you want to start), and then exit this UNIX shell whenever you want to return to `sg_shell`.

When you are in `sg_shell` prompt and you press `CTRL + Z`, then also the `sg_shell` session is suspended and you return to your native

shell, such as `tclsh`. However, on doing "fg", it should resume your `sg_shell` session. The `sg_shell` prompt may not appear until you enter some command after doing "fg".

As such, interactive session of `sg_shell` does not support the complete and usual behavior of `CTRL + Z`, as is available on most of the *UNIX* based operating systems. However, you can still use the `CTRL + Z` to suspend a running `sg_shell`'s interactive session, and use the *kill* command thereafter to terminate that interactive session of `sg_shell`. Using `CTRL + Z` in any other manner may not always give expected behavior of `CTRL + Z` on `sg_shell`'s interactive session.

## Using Escape Names in sg\_shell

An escape name is a combination of special characters. Standard SpyGlass flow has a particular way of specifying the escape names for Verilog and VHDL design units and other design objects.

The notion of escape names in sg\_shell is the same as that in standard SpyGlass flow. However, the mechanism of specifying escape names as arguments to various options and commands in sg\_shell needs to be understood from a different perspective. This is primarily because sg\_shell is a Tcl interface and Tcl has its own interpretation of special characters. Therefore, the mechanism of taking inputs from the Tcl shell needs to protect the escape names from the default interpretation by Tcl.

**NOTE:** *All the Tcl commands that accept design object names as arguments (both positional and named arguments) can take escape names as well.*

The procedure to specify escape names is Tcl-specific. You need to distinguish between an argument that takes a single string value and an argument that takes a Tcl list of multiple string values. The behavior is the same for both named and positional arguments.

### Specifying Escape Names in Arguments That Accept a Single String Value

The escape name should be surrounded by brackets without any extra leading or trailing characters or white spaces. The Verilog escape names have a white space at the end. Therefore, that space must be retained before the closing bracket. For example, if the Verilog design object is "\special\$name " (note the white space appearing at the end), use this object in the [get\\_attribute](#) command as shown below:

```
sg_shell> get_attribute -class flat_cell
{top.mid.low.\$special$name } my_attr
```

Again, note the white space appearing before the closing bracket in the above example. However, the following specification is incorrect:

```
{ top.mid.low.\$special$name }
```

This specification is not correct because it contains extra characters at the beginning and at the end. In other words, whatever is specified inside brackets { } will be taken as is.

## Specifying Escape Names in Arguments That Accept a Tcl List of Multiple String Values

Generally, a Tcl list is specified by enclosing the values within brackets `{}`. Therefore, `{a b c d}` denotes a list that contains four values, which are "a", "b", "c", and "d".

If, however, these values are escape names, these individual values should again be enclosed within brackets `{}`, as shown in the following example:

```
{{top.\special_mid$inst?name .U0.U11.wire0}
{top.\another_special$$name .U12} {top.\new_??$$name }
top.inst_without_special_chars.U0 {top.mid0.leaf0}}
```

The above scheme needs to be followed even if the list contains a single value. Consider the following example:

```
{{top.\special_mid$inst?name .U0.U11.wire0}}
```

Therefore, a typical `get_cells` command for a single name should be specified as follows:

```
sg_shell> set mycell [get_cells {{top.\special_mid$$$_name
.U01}}]
```

## Specifying Arguments That Accept Tcl's Special Characters

The above schemes of value specification for string and Tcl list options apply not only to escape names, but also to any input that contains Tcl's special characters. For example, to specify a hierarchy separator while using the `-regexp` argument in a design query command, say, `get_cells`, you need to use the above scheme. This is because, in case of the `-regexp` argument, the hierarchy separator needs to be escaped. It means, "." needs to be specified as "\." to distinguish it from the regular expression meaning of a single "." as matching any single character.

For example, consider a top module, `top`. This top module has few leaf level instances: `u_1`, `u_2`, `u_3`, and `u_4`. Use the following command to fetch `u_1`, `u_2`, and `u_3` by using the `-regexp` argument:

```
sg_shell> set cell_list [get_cells -regexp {{top\.u_[1-3]}}
```

Use the following command to fetch all of `u_` instances by using the `-regexp` argument:

```
sg_shell> set cell_list [get_cells -regexp {{top\.u_*}}
```



## Common SDC Flow

The SDC commands are now uniformly available to all products, and are used by all rules during their analysis. You can specify a subset of SDC commands, which are the most commonly used commands by different products, interactively on `sg_shell`. For further details on these commands, refer to the [SDC-Equivalent Commands](#) section.

When using `sg_shell`, all information from the SDC is automatically read. Previously, the information was only read if the `sdc2sgdc` option is set. The default value of `sdc2sgdc` option is `no`. However, for `sg_shell` runs, SpyGlass automatically sets this option to `yes`, when at least one SDC file is specified using the `sdc_data` constraint. Therefore, any information contained inside the SDC files, such as `clocks`, `set_case_analysis`, and so on, is automatically used by rules during their checking.

This flow allows you to specify these commands either in SGDC or SDC, and the software automatically picks them from both these sources. If there is any conflict between SGDC and SDC specifications, the SDC specification is given a priority. In such cases, an appropriate warning message is reported on the screen.

If you want to disable the translation from SDC to SGDC in the project scope, use the following command:

```
set_option sdc2sgdc off
```

If you want to disable the translation from SDC to SGDC in the goal scope, use the following command:

```
set_goal_option sdc2sgdc off
```

**NOTE:** *By default, the `sdc2sgdc` generated constraint files are retained in the subsequent SpyGlass run. Set the `retain_old_sgdc` parameter to `no` to remove the `sgdc` file generated in the previous SpyGlass runs.*

If you set the `sdc2sgdc` option as `off` after the `run_goal` command, the SDC commands, such as [create\\_clock](#), [create\\_generated\\_clock](#), [set\\_case\\_analysis](#), [set\\_input\\_delay](#), and [set\\_output\\_delay](#), are not available to the products that use SGDC commands. In addition, the SDC clocks are not propagated by using the [propagate\\_clocks](#) command and are not reported by the `report_clocks [ get_clocks ]` or `report_adc -sdc` command.

To use this flow in Atrenta Console, you need to either set the `sdc2sgdc` option in your project file before opening it in Atrenta Console, or enable this option from the Atrenta Console GUI. If you observe any difference in results between Atrenta Console run and `sg_shell` run, please check and ensure that the `sdc2sgdc` option is enabled in the Atrenta Console run, because it is on by default in the `sg_shell` run.

## Examples

### *Example 1*

If you specify a clock `clk1` in SDC with a period of 10, and another clock `clk2` in SGDC with a period of 20:

```
sg_shell> cat test.sdc
create_clock -name clk1 -period 10 [get_ports clk1]
```

```
sg_shell> cat test.sgdc
current_design top
sdc_data -file test.sdc
clock -name clk2 -period 20
```

Earlier, `sg_shell` used to take only the SGDC clock with a period of 20. With the Common SDC Flow feature, `sg_shell` takes a union of SGDC and SDC clocks. Therefore, in this example, both clocks, `clk1` in SDC and `clk2` in SGDC, are used by rules during their analysis.

### *Example 2*

If you specify a clock `clk1` in SDC with a period of 10, and the same clock in SGDC with a period of 20:

```
sg_shell> cat test.sdc
create_clock -name clk1 -period 10 [get_ports clk1]
```

```
sg_shell> cat test.sgdc
current_design top
sdc_data -file test.sdc
clock -name clk1 -period 20
```

Previously, `sg_shell` used to take only the SGDC clock with a period of 20. With the Common SDC Flow feature, in case of conflict, preference is given to the SDC information. Therefore, in this example, the SDC clock with a period of 10 is used by rules during their analysis.

### Example 3

If you specify multiple modes of `sgdc_data` in the SGDC constraint file, only the last active `sgdc_data` mode is retrieved. Consider the following example:

```
sg_shell> cat test.sgdc
current_design "test"
sgdc_data -file mode1.sdc -mode func
sgdc_data -file mode2.sdc -mode func1
```

In this example, only the `func1` mode will be active and only the `mode2.sdc` SDC file will be read.

## Dual Design Read Flow

`sg_shell` provides the capability to read two designs simultaneously. By using this capability, you can further compare SDCs or perform sequential equivalence (SEC) of these two designs.

The capability of reading two designs simultaneously in a single run to perform a variety of comparative analysis on these designs is known as dual design read (DDR).

The DDR feature provides the following functionalities:

- **SDC Equivalence:** SDC equivalence is used to compare two different SDCs for pre-layout and post-layout design snapshots.
- **Sequential Equivalence (SEC):** The DDR capability establishes sequential equivalence between original and power-optimized design when the original design has been auto-fixed to reduce power consumption.

In a DDR setup, the first design is known as a reference design because it is the reference input, and the second design is known as the implementation design because this design is under analysis. The implementation design is specified in the project file or on `sg_shell` command-line as any other design via commands, such as `read_file`, `set_option`, and so on.

Use the following options in the DDR goal scope to specify the reference design:

- **reference\_design\_projectfile:** This option specifies a project file containing details, such as design files, design options, and SGDC, for

the reference design while running a DDR goal. This option is complementary to the *reference\_design\_sources* and *reference\_design\_sgdc* options, and instead of these two options, you can specify a project file that contains such settings.

- ***reference\_design\_sources***: This option specifies design files and options for a reference design through the .f file.
- ***reference\_design\_sgdc***: This option specifies an SGDC file that contains constraints for a reference design.
- ***ignore\_reference\_project\_sgdc***: When a reference project file is specified, specify this option to ignore SGDC files from a reference design project file.

Various *Design Setup Commands*, *Goal Setup or Run Commands*, *ADC Setup Commands*, or *Debug Commands* are applicable to the implement design only. For example, if a file is specified by using the `read_file -type hdl test.v` command, this file is a part of the implementation design. Similarly, query or edit commands, such as *get\_file*, *get\_parameter*, *set\_parameter*, *get\_adc*, and *remove\_adc*, also work on the implementation design scope. In addition, the design query commands, such as *get\_cells* and *get\_nets*, are applicable to the implementation design only. The interaction with the reference design is limited to the above-mentioned options only and is not accessible to any other sg\_shell command.

## Examples

The following example illustrates SDC equivalence flow:

```
##implement design setup##
sg_shell> read_file -type hdl ./src/test_rtl.v
sg_shell> read_file -type sgdc imp.sgdc
sg_shell> read_file -type sglib nldm.sglib

##DDR goal setup##
sg_shell> current_methodology $env(SPYGLASS_HOME)/
Methodology
sg_shell> current_goal Constraints/pre_layout/
sdc_equiv_dual_design -top top

##reference design setup##
sg_shell> set_goal_option reference_design_projectfile
ref.prj
sg_shell> set_parameter equiv_sdc_design_equivalence_file
```

```
mapping.txt
##sdc equivalence analysis run##
sg_shell> run_goal
##sdc equivalence run results##
sg_shell> write_report more simple
```

## Using Key Combinations for Performing Actions

Editing commands in the sg\_shell prompt are based on the Emacs style of editing.

## Setting SpyGlass Preferences Using Tcl Shell Interface

You can set the SpyGlass preferences by using the [gui\\_set\\_preference](#) Tcl command.

```
gui_set_preference <preference-key> <preference-value>
```

Refer to the [gui\\_set\\_preference](#) command section for more details.

This section explains the following topics:

- List of SpyGlass GUI Preferences
- Overriding SpyGlass GUI Preferences

## List of Preferences

This section explains the various preference key values that you can use to set the SpyGlass GUI preferences.

**TABLE 1** Preference Key Values

Preference Key	Syntax	Description
menu_dialog_font	gui_set_preference menu_dialog_font {font_name,font_size}	Enable you to set the menus/dialogs font. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.
text_viewer_font	gui_set_preference text_viewer_font {font_name,font_size}	Enables you to set the font of the hdl/text viewers font. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.
tree_font	gui_set_preference tree_font {font_name,font_size}	Enables you to set the tree viewers font. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.
table_font	gui_set_preference table_font {font_name,font_size}	Enables you to set the tables font. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.
help_viewer_font	gui_set_preference help_viewer_font {font_name,font_size}	Enables you to set the help_viewer font. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.
report_font	gui_set_preference report_font {font_name,font_size}	Enables you to set the reports fonts. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.
shell_font	gui_set_preference shell_font {font_name,font_size}	Enables you to set the shell fonts. Ensure that the font name and font size values are comma-separated and are enclosed in braces. Additionally, do not use special characters in the font names.

**TABLE 1** Preference Key Values

Preference Key	Syntax	Description
wrap_message_text	gui_set_preference wrap_message_text true/false	Allows the message windows text to be wrappable or not.
auto_launch_spreadsheet_viewer	gui_set_preference auto_launch_spreadsheet_viewer true/false	Enables the spreadsheet viewer to launch automatically when selecting a violation in the message_tree.
show_message_tree_tooltip	gui_set_preference show_message_tree_tooltip true/false	Allows whether message tree tooltip should be shown or not.
show_message_id	gui_set_preference show_message_id 0/1/2	Changes the message ID visibility. Specify one of the following values: <ul style="list-style-type: none"> <li>• 0: shows the messages with unique id compatible with spyglass-reports.</li> <li>• 1: shows the messages with indexes (that are compatible with Spyglass-5.6.1 or earlier versions)</li> <li>• 2: Hides any message id if visible.</li> </ul>
auto_launch_schematic	gui_set_preference auto_launch_schematic true/false	Sets whether schematics are shown as soon as violation is selected or not.
waveform_viewer	gui_set_preference waveform_viewer 0/1/2	Specify one of the following values: <ul style="list-style-type: none"> <li>• 0: This is default value.</li> <li>• 1: Sets the GTKwave as the waveform viewer.</li> <li>• 2: Sets Debussy as the waveform viewer.</li> </ul>
schematic_single_click_probing	gui_set_preference schematic_single_click_probing true/false	Sets whether single-click probing allowed in schematic or not.
apply_waiver_on_creation	gui_set_preference apply_waiver_on_creation true/false	Specifies whether waiver must be applied as soon as they created or not.
scenario_support	gui_set_preference scenario_support true/false	Enables or disables the scenario support.

**TABLE 1** Preference Key Values

Preference Key	Syntax	Description
auto_reload_project	gui_set_preference auto_reload_project true/false	Enables or disables the auto reload project functionality.
auto_restore_session	gui_set_preference auto_restore_session true/false	Enables or disables the auto restore session functionality.
html_viewer	gui_set_preference html_viewer <path>	Specifies whether you want to use any other html viewer. To do so, set its executable path as <path>.
pdf_viewer	gui_set_preference pdf_viewer <path>	Specifies whether you want to use any other PDF viewer. To do so, set its executable path as <path>.
text_viewer	gui_set_preference text_viewer <path>	Specifies whether you want to use any other text viewer. To do so, set its executable path as <path>.
text_editor_line_arg	gui_set_preference text_editor_line_arg <argument>	Sets any line flag for your text viewer. Specify the value as a <argument>.
external_editor	gui_set_preference external_editor true/false	Enables/disables the external editor.

### Examples

The following are some of the sample `gui_set_preference` commands:

```
gui_set_preference auto_launch_schematic true
```

```
gui_set_preference menu_dialog_font {Liberation Serif,20}
```

```
gui_set_preference text_editor_line_arg ###
```

```
gui_set_preference waveform_viewer 2
```



## Overriding GUI Preferences

The GUI preferences can be overridden at the start of a SpyGlass session by using a file named `.sg_init_gui.tcl` that may include one or more `gui_set_preference` Tcl commands. This file is looked-up in the following directories sequentially at the start of a SpyGlass session and if found, it is sourced.

- Directory pointed by the `SPYGLASS_HOME` environmental variable
- User home directory
- Current working directory

For example, if the project auto reload feature should be disabled for all the users, add the following command to the `.sg_init_gui.tcl` file in the `SPYGLASS_HOME` directory.

```
gui_set_preference AutoReloadProject false
```



---

# SpyGlass Tcl Commands

---

This chapter describes the Tcl commands currently available in SpyGlass Tcl Shell Interface (or `sg_shell`). These commands are categorized in the following groups:

- *Session Commands*
- *Design Setup Commands*
- *Goal Setup or Run Commands*
- *ADC Setup Commands*
- *Reporting Commands*
- *Waiver Commands*
- *Debug Commands*
- *Miscellaneous Commands*

**NOTE:** *The usage of regular expressions through the `-regexp` argument has been described in the relevant commands. For more details on using the regular expressions, refer to the *Using Regular Expressions and Wildcard Characters* topic of the *Atrenta Console User Guide*.*

## Session Commands

Session commands are useful for session management. Use these commands to encapsulate design or goal setup information inside a user-defined project.

The following table describes the various session commands:

<b>Command</b>	<b>Description</b>
<i>gui_set_obw_dialog_labels</i>	Sets OBW dialog labels
<i>gui_restore_session</i>	Restores the session given in a Tcl file
<i>gui_save_session</i>	Saves the current SpyGlass session in a Tcl file
<i>gui_restore_session</i>	Restores the session given in a Tcl file
<i>new_project</i>	Creates a new project
<i>open_project</i>	Opens an existing project
<i>save_project</i>	Saves the specified project
<i>close_project</i>	Closes the currently active project
<i>current_project</i>	Displays data of the currently loaded project
<i>import_project</i>	Used to import existing project settings into current project
<i>exit</i>	Quits sg_shell and returns to the UNIX shell prompt
<i>set_pref</i>	Sets the specified preference variable to its specified value
<i>get_pref</i>	Displays the value of the specified preference variable (s)

## gui\_set\_obw\_dialog\_labels

Sets OBW dialog labels

### Syntax

```
gui_set_obw_dialog_labels <label_set| -all>
```

### Scope

None

### Return Value

None

### Description

Use this command to specify a set of labels to the drop-down menu of the **Message Label** column in the **Object Filter** table. The existing set of labels are hidden there after and can be retrieved using the `-all` argument.

### Arguments

This command has the following arguments:

`-all`

Sets all the available labels.

#

`<label_set>`

Label Set for OBW dialog

### Examples

```
sg_shell> gui_set_obw_dialog_labels -all
```

### See Also

None

## gui\_configure\_obw\_dialog

Configures OBW dialog

### Syntax

```
gui_configure_obw_dialog
  <-filter|-waive>
  [-rule]
  [-msg_collection]
  [-condition]
  [-tcl_var]
  [-regex]
  [-secondary_messages]
  [-force]
```

### Scope

None

### Return Value

None

### Description

Use this command to select the Fields and Checkbox to show/hide in the OBW dialog box using this Tcl command. Use of options shows the fields in dialog. To retrieve all the fields, specify the -all argument.

### Arguments

#### -condition

Show 'Condition' text field

#### -filter

Shows 'filter messages' checkbox

---

**Session Commands****-force**

Forces existing OBW dialog to close

**-msg\_collection**

Shows 'Message collection' text field

**-regex**

Shows 'regex' checkbox

**-rule**

Shows 'Rule name' text field

**-secondary\_messages**

Shows 'secondary messages' checkbox

**-tcl\_var**

Shows 'Tcl variable' text field

**-waive**

Shows 'waive messages' checkbox

**Examples**

```
sg_shell> gui_configure_obw_dialog -waive
```

**See Also**

None

## gui\_save\_session

Saves the current SpyGlass session in a Tcl file

### Syntax

```
gui_save_session <file-name>  
[ -force ]
```

### Scope

None

### Return Value

None

### Description

The *gui\_save\_session* saves the current SpyGlass session you are running in a Tcl file.

### Arguments

This command has the following arguments:

#### <file-name>

The Tcl file name to be saved.

#### -force

Overwrites the given file.

**NOTE:** *Using the -force argument may result in data loss from the previous run.*

### Examples

```
sg_shell> gui_save_session abc.tcl
```

### See Also

[gui\\_restore\\_session](#)



## gui\_restore\_session

Restores the session given in a Tcl file

### Syntax

```
gui_restore_session <file-name>
```

### Scope

None

### Return Value

None

### Description

The *gui\_restore\_session* restores the SpyGlass session mentioned in the specified Tcl file.

### Arguments

This command has the following arguments:

**<file-name>**

The Tcl file name of a previously saved session.

### Examples

```
sg_shell> gui_restore_session abc.tcl
```

### See Also

[\*gui\\_save\\_session\*](#)

## new\_project

Creates a new project

### Syntax

```
new_project <project-name>
  [-projectwdir <project-path>]
  [-force]
```

### Scope

None

### Return Value

None

### Description

The *new\_project* command creates a new project with the specified name.

For design analysis, your first step inside *sg\_shell* should be to create a project. The project stores design setup information and the goal setup information, which is defined during an interactive *sg\_shell* session. After the project is saved, it can be loaded in subsequent *sg\_shell* sessions to start from where it was left in the earlier session.

As you run various goals inside the given project, *sg\_shell* saves the output on the disk inside the project working directory. By default, the project working directory is the same area from where the project file is picked. However, you can specify a different working directory by using the *-projectwdir* argument of this command. Inside the project working directory, *sg\_shell* creates a directory by the name of the project to store the output of various runs.

The *new\_project* command fails to execute if some other project is already open. You should close any existing project by using the [close\\_project](#) command before executing the *new\_project* command.

### Arguments

This command has the following arguments:

**<project-name>**

Specifies the name of the project to be created

**-projectwdir <project-path>**

Specifies the name of the project working directory where the project output would be created. You can use the same project work directory for multiple projects. `sg_shell` creates all its output directories inside the project directory.

All intermediate directories specified by using the `-projectwdir` argument should be already present, otherwise the `new_project` command would fail. In addition, the output area specified with the `-projectwdir` argument should have write permission.

**-force**

Removes an already existing project (both the project file and the project directory) and starts a new project with the same name.

**NOTE:** *Be careful while using the `-force` argument because you may lose any previous run data.*

**Examples**

```
sg_shell> new_project new -projectwdir new_dir
# will create a project file new.prj in cwd
# project output is stored inside new_dir/new
```

```
sg_shell> new_project new
new_project: error: please save and close the current project
before opening any new project
```

```
sg_shell> new_project new -projectwdir new_dir -force
```

**See Also**

[current\\_project](#), [open\\_project](#), [save\\_project](#), [close\\_project](#)

## open\_project

Opens an existing project

### Syntax

```
open_project <project-name>  
  [-projectwdir <project-path>]
```

### Scope

None

### Return Value

None

### Description

The *open\_project* command opens an existing project. You can use this command to open any Tcl-compliant project file, which is saved in an earlier *sg\_shell* or Atrenta Console session.

**NOTE:** *Tcl interface does not support a project file that is created prior to the SpyGlass 4.2.0 release, because it is not Tcl-compliant. To make this file Tcl-compliant, open that project file in Atrenta Console UI 4.2.0 or 4.2.1 and save it.*

When you open an existing project in *sg\_shell*, it brings you back to the same stage in which you last saved it. For example, in your last session, if you left at the *initial\_rtl/lint/simulation* goal inside the *GuideWare/New\_RTL* methodology, the *open\_project* command would restore the same stage and any data relevant for this goal would be loaded.

In case you have specified *link\_design* as the last command before saving the project, *sg\_shell* loads the *Design\_Read* goal on opening that project because *Design\_Read* is the goal associated with the *link\_design* command.

You can configure the top setting at the time of using the *open\_project* command by setting an appropriate top using the *set\_option top <top-name>* command inside the project file. By default, this command has the last active top at the time of saving the project. You can work on a

different top by changing this command as required.

It is recommended that you open the project from the same directory where it was last saved. This is required to take care of relative path references because these references would become invalid if you open the project from a different path. If there are relative paths in your project and a message is flagged about directory mismatch while opening it, you should perform the following steps:

1. Close the project.
2. Change your working directory as suggested in the message.
3. Open it again inside the `sg_shell`.

At the time of opening a project, `sg_shell` can flag some messages for dirty goals. Dirty goals are the goals in which run data is inconsistent with the setup information. This would happen if you have not saved the project in the last session after changes being made to the goal setup. It is recommended to rerun dirty goals, because their message database is otherwise out-of-sync with the saved setup information.

If there are any errors in the project file, the `open_project` command fails to execute. But, if there is any error in these older project files, the error messages would show the name of the new command name only. Refer to the [Appendix A: Deprecated Command Names and Their Corresponding New Commands](#) section for details of these command name changes. If you have an old project file, it is recommended that you open it in `sg_shell` and immediately save it to get the updated project file with new command names.

## Arguments

This command has the following arguments:

**<project-name>**

Specifies the name of the project file (.prj).

**-projectwdir <project-path>**

Specifies the project work directory where project run data would be created. If you do not specify this argument, `sg_shell` considers the project work directory as specified in the project file. If no project work directory is found in the project file, `sg_shell` considers that directory where the project file is present.

All intermediate directories in the *-projectwdir* specification should be already present, otherwise the *open\_project* command would fail. In addition, the output area specified by using the *-projectwdir* argument should have write permission.

## Examples

```
sg_shell> open_project myProject
# opens existing project file - myProject.prj

sg_shell> open_project myProject -projectwdir new_dir
# output is stored inside new_dir/myProject directory
```

## See Also

[\*new\\_project\*](#), [\*current\\_project\*](#), [\*save\\_project\*](#), [\*close\\_project\*](#)

## save\_project

**Saves the specified project**

### Syntax

```
save_project [<destination-project-file-name>][ -force]
```

### Scope

Project

### Return Value

None

### Description

The *save\_project* command saves the active project file. If the active project file does not have Write permissions, the *save\_project* command fails to execute. Optionally, you can also specify a destination project file name to save the current project. Otherwise, the current project is saved with the same name as the current project name. The *save\_project <destination\_project\_file\_name>* command will fail when the leaf level directory of the destination project file provided does not exist or does not have write permission.

If the project is a default project, the *save\_project* command does not save the project and informs you that there is no project to save. In case you want to save the default project, specify *save\_project -force <project\_name>.prj >* to save the project as *<project\_name>.prj*. Here the *<project\_name>* is an optional argument.

If the project is a read only project, the *save\_project* command does not save the project and informs you that there is no project to save. In case you want to save the read only project, specify *save\_project -force <project\_name>.prj >* to save the project as *<project\_name>.prj* (irrespective of whether *<project\_name>.prj* is the same as the original project. There is no backup of the original project.

If the project is not a read only project, the *save\_project* command saves the project and backup of the original project is created. In case you want to save the project with different file name, use the option

`save_project -force <project_name.prj>`. In this case, the backup of the original project would not be created. If you do not specify `<project_name.prj>` and use `save_project -force` option, the original project is saved without any backup.

In case a project is saved to a different project file, the current project is considered as saved only when the project file with which the project was opened does not have Write permissions. Otherwise, the project is considered as unsaved. When you attempt to close the project or exit the shell without saving the project, `sg_shell` prompts you to save the project.

## Arguments

This command has the following arguments:

**<destination-project-file-name>**

Specifies the project file name to be saved

**<-force>**

Saves the project forcefully.

## Examples

### Example 1

```
sg_shell> read_file -type hdl test.v
sg_shell> save_project
save_project: error: no project to save
sg_shell> save_project -force abc.prj
```

### Example 2

```
sg_shell> new_project new
sg_shell> current_methodology $::env(SPYGLASS_HOME)
GuideWare/New_RTL
sg_shell> read_file test.v
sg_shell> current_goal initial_rtl/lint/structure -top test
sg_shell> run_goal
sg_shell> close_project
close_project: warning: project still unsaved
```



Following goals were run but have not been saved  
-initial\_rtl/lint/structure (Top: test,  
Methodology: .)  
Run results and settings for these goals might  
not match on opening the project next time if you  
still want to close project, specify close\_project  
-force, or save the project first

```
sg_shell> save_project project_copy          # save this
           project with name as project_copy.prj
sg_shell> close_project
close_project: warning: project still unsaved
           Following goals were run but have not been saved
           - initial_rtl/lint/structure (Top: test,
           Methodology: .)
           Run results and settings for these goals
           might not match on opening the project next time
           if you still want to close project,
           specify close_project -force, or save the
           project first

sg_shell> save_project
sg_shell> close_project
```

### Example 3

```
sg_shell> open_project new.prj
sg_shell> set_option abc 1
sg_shell> save_project
save_project: info: sg_shell is going to save current project
              `abc.prj'. Some of the original content
              might be lost or reformatted. The original
              project file is saved as -
              abc_27Nov13_12.05.prj
```

### Example 4

```
sg_shell> open_project new.prj
```

```
sg_shell> set_option abc 1  
sg_shell> save_project -force
```

### **Example 5**

```
sg_shell> open_project new.prj  
sg_shell> set_option abc 1  
sg_shell> save_project -force b1.prj
```

### **See Also**

*[open\\_project](#), [current\\_project](#), [new\\_project](#), [close\\_project](#)*

## close\_project

**Closes the currently active project**

### Syntax

```
close_project  
[ -force ]
```

### Scope

Project

### Return Value

None

### Description

The *close\_project* command closes the currently active project. If there are any unsaved changes in the currently active project, this command prompts you to save those changes before closing the project. In addition, if there are any goals that were run but not saved, *sg\_shell* informs the same at the time of closing the project.

You can either save the project by using the *save\_project* command or you can forcefully close the project without saving the changes by using the *-force* argument of this command.

### Arguments

This command has the following arguments:

#### **-force**

(Optional) Closes the project forcefully even if it is not saved. If a project is closed forcefully, any settings done after the last save are lost. If some goals have been run but not saved, run results, and settings for such goals might not match on opening the project next time.

### Examples

```
sg_shell> new_project new  
sg_shell> current_methodology $::env(SPYGLASS_HOME)/
```

```
GuideWare/New_RTL
sg_shell> read_file test.v
sg_shell> current_goal initial_rtl/lint/connectivity -top
test
sg_shell> run_goal
sg_shell> close_project
close_project: warning: project still unsaved
    Following goals were run but have not been saved
    - initial_rtl/lint/connectivity (Top: test,
      Methodology:.)
    Run results and settings for
    these goals might not match on opening the
    project next time
    if you still want to close project,
      specify close_project -force, or save the
    project first
sg_shell> close_project -force
```

## See Also

*[new\\_project](#), [open\\_project](#), [save\\_project](#), [current\\_project](#)*

## current\_project

Displays data of the currently loaded project

### Syntax

```
current_project
```

### Scope

Project

### Return Value

Returns *<project name>* and *<project directory>* strings in an array

### Description

The *current\_project* command displays the name of the currently loaded project and working directory of that project.

### Examples

```
sg_shell> open_project myProject.prj -projectwdir new_dir
sg_shell> set prj [ current_project ]
myProject.prj new_dir
sg_shell> set prj_name [ lindex $prj 0 ]
myProject.prj
sg_shell> set prj_dir [ lindex $prj 1 ]
new_dir
sg_shell> puts "Project Name: $prj_name\nProject Directory :
$prj_dir
Currently selected project's info
-----

Project Name      : myProject.prj
Project Directory : new_dir
```

## See Also

*new\_project, open\_project, save\_project, close\_project*

## import\_project

Used to import existing project settings into current project

### Syntax

```
import_project <src_project>  
  -design|-goal <goal_name>  
  [-goal_constraint]
```

**NOTE:** *It is mandatory to specify either a goal or a design.*

### Scope

Project, Goal

### Return Value

None

### Description

The *import\_project* command is used to import existing project settings from a source project given through the *<src\_project>* argument into the current project.

All option values during import are retained as per the original specification, which includes the relative or absolute path. If `projectcwd` of original and current projects are different, it is recommended that you manually change all relative paths in imported goal-specific options to absolute, otherwise they may not get resolved properly. You would be alerted in such a scenario.

**NOTE:** *Importing design options and global settings from a source project to the current project is only allowed in global scope.*

### Arguments

The *import\_project* command has the following arguments:

#### **<src\_project>**

The *<src\_project>* argument is the first positional argument to the *import\_project* command and is a mandatory option. Use this argument to specify a source project name.

**-design**

Use this argument to import all design data, such as design files and all global options given with the *set\_option* command, from a source project to the current project.

**-goal <goal\_name>**

Use this argument to import settings from a given goal of a source project to a project or a currently selected goal. Because the *import\_project* command can be specified both outside a goal and after selecting a goal, specifying *<goal\_name>* would have the following impacts:

**If the *import\_project* command is specified after selecting a goal:**

- Parameter settings from the methodology, which contains the *<goal\_name>* goal, are imported from the source project in the currently selected methodology.
- Goal options and parameters for the *<goal\_name>* goal are imported in the currently selected goal.

**If the *import\_project* command is specified outside a goal:**

- Parameter settings from the methodology, which contains the *<goal\_name>* goal, are imported from the source project in the currently selected methodology.
- Goal options and parameters for the *<goal\_name>* goal are imported in the current project.

**-goal\_constraint**

Use this argument to import SGDC or ADC files from a goal given with the *-goal <goal\_name>* argument in a source project to a currently selected goal.

**NOTE:** *While importing settings for an option, the option value(s) is overwritten if the option is of scalar type. Otherwise, it is appended with the options of list type.*

**Examples****Example 1**

```
sg_shell> import_project src.prj -goal "connectivity"
```



```
sg_shell> import_project src.prj -goal connectivity -  
goal_constraint
```

### Example 2

```
source project file (src.prj) :
```

```
....  
read_file -type hdl test.v  
set_option lib L1 ./P1  
set_option libhdlfiles L1 { 1.v 2.v 3.v }  
...  
set_option projectcwd /my/mycwd/srcCwd  
.....
```

```
projectcwd: /my/mycwd
```

```
sg_shell> import_project srcCwd/project.prj -design  
import_project: error: `hdl' file `test.v' does not exist  
`libhdlfiles' file `3.v' does not exist  
import_project: info: projectcwd of current & source project  
are different  
design files & libmaps may be present relative to source  
projectcwd:  
`/my/mycwd/srcCwd'  
import_project: info: settings get partially imported to  
current project
```

### See Also

[set\\_goal\\_option](#), [new\\_project](#), [open\\_project](#), [save\\_project](#)

## exit

Quits `sg_shell` and returns to the UNIX shell prompt

### Syntax

```
exit [-save][-force][exit_code][-dest_prj <prj_file_name>]
```

### Scope

Any

### Return Value

None

### Description

The *exit* command is used to quit the `sg_shell` session and returns to the UNIX shell prompt.

If there are any unsaved changes in the currently active project, `sg_shell` prompts you to save those changes before closing the session. In case the project is a default project, it is saved without any message on the screen. Further, if there are any goals that were run but not saved, `sg_shell` informs the same at this point.

You can either save the project by using the *save\_project* command or forcefully close the session without saving the changes by using the *-force* argument of this command. Optionally, you can specify the *-save* argument to save the project while exiting. You will be prompted if the project can not be saved. At this point, you cannot exit unless you have specified the *-force* argument.

Additionally, you can also give a destination for the project where you can save your project before exiting by using options `-dest_prj <prjname>` along with `-save` option

Optionally, you can specify an integer code value to be returned to the calling UNIX shell.

### Arguments

This command has the following arguments:

## Session Commands

**-save**

Saves the current project if it is not saved.

**-force**

Exits `sg_shell` even if the current project is not saved.

**-dest\_prj**

Specifies the destination project file name to be saved.

**exit\_code**

Specifies the return code to the calling UNIX shell. The default value is 0.

**Examples****Example 1**

```
sg_shell> read_file -type hdl DEST.prj
read_file: info: using default project `spyglass-7.prj'
DEST.prj
sg_shell> exit
```

**Example 2**

```
sg_shell> new_project new
sg_shell> current_methodology $::env(SPYGLASS_HOME)/
GuideWare/IP_netlist
sg_shell> set_parameter abc 1
sg_shell> exit
exit: warning: project 'new.prj' is not saved
(Use 'save_project' to save it), or
(Use 'exit -save' to save the project before exiting), or
(Use `exit -save -dest_prj <project_name.prj>' to save the
project as project_name.prj before exiting), or
(Use 'close_project -force' to close it forcibly), or
(Use 'exit -force' to exit anyway, that is, without
saving the project)
sg_shell> exit -save
```

```
% echo $status  
0
```

### Example 3

```
sg_shell> new_project new1  
sg_shell> current_methodology $::env(SPYGLASS_HOME)/  
GuideWare/IP_netlist  
sg_shell> set_parameter abc 1  
sg_shell> exit  
exit: warning: project 'new1.prj' is not saved  
(Use 'save_project' to save it), or  
(Use 'exit -save' to save the project before exiting), or  
(Use `exit -save -dest_prj <project_name.prj>' to save the  
project as project_name.prj before exiting), or  
(Use 'close_project -force' to close it forcibly), or  
(Use 'exit -force' to exit anyway, that is, without  
saving the project)  
sg_shell> exit -force
```

### Example 4

```
sg_shell> new_project new1  
sg_shell> current_methodology $::env(SPYGLASS_HOME)/  
GuideWare/IP_netlist  
sg_shell> set_parameter abc 1  
sg_shell> exit  
exit: warning: project 'new1.prj' is not saved  
(Use 'save_project' to save it), or  
(Use 'exit -save' to save the project before exiting), or  
(Use `exit -save -dest_prj <project_name.prj>' to save the  
project as project_name.prj before exiting), or  
(Use 'close_project -force' to close it forcibly), or  
(Use 'exit -force' to exit anyway, that is, without  
saving the project)  
sg_shell> exit -save -dest_prj abc.prj
```

**Example 5**

```
sg_shell> exit 5
% echo $status
5
```

**See Also**

*save\_project, close\_project*

## set\_pref

**Sets the specified preference variable to its specified value**

### Syntax

```
set_pref <variable-name> <value>
```

### Scope

Any

### Return Value

Returns the preference value being set

### Description

The *set\_pref* command is used to set the specified preference variable to the specified value. The effect of the setting done is usually visible immediately after setting the variable. Though this command can be specified anywhere, the usual location is in *sg\_shell*'s startup file (i.e., `$HOME/sg_shell.startup` or `$CWD/sg_shell.startup`). Usually, the settings done through *set\_pref* commands are expected to remain for the entire session of *sg\_shell*, and that is why they make more sense in the startup files.

The list of recognized preference variables can be viewed by using the [help -preferences](#) command. In addition, help for any particular preference variable can be seen by using [help -preferences <variable\\_name>](#). Alternatively, you can enter `man <variable_name>` to see the details of a particular preference variable.

### Arguments

The *set\_pref* command has the following arguments:

**<variable-name>**

Use this argument to specify the variable name.

**<variable-value>**

Use this argument to specify the variable value.

**NOTE:** *Both the above arguments are mandatory.*

## Examples

The following example shows how to set the name of the command log file to a unique file name.

```
sg_shell> set_pref sh_command_log_file  
"sg_shell_command_[clock seconds].log"  
  
set_pref: info: command log file is now at  
'sg_shell_command_1255369133.log'  
  
(Previously, it was at './sg_shell_command.log')
```

**NOTE:** *The above informational message is going to be specific to each preference variable.*

## See Also

[get\\_pref](#), [help](#), [sh\\_command\\_log\\_file](#), [goal\\_enforce\\_prerequisite](#),  
[goal\\_show\\_hidden](#), [dq\\_design\\_view\\_type](#)

## get\_pref

**Displays the value of the specified preference variable (s)**

### Syntax

```
get_pref <variable-name>
```

### Scope

Any

### Return Value

Returns a list of variable or value pairs

### Description

The *get\_pref* command is used to display the variable or value pairs for known preference variables. If the *variable\_name* is not specified, then variable or value pairs for all those preference variables that have been set either through the startup file, the *sg\_shell* prompt, or any Tcl source file is returned in a list. Otherwise, if the *variable\_name* is specified, the variable or value pair for the specified preference variable is returned in a list. If a preference variable has not been set by the user, then its default value (as in the software) is returned as variable or value pair in a list. Currently, the use of wildcard characters is not supported in *variable\_name*.

### Arguments

This command has the following argument:

**<variable-name>**

Specifies the name of the variable

### Examples

The following example shows how to display the variable or value pairs.

```
sg_shell> get_pref  
sh_command_log_file ./sg_shell_command.log  
sg_shell> get_pref sh_command_log_file
```



```
sh_command_log_file ./sg_shell_command.log
sg_shell> set_pref sh_command_log_file ./new_file.log
set_pref: info: command log file is now at './new_file.log'
          (Previously, it was at './sg_shell_command.log')
sg_shell> set_pref sh_command_log_file ./new_file.log
sg_shell> get_pref
sh_command_log_file ./new_file.log
sg_shell> get_pref sh*
get_pref: warning: 'sh*' is not a recognized preference
variable
sg_shell>
```

## See Also

[set\\_pref](#), [help](#), [sh\\_command\\_log\\_file](#)

## Design Setup Commands

Design setup commands allow you to set up the design by providing design files, libraries, and design options. You can clean-up basic design read errors by updating the design data using these commands.

The following table describes the various design setup commands:

<b>Command</b>	<b>Description</b>
<i>read_file</i>	Reads in the specified file for analysis
<i>get_file</i>	Displays the names of various types of files added
<i>remove_file</i>	Removes files of the specified type
<i>set_option</i>	Sets the specified option to the specified value
<i>get_option</i>	Displays value(s) set in the project for the specified option
<i>remove_option</i>	Removes or unsets the specified option in the project scope
<i>link_design</i>	Reads the design to check for design read errors
<i>compile_design</i>	Synthesizes the design to check synthesis errors
<i>read_power_data</i>	Provides the UPF files
<i>read_sdc_data</i>	Provides the SDC files
<i>read_activity_data</i>	Provides the activity data files, such as FSDB, VCD or SAIF files

## read\_file

**Reads in the specified file for analysis**

### Syntax

```
read_file
  [-type <file-type>]
  <file-list>
```

To know more about specifying the design files to be read during SpyGlass analysis in the Atrenta Console, refer to the *read\_file* section in the *Atrenta Console Reference Guide*.

### Scope

Project, Goal

### Return Value

Returns a list of files that are read

### Description

The *read\_file* command provides the various types of files to be read by *sg\_shell*. Currently, all files, except *sourcelist* type files, are not read immediately but rather stored in some object model. These files are read when you run a goal. Files of the *sourcelist* type are read at the time of command execution itself.

All files are applicable to the currently selected methodology and goal as well as to any methodology or goal that would be selected in the future. However, for SGDC and waiver files, if you have specified the *read\_file* command after the *current\_goal* command, *sg\_shell* adds the specified SGDC file to the currently selected goal only, and not to the other goals. If you specify the *read\_file -type sgdc* command outside the *current\_goal* command, *sg\_shell* adds SGDC files to the complete project and these SGDC files would be applicable to any goal in the current project.

**NOTE:** *Earlier the SGDC files were read immediately on execution of the command. Now, they are read during the run\_goal execution.*

## Arguments

The `read_file` command has the following arguments:

### **-type <file-type>**

Use this argument to specify the type of file to be read. This argument accepts any of the following values:

Type	Description
verilog	Specifies Verilog files
vhdl	Specifies VHDL files
def	Specifies DEF files
hdl	Specifies HDL files (Verilog, VHDL, or DEF)
gateslib	Specifies gateslib files
gateslibdb	Specifies gateslib db files
sglib	Specifies sglib files
lef	Specifies LEF files
plib	Specifies plib files
sgdc	Specifies SGDC files
waiver	Specifies waiver files
sourcelist	Specifies command files. Only options allowed in the <code>set_option</code> command and file types allowed in the <code>read_file</code> command are read from the files passed. If there is any syntax error in some file due to an error in a command file or some undefined variable in the file, contents of this file are dropped, and no further files passed in the command are read.

**NOTE:** *The above options are not mandatory. If you do not specify any of these options, the file type is assumed to be HDL.*

### **<file-list>**

Use this argument to specify a space-separated list of file names.

The file paths specified in this argument can be relative or absolute. If the file paths are relative, the current `sg_shell` working directory should match with the project current working directory at the time of opening the project. All relative paths should be relative to a single base working

directory, that is, the project's current working directory.

The file paths can be specified as wildcard patterns containing wildcard characters (\*, ?). If you want to refer to two files, for example "abc1d.v" and "abc2d.v", you can specify them using SpyGlass pattern matching support, that is, you can specify `read_file -type hdl abc*d.v` in this case.

**NOTE:** *If your filename includes wildcard characters (\*, or ?), the name should be enclosed in curly brackets, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your filename is "abc\*d.v", you need to specify it as "read\_file -type hdl {{abc\*d.v}}".*

The following sanity checks are performed on the files that are specified in the `read_file` command:

- File should exist
- File path should not be a directory
- File should have read permissions for the current user
- If wildcard patterns are specified, then they should match at least one file

If any of the sanity checks fail, an error is flagged and the current `read_file` command is ignored. Any file that has been already added or any duplicate entries in the current `read_file` command is ignored and a message will be displayed.

## Examples

```
sg_shell> read_file test.v mid.v bot.v
sg_shell> read_file test.vhd
sg_shell> get_file -type hdl
test.v mid.v bot.v test.vhd
```

```
sg_shell> read_file -type gateslib cells.lib
sg_shell> get_file -type gateslib
cells.lib
```

```
sg_shell> read_file -type sgdc global.sgdc
sg_shell> get_file -type sgdc
global.sgdc
sg_shell> current_goal initial_rtl/lint/structure
```

```
sg_shell> read_file -type sgdc local.sgdc
sg_shell> get_file -type sgdc
global.sgdc local.sgdc
sg_shell> current_goal none
sg_shell> get_file -type sgdc
global.sgdc
```

## See Also

[remove\\_file](#), [get\\_file](#)

## get\_file

**Displays the names of various types of files added**

### Syntax

```
get_file
  -type <file_type>
```

### Scope

Project, Goal

### Return Value

An array of strings specifying a list of files of the specified type

### Description

The *get\_file* command displays the names of various types of files added by using the [read\\_file](#) command. For any file other than the SGDC type of file, the *get\_file* command displays the names all the files of a given type as present in the project, and is not context sensitive.

After selecting a goal, if you specify the value of the *-type* argument as *sgdc*, *sg\_shell* displays the names of SGDC files that are valid for the selected goal only. However, if you specify outside a goal, *sg\_shell* displays the names of all the global SGDC files present in the project.

### Arguments

The *get\_file* command has the following arguments:

#### **-type <file\_type>**

Use this argument to specify the type of files to be displayed. You can specify any of the following values to this argument:

Type	Description
hdl	Displays all the HDL files (Verilog, VHDL, or DEF)
gateslib	Displays gateslib files
sglib	Displays sglib files

---

lef	Displays LEF files
plib	Displays plib files
sgdc	Displays SGDC files in given scope (project or goal)
waiver	Displays waiver files

---

**NOTE:** *This option is mandatory.*

## Examples

```
sg_shell> read_file test.v
sg_shell> read_file test1.v
sg_shell> set x [get_file -type hdl]
test.v test1.v
sg_shell> puts $x
test.v test1.v
sg_shell> set y [lindex $x 0]
test.v
sg_shell> puts $y
test.v

sg_shell> read_file -type sgdc g1.sgdc
# global sgdc file
sg_shell> read_file -type sgdc g2.sgdc
# global sgdc file

sg_shell> current_goal initial_rtl/lint/simulation
sg_shell> read_file -type sgdc a.sgdc
# local sgdc file
sg_shell> read_file -type sgdc b.sgdc
# local sgdc file
sg_shell> get_file -type sgdc
g1.sgdc g2.sgdc a.sgdc b.sgdc

sg_shell> remove_file -type sgdc g1.sgdc b.sgdc
# remove global and local sgdc files
sg_shell> get_file -type sgdc
g2.sgdc a.sgdc
```



**See Also**

*[read\\_file](#), [remove\\_file](#)*

## remove\_file

**Removes files of the specified type**

### Syntax

#### Usage 1

```
remove_file -type <file-type>
```

#### Usage 2

```
remove_file -type sgdc [<file-list>]
```

To know more about removing design files during SpyGlass analysis in the Atrenta Console, refer to the *remove\_file* section in the *Atrenta Console Reference Guide*.

### Scope

Project, Goal

### Return Value

**Usages 1 and 2:** Returns a list of files that are removed

### Description

The *remove\_file* command removes files that have been added by using the *read\_file* command. Except for the SGDC type of file, the *remove\_file* command removes all the files of a given type from the project, and it is not context-sensitive.

Optionally, in the case of SGDC types of files, you can provide a list of files (*<file-list>*) to be specifically removed. The *remove\_file -type sgdc* command is context-sensitive. If this command is specified after selecting a goal, *sg\_shell* removes all or given SGDC files from the selected goal only. These removed files can be both inherited from the project and added locally to a goal. However, if this command is specified outside a goal and without a list of files, *sg\_shell* removes all the SGDC files that have been added globally. SGDC files locally added to a goal are not removed. If this command is specified outside a goal with a list of files, it removes the specified SGDC files from the whole project and from all its goals even if these SGDC files were added locally to a goal.

**NOTE:** *The impact of removing the AWL file is seen immediately while the impact of*

removing the SGDC/waiver files is seen during/after the `run_goal` execution.

## Arguments

The `remove_file` command has the following arguments:

### **-type <file-type>**

Use this argument to specify the type of file to be removed. The following is the list of file types that can be specified with this argument:

Type	Description
hdl	Removes all HDL files (Verilog, VHDL, or DEF). <b>NOTE:</b> You cannot remove individual HDL file types, such as Verilog or VHDL files. The complete HDL file set is removed.
gateslib	Removes gateslib files
sglib	Removes sglib files
lef	Removes LEF files
plib	Removes plib files
sgdc	Removes all or given SGDC files
waiver	Removes waiver files

**NOTE:** *This option is mandatory.*

### **<file-list>**

Use this argument to specify a list of SGDC files to be removed. The list of files is not allowed with any other file type other than SGDC. You will be informed in case any of the file specified as part of the file-list has not been already added by a corresponding [read\\_file](#) command.

The file paths can be specified as wildcard patterns containing wildcard characters (\*, ?). If you want to refer to two files, for example, "abc1d.sgdc" and "abc2d.sgdc", you can specify them using SpyGlass pattern matching support, that is, you can specify as `remove_file -type sgdc abc*d.sgdc` in this case.

**NOTE:** *If your filename includes wildcard characters (\*, or ?), the name should be enclosed in curly brackets, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your filename is "abc\*d.sgdc", you need to specify it as "remove\_file -type sgdc {{abc\d.sgdc}}."*

## Examples

```
sg_shell> read_file test.v
sg_shell> read_file test.vhd
sg_shell> remove_file -type hdl
sg_shell> read_file test.v
sg_shell> get_file -type hdl
test.v
sg_shell> read_file -type sgdc global1.sgdc
# global sgdc file
sg_shell> read_file -type sgdc global2.sgdc
# global sgdc file
sg_shell> get_file -type sgdc
global1.sgdc global2.sgdc
sg_shell> current_goal initial_rtl/lint/structure
sg_shell> read_file -type sgdc local1.sgdc
# local sgdc file
sg_shell> read_file -type sgdc local2.sgdc
# local sgdc file
sg_shell> get_file -type sgdc
global1.sgdc global2.sgdc local1.sgdc local2.sgdc          #
both global and local sgdc files
sg_shell> remove_file -type sgdc global2.sgdc local2.sgdc
# global2.sgdc removed from current goal
sg_shell> get_file -type sgdc
global1.sgdc local1.sgdc
sg_shell> current_goal none
sg_shell> get_file -type sgdc
global1.sgdc global2.sgdc
sg_shell> remove_file -type sgdc
# removes global sgdc files
sg_shell> read_file -type sgdc global3.sgdc
sg_shell> get_file -type sgdc
global3.sgdc
sg_shell> current_goal initial_rtl/lint/structure
sg_shell> get_file -type sgdc
global3.sgdc local1.sgdc
sg_shell> current_goal none
```

## Design Setup Commands

```
sg_shell> remove_file -type sgdc local1.sgdc
# removed local1.sgdc from goal 'initial_rtl/lint/structure'
as well
sg_shell> current_goal initial_rtl/lint/structure
sg_shell> get_file -type sgdc
global3.sgdc
```

**See Also**

[read\\_file](#), [get\\_file](#)

## set\_option

**Sets the specified option to the specified value**

### Syntax

```
set_option <option-name> <value>
```

### Scope

Project

### Return Value

Returns the value being set

### Description

The *set\_option* command sets the specified option to the specified value.

The settings specified by this command are applicable for all goals. To set an option for a particular goal scope, use the [set\\_goal\\_option](#) command.

The option value can be of the following types:

Type	Description
Boolean	If the option is of the Boolean type, the value for that option can be <code>&lt;true on yes 1&gt;</code> , which is evaluated to <code>true</code> . Similarly, the value <code>&lt;false off no 0&gt;</code> is evaluated to <code>false</code> . If more than one value is specified for the Boolean option, the last value is considered as the final value. You will get a warning about the multiple values specified in the Boolean option.
List	If the option is of type list, the values will be appended in the form of a string list. The option value should be specified in curly brackets <code>{ }</code> .
String	If the option is of type, string, and you specify more than one value for the given string option, then the last value will be considered as the final value. You will get a warning about the multiple values specified in the scalar option.

If the option specified by using the [set\\_option](#) command is already set and the option is of list type, `sg_shell` appends the value specified in the

option's value-list. Otherwise, it overwrites the existing option value.

The following sanity checks are performed on value(s) passed before an option is set:

- Invalid value specified in a Boolean option
- More than one value specified in a Boolean or scalar option
- Invalid enum value specified in an enum type of option
- String type of value specified for integer or float type of option
- Out of range value specified for options that accept a range of values
- More than two values specified for options that accept a pair of values, such as `lib` and `define_incr_dirmap`.

The following file or directory checks are performed for file or directory type of options:

- File or directory does not exist
- Read or write permissions are not there on the file directory
- File given for directory or vice versa
- Directory specified for directory type of option is empty
- File or directory that has been already added or duplicate entries in value list are ignored for list type of options
- Wildcard pattern used to specify file/directory does not match any file/directory

Above checks can be either of the Error or Warning severity. If check is flagged with Error severity, the current *set\_goal\_option* is ignored. Otherwise, the option will be set as per the values passed in the command.

The remaining sanity checks as done by SpyGlass are performed at the time of the *run\_goal* command.

To get a list of options that can be set by using the *set\_option* and *set\_goal\_option* commands, use the following command:

```
help -option
```

There are a few options, such as `check_celldefine`, that can be set by both the *set\_option* and *set\_goal\_option* commands, because these options can be set both for the entire project and for a particular goal. These options are listed twice by the *help -option* command, once with the list of options that can be set by the *set\_option* command and again with the

list of options that can be set by the [set\\_goal\\_option](#) command. In addition, you can also view man pages for each option in `sg_shell` by using the `man <option-name>` command.

**NOTE:** *An alias name, if available for a particular option, is not displayed in the corresponding man page or in the list of options displayed by using the [help -option](#) command.*

If there is a Boolean option set in global scope for all goals using [set\\_option](#), it cannot be unset in goal scope using [set\\_goal\\_option](#).

For example, consider the following:

```
sg_shell> set_option ignorelibs yes
sg_shell> current_goal initial_rtl/lint/synthesis
sg_shell> set_goal_option ignorelibs no
sg_shell> run_goal
```

In the above case, even if `ignorelibs` is being set to `no` inside goal scope, it would still be treated as `yes` during `run_goal`.

It is currently recommended that if a specific Boolean option is intended to be turned on/off on per goal basis, then set it inside goal scope only, and not have it set globally using [set\\_option](#).

The following table lists various options that you can set by using this Tcl command. The alias name, if available for a particular option, is written in parentheses.

Option	Description
87	Checks the design for IEEE standard 1076-1987 compliance
DEBUG	Decompiles debug information about various stages during a SpyGlass run
I	Specifies the directories search path
LICENSEDEBUG	Prints license debug information on the screen
addrules (addrule)	Selectively adds a rule in current goal



Option	Description
aggregate_report	Specifies the aggregate report format type(s) to be generated in batch console mode
aggregate_reportdir	Specifies the directory path where the data for the aggregate report gets generated
aggregate_report_config_file	Specifies the aggregate report config file to be used
allow_celldefine_as_top	Specifies to perform rule checking on 'celldefine module top's hierarchy
allow_module_override	Allows duplicate module or UDP definitions
auto_save	Enables save of design query data as part of <i>run_goal</i> command
auto_restore	Enables restore of design query data as part of <i>current_goal/open_project</i> commands
block_abstract_directory	Specifies a directory in which the abstract view of a block should be saved
cachedir	Specifies the directory where library compilation will be performed
cell_library (cell_libraries)	Skips rule checking on design units loaded from precompiled libraries
check_celldefine	Turns on the rule checking on all the celldefine modules
check_if_else_reset	Detects the "sync preset" even if "sync clear" is present in an always block or vice-versa
checkdu	Specifies the design hierarchy (level) for rule checking
checkip	Specifies the design units for rule checking
convert_udp_to_latch	Enables SpyGlass to infer UDP as a latch while translating the UDP with both edge and level sensitiveness
default_adc_file	Specifies a default file to save the ADC constraint applied on <i>sg_shell</i> . If you do not specify a file by using this command, SpyGlass considers the <i>&lt;project-wdir&gt;/&lt;project_name&gt;_adc_file.adc</i> file as the default ADC file.

Option	Description
default_waiver_file	Specifies a default waiver file for saving interactive waiver commands. If you do not specify a default waiver file by using this command, SpyGlass considers the <i>&lt;project-wdir&gt;/&lt;project_name&gt;_waiver_file.awl</i> file as the default waiver file.
define	Adds the specified macro definitions
define_cell_sim_depth	Specifies the threshold limit for the size of the macro in order to completely simulate the macro. The value can range from 1 to 30, inclusive of the range.
define_incr_dirmap	Provides mapping for different locations of RTL files
designread_disable_flatten	Disables flattening during the <i>compile_design</i> command in <i>sg_shell</i> . Also disables flattening while opening a project if the project was closed with a flattened view
designread_synthesis_mode	Specifies mode in which synthesis should be performed
disable_encrypted_hdl_checks	Disables RTL rule checking on the encrypted design units
disable_hdllibdu_lexical_checks	Disallows lexical rule checking on precompiled libraries
disable_hdlin_synthesis_off_skip_text	Disables the interpretation of VHDL design code between Synopsys <i>synthesis_off/synthesis_on</i> pragma pair as comments
disable_hdlin_translate_off_skip_text	Disables the interpretation of VHDL design code between Synopsys <i>translate_off/translate_on</i> pragma pair as comments
disable_infer_async_rst_latches	By default, SpyGlass synthesis infers latches with asynchronous set/reset. When this command is set for specified modules, SpyGlass synthesis infers latches without asynchronous set/reset, for such modules.
disable_sgdc_dump	Disables the conversion of certain SDC commands to their respective SGDC commands in the <i>sdc2sgdc</i> flow

Option	Description
disallow_view_delete	Disables the Large Design Processing Mode
dump_precompile_builtin	Saves design parsing builtin messages in precompile dump
dw	Enables DesignWare <sup>®</sup> module support. SpyGlass can expand DesignWare <sup>®</sup> modules and generate logic for these modules during SpyGlass analysis. For further details, refer to the <i>Working with DesignWare<sup>®</sup> Modules</i> section in the <i>Atrenta Console User Guide</i>
elab_precompile	Enables elaboration in single step precompilation
enableSV	Enables parsing of SystemVerilog constructs
enableSVA	Enables parsing of SystemVerilog constructs and also handles SV Assert logic. By default, the value of this option is set to 0 and SpyGlass reports SystemVerilog constructs as syntax errors.
enable_abstract_block_schematic	Enables schematic debugging of abstracted modules. This option accepts abstracted module names, for which you want to view schematic to debug issues in the abstracted IPs of SoC. You can specify wildcard characters in the module names as an argument value of this option.
enable_const_prop_thru_seq	Allows constant propagation beyond sequential elements during logic simulation
enable_gateslib_autocompile	Enables automatic compilation of Synopsys Liberty <sup>™</sup> files (.lib files) to a SpyGlass-compatible library file form (.sglib files)
enable_hdl_encryption	Enables encryption of VHDL or Verilog libraries during compilation

Option	Description
enable_hier_flattening	Enables hierarchical flattening during the <i>compile_design</i> or <i>run_goal</i> command in <i>sg_shell</i> . By default, hierarchical flattening is off. Compile your design by using this option if you want to use design query commands on the hierarchical netlist. Using this option may consume more time and memory during the flattening stage of the design.
enable_inactive_rtl_checks (enable_inactive_rtl_check)	Enables semantic checking capability in SpyGlass
enable_pass_exit_codes	Causes SpyGlass to print more detailed exit status codes and messages
enable_pgnetlist	Enables power and ground pin information to be taken into consideration from specified physical libraries
enable_precompile_vlog	Enables the Precompiled Verilog library feature
enable_save_restore	Enables the Design Save or Restore feature
enable_save_restore_builtin	Restores design parsing, elaboration, and synthesis messages in restore run. By default, this option is ON in <i>sg_shell</i>
enable_sglib_debug	Provides SpyGlass Library Compiler debug Information
force_compile	Forces compilation of libhdlfiles
force_gateslib_autocompile	Forces automatic compilation of Synopsys Synopsys Liberty™ files (.lib files) to a SpyGlass-compatible library file form (.sglib files)
gen_block_options	Generates the block dependency report
gen_block_expand_lib_sources	Lists specification information about source files, as comments, for precompiled <i>libs</i> , <i>incdir</i> , and <i>-y/v</i> directories. It also filters out irrelevant <i>.lib</i> specifications for the specified design unit or a block.
gen_blk_sgdc	Enables generation of an abstract view of a block

Option	Description
handlememory	Specifies to process memories in an optimized manner
hdlin_synthesis_off_skip_text	Interprets VHDL design code between Synopsys synthesis_off or synthesis_on pragma pairs as comments
hdlin_translate_off_skip_text	Interprets VHDL design code between Synopsys translate_off or translate_on pragma pair as comments
hdlilibdu	Enables RTL and lexical rule checking on precompiled Verilog or VHDL design units
higher_capacity	Disables rules designed for SpyGlass version 3.2.0
ignore_builtin_rules	Disables SpyGlass built-in rules
ignore_builtin_spqdir	Specifies directory in which the ignore_builtin-<language>.spq file (containing <i>ignorerule</i> specifications of SpyGlass built-in rules) exists
ignoredu	Ignores the specified VHDL design unit or the Verilog module at the design read (parsing) stage
ignorefile	Ignores the specified design file at the design read stage
ignoredir	Ignores design files in the specified directory and its subdirectories at the design read stage.
ignorelibs (ignorelib)	Skips the rule checking for modules in the library files specified through v/y option
ignorerules (ignorerule)	Specifies the rule names or rule group names to be ignored in rule checking
ignore_undefined_rules (ignore_undefined_rule)	Continues after issuing a warning message if an undefined rule is specified
ignorewaivers (ignorewaiver)	Causes SpyGlass to ignore waivers supplied as embedded SpyGlass Waiver pragmas
incdir	Searches the specified path for include files
inferblackbox	Infers black box module interface based on the black box instances in the synthesized netlist

Option	Description
inferblackbox_iterations	Specifies the effort in terms of the total number of iterations that SpyGlass should make before finalizing port directions for black boxes
inferblackbox_rtl	Infers black box module interface based on the black box instances in the RTL description in addition to the synthesized netlist
infer_enabled_flop	By default synthesis creates flip-flops with enable wherever possible. When this switch is turned off, synthesis creates flops without enable together with a MUX to get enable functionality
lang	Specifies the display language for messages and waivers
language_mode	Specifies language specification to use
lib	Defines the logical to physical mapping for referenced libraries
libext	Specifies library file extensions
libhdlf	Specifies mapping between specified logical library and source-files for precompiled library use
libhdlfiles (libhdlfile)	Specifies mapping between specified logical library and HDL files for precompiled library use
libmap	Defines the logical to intermediate library mapping for referenced libraries
lvpr	Specifies the maximum number of messages to report
macro_synthesis_off	Turns off the macro SYNTHESIS during design read. By default, the macro is set on.
mthresh	Specifies the bit-count threshold for the compilation of net or variables in a design unit
net_osc_count_limit	Overrides the oscillation limits for nets. By default, the oscillation count is set to 100.
nobb	Forces SpyGlass to exit without processing if any black box is found in the design

<b>Option</b>	<b>Description</b>
nodefparam	Ignores explicit parameter re-definition given by defparam Verilog construct
noelab	Exits after design analysis and without elaborating the design
noispy	Suppresses the generation of module schematic data
nopreserve	Forces SpyGlass to remove hanging or unconnected instances and nets
noreport	Suppresses report generation
norules (norule)	Suppresses rule checking
nosavepolicy (nosavepolicies)	Specifies the product or products that should not be saved during design save
nosch	Suppresses the generation of schematic data by rules
oem_mode	Runs Atrenta Console GUI in the OEM (Original Equipment Manufacturer) mode
operating_mode	Enables setting value of special variable SG_OPERATING_MODE for SGDC conditional compilation
overloadrules (overloadrule)	Overloads the severity or weight of a rule
param	Sets the new user-specified values of the VHDL generics and Verilog parameters
perflog	Specifies to generate the SpyGlass performance log
pragma	Specifies a prefix-string for synthesis directives
prefer_tech_lib	Sets higher preference for technology library definitions over HDL definitions while resolving master of instances
preserve_mux	Enables SpyGlass to pick MUX cells from the technology library

Option	Description
prohibit_waiver	Prohibits waiver from waiving the violation of specified rules without any need to edit the waive commands in the pre-existing waiver files. This option accepts a list of rules.
print_sortorder_only	Prints the list of sorted VHDL files and exits
relax_hdl_parsing	Performs relaxed VHDL semantic checking
remove_work	Deletes the contents of the WORK directory
report	Specifies the report format type to be generated
report_command_mismatch	Enables strict checking on a command given in constraints (ADC) and waiver file on sg_shell. On enabling this option, if constraint is specified in a waiver file or waiver is specified in a constraint file through the <a href="#">read_file</a> -adc   sgdc   waiver  awl and <a href="#">convert_sgdc2adc</a>   <a href="#">convert_swl2awl</a> commands, TCL_ERROR is returned as the return status of the command.
report_incr_messages	Enables reporting of incremental messages
report_inst_backref	Prints back-reference information, such as file name and line number, containing definition for a design and its instances in the elab_summary.rpt report
report_max_size	Specifies the maximum number of messages for sorted reports
report_style	Enables customization of report format
report_unreachable_default_case	Use this option to suppress the SYNTH_5039 messages reported by SpyGlass Synthesis for scenarios in which case-conditions are complete and case-default statement is unreachable
resetall	Resets the Verilog compiler directive default_nettype to language default, which is wire



Option	Description
rules (rule)	Specifies the list of rules or rule group names for which rule checking should be done
sca_on_net	Use this option to apply set_case_analysis constraints on a net when the port/pin name matches the net name. This impacts the logic cone inside the hierarchy, which is driven by the net connected to pin/port.
savepolicy (savepolicies)	Specifies the products that run during design restore
sdc2sgdc	Enables the SDC-to-SGDC feature. The default value of sdc2sgdc option is no. However, for sg_shell runs, SpyGlass automatically sets this option to yes, when at least one SDC file is specified using the sdc_data constraint.
sdc2sgdc_mode	Specifies the mode of the SDC file to be translated to SGDC
sdc2sgdcfile	Specifies the file to save the output of SDC-to-SGDC translation
sfcu	Enables each file to be compiled as a separate compilation unit
sgsyn_clock_gating	Causes SpyGlass to create a simple clock gating logic instead of creating a MUX-based enable logic
sgsyn_clock_gating_threshold	Specifies threshold number (default 16) of flip-flops beyond which SpyGlass creates a simple clock gating logic
sgsyn_loop_limit	Specifies the loop rolling limit during design synthesis
show_lib	Enables generation of messages for each library module
skip_rules_for_fast_restore	Enables skip of design's re-parsing and/or resynthesis during design restore

Option	Description
sort	Sorts and prints the design files before analyzing
sortrule (sortrules)	Specifies the sort order for messages in SpyGlass reports
stop	Skips rule checking on the specified VHDL design unit or Verilog Module
stopdir	Skips rule checking on all design units located in a specified directory
stopfile	Skips rule checking on all design units described in a specified file
support_sdc_style_escaped_name	Enables Synopsys-style escaped names in SpyGlass Design Constraints files
target	Specifies libraries to be used for technology mapping out of the specified SGLIB libraries
top	Specifies top of the design for sg_shell
treat_priority_pin_as_obs	Use this command to consider the highest priority asynchronous pin of sequential block as observable/unblocked
unify_sdc2sgdc	Enables unification of mutually exclusive information from different sources, SDC and SGDC.
use_du_sch_hier	Enables SpyGlass to use schematic highlight information of a violation to waive violations on a design unit.
use_goal_rule_sort	Sorts violation messages in SpyGlass reports based on the order of rules specified in a goal file
use_scan_flops (use_scan_flop)	Enables SpyGlass to pick scan flip-flops from the technology library
v	Specifies the library file used in the source design
vlog2001_generate_name	Specifies if all Verilog generate statements in the Verilog 2001 syntax should be unrolled
vlog2005_irm_naming	Specifies whether to unroll all Verilog generate statements in the Verilog 2005 syntax.

Option	Description
w	Turns on generation of warnings for PERL-level compilation
waivers_translate_generate_names	Enables the use of a non escaped "generate block" name or an "instance array" name in the -msg field of the waive command
work	Specifies the logical library directory for compilation of Verilog or VHDL libraries
y	Specifies the library directory containing libraries

## Arguments

The *set\_option* command has the following arguments:

### <option-name>

Use this argument to specify the name of the option that needs to be set.

### <value>

Use this argument to specify the value of the option being set.

## Examples

To generate the summary and inline reports, the *report* option can be set by using the following command:

```
sg_shell> set_option report { "summary" "inline" }
```

To enable SystemVerilog constructs in Verilog files, use the following command:

```
sg_shell> set_option enableSV 1
```

Consider an SoC having mtscore, mts\_block1, mts\_block2 and mts\_blk abstracted IPs. To view the schematic of abstracted blocks, mtscore and mts\_blk, use the following command

```
sg_shell> set_option { mtscore mts_blk }
```

To view the schematic of abstracted blocks having prefix, mts\_..., use the following command:

```
sg_shell> set_option  
enable_abstract_block_schematic{ mts_* }
```

To view the schematic of all the abstracted blocks, use the following command:

```
sg_shell> set_option enable_abstract_block_schematic *
```

## See Also

[set\\_goal\\_option](#), [get\\_option](#), [get\\_goal\\_option](#), [get\\_run\\_option](#), [remove\\_option](#),  
[remove\\_goal\\_option](#)

## get\_option

Get values set in the project for the specified option

### Syntax

#### Usage 1

```
get_option
```

#### Usage 2

```
get_option <option-name>
```

### Scope

Project

### Return Value

**Usage 1:** Returns a list of option or value pairs

**Usage 2:** Returns the value set for the specified option, *<option-name>*

### Description

The *get\_option* command displays the values set for the specified option. If the option name is not given, *sg\_shell* returns a list of option-value pairs set in the project scope. Otherwise, it returns the value of the specified option.

If the option value is not set by using the *set\_option* command, *sg\_shell* displays the default value, if present, for the specified option. If there is no default value for that option, *sg\_shell* displays an information message on the screen informing you that the option has not been set. The control is then returned to the *sg\_shell* prompt. If you specify the wrong option name, *sg\_shell* displays an error message.

### Arguments

The *get\_option* command has the following arguments:

**<option-name>**

Use this argument to retrieve the value of the specified option.

## Examples

```
sg_shell> set_option report { "simple" "drag" }  
sg_shell> puts [ get_option report ]  
simple drag  
sg_shell> set_option report { inline }  
sg_shell> puts [ get_option report ]  
simple drag inline
```

## See Also

[set\\_option](#), [set\\_goal\\_option](#), [get\\_goal\\_option](#), [get\\_run\\_option](#), [remove\\_option](#),  
[remove\\_goal\\_option](#)

## report\_option

### Report options in tabular format

### Syntax

#### Usage 1

```
report_option [get_option]
```

#### Usage 2

```
report_option [get_goal_option]
```

#### Usage 3

```
report_option [get_run_option]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *report\_option* command displays options in a tabular format. You can specify the following commands without arguments as input for this command:

- *get\_option*: Use this command as an argument to display options set in the project in a tabular format.
- *get\_goal\_option*: Use this command as an argument to display options set in the goal scope in a tabular format.
- *get\_run\_option*: Use this command as an argument to display options to be used during the current run in a tabular format.

**NOTE:** *One of the above commands is mandatory to provide as an input to this command.*

### Arguments

None

## Examples

```
sg_shell> report_option
report_option: error: invalid usage (mandatory options
missing, or wrong combination of options)
Try 'report_option -help' for more details.
sg_shell> get_option
language_mode mixed enable_save_restore 1
enable_save_restore_builtin true
sg_shell> report_option [ get_option ]
Options set in current context:
+++++
Option Name                Current Value
=====
language_mode              mixed
enable_save_restore        1
enable_save_restore_builtin true
+++++
sg_shell>
```

## See Also

[get\\_option](#), [get\\_goal\\_option](#), [get\\_run\\_option](#), [report\\_parameter](#), [get\\_parameter](#)



## remove\_option

Removes or unsets the specified option in the project scope

### Syntax

```
remove_option <option-name>
```

### Scope

Project

### Return Value

Returns the option value set before it is unset.

### Description

The *remove\_option* command removes or unsets the specified option from the project scope. *sg\_shell* does not perform any sanity check on the specified option name. Therefore, if you provide an incorrect option name, *sg\_shell* does not print anything on the screen and returns the control back to the *sg\_shell* prompt.

To remove an option from the scope of a particular goal, specify the [remove\\_goal\\_option](#) command.

### Arguments

The *remove\_option* command has the following argument:

**<option>**

Use this argument to specify the name of an option that needs to be removed or unset.

### Examples

The following example unsets the wrongly set *mtthresh*:

```
sg_shell> set_option mtthresh "abcd"  
sg_shell> remove_option mtthresh  
sg_shell> get_option mtthresh  
4096
```

**See Also**

*set\_option, set\_goal\_option, get\_option, get\_goal\_option, get\_run\_option, remove\_goal\_option*

## link\_design

Reads the design to check design read errors

### Syntax

#### Usage 1

```
link_design -top <top-name>
```

#### Usage 2

```
link_design -alltop
```

#### Usage 3

```
link_design
```

### Scope

Methodology

### Return Value

**Usages 1–3:** Returns 'exit\_code {error string}' pair

**NOTE:** Refer to the [Exit Codes Reported by sg\\_shell](#) section for more information on exit codes generated by `sg_shell`.

### Description

The `link_design` command reads the complete user design as per design files and options settings. This includes analyzing RTL files and performing elaboration of the design.

You cannot execute this command in scope of a goal. You first need to select the `current_goal` command with `none` before executing this command. The `link_design` command internally invokes the `Design_Read` goal and runs it.

You can configure the design hierarchy to be analyzed by using the `link_design` command in the following manner:

- If you want to analyze a specific top-level module hierarchy, specify that top-level module by using the `-top` argument of this command.
- If your design has multiple top-level modules and you want to analyze all of them, specify the `-alltop` argument of this command.

- If you want to focus on a specific top-level module during the current Tcl session, it is best to set that top-level module by using the `set_option top <top-name>` command. After the top-level module is set, the `link_design` or `current_goal/run_goal` commands automatically work on this top-level module only. Therefore, you need not explicitly specify this top-level module in each of these commands.
- If you want to determine a top-level module in your design in case the top-level module is not specified by using the `set_option top <top-name>` command, use the `link_design` command.

This command enables you to determine multiple top-level modules in your design so that you can set one top-level module for subsequent `current_goal` commands.

In case there is a single top-level module in the design, it would be automatically set as a session top-level module once the `link_design` command completes. Subsequent `link_design` or `current_goal/run_goal` commands automatically work on this top-level module only, and you need not explicitly specify this top-level module in each of these commands.

When you have defined an implicit top-level module by using the `set_option top <top-name>` command and you want to alter top-level module settings during a specific `link_design` operation, you can specify the `-top` or `-alltop` argument, as desired, to work on a top-level module different from the implicit top-level module or all top-level modules in the design, respectively.

**NOTE:** *You can either analyze a specific top-level module or all the top-level modules in the design, but not anything in between, such as two top-level modules if there are ten top-level modules in the design.*

You can generate a report after specifying the `link_design` command to view the design read messages and fix them before proceeding with the detailed analysis of your design. It is recommended that you analyze all the black boxes and provide appropriate models or constraints for them to have more meaningful goal runs.

## Arguments

The `link_design` command has the following arguments:

## Design Setup Commands

**-top <top-name>**

Use this argument to specify the top-level design modules with which the design needs to be run

**-alltop**

Use this argument to specify that the *link\_design* command should be done with all the top-level design modules in the design

**NOTE:** *You cannot specify the -top and -alltop arguments together.*

**Examples**

```
sg_shell> read_file test1.v test2.v
# test1.v has top test1 and test2.v has top test2
sg_shell> link_design -top test1
# run with top test1
sg_shell> set_option -top test2
# set test2 as top for link_design done without
# -top/-alltop
sg_shell> link_design -alltop
# run with both top test1 and test2
sg_shell> link_design
# run with top test2
```

**See Also**

[\*write\\_report\*](#)

## compile\_design

Synthesizes the design to check synthesis errors

### Syntax

#### Usage 1

```
compile_design [-top <top-name>]
```

#### Usage 2

```
compile_design [-alltop]
```

#### Usage 3

```
compile_design
```

#### Usage 4

```
compile_design [-force]
```

### Scope

Project

### Return Value

**Usages 1–3:** Returns 'exit\_code {error string}' pair

**NOTE:** Refer to the [Exit Codes Reported by sg\\_shell](#) section for more information on exit codes generated by *sg\_shell*.

### Description

The *compile\_design* command synthesizes the complete user design as per the design files and options settings. You cannot execute this command in the scope of a goal. In addition, select *current\_goal* none before executing this command. The *compile\_design* command internally invokes the *Design\_Read* goal and runs it.

You can configure the design hierarchy to be synthesized by using the *compile\_design* command in the following manner:

- If you want to synthesize a specific top-level module hierarchy, specify that top-level module with the *-top* argument of this command.
- If the design has multiple top-level modules and you want to synthesize all of them, specify the *-alltop* argument of this command.

- If you want to focus on a specific top-level module during the current Tcl session, it is best to set that top-level module by using the `set_option top <top_name>` command. After the top-level module is set, `compile_design` or `current_goal/run_goal` automatically works on this top-level module only, and you need not explicitly specify this top-level module in each of these commands.
- If you want to determine a top-level module in your design in case the top-level module is not specified by using the `set_option top <top_name>` command, use the `compile_design` command.

This command enables you to determine multiple top-level modules in your design so that you can set one top-level module for subsequent `current_goal` commands.

If there is a single top-level module in the design, it would be automatically set as a session top-level module once the `compile_design` command is completed. Subsequent `compile_design` or `current_goal/run_goal` commands automatically work on this top-level module only, and you need not explicitly specify this top-level module in each of these commands.

When you have defined an implicit top-level module using the `set_option top <top_name>` command and you want to alter top-level module settings during a specific `compile_design` operation, you can specify the `-top` or `-alltop` arguments, as desired, to work on a top-level module different from the implicit top-level module or all top-level modules in the design, respectively.

**NOTE:** *You can synthesize either a specific top-level module or all top-level modules in the design, but not anything in between, such as two top-level modules if there are ten top-level modules in the design.*

You can choose the type of synthesis to be performed in the `compile_design` command by using the following command:

```
set_option designread_synthesis_mode <synthesis_mode>
```

Depending on the type of synthesis to be performed, `<synthesis_mode>` can take one of the following values:

- **base:** Performs classical synthesis
- **opt:** Performs optimized synthesis
- **techmap:** Enables technology mapping during synthesis

You can generate a report after specifying the *compile\_design* command to view the design read messages, and fix them before proceeding with the detailed analysis of your design. It is recommended that you analyze all black boxes and provide appropriate models or constraints for them to have more meaningful goal runs.

**NOTE:** *Some setup rules do not run when you specify the `compile_design` command after the `link_design` command. SpyGlass may report the SDC\_55 violation in this case.*

## Arguments

The *compile\_design* command has the following arguments:

### **-top** <top-name>

Use this argument to specify the top-level design modules with which the design needs to be run.

### **-alltop**

Use this argument to specify that the *compile\_design* command should be run with all top-level design modules in the design.

**NOTE:** *You cannot specify the `-top` and `-alltop` arguments together.*

### **-force**

The *compile\_design* command does not perform any action in case the design has not changed across successive commands. In case, you still want to compile your design forcefully, specify the `-force` argument.

## Examples

```
sg_shell> read_file test1.v test2.v
# test1.v has top test1 and test2.v has top test2
sg_shell> compile_design -top test1
# run with top test1
sg_shell> set_option top test2
# set test2 as top for compile_design done without
# -top/-alltop
sg_shell> compile_design -alltop
# run with both top test1 and test2
sg_shell> compile_design
# run with top test2
```



**See Also**

*[write\\_report](#)*

## read\_power\_data

Provides the UPF files

### Syntax

```
read_power_data <-type <upf> > [-top <top_name>] [-version  
<version>] <file>
```

### Scope

Project

### Return Value

Boolean

### Description

The *read\_power\_data* command is used to provide the UPF files to be read by *sg\_shell*. These files are read when you run a goal. All the files are applicable to the whole project scope. This command cannot be specified after the *current\_goal* command.

The file path specified in *<file>* can be relative or absolute. In case it is relative, the current *sg\_shell* working directory should match with the project current working directory at the time of opening the project. All the relative paths should be relative to a single base working directory that is the project current working directory.

The following sanity checks are performed on files specified in the *read\_power\_data* command:

- File should exist
- Multiple specification of *read\_power\_data*

In case any of the above checks fails, an error is reported. Any file that has been already added (or any duplicate entries in the current *read\_power\_data* command) would be ignored and you would be appropriately informed. Also, if there are multiple occurrences of the *read\_power\_data* command, then only the last command would be considered and you would be appropriately informed.

## Arguments

The *read\_power\_data* command has the following arguments:

**-top <top-name>**

Use this argument to specify the name of the top module.

**-version <version>**

Use this argument to specify the UPF version.

**<file>**

Use this argument to specify the path of the file to be read.

## Examples

```
sg_shell> read_power_data -type upf example_upf_file.upf -  
top TOP -version X  
sg_shell> read_power_data -type upf example_upf_file.upf  
sg_shell> read_power_data -type upf example_upf_file.upf -  
top TO  
sg_shell> read_power_data -type upf example_upf_file.upf -  
version X
```

## See Also

[read\\_sdc\\_data](#), [read\\_activity\\_data](#)

## read\_sdc\_data

Provides the SDC files

### Syntax

```
read_sdc_data <-top <top_name> > [-mode <mode_name>] [-corner  
corner_name] [-level level_value] <file_list>
```

### Scope

Project

### Return Value

Boolean

### Description

The *read\_sdc\_data* command is used to provide the SDC files to be read by *sg\_shell*. These files are read when you run a goal. If the *read\_sdc\_data* is specified after the *current\_goal* command, the given SDC file is added to the currently selected goal only, and not for other goals. On the other hand, the *read\_sdc\_data* command specified outside the *current\_goal* command adds the SDC files to the complete project, and these SDC files are applicable to any goal in the current project.

The file path specified in *<file>* can be relative or absolute. In case it is relative, the current *sg\_shell* working directory should match with the project current working directory at the time of opening the project. The idea is all relative paths should be relative to a single base working directory which is the project current working directory.

The following sanity checks are performed on files specified in the *read\_sdc\_data* command:

- File should exist
- *-top* is a mandatory field
- Validation of *top* specified in the command

In case if any of the above checks fails, an error is reported.

## Arguments

The *read\_sdc\_data* command has the following arguments:

**-top** <top\_name>

Use this argument to specify the association of SDC file with design top and analogues to current\_design in SGDC.

**-mode** <mode\_name>

Use this argument to specify the association of SDC file with a mode.

**-corner** <corner\_name>

Use this argument to specify the association of SDC file with a corner.

**-level** <level\_value>

Use this argument to specify the association of SDC file with a level value.

**<file\_list>**

Use this argument to specify the files to be read.

## Examples

```
sg_shell> read_sdc -files a.sdc -top TOP -mode A -corner  
worst  
sg_shell> read_sdc -files a.sdc -top top  
sg_shell> read_sdc -files a.sdc -top top -mode A  
sg_shell> read_sdc -files a.sdc -top top -corner worst  
sg_shell> read_sdc -files a.sdc b.sdc -top top -corner worst
```

## See Also

[read\\_power\\_data](#), [read\\_activity\\_data](#)

## read\_activity\_data

Provides the activity data files, such as FSDB, VCD or SAIF files

### Syntax

```
read_activity_data <-type <fsdb | saif | vcd >> <-top  
<top_name> > [-starttime <starttime_value>] [-endtime <end-  
time_value>] [-instname <instance_name>] <file_list>
```

### Scope

Project

### Return Value

Boolean

### Description

The *read\_activity\_data* command is used to provide the activity data files to be read by *sg\_shell*. These files are read when you run a goal. This command supports three types of formats, FSDB, SAIF, and VCD. If the *read\_activity\_data* is specified after *current\_goal* command, then the given activity file is added to the currently selected goal only, and not to other goals. On the other hand, the *read\_activity\_data* command specified outside the *current\_goal* command adds SDC files to the complete project, and these SDC files would be applicable to any goal in the current project.

The file path specified in *<file>* can be relative or absolute. In case it is relative, the current *sg\_shell* working directory should match with the project current working directory at the time of opening the project. All relative paths should be relative to a single base working directory, which is the project current working directory.

The following sanity checks are performed on files specified in *read\_activity\_data* command:

- File should exist
- *-top* is a mandatory field
- Validation of *top* specified in the command

In case if any of the above checks fails, an error is reported.

## Arguments

The *read\_activity\_data* command has the following arguments:

**<type>**

Use this argument to specify the type of activity file to be read.

**-top <top\_name>**

Use this argument to specify the name of the top module in the simulation file.

**-starttime <starttime\_value>**

Use this argument to specify the start time for simulation file with a time unit.

**-endtime <endtime\_value>**

Use this argument to specify the end time for simulation file with a time unit.

**-instname <instance\_name>**

Use this argument to specify the name of the hierarchical instance for which the simulation file is applicable.

**<file\_list>**

Use this argument to specify the files to be read.

## Examples

```
sg_shell> read_activity_data -type fsd example.fsdb -  
starttime 1s -endtime 2s -sim_topname TOP -instname TOP.MID.I  
sg_shell> read_activity_data -type vcd example.vcd -  
starttime 1s -endtime 2s -sim_topname TOP -instname TOP.MID.I  
sg_shell> read_activity_data -type saif example.saif -  
sim_topname TOP -instname TOP.MID.I
```

## See Also

[read\\_power\\_data](#), [read\\_sdc\\_data](#)





## Goal Setup or Run Commands

Goal setup or run commands configure goal settings, such as parameters and SGDC. You can run the selected goal and view its messages.

The following table describes the various goal setup or run commands:

<b>Command</b>	<b>Description</b>
<i>current_methodology</i>	Loads the specified methodology
<i>addpolicy</i>	Adds policies to an existing goal
<i>current_goal</i>	Loads the specified goals
<i>define_goal</i>	Used to define custom goal in current selected methodology
<i>set_goal_option</i>	Sets the specified goal option to the specified value
<i>get_goal_option</i>	Gets value(s) in the goal scope for the specified option
<i>remove_goal_option</i>	Removes or unsets the specified option in the goal scope
<i>get_run_option</i>	Gets values for the current run
<i>set_run_option</i>	Sets option/parameter to specified value
<i>set_parameter</i>	Sets the value of the specified parameter
<i>get_parameter</i>	Gets the value of the specified parameter set in the current goal
<i>get_messages</i>	Returns a collection of message objects that have been reported for the goal run
<i>get_message_arg</i>	Creates a list of message arguments for a message object(s) collection
<i>get_message_labels</i>	Returns a string list of logical labels for message arguments of a message object(s) collection
<i>get_rules</i>	Returns a collection of rule objects that were loaded for the goal run
<i>run_goal</i>	Runs the selected goals
<i>save_goal</i>	Used to save design query data for the currently selected goal
<i>restore_goal</i>	Used to restore design query data for the currently selected goal

## current\_methodology

Selects a methodology

### Syntax

#### Usage 1

```
current_methodology
```

#### Usage 2

```
current_methodology [ <methodology-name> ]
```

To know more about specifying the current methodology to be used during SpyGlass analysis in the Atrenta Console, refer to the *Specifying Current Methodology* section in the *Atrenta Console User Guide*.

### Scope

Project

### Return Value

**Usage 1:** Returns current selected methodology name

**Usage 2:** Returns nothing

### Description

The *current\_methodology* command loads the specified methodology. If you do not specify the name of the methodology, the currently selected methodology name is returned.

The methodology name specified in this Tcl command should be the complete path till the top-level directory where various goals are present. `sg_shell` automatically selects a default methodology whenever you start a project. This command can be used to change it.

Use the following [help](#) commands to view information regarding methodologies:

- `help -methodology`: View the list of available methodologies.
- `help -methodology <methodology-name>`: View the help of a particular methodology.

- `help -goals`: View the list of available goals inside the currently selected methodology.
- `help -goals <goal-name>`: View the help of a specific goal.
- `write_report goal_summary`: View the summary information about the goals in the currently selected methodology.

## Arguments

This command has the following arguments:

**<methodology-name>**

Specifies the name of the methodology to be loaded

## Examples

```
sg_shell> new_project new
sg_shell> current_methodology $::env(SPYGLASS_HOME)/
GuideWare/IP_netlist
current_methodology: info: methodology is now
'SPYGLASS_HOME/GuideWare/IP_netlist'
```

```
sg_shell> set curr_meth [ current_methodology ]
sg_shell> puts $curr_meth [ current_methodology ]
SPYGLASS_HOME/GuideWare/IP_netlist
```

## See Also

[\*current\\_goal\*](#)

## addpolicy

**Adds policies to an existing goal**

### Syntax

```
addpolicy <goal_name> <policy_list>
```

### Scope

Methodology

### Return Value

Returns nothing

### Description

The *addpolicy* command is used to add policies to an existing goal.

All the new policies are added to the list of existing policies during the execution of the *current\_goal* command and then executed during execution of the *run\_goal* command.

### Arguments

This command has the following arguments:

**<goal-name>**

Specifies the name of goal to which new polices are to be added.

**<policy-list>**

Specifies the policies to be added to the goal.

### Examples

```
#adding single policy
sg_shell> current_methodology $SPYGLASS_HOME/Methodology/
rtl_handoff
sg_shell> addpolicy lint/lint_rtl clock-reset
#adding policy list
```

---

**Goal Setup or Run Commands**

```
sg_shell> current_methodology $SPYGLASS_HOME/Methodology/  
rtl_handoff  
sg_shell> addpolicy cdc/cdc_verify_struct { openmore starc  
starc2005 }
```

**See Also**

*[current\\_goal](#), [current\\_methodology](#)*

## current\_goal

Selects a goal

### Syntax

#### Usage 1

```
current_goal <goal_name>  
  [ -top <top-name> | -alltop ]
```

#### Usage 2

```
current_goal <goal_name>  
  [ -top <top-name> | -alltop ]  
  [-scenario <scenario_name>]
```

#### Usage 3

```
current_goal Group_Run -goal <goal_list> -group_name  
<group_name>
```

#### Usage 4

```
current_goal Group_Run -goal <goal list>
```

#### Usage 5

```
current_goal none
```

#### Usage 6

```
current_goal
```

**NOTE:** *The Group\_Run option is not a recommended feature.*

To know more about specifying the current goal during SpyGlass analysis in Atrenta Console, refer to the *Goal Setup* section under the *Structure of a Project File* section in the *Atrenta Console User Guide*.

### Scope

Methodology

### Return Value

**Usages 1–5:** Returns nothing

**Usage 6:** Returns an array of selected goals

## Description

The *current\_goal* command selects a goal to be run by the consequent *run\_goal* command. The selected goal should be a part of the currently active methodology, which is the methodology currently selected by the *current\_methodology* command or default methodology in setup file in case not set explicitly. You can view the help for goals in the currently selected methodology by using the *help -goals <goal-name>* command.

If you specify a substring for the complete goal name, *sg\_shell* displays a list of matching goals from which you can choose the required goal. Any previously selected goal is deselected when this command is issued.

If you select a goal that has already been run, its message database is loaded, and you can create or view its report containing messages as reported while running this goal.

Goals can have multiple scenarios, and each scenario can have a different setting for that particular goal. For running different scenarios of goals defined in a project file, specify *-scenario <scenario\_name>* with the *current\_goal* command. For more details on scenarios, refer to the *Atrenta Console User Guide*.

*sg\_shell* provides you the following flavors of the *current\_goal* specification for different purposes:

- **current\_goal with specific goal name:** This specification enables you to run a specified goal.
- **current\_goal with specific goal name and scenario name:** This specification enables you to run specified scenarios of the goal.
- **current\_goal with Group\_Run:** *Group\_Run* is a special keyword, which when specified as part of the *current\_goal* command, defines a regression goal that can be a named or un-named regression goal. If a *Group\_Run* goal is already defined in a project file, you can load the goal in *sg\_shell* using the same *current\_goal* command as used to define the goal. In this case, there is no need to specify the goal list again if it is a named group goal. You may now use the *run\_goal* command to run the goal. Alternatively, to run named group goals, you can directly specify name of the group goal in *group* option in the *run\_goal* command.

**NOTE:** *The Group\_Run option is not a recommended feature.*

- **current\_goal with none:** This specification enables you to switch to no goal selected mode. It is useful if you want to set parameter or SGDC applicable for all goals, or run the [link\\_design](#) command.
- **current\_goal without any argument:** This specification enables you to return currently selected goals in an array.

The `current_goal` command fails if none of the `-top` or `-alltop` options is specified and no session top-level module is set by the [set\\_option](#) `top <top-name>` command. If you have specified the `-alltop` argument, `sg_shell` ignores the top-level module set by using the [set\\_option](#) `top <top-name>` command, and the goal is run with all top-level modules in the design. If the top-level module is specified with the `-top` option, that top-level module is used for running the goal. Top-level module specification given with this command (`-alltop` or `-top`) overrides the current session top-level module.

You can get information about rules and their dependent parameters inside the currently selected goal by using the [write\\_report](#) `goal_setup` command. This command dumps a table comprising rules, their dependent parameters, and current or default value for each parameter.

If the selected goal has already been run in a previous session and there is a mismatch between its saved results and the setup information saved in the project, a message is flagged while loading this goal. It is recommended that you rerun it by using the [run\\_goal](#) command to ensure that results are consistent with its saved setup information.

## Arguments

This command has the following arguments:

**<goal\_name>**

Use this argument to specify a goal to be selected.

The `<goal-name>` argument is not treated like a path. This argument should be provided as a list in the following command:

```
help -goals
```

The following modifications to the `<goal-name>` argument are not allowed:

- ❑ **Starting with a dot**



```
./initial_rtl/lint/simulation
```

❑ **Using ..**

```
initial_rtl/lint/../../lint/simulation
```

❑ **Using double slashes**

```
initial_rtl/lint//simulation
```

The *<goal-name>* argument should be specified exactly as follows:

```
initial_rtl/lint/simulation
```

The path modifiers are not supported in the *<goal-name>* argument. If you use a path modifier, the *current\_goal* command fails to run.

**<scenario\_name>**

Use this argument to specify the scenario of the goal to be selected.

**-goal <goal-list>**

Use this argument to specify a list of goals to be run as part of Group\_Run.

**<group\_name>**

Use this argument to specify the name of group run.

**-top <top-name>**

Use this argument to specify the top design unit with which a goal needs to be run.

**-alltop**

Use this argument to specify that a goal should be run with all top-level modules in the design. This option is specifically helpful if there is already a top-level module set for the current session by using the [set\\_option](#) *top <top-name>* command and if you want to run the given goal with all top-level modules in the design.

**NOTE:** *The -top and -alltop arguments cannot be specified together.*

## Examples

### Example 1

```
# test1.v has top test1 and test2.v has top test2
sg_shell> read_file test1.v test2.v

# run with top test1
sg_shell> current_goal initial_rtl/lint/connectivity -top
test1
sg_shell> run_goal

# run with both top test1 and test2
sg_shell> current_goal initial_rtl/lint/connectivity -alltop
sg_shell> run_goal

# session top overridden by -alltop, no session top selected
so current_goal would fail
sg_shell> current_goal initial_rtl/lint/connectivity

# set test2 as top for current_goal specified without -top/-
alltop
sg_shell> set_option top test2

# run with top test2 and scenario scenario1 of goal
'connectivity'
sg_shell> current_goal initial_rtl/lint/connectivity
-scenario scenario1
sg_shell> run_goal

# inform currently active goal
sg_shell> set goal_sel [ current_goal ]
sg_shell> puts $goal_sel
initial_rtl/lint/connectivity

# inform currently active goal
sg_shell> current_goal
initial_rtl/lint/synthesis initial_rtl/lint/simulation

# deselect currently selected goal and switch to methodology
scope
sg_shell> current_goal none
current_goal: info: removed current goals (sg_shell is now
back to methodology scope)
```

```
(selected methodology is '/u/release/spyglass/SPYGLASS_HOME/
GuideWare/New_RTL')
```

```
# inform currently active goal i.e. none
```

```
sg_shell> current_goal
```

```
current_goal: info: no goal is currently selected
```

### Example 2

You can create scenarios by using the *-scenario* argument of the *current\_goal* command, as shown in the following example:

```
sg_shell> new_project demo
```

```
sg_shell> read_file -type hdl test.v
```

```
sg_shell> current_goal "initial_rtl/lint/connectivity"
```

```
-scenario demo_scenario_1
```

```
sg_shell> set_option show_no_msg_rule_help yes
```

```
sg_shell> current_goal none
```

```
sg_shell> current_goal "initial_rtl/lint/connectivity"
```

```
-scenario demo_scenario_2
```

```
sg_shell> set_option show_no_msg_rule_help no
```

```
sg_shell> save_project
```

```
sg_shell> close_project
```

In demo project, two scenarios *demo\_scenario\_1* and *demo\_scenario\_2* are created for goal *initial\_rtl/lint/connectivity*.

### Example 3

You can run scenarios by using the *-scenario* argument of the *current\_goal* command, as shown in the following example:

```
sg_shell> open_project demo.prj
```

```
sg_shell> current_goal "initial_rtl/lint/connectivity"
```

```
-scenario demo_scenario_2
```

```
sg_shell> run_goal
```

## See Also

[\*run\\_goal\*](#)

## define\_goal

Used to define custom goal in current selected methodology

### Syntax

```
define_goal <goal_name> [-policy <policy_list>] [-base_goal  
<base-goal-name>] [-strict] {...  
set_goal_option <option> <value>  
set_parameter <param> <value> ...  
}
```

To know more about defining a custom goal during SpyGlass analysis in the Atrenta Console, refer to the *Defining Custom Goals* section under the *Structure of a Project File* section in the *Atrenta Console User Guide*.

### Scope

Methodology

### Return Value

None

### Description

The *define\_goal* command defines a new custom goal with a given name. Once a goal is defined, it is a part of the currently selected methodology. The normal mechanism to select and execute this goal can then be used: *current\_goal* <goal\_name> followed by *run\_goal*. This command is saved as part of the project file.

The usage of custom goals is similar to that of normal goals. The batch console mode also supports these user-defined goals. If custom goal definition is present in a project file, *sg\_shell* displays these custom goals in the *goal\_summary* report by using the *write\_report goal\_summary* command. You can see these goals in batch console by using the *-showgoals* switch.

Only syntax checks are performed when you define a goal. All semantic checks on goal options, parameters, and rules added to the goal while defining it are performed when you select the goal by the *current\_goal* command. The *define\_goal* command fails if some syntax errors are found in the goal definition. Similarly, the *current\_goal* command fails if some

semantic errors are found while loading the goal.

To add additional settings of any existing goal to a new custom goal, use the `-base_goal` option to specify an existing goal. For example, consider the following `define_goal` command:

```
define_goal test_goal -policy { lint } -base_goal lint/
lint_rtl {
...
...
}
```

The `-base_goal` option in the above command applies all the settings of the existing `lint/lint_rtl` goal, such as rules, policies, options, and parameters, to the newly defined `test_goal` custom goal.

If the `-strict` option is used with the `define_goal` command, the command generates an error and stops executing the project file if any command specified in the body of `define_goal` specification fails.

All the options allowed in a goal (.SPQ) file for goal setup can be a part of the `define_goal` body, which is enclosed within curly brackets. The [set\\_goal\\_option](#) and [set\\_parameter](#) commands can be used to set custom goal-specific options.

**NOTE:** The [set\\_option](#) and [read\\_file](#) commands are not allowed to be a part of custom goal definitions.

## Arguments

**<goal\_name>**

The `<goal_name>` argument is the first positional argument to the `define_goal` command and is a mandatory option. Use this argument to specify a custom goal name.

**-policy <policy\_list>**

Use this argument to specify a list of products that you want to be a part of the custom goal. If no product is given, `-policy = none` is taken as a part of custom goal definition.

**<define\_goal\_body>**

Use this argument to specify the goal-specific settings or options, such

as rules, parameters, overload-rule, reports, and so on. The settings must be specified within brackets.

## Examples

```
sg_shell> define_goal my_goal -policy {lint clock-reset} {  
? set_goal_option rules MultipleDriver  
? set_parameter fast yes  
? }  
sg_shell> current_goal my_goal -alltop
```

## See Also

[\*import\\_project\*](#), [\*set\\_parameter\*](#), [\*set\\_goal\\_option\*](#)

## define\_regression

Used to define to define regression either by specifying a list of goals or by assigning a name to the goal list.

### Syntax

```
define_regression <regression-name> -goals {<goals-list>}
```

To know more about defining regression during SpyGlass analysis in the Atrenta Console, refer to the *define\_regression* section in the *Atrenta Console Reference Guide*.

### Scope

Methodology

### Return Value

None

### Description

Use this command to define regression by:

- Specifying a list of goals to be run in the sequential mode, and
- Assigning a name <regression-name> to the goals list.

Once saved by `save_project` command, you can specify the goal name defined by the `define_regression` command during batch run with `-goals <goals>` command-line option. This enables you to run the goals specified using the `define_regression` command in the sequential mode.

**NOTE:** *You can only define a goal in `sg_shell`. You can run a goal only in batch or Console.*

This command is useful when the list of goals to be run in the sequential mode is huge. Specifying such huge list of goals every time you run the `-goals <goals>` command-line option can be time-consuming. Therefore, specify this list in one go by using the `define_regression` command and later specify the list name (<regression-name>) to the `-goals <goals>` command-line option.

If a regression name specified by the `-goals <goals>` batch command is not declared in the project file, SpyGlass searches the specified goal in the current methodology.

## Arguments

**<regression-name>**

Regression name that you want to assign to the specified goals.

**<goal-list>**

List of goals to be run in the sequential mode.

## Examples

The following example shows the usage of this command:

```
// Project1.prj  
define_regression Reg1 -goals {G1 G2 G3}
```

Now, consider that you specify the above project file in batch by using the following command:

```
spyglass -projectfile Project1.prj -goals Reg1
```

When you run the above command, SpyGlass searches Reg1 in the project file. As the project file contains the declaration of Reg1 through the define\_regression command, SpyGlass runs the G1, G2, and G3 goals in the sequential mode.

If you want to exempt any goal from the list of goals belonging to a regression, create a new regression by using the define\_regression command such that the new regression does not have that goal.

### Creating a Regression File

A regression file is a Tcl file in which you can define regressions by using the define\_regression commands. You must keep this file parallel to the order file in the current methodology.

After creating the regression file, when you specify a regression name to the -goals <goals> command-line option, SpyGlass looks for that regression name in the regression file and runs the goals of that regression in the sequential mode.

Consider an example in which Methodology/New\_RTL is the current methodology and you have created the following regression\_run.tcl file in this methodology directory:

```
define_regression Reg1 -goals {G1 G2 G3}
```



```
define_regression Reg2 -goals {G4 G5 G6}
```

Now, when you specify Reg1 to the `-goals <goals>` command-line option, SpyGlass searches Reg1 in the `regression_run.tcl` file and runs the G1, G2, and G3 goals in the sequential mode.

However, if a regression name defined in a regression file matches with the regression name defined in a project file, SpyGlass gives preference to the regression defined in the project file. For example, if a regression file defines the A, B, and C regressions and a project file defines the C and D regressions, SpyGlass considers all the A, B, C, and D regressions where the C regression is picked from the project file.

## set\_goal\_option

**Sets the specified goal option to the specified value**

### Syntax

```
set_goal_option <option_name> <value(s)>
```

### Scope

Goal

### Return Value

Returns the value being set

### Description

The *set\_goal\_option* command sets the specified goal option to the specified value to be used for the current goal. Before using this command, you must select a goal first. Use the *set\_option* command to set an option that is not bounded to a goal.

The option value can be any of the following types:

Type	Description
BOOLEAN	If the option is of the Boolean type, the value for the option can be <true on yes 1>, which is evaluated to <code>true</code> . Similarly, the value <false off no 0> is evaluated to <code>false</code> . If more than one value is specified for the Boolean option, the last value is considered as the final value. You will get a warning about the multiple values specified in the Boolean option.
LIST	If the option is of type, list, the values are appended in the form of a string list. The option value should be specified in curly brackets { }.
STRING	If the option is of type, string, and you specify more than one value for the given string option, the last value is considered as the final value. You will get a warning about the multiple values specified in the scalar option.

If the option specified by the *set\_goal\_option* command already exists and

the option is of list type, `sg_shell` appends the value specified in the option's value-list. Otherwise, it overwrites the existing option value.

The following sanity checks are performed on values passed before an option is set:

- Invalid value specified in a Boolean option
- More than one value specified in a Boolean or scalar option
- Invalid enum value specified in an enum type of option
- String type of value specified for integer or float type of option
- Out of range value specified for options that accept a range of values
- More than two values specified for options that accept a pair of values, such as `lib` and `define_incr_dirmap`.

The following file or directory checks are performed for file or directory types of options:

- File or directory does not exist
- Read or write permissions are not there on the file directory
- File given for directory or vice versa
- Directory specified for directory type of option is empty
- File or directory that has been already added or duplicate entries in value list are ignored for the list type of options
- Wildcard pattern used to specify file/directory does not match any file/directory

The above checks can be either of the Error or Warning severity. If check is flagged with the Error severity, the current `set_goal_option` is ignored. Otherwise, the option is set as per the values passed in the command.

The remaining sanity checks as done by SpyGlass are performed at the time of the `run_goal` command.

You can view the list of options, which can be set using this command, by using the `help -option` command. This command provides a list of options that can be set by using the `set_option` and `set_goal_option` commands, respectively. There are a few options, such as `check_celldefine`, that can be set by both the `set_option` and `set_goal_option` commands, because these options can be set both for the entire project and for a particular goal. These options are listed twice by

the *help* -option command, once with the list of options that can be set by the *set\_option* command and again with the list of options that can be set by the *set\_goal\_option* command. In addition, you can also view man pages for each option in *sg\_shell* by using the *man <option-name>* command.

If there is a Boolean option set in global scope for all goals using *set\_option*, it cannot be unset in goal scope using *set\_goal\_option*.

For example, consider the following:

```
sg_shell> set_option ignorelibs yes
sg_shell> current_goal initial_rtl/lint/synthesis
sg_shell> set_goal_option ignorelibs no
sg_shell> run_goal
```

In the above case, even if *ignorelibs* is being set to *no* inside goal scope, it would still be treated as *yes* during *run\_goal*.

It is currently recommended that if a specific Boolean option is intended to be turned on/off on per goal basis, then set it inside goal scope only, and not have it set globally using *set\_option*.

The following table lists various options that you can set by using this Tcl command:

Option	Description
DEBUG	Decompiles debug information about various stages during a SpyGlass run
addrules	Selectively adds a rule in Current Goal
cell_library	Skips rule checking on design units loaded from precompiled libraries
check_celldefine	Turns on the rule checking on all the 'celldefine' modules
convert_udp_to_latch	Enables SpyGlass to infer UDP as a latch while translating the UDP with both edge and level sensitiveness
define_incr_dirmap	Provides mapping for different locations of RTL files
define_severity	Defines a user-specified severity label

## Goal Setup or Run Commands

<b>Option</b>	<b>Description</b>
<code>disable_encrypted_hdl_checks</code>	Disables RTL rule checking on the encrypted design units
<code>disable_hdllibdu_lexical_checks</code>	Disallows lexical rule checking on precompiled libraries
<code>dump_precompile_builtin</code>	Saves design parsing builtin messages in precompile dump
<code>enable_abstract_block_schematic</code>	Enables schematic debugging of abstracted modules.
<code>enable_const_prop_thru_seq</code>	Allows constant propagation beyond sequential elements during logic simulation
<code>enable_inactive_rtl_checks</code>	Enables semantic checking capability in SpyGlass
<code>enable_pass_exit_codes</code>	Causes SpyGlass to print more detailed exit status codes and messages
<code>enable_save_restore_builtin</code>	Restores design parsing, elaboration, and synthesis messages in restore run. By default, this option is ON in <code>sg_shell</code>
<code>enable_sglib_debug</code>	Provides SpyGlass Library Compiler debug information
<code>gen_blk_sgdc</code>	Enables generation of an abstract view of a block
<code>hdlldu</code>	Enables RTL and lexical rule checking on precompiled Verilog or VHDL design units
<code>ignore_reference_project_sgdc</code>	When a reference project file is specified, specify this option to ignore SGDC files from a reference design project file
<code>ignore_undefined_rules</code>	Continues after issuing a warning message if an undefined rule is specified
<code>ignorelibs</code>	Skips the rule checking for modules in the library files specified through <code>v/y</code> option
<code>ignorerules</code>	Specifies the rule names or rule group names to be ignored in rule checking
<code>ignorewaivers</code>	Causes SpyGlass to ignore waivers supplied as embedded SpyGlass Waiver pragmas
<code>lvpr</code>	Specifies the maximum number of messages to report

Option	Description
net_osc_count_limit	Overrides the oscillation count limit for nets. By default, oscillation count is set to 100
nodefparam	Ignores explicit parameter re-definition given by defparam Verilog construct
noispy	Suppresses the generation of module schematic data
noreport	Suppresses report generation
norules	Suppresses rule checking
nosch	Suppresses the generation of schematic data by rules
old_vdbfile	Specifies path of previous Violation Database file for consideration in Incremental Mode
operating_mode	Enables setting value of special variable SG_OPERATING_MODE for SGDC conditional compilation
overload	Runs the specified named overloads for all specified products
overloadpolicies	Runs the specified products with the overloaded components
overloadpolicy	Runs the specified products with the overloaded components
overloadrules	Overloads the severity or weight of a rule
perflog	Specifies to generate the SpyGlass performance log
preserve_mux	Enables SpyGlass to pick MUX cells from the technology library
reference_design_projectfile	Specifies a project file containing details, such as design files, design options, and SGDC, for the reference design while running a DDR goal. This option is complementary to the <i>reference_design_sources</i> and <i>reference_design_sgdc</i> options, and instead of these two options, you can specify a project file that contains such settings.
reference_design_sgdc	Specifies an SGDC file that contains constraints for a reference design

## Goal Setup or Run Commands

Option	Description
reference_design_sources	Specifies design files and options for a reference design through the .f file
report	Specifies the report format type to be generated
report_incr_messages	Enables reporting of incremental messages
report_inst_backref	Prints back-reference information, such as file name and line number, containing definition for a design and its instances in the <code>elab_summary.rpt</code> report
report_max_size	Specifies the maximum number of messages for sorted reports
report_style	Enables customization of report format
rules	List of rules or rule group names for which rule checking should be done
sdc2sgdc	Enables the SDC-to-SGDC feature. The default value of <code>sdc2sgdc</code> option is <code>no</code> . However, for <code>sg_shell</code> runs, SpyGlass automatically sets this option to <code>yes</code> , when at least one SDC file is specified using the <code>sdc_data</code> constraint.
sdc2sgdc_mode	Specifies the mode of the SDC file to be translated to SGDC
sdc2sgdcfile	Specifies the file to save the output of SDC-to-SGDC translation
sortrule	Specifies the sort order for messages in SpyGlass reports
sgsyn_clock_gating	Causes SpyGlass to create a simple clock gating logic instead of creating a MUX-based enable logic
sgsyn_clock_gating_threshold	Specifies threshold number (default 16) of flip-flops beyond which SpyGlass creates a simple clock gating logic
support_sdc_style_escaped_name	Enables Synopsys-style escaped names in SpyGlass Design Constraints files
target	Specifies libraries to be used for technology mapping out of the specified SGLIB libraries

Option	Description
<code>unify_sdc2sgdc</code>	Enables unification of mutually exclusive information from different sources, SDC and SGDC
<code>use_goal_rule_sort</code>	Sorts violation messages in SpyGlass reports based on the order of rules specified in a goal file
<code>w</code>	Turns on generation of warnings for PERL-level compilation

## Arguments

The `set_goal_option` command has the following arguments:

**<option-name>**

Use this argument to specify the name of the goal option.

**<value(s)>**

Use this argument to specify a space-separated list of values for the goal option.

## Examples

To generate a summary report for the currently selected goal, the `report` option can be set by using the following command on the `sg_shell` prompt:

```
sg_shell> set_goal_option report { "summary" }
```

To ignore rules W19 and W467 in the currently selected goal, the `ignorerules` option can be set as follows:

```
sg_shell> set_goal_option ignorerules { "W19" "W467" }
```

To view the schematic of abstracted blocks having prefix, `mts_...`, use the following command:

```
sg_shell> set_goal_option enable_abstract_block_schematic { mts_* }
```

To view the schematic of all the abstracted blocks, use the following command:



```
sg_shell> set_goal_option enable_abstract_block_schematic *
```

## See Also

[set\\_option](#), [get\\_option](#), [get\\_goal\\_option](#), [get\\_run\\_option](#), [remove\\_option](#),  
[remove\\_goal\\_option](#)

## get\_goal\_option

Get values in the goal scope for the specified option

### Syntax

#### Usage 1

```
get_goal_option
```

#### Usage 2

```
get_goal_option <option-name>
```

### Scope

Goal

### Return Value

**Usage 1:** Returns a list of option or value pairs

**Usage 2:** Returns the value set for the specified option, *<option-name>*

### Description

The *get\_goal\_option* command gets the option values set for the specified option in the current goal. To use this command, some goals must be selected first. To get option value set at project scope, use the *get\_option* command. When the *<option\_name>* is not specified, *sg\_shell* returns a list of option-value pairs set in the current goal scope. Otherwise, it returns the value set for the specified option in the current goal.

In case the option value is not set by using the *set\_goal\_option* command, *sg\_shell* returns the default value, if present, for the specified option. If the option does not have a default value, *sg\_shell* does not print anything on the screen, and returns the control back to the *sg\_shell* prompt. If you specify the wrong option name, *sg\_shell* displays an error message.

### Arguments

The *get\_goal\_option* command has the following arguments:

**<option-name>**

Use this argument to retrieve the value of the specified option.

## Examples

```
# To get values of report option set in current goal:  
sg_shell> set_goal_option report { simple drag }  
sg_shell> puts [ get_goal_option report ]  
simple drag  
sg_shell> set_goal_option report { inline }  
sg_shell> puts [ get_goal_option report ]  
simple drag inline
```

## See Also

[set\\_goal\\_option](#), [get\\_option](#), [set\\_option](#), [get\\_run\\_option](#), [remove\\_option](#),  
[remove\\_goal\\_option](#)

## get\_messages

Returns a collection of message objects that have been reported for the goal run

### Syntax

```
get_messages
  [<message_collection>]
  [-of_rule <rule_collection>]
  [-of_du <du_name>]
  [-msg <message_pattern>]
  [-invert]
  [-include_waived]
  [-regex | -exact]
  [-filter <filter expression>]
  [-include_data]
```

### Scope

Project

### Return Value

Return a collection of messages in case of successful execution. An empty string is returned, if no matches are found for the filtered criteria.

### Description

The `get_messages` command returns a set of messages, if any message matches one of the following criteria specified using the `<message_collection>`, `<rule_collection>`, and `<du_name>` arguments. You can specify the message pattern using the `-msg` option.

### Arguments

#### `<message_collection>`

Returns all messages which are part of the input collection. This option is useful while filtering already reduced set of messages.

## Goal Setup or Run Commands

**-of\_rule <rule\_collection>**

Return all messages of the specified rules that are flagged for the current run.

**[-of\_du <du\_name>] [-msg <message\_pattern>] [-invert]  
[-regex | -exact]**

Returns all messages flagged based on conditions and combinations of above arguments.

**NOTE:** *The -invert and the -regex/-exact options are only valid if the -of\_du or -msg options are provided.*

**-filter <filter expression>**

Returns all messages that match the filtering criteria, which is a boolean expression based on attributes defined in the message. This argument can be given along with the <rule\_collection> argument.

**-include\_data**

Includes secondary messages.

**Examples**

```
sg_shell> get_messages
```

```
sg_shell> get_messages -of_rule [get_rules]
```

```
sg_shell> get_messages -msg <pattern> -regex
```

```
sg_shell> get_messages -msg <pattern> -exact
```

```
sg_shell> get_messages [get_messages -filter {is_waived ==  
true} ]
```

```
sg_shell> get_messages -of_rule [get_rules $rules ] -  
include_data
```

## get\_message\_arg

Creates a list of message arguments for a message object(s) collection

### Syntax

```
get_message_arg [-label <arg_label>] [-as_object] <objects>
```

### Scope

Project

### Return Value

This command can be used in the following three ways:

- *get\_message\_arg < objects >*
  - ❑ Shows a list of message arguments in input message collection object.
  - ❑ Returns string list of message arguments in input message collection object.
- *get\_message\_arg < objects > -label <arg\_label>*
  - ❑ Shows a message argument string corresponding to message argument label of the message collection object.
  - ❑ Returns string of the message argument corresponding to message argument label of the message collection object.
- *get\_message\_arg < objects > -label <arg\_label> -as\_object*
  - ❑ Shows nothing.
  - ❑ Returns a collection of design object referred by message argument label of the message collection object.

### Description

The *get\_message\_arg* command returns the list of message arguments for a message object(s) collection.

## Arguments

### <objects>

Input collection of message object(s) returned by *get\_messages* command.

### -label <arg\_label>

Message argument's label(logical name) for which a message argument is being queried.

### <-as\_object>

Specify to get collection of design object corresponding to a message argument label, which refers to some design object of the current design.

**NOTE:** *The -label <arg\_label> command argument is mandatory to be specified with this argument.*

## Examples

```
## set rule_col [get_rules <RULE NAMES> ]
sg_shell> set rule_col [get_rules Ar_syncrst_setupcheck01]
sg_shell> set msg_col [get_messages -of_rule $rule_col ]

sg_shell> foreach_in_collection m $msg_col {

        #Print static string of the flagged message
        #NOTICE: the message argument position should
not refer logical labels
        puts "===== message static parts
===== "
        puts [ get_attribute $m static_message ]

        #Now Print the exact violation message of this
msg collection object
        puts "===== rule messages
===== "
        puts [ get_attribute $m msg ]

        #Check if a rule message has logical labels
associated?
```

```

        puts "===== has message labels
=====
        puts [ get_attribute $m has_arglabels ]
        if { [get_attribute $m has_arglabels ] } {

            puts "===== get message All
labels ====="
puts [ get_message_labels $m ]
            set myLabels [ get_message_labels $m ]

            puts "===== get_message_arg All
arguments ====="
            puts [ get_message_arg $m ]

            #Iterate over various labels
            foreach mylabel "$myLabels" {
                puts "===== get_message_arg
===== "

                #Get string values of the message
argument corresponding to a label
                puts "with mylabel: $mylabel : [
get_message_arg $m -label $mylabel ]"

                #Get design object corresponding to a
label
                set obj [ get_message_arg $m -label
$mylabel -as_object ]
                if {[sizeof_collection $obj] != 0} {
                    #Yeah! we have design object.. so
we can run any design query command
                    #on this object now!!
                    puts "Full Name of Obj with label
<$mylabel>: [ get_attribute $obj full_name ] "
                    #lets try to print object_type of
the object ..
                    puts "Type of object found with
label <$mylabel>: [ get_attribute $obj object_class ] "
                } else {

```



```

                                puts "Info: Design object not found
or argument is of string type."
                                }
                                }
                                } else {
                                puts "No labels associated with the
message"
                                }
                                }
                                }

===== message static parts=====
{At deassertion of synchronous reset
'<RESET>', '<TYPE>' pin of '<FLOP>' is held constant at
'<BIT_VAL>'}
===== rule messages =====
    {At deassertion of synchronous reset
'top.inst1.rst1', 'data' pin of 'top.q2' is held constant at
'0'}
===== has message labels=====
    true
===== get message All labels =====
    RESET TYPE FLOP BIT_VAL
puts "No labels associated with the message"
    }
}

===== message static parts =====
{At deassertion of synchronous reset '<RESET>', '<TYPE>' pin
of '<FLOP>' is held constant at '<BIT_VAL>'}
===== rule messages =====
{At deassertion of synchronous reset 'top.inst1.rst1', 'data'
pin of 'top.q2' is held constant at '0'}
===== has message labels =====
    true
===== get message All labels =====
    RESET TYPE FLOP BIT_VAL
===== get_message_arg All arguments =====
    top.inst1.rst1 data top.q2 0

```

```
===== get_message_arg=====
      with mylabel: RESET : top.inst1.rst1
      Full Name of Obj with label <RESET>: top.r1
      Type of object found with label <RESET>: flat_net
===== get_message_arg =====
      with mylabel: TYPE : data
      get_message_arg: error: object not found for label
'TYPE'!
      Info: Design object not found or argument is of
string type.
===== get_message_arg =====
      with mylabel: FLOP : top.q2
      Full Name of Obj with label <FLOP>: top.q2
      Type of object found with label <FLOP>: flat_net
===== get_message_arg=====
      with mylabel: BIT_VAL : 0
      get_message_arg: error: object not found for label
'BIT_VAL'!
      Info: Design object not found or argument is of
string type.
```

## See Also

[get\\_messages](#), [get\\_message\\_labels](#)

## get\_message\_labels

Returns a string list of logical labels for message arguments of a message object(s) collection

### Syntax

```
get_message_labels [message_collection]
```

### Scope

Project

### Return Value

Returns a string list of logical labels for message arguments of a message object(s) collection in case of successful execution. The empty string list is returned if no message labels are present for the specified message collection object.

### Description

The *get\_message\_labels* command returns a string list of logical labels for message arguments of a message object(s) collection. If no messages are found, an empty Tcl string list is returned.

### Arguments

**<message\_collection>**

Input collection of message object(s) returned by the *get\_messages* command.

### Examples

```
sg_shell> get_message_labels [get_messages -of_rule  
[get_rules] ]  
sg_shell> set rule_col [get_rules <RULE_NAME> ]  
sg_shell> get_message_labels [get_messages -of_rule  
$rule_col -filter {is_waived == true} ]
```

## See Also

[\*get\\_messages\*](#), [\*get\\_message\\_arg\*](#)

## get\_rules

Returns a collection of rule objects that were loaded for the goal run

### Syntax

```
get_rules  
  [pattern | -of_messages <message_collection>]  
  [-filter <filter expression>]
```

### Scope

Project

### Return Value

Returns an empty string or a collection of rules in case of successful execution. The empty string is returned if nothing matches the filtering criterion.

### Description

The `get_rules` command returns a collection of rule objects that are loaded for the goal run. The command returns a collection of rules if any rule matches the pattern or the `-of_messages` specification and also passes the filtering criteria, if specified. If no rules match the criteria, an empty string is returned.

### Arguments

**<pattern>**

Matches rule names with the specified pattern. You can use wildcard characters, `*` and `?`, for pattern matching.

**<-of\_messages message\_collection>**

Returns a collection of rules connected to the messages specified using this argument. You can only specify the messages returned by the `get_messages` command as an input to this argument.

**[ -filter expression ]**

Returns all messages that match the filtering criteria, which is a boolean expression based on rule attributes. If the expression evaluates to true, rule is included in the result.

**Examples**

```
sg_shell> get_rules
sg_shell> get_rules STARC*
sg_shell> get_rules -of_messages [get_messages]
sg_shell> get_rules -filter {rule_language == "Verilog"}
sg_shell> get_rules -filter {is_enabled == true}
```

**See Also**

[get\\_messages](#), [filter\\_collection](#)

## remove\_goal\_option

Removes or unsets the specified option in the goal scope

### Syntax

```
remove_goal_option <option-name>
```

### Scope

Goal

### Return Value

Returns the option value set before it is unset

### Description

The *remove\_goal\_option* command removes or unsets the specified option from the goal scope. *sg\_shell* does not perform any sanity check on the specified option name. Therefore, if you specify a wrong option name, *sg\_shell* does not print anything on the screen and returns the control back to the *sg\_shell* prompt.

To remove an option from the project scope, use the [remove\\_option](#) command.

### Arguments

The *remove\_goal\_option* command has the following argument:

**<option-name>**

Use this argument to specify the option name.

### Examples

```
sg_shell> set_goal_option report { drag inline }
sg_shell> get_goal_option report
drag inline
sg_shell> remove_goal_option report
sg_shell> get_goal_option report
sg_shell>
```

**See Also**

*[set\\_goal\\_option](#), [get\\_option](#), [set\\_option](#), [get\\_goal\\_option](#), [remove\\_option](#), [get\\_run\\_option](#)*



## get\_run\_option

**Gets values for the current run**

### Syntax

#### Usage 1

```
get_run_option
```

#### Usage 2

```
get_run_option <option-name>
```

### Scope

Project

### Return Value

#### Usage 1:

- ❑ Returns a list of option or value pairs
- ❑ Returns parameters in tabular format

**Usage 2:** Returns the value of the specified option

### Description

The *get\_run\_option* command gets the option value(s) that would be used in the current run for the specified option name. When the *<option\_name>* is not specified, *sg\_shell* displays all the options with their values and parameters in tabular format that *sg\_shell* would use for the current run, otherwise it returns the value (s) for the specified option only.

In case you need to get the value of a particular parameter, use the *get\_parameter* command.

In case the option value is not set by either using the *set\_option* command or *set\_goal\_option* command, *sg\_shell* returns the default value, if present, for the specified option. However, if the specified option does not have a default value, *sg\_shell* does not print anything on the screen and returns the control back to the *sg\_shell* prompt. If you specify the wrong option name, *sg\_shell* displays an error message.

The *get\_run\_option* command combines the results of the *get\_option*,

[get\\_goal\\_option](#), and [report\\_parameter](#) commands. If an option can be set by using the [set\\_option](#) or [set\\_goal\\_option](#) command only, the result of this command is identical to that of the [get\\_option](#) or [get\\_goal\\_option](#) commands respectively.

## Arguments

This command has the following arguments:

**<option-name>**

Use this argument to specify the name of the run option whose value(s) needs to be retrieved.

## Examples

```
sg_shell> get_option mthresh
# returns value set by the "set_option" command, since
# nothing is set so far, so empty string is returned
{ }
```

```
sg_shell> get_run_option mthresh
# returns value as going to be used during run, since
# default for 'mthresh' is 4096, so it is printed
4096
```

```
sg_shell> set_option report { inline }
sg_shell> set_goal_option report { drag }
sg_shell> puts [get_option report]
inline
sg_shell> puts [get_goal_option report]
drag
sg_shell> set x [get_run_option report]
inline drag
sg_shell> puts $x
inline drag
```

## See Also

[set\\_goal\\_option](#), [get\\_option](#), [set\\_option](#), [get\\_goal\\_option](#), [remove\\_option](#), [remove\\_goal\\_option](#), [report\\_parameter](#), [set\\_run\\_option](#)

## set\_run\_option

Sets option/parameter to specified value

### Syntax

```
set_run_option <option_name> <value(s)> | <option_name> -  
default
```

### Scope

Project

### Return Value

Returns a value being set

### Description

The *set\_run\_option* command gets translated into the *set\_option* command if used in global scope. It gets translated into the *set\_goal\_option* if used in goal scope, and into the *set\_parameter* command if the argument is a parameter.

This command is used to internally call the *set\_option*, *set\_goal\_option*, or *set\_parameter* command depending upon the option specified by the user.

- If the argument is a parameter, it gets converted into the *get\_parameter* command
- If the argument is an option used in global scope, it gets converted into the *get\_option* command
- If the argument is an option used in goal, it gets converted into the *get\_goal\_option* command
- If *-default* parameter is specified, it gets converted into *set\_parameter* command if used in goal scope

The following sanity checks are performed on the value(s) passed before an option is set:

- Option name is not specified
- Option value is not specified if *-default* is not given
- If *-default* is specified, then option should be a parameter used in *goal\_scope*

Above checks can be either of Error or Warning severity. If check is flagged with Error severity, current *set\_run\_option* command is ignored. Otherwise option would be set as per values passed in the command.

Remaining sanity checks that are performed by SpyGlass would be performed at the time of executing the *run\_goal* command. If *set\_option* is successful then it will return the value being set.

## Arguments

This command has the following arguments:

### <option-name>

Use this argument to specify the option/parameter to be set

### <option-value>

Use this argument to specify value of the option/parameter

## Examples

```
sg_shell>set_run_option verbosity 1
set_run_option: info: using default project `spyglass-15.prj'
set_run_option: info: using `set_option\u2019 to set option `verbosity\u2019 in project scope
1

sg_shell> set_run_option strict 1
set_run_option: info: using `set_parameter\u2019 to set parameter `strict\u2019 in methodology scope
1

sg_shell> set_run_option stict 1
set_run_option: info: using `set_parameter\u2019 to set parameter `stict\u2019 in methodology scope
set_parameter: error: parameter `stict' is not registered

sg_shell> current_goal lint/lint_rtl -alltop
current_goal: info: loading goal `lint/lint_rtl' (in progress)
current_goal: info: finished loading goal `lint/lint_rtl' (ok)
```

```
sg_shell> set_run_option verbosity 1
set_run_option: info: using `set_goal_option` to set
option `verbosity` in goal scope
option `verbosity` is not a goal specific option, please set
it by set_option

sg_shell> set_run_option strict 1
set_run_option: info: using `set_parameter` to set
parameter `strict` in goal scope
1

sg_shell> set_run_option strict -default
set_run_option: info: using `set_parameter` to set
parameter `strict` in goal scope
W342,W343

sg_shell> set_run_option strit -default
set_run_option: info: using `set_parameter` to set
parameter `strit` in goal scope
set_parameter: error: parameter `strit` is not registered

sg_shell> set_run_option set_parameter rme_active 1
set_run_option: info: using `set_parameter` to set
parameter `set_parameter` in goal scope
set_parameter: error: parameter `set_parameter` is not
registered

sg_shell> set_run_option rme_active 1
set_run_option: info: using `set_parameter` to set
parameter `rme_active` in goal scope
1
```

## See Also

[set\\_goal\\_option](#), [get\\_option](#), [set\\_option](#), [get\\_goal\\_option](#), [remove\\_option](#),  
[remove\\_goal\\_option](#), [get\\_run\\_option](#)

## set\_parameter

Sets the value of the specified parameter

### Syntax

#### Usage 1

```
set_parameter <param-name> <param-value>
```

#### Usage 2

```
set_parameter <param-name> -default
```

### Scope

Methodology, Goal

### Return Value

**Usages 1 and 2:** Returns the parameter value being set

### Description

The *set\_parameter* command sets the value of the specified parameter. A parameter that is set after selecting a methodology is applicable to all goals of the selected methodology. A parameter set in methodology scope can be overridden with a new value after selecting a goal. If the value of the parameter is changed in goal scope, it does not affect the value of that parameter in the methodology scope. If a parameter is not set in the current goal, its value is inherited from the methodology scope of the goal.

A parameter may have a default value. You can review the current value versus the default value for each parameter as part of the report generated by using the *write\_report goal\_setup* command. Alternatively, you can use the *get\_parameter ALL* command to view this information. Please note that these commands are valid for goal scope only.

It is recommended that you enclose the parameter value in curly brackets ({}), so that any Tcl-specific interpretation is not applied on it. The following example specifies the value of the *synchronize\_data\_cells* parameter:

```
sg_shell> set_parameter synchronize_data_cells  
{sync2sdffcq_f4_d?}
```

If the parameter is Boolean, that is, specified with no =<param-value> like `strict`, then it should be specified with value, 1, as shown below:

```
sg_shell> set_parameter strict 1
```

The Boolean values (true, on, 1, or yes; and false, off, 0, or no), which are supported for Boolean option set in [set\\_option](#) or [set\\_goal\\_option](#) command, are also supported by the [set\\_parameter](#) command in `sg_shell`. When the parameter value specified is different from the allowed values specified in the parameter registration (in case the parameter registration restricts allowable values), the given value is converted to the corresponding true or false value as specified in the parameter registration. In such a case, [save\\_project](#) saves the converted parameter value in the project file.

When the project is opened in Atrenta Console, the parameter value specified in the project file should be one of the allowed values in the parameter registration.

If a parameter has been set multiple times, `sg_shell` considers the last set value and uses that value during the [run\\_goal](#) command.

`sg_shell` performs the following sanity checks when a parameter is set in methodology and goal scopes:

- Whether parameter specified is registered in the set of products loaded
- Whether parameter has been made obsolete
- Invalid enum value specified in an enum type of parameter
- String type of value specified for integer or float type of parameter
- Out of range value specified for parameters that accept a range of values

The following file or directory checks are performed for file or directory type of parameters:

- File or directory does not exist
- Read or write permissions are not there on the file directory
- File given for directory or vice versa
- Directory specified for directory type of parameter is empty

The remaining sanity checks as done by SpyGlass are performed at the time of the [run\\_goal](#) command.

You can view the list of parameters available in the current scope by using

the `help -params` command. To view the help of a particular parameter, use the `help -params <parameter-name>` command.

## Arguments

The `set_parameter` command has the following arguments:

**<param-name>**

Use this argument to specify the parameter name that needs to be set.

**<param-value>**

Use this argument to specify the parameter value.

**-default**

Use this argument to set the default value for the specified parameter in the goal scope. This option is not allowed in the methodology scope.

## Examples

```
sg_shell> new_project new
```

```
sg_shell> current_methodology $::env(SPYGLASS_HOME)/  
GuideWare/New_RTL
```

```
sg_shell> set_parameter clock_reduce_pessimism mux_sel  
# sets parameter in methodology scope
```

```
sg_shell> get_parameter clock_reduce_pessimism  
mux_sel
```

```
sg_shell> current_goal initial_rtl/cdc_exhaustive/  
cdc_verif_base_strict
```

```
sg_shell> get_parameter clock_reduce_pessimism  
# inherits settings from the methodology scope  
mux_sel
```

```
sg_shell> set_parameter clock_reduce_pessimism latch_en  
# overrides parameter value with local goal setting
```

```
sg_shell> get_parameter clock_reduce_pessimism  
latch_en
```



```
sg_shell> current_goal none
sg_shell> get_parameter clock_reduce_pessimism
# value in methodology scope
mux_sel

sg_shell> current_goal initial_rtl/cdc_exhaustive/
cdc_verif_base_strict
sg_shell> get_parameter clock_reduce_pessimism
latch_en
sg_shell> set_parameter clock_reduce_pessimism -default
# default setting as part of goal, which
# is "all" for clock_reduce_pessimism
sg_shell> get_parameter clock_reduce_pessimism
all

sg_shell> current_goal none
sg_shell> get_parameter clock_reduce_pessimism
# value in methodology scope
mux_sel
```

## See Also

[\*get\\_parameter\*](#)

## get\_parameter

**Gets the value of the specified parameter set in the current goal**

### Syntax

#### Usage 1

```
get_parameter
```

#### Usage 2

```
get_parameter <param-name>
```

### Scope

Methodology, Goal

### Return Value

**Usage 1:** Returns a list of option/(value/default-value) pairs

**Usage 2:** Returns the value set for the specified option, *<param-name>*

### Description

The *get\_parameter* command gets the value of the specified parameter in the current scope (methodology or goal).

In the methodology scope, this command returns parameter value as applicable to all goals of the methodology. In the goal scope, this command returns parameter value as applicable to the current goal. If a parameter is not set in the current goal, its value is inherited from the methodology scope. If the parameter name is not specified, the *get\_parameter* command returns the value for all the parameters applicable in current scope.

In case the parameter value is not set by using the *set\_parameter* command, *sg\_shell* returns the default value, if present, for the specified parameter. However, if the parameter does not have a default value, *sg\_shell* does not print anything on the screen and returns the control back to the *sg\_shell* prompt. The same thing happens if you specify a wrong parameter name because there is no sanity check done on the parameter name. However, a message is printed on the screen when you specify an unregistered or obsolete parameter after selecting a goal.

## Arguments

The *get\_parameter* command has the following arguments:

**<param-name>**

Use this argument to retrieve the value of the specified parameter.

## Examples

```
sg_shell> new_project new
sg_shell> current_methodology $::env(SPYGLASS_HOME)/
GuideWare/New_RTL
sg_shell> set_parameter clock_reduce_pessimism mux_sel
sg_shell> get_parameter clock_reduce_pessimism
mux_sel

sg_shell> current_goal initial_rtl/cdc_exhaustive
/cdc_verif_base_strict
sg_shell> get_parameter clock_reduce_pessimism
# inherits settings from the methodology scope
mux_sel
sg_shell> set_parameter clock_reduce_pessimism latch_en
sg_shell> get_parameter clock_reduce_pessimism
latch_en

sg_shell> current_goal none
sg_shell> get_parameter clock_reduce_pessimism
# value in methodology scope
mux_sel

sg_shell> current_goal initial_rtl/cdc_exhaustive
/cdc_verif_base_strict
sg_shell> get_parameter clock_reduce_pessimism
latch_en
sg_shell> set_parameter clock_reduce_pessimism -default
# default setting as part of goal, which
# is "all" for clock_reduce_pessimism
```

```
sg_shell> get_parameter clock_reduce_pessimism  
all
```

```
sg_shell> current_goal none  
sg_shell> get_parameter clock_reduce_pessimism  
# value in methodology scope  
mux_sel
```

## See Also

[\*set\\_parameter\*](#)

## report\_parameter

Report parameters in tabular format

### Syntax

```
report_parameter [get_parameter]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *report\_parameter* command displays parameters in a tabular format. Provide the *get\_parameter* command without argument as an input to this command.

### Arguments

None

### Examples

```
sg_shell> report_parameter
report_parameter: error: invalid usage (mandatory options
missing, or wrong combination of options)
Try 'report_parameter -help' for more details.
sg_shell> get_parameter
checkInHierarchy {yes yes} checkRTLInst {yes yes}
check_default_value {yes yes}
sg_shell> report_parameter [ get_parameter ]
Parameters in the current scope:
+++++
Parameter Name      Current Value Default Value
=====
checkInHierarchy    yes           yes
checkRTLInst        yes           yes
```

```
check_default_value  yes          yes
+++++
```

**See Also**

[get\\_option](#), [get\\_goal\\_option](#), [get\\_run\\_option](#), [report\\_parameter](#), [get\\_parameter](#)

## run\_goal

**Runs the selected goal**

### Syntax

#### Usage 1

```
run_goal
```

#### Usage 2

```
run_goal -goal <goal_list>
```

#### Usage 3

```
run_goal -goal <goal_list> -group_name <group_name>
```

#### Usage 4

```
run_goal -goal <goal_list> -group_name <group_name>  
-host_config_file <parallel_run_config_file>
```

#### Usage 5

```
run_goal -goal <goal_list> -host_config_file  
<parallel_run_config_file>
```

#### Usage 6

```
run_goal -group_name <group_name> -host_config_file  
<parallel_run_config_file>
```

#### Usage 7

```
run_goal <goal_list>
```

### Scope

Goal

### Return Value

**Usages 1–7:** Returns 'exit\_code {error string}' pair

**NOTE:** Refer to the [Exit Codes Reported by sg\\_shell](#) section for more information on exit codes generated by `sg_shell`.

## Description

The `run_goal` command runs the currently selected goal with the currently selected top-level module. This command runs all the rules as specified in the goal.

**NOTE:** *By default, the SpyGlass save-restore feature is enabled during `run_goal`. To turn off this feature, specify the `set_option enable_save_restore false` command. This command turns off the save-restore feature for the complete project. Therefore, goals will be run with this feature turned off.*

The `run_goal` command normally prints a number of messages on the screen. A user can redirect its output to a file by using shell redirection operator, `>`, or capture it in some file by using the `capture` command.

### Parallel Goal Execution in SpyGlass

SpyGlass is enhanced with the parallel goal execution feature. A user can run multiple goals that perform various checks, such as clock domain analysis, timing analysis, testability analysis, power analysis and reduction, and equivalence checking, to ensure that RTL meets all design requirements. It also ensures that there are no clock, test, power, or timing violations. This feature is available in console, batch console, and `sg_shell`, but not in traditional SpyGlass.

Multiple goals can be run in parallel by using the `run_goal` command in `sg_shell`:

```
sg_shell> run_goal G1 G2 -host_config_file <file_path>
```

Where:

- `run_goal` command can be used inside methodology scope or after the `current_goal` command. All goals that are run in parallel should belong to the same methodology.
- G1 and G2 are goals that need to be run sequentially or in parallel.
- `-host_config_file <file_path>`: Specify the path of configuration file containing settings for running multiple goals on several machines in parallel. These settings include protocol to follow for remote login and other configuration, such as maximum parallel goals, to run.

For remote login, the following protocols are supported:

- **LSF:** Run goals using LSF as login protocol. It takes LSF command as an input.



**NOTE:** *Currently, only the `bsub` command is supported for LSF protocol. To use `qsub`, see [Using the `qsub` Command During Parallel Goal Run through LSF](#).*

- **RSH:** Run goals using RSH as login protocol. It takes a list of machines with a maximum process limit for each machine as an input separated by a colon from the machine name. The default number of processes that can be run on a machine is 1.
- **SSH:** Run goals using SSH as login protocol. It takes a list of machines with a maximum process limit for each machine as an input separated by a colon from the machine name. The default number of processes that can be run on a machine is 1.

The configuration keys that are provided in `.spyglass.setup` to turn ON the parallel run feature are as follows:

- **ENABLE\_PARALLEL\_RUN:** This configuration key enables the parallel run feature. Currently, only the following keywords are supported for parallel goal run:

- none
- goal

By default, this configuration key is set to `none` in the installation `.spyglass.setup` file. This indicates the parallel goal feature is disabled. From the user interface perspective, the parallel run toggle button in Goal Setup window is hidden if the key is set to `none`. Similarly, the toggle button is displayed if the key is set to `goal`.

- **HOST\_CONFIG\_FILE:** This configuration key provides the machine details for parallel execution. This key is kept as commented in the installation `.spyglass.setup` file, which can be changed if desired. This key is ignored if the `ENABLE_PARALLEL_RUN` key is set to `none`.

### ***Format of Parallel Goal Run Settings Configuration File***

The format of the parallel goal run settings configuration file is the same as that of the configuration file used to perform distributed runs of Advanced SpyGlass CDC rules on several machines. The configuration file is an ASCII text file that contains specific lines for different methods, as discussed below:

- The LSF method contains the following lines:

```
LOGIN_TYPE: lsf
MAX_PROCESSES: <num>
```

LSF\_CMD: <bsub-command>

Details of various arguments and keywords are as follows:

- ❑ Specify the value of the LOGIN\_TYPE keyword as `lsf`.
- ❑ The <num> argument of the MAX\_PROCESSES keyword specifies the maximum number of processes to be spawned. This argument is mandatory.
- ❑ The <bsub-command> argument of the LSF\_CMD keyword specifies the LSF invocation command. (default is `bsub`).
- ❑ A space is required after "LOGIN\_TYPE:" and "MAX\_PROCESSES:".
- ❑ The # and // symbols are supported as comments in the config file

Following is an example of the LSF method:

```
LOGIN_TYPE: lsf
//LOGIN_TYPE: rsh
MAX_PROCESSES: 5
LSF_CMD: bsub -q "normal | priority"
//LSF_CMD: bsub -q bsub
```

In the above example, the `-q` option is used to specify the queue as normal or priority.

- The RSH and SSH methods contain the following lines:

```
LOGIN_TYPE: rsh | ssh
MAX_PROCESSES: <num>
MACHINES:
<machine1-name>[:<num-processes>]
<machine2-name>[:<num-processes>]
...
```

Details of various arguments and keywords are as follows:

- ❑ Specify the value of the LOGIN\_TYPE keyword as `rsh` or `ssh`.
- ❑ The <num> argument for the MAX\_PROCESSES keyword specifies the maximum number of processes to be spawned. This argument is mandatory.
- ❑ The arguments, such as <machine1-name>, <machine2-name>, refer to the machine names. If the machine keyword is not specified,

it indicates you want to run goals on the current machine itself. In that case, the LOGIN\_TYPE keyword is ignored.

- ❑ The `<num-process>` argument refers to the number of processes to be spawned on the specified machine. Default value is 1.
- ❑ A space is required after “LOGIN\_TYPE:” and “MAX\_PROCESSES:”.
- ❑ The # and // symbols are supported as comments in the config file.

**NOTE:** *If the login type is SSH, the SSH\_ID\_FILE file specifies the login details for SSH login so that login does not require user name and password. Please check the man page for 'ssh' login on how to generate this file.*

Following is an example of the SSH method:

```
LOGIN_TYPE: ssh
#LOGIN_TYPE: rsh
MAX_PROCESSES: 5
MACHINES:
engr1: 1
#engr2: 1
ae3: 1
condor1: 4
```

### ***Files Generated After Parallel Goal Execution***

- **<project\_name>\_modified.prj:** This is the modified project file, using which you can run the goals in parallel. This file contains all the user settings and data required for all the goals.
- **<project\_name>\_temp.prj:** This file contains the default settings. Initially, the default settings are saved and goals are run with the new (modified) project <project\_name>\_modified.prj. Once the parallel run is complete, the settings specified in the <project\_name>\_temp.prj file are used for the subsequent runs.

### ***Using the run\_goal Command for Parallel Goal Execution***

The `run_goal` command is used for parallel goal execution in SpyGlass. This command can be used both outside and inside the scope of the `current_goal` command as follows:

- **run\_goal outside of current\_goal:** In the following command, goals G1 and G2 run sequentially one after another on the current machine if the ENABLE\_PARALLEL\_RUN configuration key is set to none.

```
run_goal -goal {G1 G2}
```

If `ENABLE_PARALLEL_RUN` is set to `goal`, the `run_goal` command generates an error message asking you to supply the parallel run configuration file if not supplied in the configuration files. Use the following command to supply the parallel run configuration file:

```
run_goal -goal {G1 G2} -host_config_file <m/c info file>
```

As a result, goals `G1` and `G2` run in parallel depending on the machine details specified in the configuration file. Once the run is complete, results of the first goal specified in goal list is loaded with the goals specified by `-goal` option or those specified as positional arguments given higher preference over groups specified over `-group_name` option. Once control is returned, you can switch to the intended goal as in Console.

However, if a regression goal needs to be run, only named regression goals can be run in parallel with other goals.

```
run_goal -goal {G1 G2} -group_name {X Y} -host_config_file  
<m/c info file>
```

This means four goals, including independent goals `G1` and `G2`, and regression goals with group names `X` and `Y`, run in parallel on the machines specified in the `-host_config_file` option. If

`ENABLE_PARALLEL_RUN` is set to `none`, the above command would run these four goals sequentially on the current machine after prompting you that the `-host_config_file` option is ignored.

- **run\_goal inside current\_goal:** The `run_goal` command inside a selected goal is functionally the same as running this command outside a selected goal, except for the following differences:
  - The goal selected for parallel execution would override currently selected goal such that currently selected goal is unloaded.
  - If `run_goal` command fails, last selected goal is loaded back. Else, results of first goal is loaded (as in `run_goal` outside `current_goal`, goals specified by `-goal` option or those specified as positional arguments given higher preference over groups specified over `-group_name` option).
  - If the `run_goal` command is performed without any `-goal` or `-group_name`, currently selected goal is run on current machine and `-host_config_file` option, if specified, is ignored.

### ***Running Goals Sequentially Using Remote Login***

A user can use this mechanism to run multiple goals, but on a separate machine without exiting the current session on the current machine. Set the `MAX_PROCESSES` variable in the configuration file to 1 to run goals sequentially using remote login. In such case, goals are first ordered into the goal dependency tree. The goal dependency tree checks whether a goal depends on some prerequisite goals that must be run before. The goals are run sequentially as per settings specified in configuration file.

#### ***Examples***

```
sg_shell> run_goal G1 G2 G3 G4 -host_config_file
<settings.txt>
```

The content of `settings.txt` changes in the following examples:

1. Running multiple goals sequentially on the same machine

```
LOGIN_TYPE : rsh
MAX_PROCESSES: 1
```

2. Running multiple goals sequentially on a different machine, identified by login protocol given you

```
LOGIN_TYPE : lsf
MAX_PROCESSES: 1
LSF_CMD: bsub
```

OR

```
LOGIN_TYPE : rsh | ssh
MAX_PROCESSES: 1
MACHINE :
<machine1>
```

Here, only one task is submitted on LSF at a time and the next task is submitted once the first task is complete. For RSH and SSH, all goals run sequentially on the first machine in the machine list given above. If no machine is specified, all goals run sequentially on the local user machine. In this case, `MAX_PROCESS` should be set to 1.

3. Running multiple goals in parallel across various machines

In the following example, a maximum of two goals can be run in

parallel on LSF queues normal and priority

```
LOGIN_TYPE : lsf
MAX_PROCESSES: 2
LSF_CMD: bsub -q "normal | priority"
```

In the following example, a maximum of two goals can be run in parallel, a maximum of one goal can be run on *<machine1>* at a time and four goals can be run in parallel on *<machine2>*

```
LOGIN_TYPE : rsh | ssh
MAX_PROCESSES: 2
MACHINE :
<machine1>
<machine2>:4
```

In the examples given above, because MAX\_PROCESS limit is set to 2, only two goals can be run in parallel at a time. Goals G1 and G2 run in parallel, with the first followed by running goals G3 and G4 in parallel in the following order (in each of examples a and b).

**Example a:** Goals G1 and G2 run on LSF. LSF manages queue normal and priority.

**Example b:** Goal G1 runs on *<machine1>* on which a maximum of one process can be performed, and goal G2 runs on *<machine2>*. As MAX\_PROCESS limit is 2, despite the fact that four processes can be run on *<machine2>*, the next goal (G3 here) is run only when one of goals G1 and G2 returns. It runs on whichever machine is available first.

Similarly, goal G4 runs on whichever machine is available first, maintaining the total process count always as 2.

Ordering of machines in the list has an impact on machine selection for execution. If *<machine2>:4* comes first, and the maximum process limit is 2, both goals always run on *<machine2>*, and *<machine1>* is used if *<machine2>* is down.

## Notes

Consider the following points while running and analyzing results of goals run in parallel:

- During parallel goal run, all the specified design files are first precompiled. At the same time, gateslib files are compiled to aggregate

sglib if the *enable\_gateslib\_autocompile* option is specified. This is required so that all the precompiled design data (RTL) become read only during parallel goal runs and gateslib files are not recompiled during goal runs because otherwise it may cause disk write race condition.

- Once precompilation and automatic compilation of gateslib are over, the input command line is changed. Top-level design unit, if not present in the project, is set for running parallel goals if the design is a single-top design.
- Multiple top-level design modules in parallel goal run are not supported. In such cases, flow stops and the user needs to set the top-level design module explicitly before proceeding to run goals in parallel.
- By default, the *enable\_precompile\_vlog* and *dump\_precompile\_builtins* project file commands are enabled while precompiling design files.
- The *hdlldu* option is added while running a parallel goal so that all messages reported at the RTL stage during the precompilation run are also reported during the goal run. These messages are reported in absolute file paths.
- DDR goals, which require dual design read capability to run, are currently unsupported in *sg\_shell*.
- Only goals belonging to the same synthesis mode (classic, optimized, or technology mapped) are allowed to be run in parallel.
- Each run separately checks out all the licenses that are required to run corresponding goal along with license for checker.
- If the login type in the specified host configuration file is `lsf`, do not specify the `-I` option of the `bsub` command in the `LSF_CMD` keyword.
- Parallel goal run is not supported in the DEF mode.

### Sanity Checks During Parallel Goal Run

The following checks are performed during parallel goal run on the parallel run configuration file and on the goals need to be run in parallel:

#### **Check 1**

The following message appears if parse errors are found in the parallel run configuration file.

```
run_goal : error: Could not open parallel run file '<file-name>'
```

#### **Check 2**

The following message appears for an invalid login type specified in the LOGIN\_TYPE keyword.

```
run_goal: error: <type> is not a supported login type
```

**Check 3**

The following message appears for unsuccessful LSF run with the specified command.

```
run_goal: error: Lsf run with specified command is not successful
```

**Check 4**

The following message appears if process count is not a positive integer value.

```
run_goal: error: Process count must be a positive integer
```

**Check 5**

The following message appears if none of the machines specified in RSH or SSH protocol is accessible.

```
run_goal: error: None of the machines specified is accessible
```

**Check 6**

The following message appears to indicate the machines that are not accessible.

```
run_goal: warning: Machines '<machines>' are not accessible
```

**Check 7**

The following message appears if the LOGIN\_TYPE keyword is not specified in the parallel file.

```
run_goal: error: LOGIN_TYPE: is not specified
```

**Check 8**

The following message appears if an error occurs while executing the lsf bsub command.

```
run_goal: error: Error executing lsf bsub command
```

**Check 9**

The following message appears if no goal list is specified in the run\_goal command, but the HOST\_CONFIG\_FILE key is set.



run\_goal: info: ignoring '-host\_config\_file' option as goal list/group list not specified, currently selected goal would run on current machine

### **Check 10**

The following message appears if the ENABLE\_PARALLEL\_RUN key is set to none and goal list is specified in the *run\_goal* command, but the HOST\_CONFIG\_FILE key is set.

run\_goal: warning: ignoring '-host\_config\_file' option as parallel run is not enabled, goals would be run sequentially on current machine

### **Check 11**

The following message appears if the ENABLE\_PARALLEL\_RUN key is set to none, but goal list is specified in the *run\_goal* command.

run\_goal: warning: parallel run is not enabled, goals would be run sequentially on current machine

### **Check 12**

The following message appears if no goal list is specified in the *run\_goal* command, but the HOST\_CONFIG\_FILE key is set.

run\_goal: info: ignoring '-host\_config\_file' option as goal list/group list not specified, currently selected goal would run on current machine

### **Check 13**

Goals pertaining to different synthesis modes cannot be run together. In such cases, the *run\_goal* command fails and generates the following error message.

run\_goal: error: goals specified belong to different synthesis mode, please specify goals of one mode only-

Following goals belong to classic synthesis mode

goal 1, goal 2 ..

Following goals belong to optimized synthesis mode

goal 3, goal 4 ..

Following goals belong to techmapped synthesis mode

goal 5, goal 6 ..

```
given goals can not be run in parallel, aborting parallel run
...
```

#### **Check 14**

If all goals pertain to the same synthesis mode, they may contain design options or goal options that can cause or netlist object model database (NOMDB) to be saved again. Such mixing of goals is not allowed and generates the following error message.

```
run_goal: error: following goals can not be run in partial/full
restore mode, please specify all goals that can be run on
restored netlist-
```

```
goal 1, goal 2...
```

```
given goals can not be run in parallel, aborting parallel run
...
```

#### **Using the qsub Command During Parallel Goal Run through LSF**

The `qsub` command is not inherently supported while running goals in parallel through LSF protocol. However, you can still use `qsub` by writing a wrapper script (say `qsub_wrapper`) over `qsub` and specifying it as an LSF command in the parallel run configuration file. This wrapper script would dissect the inputs sent to it by SpyGlass and create a command line suited to `qsub`.

Parallel run configuration file appears as the following:

```
LOGIN_TYPE: lsf
MAX_PROCESSES: <num>
LSF_CMD: qsub_wrapper
```

Ensure that the directory containing `qsub_wrapper` script is present in your path variable.

The `qsub_wrapper` script appears as follows:

```
#!/bin/sh
script=/tmp/my_script$$
outputfile=
spyglass_cmd=
while [ $# -gt 0 ];
```

```

do
  case $1 in
  -o)
    shift;
    outputfile=$1
    ;;
  -K) ;;
  *)
    if [ "X${spyglass_cmd}" != "X" ]; then
      spyglass_cmd="${spyglass_cmd} $1"
    else
      spyglass_cmd=$1
    fi
  esac
  shift;
done
\rm -f ${script}
echo "#!/bin/sh" > ${script}
echo "#PBS -o ${outputfile}" >> ${script}
echo "${spyglass_cmd}" >> ${script}
qsub -V ${script}
\rm -f ${script}

```

The above `qsub_wrapper` script generates the `/tmp/my_script<process_id>` file, which is used as an input to the `qsub` LSF command. This file appears like the following:

```

#!/bin/sh
#PBS -o output.txt
$SPYGLASS_HOME/bin/sg_shell -32bit -tcl test.tcl

```

### Run-Time Advantage from a Parallel Goal Run

Parallel goal run should give significant time improvement over running the goals sequentially. In an ideal scenario, if all goals are run in parallel, you should see the overall parallel run-time equal to the run-time of the goal that takes maximum time when run individually.

However, in actual parallel run environment, the run-time is more than the ideal situation because of the following factors:

- Parallel goal run is limited by the number of available machines, and also by the number of processes allowed to be run on a given machine. If you want to reduce the parallel run-time further, increase the machine pool available for parallel goal run, and update your parallel run configuration file accordingly.
- There may be some interdependencies among the goals specified for parallel run, which could delay the running of a goal till its dependent goals have been run.
- Parallel goal run requires some initial setup stage where goals are checked for their synthesis view requirement, and any disk write operations, such as design precompilation and design save are performed. Such setup activities are critical to ensure there are no disk read/write operations at the same time from different goals when these are running in parallel.

In parallel goal run, each goal loads policies/design independently. The time spent in parallel run setup plus the time taken by policies/design load for each parallel goal should get offset in parallel goal run if rule-checking time is significant, because rules are running in parallel.

### Peak Memory Reduction During Parallel Goal Run

Normally, during parallel goal run, design read happens during first goal run. You can use the `parallel_run_options` option to specify the mode in which parallel run design read is performed. Currently, the `separate_design_read` value is supported for this option.

For example:

```
sg_shell> set_option parallel_run_options  
separate_design_read
```

When you set the value of this option to `separate_design_read`, a separate design read is performed and NOM DB is saved, during design read. This can help in reducing the PEAK memory requirement. Goals are then run in parallel in NOM restore mode.

**NOTE:** *Currently, separate design read is done for goals that use the classic view of synthesis and is not available for the goals that need techmapped or optimized synthesis.*

## See Also

[current\\_goal](#)

## save\_goal

Used to save design query data for the currently selected goal

### Syntax

```
save_goal
```

### Scope

Goal

### Return Value

Returns an empty string when the saving of design query data is successful. In case of an unsuccessful execution, an error is returned that you can trap using the *catch* command.

### Description

This command is used to save design query data for the currently selected goal, so that it can be queried when user comes back to the same goal in the same/different session. As part of the save for the design query data, the product attributes defined on object classes, such as *design*, *du\_cell*, *du\_port*, *du\_net*, *du\_pin*, *flat\_pin*, *flat\_port*, *flat\_cell*, and *flat\_net* are saved.

If *auto\_save* option is enabled prior to running the goal, then the design query data for the currently selected goal is saved as part of *run\_goal* command. In this case, user does not need to explicitly save design query data of individual goal using this command, as the save happens automatically in the *run\_goal* command.

The *auto\_save* option is also honored by the *current\_goal*, *close\_project*, and *exit* commands. These commands also save the design query data of the last active goal if it has not been saved earlier. This option basically ensures that the goal design query data is always saved before coming out of the currently selected goal. If this option is set, the *save\_goal* command need not have to be explicitly specified to save the design query data for individual goals.

## Arguments

The *save\_goal* command has the following argument:

### **[-quiet]**

Use this argument to turn the silent mode on. When this argument is specified, no progress information is displayed while the attributes are being saved.

## Examples

Consider the following commands:

```
sg_shell> current_methodology $SPYGLASS_HOME/Methodology/DFT
sg_shell> current_goal dft_scan_ready -top top
sg_shell> run_goal
sg_shell> save_goal
save_goal: info: flat root information for module 'top' is
already saved
save_goal: info: saving attribute 'static_controllability'
of product 'dft' ...
save_goal: info: attribute 'static_controllability' saved
successfully (Time = 2.10s, Memory = 0.8K)
save_goal: info: saving attribute 'static_observability' of
product 'dft' ...
save_goal: info: attribute 'static_observability' saved
successfully (Time = 0.01s, Memory = 0.4K)
```

## See Also

[\*restore\\_goal\*](#), [\*run\\_goal\*](#)

## restore\_goal

Used to restore design query data for the currently selected goal

### Syntax

```
restore_goal
```

### Scope

Goal

### Return Value

Returns an empty string when the restoration of design query data is successful. In case of an unsuccessful execution, an error is returned that you can trap using the *catch* command.

### Description

This command is used to restore design query data for the currently selected goal. In general, product attributes are restored when user queries for a specific attribute. However, if this command is given with *-attribute\_also* option, then all the product attributes are also restored.

It is required that user should have saved the product attributes earlier while working on the currently selected goal. This command allows to restore the previously saved design query data including all the product attributes on object classes, such as *design*, *du\_cell*, *du\_port*, *du\_net*, *du\_pin*, *flat\_pin*, *flat\_port*, *flat\_cell*, and *flat\_net*.

If *auto\_restore* option is enabled, then the design query data for the active goal is restored as part of *current\_goal* and *open\_project* commands. In this case, only design data is restored, and the product attributes are restored when user queries for a specific attribute.

### Arguments

The *restore\_goal* command has the following arguments:

**[-attributes\_also]**

Enables the restore of product attributes also.

**[-quiet]**

Use this argument to turn the silent mode on. When this argument is specified, no progress information is displayed while the attributes are being restored.

**Examples**

Consider the following commands:

```
sg_shell> current_methodology $SPYGLASS_HOME/Methodology/DFT
sg_shell> current_goal dft_scan_ready -top top
sg_shell> restore_goal -attributes_also
restore_goal: info: restoring synthesized design view from
saved design database ...
restore_goal: info: synthesized design view restored
successfully (Time = 5.52s, Memory = 234.5K)
restore_goal: info: creating flat root from saved information
for module 'top' ...
  Flattening top ....
  Flattening completed
restore_goal: info: flat root created successfully for module
'top' (Time = 13.25s, Memory = 402.6K)
restore_goal: info: restoring attribute
'static_controllability' of product 'dft' ...
restore_goal: info: attribute 'static_controllability'
restored successfully (Time = 0.90s, Memory = 9.8K)
restore_goal: info: restoring attribute
'static_observability' of product 'dft' ...
restore_goal: info: attribute 'static_observability'
restored successfully (Time = 0.10s, Memory = 0.8K)
```

**See Also**

[save\\_goal](#), [current\\_goal](#), [open\\_project](#)



## ADC Setup Commands

Commands under this group allow you to set up the design by introducing Atrenta Design Constraints, or ADC.

These commands can be categorized in the following groups:

- [ADC Commands](#)
- [Utility Commands](#)

## ADC Commands

Atrenta Design Constraints, or ADC, commands are Tcl shell based constraints that can be used directly in `sg_shell`. Most of the SGDC commands have corresponding ADC commands with the same constraint and field names.

**NOTE:** *ADC commands and their option usages may change across releases, with proper notification, because the use model evolves to improve productivity and effectiveness. For scripting purpose, use your own Tcl wrappers for such commands to enable easy adoption to command changes.*

### List of ADC Commands

The following table lists the ADC commands supported in Tcl shell. This list includes new commands and the SGDC commands supported as a part of the ADC command set.

<a href="#">create_clock</a>	<a href="#">create_clock_attrib</a>	<a href="#">create_generated_clock</a>	<a href="#">set_annotated_transition</a>
<a href="#">set_case_analysis</a>	<a href="#">set_dft_signal</a>	<a href="#">set_dont_touch_network</a>	<a href="#">syn_set_dont_use</a>
<a href="#">set_driving_cell</a>	<a href="#">set_false_path</a>	<a href="#">set_ideal_network</a>	<a href="#">set_input_delay</a>
<a href="#">set_load</a>	<a href="#">set_multicycle_path</a>	<a href="#">set_output_delay</a>	<a href="#">set_scan_group</a>
<a href="#">set_wire_load_model</a>	<a href="#">set_wire_load_model</a>	<a href="#">abstract_port</a>	<a href="#">activity</a>
<a href="#">activity_data</a>	<a href="#">allow_combo_logic</a>	<a href="#">antenna_cell</a>	<a href="#">aon_buffered_signals</a>

<i>assume_path</i>	<i>atspeed_clock_frequency</i>	<i>balanced_clock</i>	<i>blackbox_power</i>
<i>block</i>	<i>breakpoint</i>	<i>bypass</i>	<i>cdc_false_path</i>
<i>cell_hookup</i>	<i>clock_buffer</i>	<i>clockgating</i>	<i>clock_group</i>
<i>clock_pin</i>	<i>clock_root</i>	<i>clock_shaper</i>	<i>complex_cell</i>
<i>compressor</i>	<i>dbist</i>	<i>decompressor</i>	<i>define_library_group</i>
<i>define_reset_order</i>	<i>define_tag</i>	<i>delay_buffer</i>	<i>deltacheck_stop_instance</i>
<i>deltacheck_ignore_module</i>	<i>deltacheck_start</i>	<i>deltacheck_stop_instance</i>	<i>deltacheck_stop_module</i>
<i>deltacheck_stop_signal</i>	<i>design_map_info</i>	<i>domain</i>	<i>dont_touch</i>
<i>expect_frequency</i>	<i>false_path</i>	<i>fifo</i>	<i>force_ta</i>
<i>gating_cell</i>	<i>gating_cell_enable</i>	<i>initstate</i>	<i>input_drive_strength</i>
<i>instance_trace</i>	<i>ip_block</i>	<i>isolation_wrapper</i>	<i>keeper</i>
<i>mapped_pin_map</i>	<i>memory</i>	<i>memory_port</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>meta_design_hier</i>	<i>mode_condition</i>	<i>module_bypass</i>	<i>module_pin</i>
<i>multivt_lib</i>	<i>network_allowed_cells</i>	<i>no_atspeed</i>	<i>no_fault</i>
<i>force_no_scan</i>	<i>noclockcell_start</i>	<i>noclockcell_stop_instance</i>	<i>noclockcell_stop_module</i>
<i>noclockcell_stop_signal</i>	<i>non_pd_inputcells</i>	<i>num_flops</i>	<i>operating_mode_set</i>
<i>output_not_used</i>	<i>pg_cell</i>	<i>pg_pins_naming</i>	<i>pll</i>
<i>port_time_delay</i>	<i>power_data</i>	<i>power_down</i>	<i>power_management_test_control_cell</i>
<i>power_management_unit</i>	<i>power_rail_mapping</i>	<i>pr_safe_clocks</i>	<i>pulldown</i>
<i>pullup</i>	<i>qualifier</i>	<i>quasi_static</i>	<i>ram_instance</i>
<i>ram_switch</i>	<i>repeater_buffer</i>	<i>require_path</i>	<i>require_pulse</i>

## ADC Setup Commands

<i>require_structure</i>	<i>require_value</i>	<i>reset</i>	<i>reset_pin</i>
<i>require_strict_path</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>sdv_data</i>	<i>select_wireload_model</i>	<i>seq_atpg</i>
<i>set_clock_gating_type</i>	<i>syn_set_dont_use</i>	<i>set_pin</i>	<i>sgdc</i>
<i>shadow_ratio</i>	<i>abstract_port</i>	<i>special_cell</i>	<i>special_module</i>
<i>spdef_data</i>	<i>switchoff_wrapper_instance</i>	<i>test_mode</i>	<i>test_point</i>
<i>tie_x</i>	<i>tristate_cell</i>	<i>ungroup_cells</i>	<i>use_library_group</i>
<i>vt_mix_percentage</i>	<i>watchpoint</i>	<i>wireload_selection</i>	<i>require_strict_path</i>
<i>illegal_path</i>	<i>always_on_buffer</i>	<i>always_on_cell</i>	<i>always_on_pin</i>
<i>assertion_signal</i>	<i>cell_pin_info</i>	<i>cell_tie_class</i>	<i>domain_inputs</i>
<i>domain_outputs</i>	<i>domain_signal</i>	<i>ignore_crossing</i>	<i>input_isocell</i>
<i>isolation_cell</i>	<i>levelshifter</i>	<i>pin_voltage</i>	<i>power_down_sequence</i>
<i>power_switch</i>	<i>retention_cell</i>	<i>retention_instance</i>	<i>power_state</i>
<i>supply</i>	<i>voltage_domain</i>	<i>define_sgdc_severity_class</i>	<i>end_sgdc_severity_class</i>
<i>illegal_value</i>	<i>illegal_constraint_message_tag</i>	<i>require_constraint_message_tag</i>	<i>sg_clock_group</i>
<i>abstract_block_violation</i>	<i>simulation_data</i>	<i>abstract_file</i>	<i>abstract_interface_param</i>
<i>abstract_interface_port</i>	<i>voltage_domain</i>	<i>memory_inst_port</i>	<i>power_state</i>

**NOTE:** The *clock SGDC* command is not supported in Tcl shell. Instead, you can use the *create\_clock*, *create\_clock\_attribute*, and *create\_generated\_clock* commands to specify clock and its attributes. Similarly, the *input* and *output* commands are not supported in Tcl shell. Instead, you can use the *set\_input\_delay* and *set\_output\_delay* commands to specify input and output delay.

## How to Specify ADC Commands

ADC commands can be specified in the following ways:

- **Mode 1:** ADC commands can be specified directly in `sg_shell` as follows:

```
sg_shell> test_mode -name {in1} -value {1}
```

- **Mode 2:** The following command adds an ADC file, which has ADC commands, to the project:

```
sg_shell> read_file -type adc <file_name> for adc  
constraints
```

## Tcl Format of ADC Commands

ADC commands follow Tcl syntax. Please refer to the [Using Escape Names in `sg\_shell`](#) section for more details on how to specify escape names in `sg_shell`.

- For ADC-constraint fields supporting scalar value, specify ADC commands in the following format:

```
sg_shell> sgdc_cmd {value}
```

For instance, consider the `test_mode` constraint given below:

```
sg_shell> test_mode -name {pin1} -value 1
```

You can also specify the above `test_mode` constraint without curly brackets, as there is no escape name:

```
sg_shell> test_mode -name pin1 -value 1
```

To specify an escape name or a wildcard pattern, curly brackets are mandatory, as shown below:

```
sg_shell> test_mode -name  
{test1.sub1_inst.\label2.sub3_inst .clk} -value 1
```

For scalar fields, anything specified inside curly brackets is considered a part of the value. Therefore, no space, other than the part of an escape name, can be specified inside curly brackets. Therefore, the following is the incorrect `test_mode` specification:

```
sg_shell> test_mode -name { pin1 } -value 1
```

Note the white space in the above `test_mode` constraint.

- For fields supporting multiple values, specify ADC commands in the following format:

```
sg_shell> sgdc_cmd {{value1} {value2}}
```

For instance, consider the following *cdc\_false\_path* constraint, which accepts multiple values in the port-pin list:

```
sg_shell> cdc_false_path -from {{clk1} {clk2}} -to {{out1}
{out2}}
```

Brackets within brackets are needed to specify escape names and wildcard patterns. The inner set of brackets can be ignored if there are no escape names or wildcard patterns. For instance, the above *cdc\_false\_path* specification can also be written as given below:

```
sg_shell> cdc_false_path -from {clk1 clk2} -to {out1 out2}
```

To specify escape names, an inner set of curly brackets is mandatory:

```
sg_shell> cdc_false_path -from
{{test1.sub1_inst.\label2.sub3_inst .q1}} -to
{{test1.sub1_inst.\label2.sub3_inst .q}}
```

The following *cdc\_false\_path* constraint uses wildcard pattern to specify a constraint:

```
sg_shell> cdc_false_path -from {{ram*.DO}}
```

The above wildcard specification can be expanded to the following constraint:

```
sg_shell> cdc_false_path -from {{foo.ram_i.DO[31:0]}}
```

- For fields of the Boolean types, there is no value associated and the field is directly referred to enable it, as shown below:

```
sg_shell> sgdc_cmd -field
```

For instance, consider the *reset* constraint that accepts *sync* and *async* as Boolean fields. Both fields cannot be specified together.

```
sg_shell> reset -name {CP} -sync
```

or

```
sg_shell> reset -name {CP} -async
```

- For more information regarding specifying escape names in ADC commands, refer to [Using Escape Names in sg\\_shell](#).
- For further information about ADC commands or their fields, refer to their corresponding man pages or the `-help` option.

## Notes

- To use an ADC command, it is necessary to set the *current\_design* Tcl command. Otherwise, it displays an error message on the screen, as shown in the following example:

```
sg_shell> test_mode -name {u2} -value {0}
test_mode: error: Rule `SGDCSTX_010': Missing
current_design specification for SGDC command 'test_mode'

sg_shell> current_design top
top
sg_shell> test_mode -name {u2} -value {0}
sg_shell>
```

- Any sanity check that fails is reported on the screen, as shown in the following examples:

```
sg_shell> test_mode -name {u3} -value {1}
test_mode: error: Rule `SGDC_testmode01': 'u3'[TopPort +
Net + HierTerminal] not found on/within module 'top'

sg_shell> test_mode -name {u2}
test_mode: error: Rule `SGDCSTX_004': Missing mandatory
field '-value' in SGDC command 'test_mode'
```

- Some sanity checks, such as design object existence check, are done only if the synthesis has happened at the time of adding a command. As shown in the following example, though pin does not exist, no error is reported when the command is added. However, these sanity errors will be reported once synthesis is done.

```
sg_shell> new_project small -force
current_methodology: info: methodology is now `~/delsoft/
spyint/integration/4.5.0-FCS-C4/RELEASE/SpyGlass-4.5.0/
SPYGLASS_HOME/GuideWare/New_RTL'
sg_shell> current_design top
top
sg_shell> test_mode -name {u3} -value {1}

sg_shell> report_adc test_mode
```

```
*****
```

```

Report : ADC command(s)
*****

+++++
ID      Command
=====
0      test_mode -name {u3} -value {1}
+++++

```

In the above example, the *test\_mode* constraint, on non-existent object *u3*, is specified before synthesis. Therefore, no error will be reported. An error will be reported for non-existent object when *Design\_Read* or *run\_goal* is performed.

#### ■ Removal of ADC Commands

- ❑ Global-level ADC commands can be removed by using the [remove\\_adc](#) and [remove\\_file](#) commands in the global scope. For scopes other than global, global-level ADC commands can be removed only by using the [remove\\_file](#) command.
- ❑ Goal-specific commands, having scope in some goal, cannot be removed in other scopes.
- ❑ SDC-equivalent constraints cannot be removed by using the [remove\\_adc](#) command. To remove all of these constraints, remove the corresponding [sdc\\_data](#) constraint shown by the [report\\_adc](#) command. However, the [create\\_clock](#) command can be removed by using the [remove\\_adc](#) command after being applied on the design, by using the [compile\\_design](#) or [run\\_goal](#) command.

**NOTE:** Line number in violation messages for all ADC and AWL commands is reported as 0.

## SDC-Equivalent Commands

You can specify the SDC-equivalent commands directly in `sg_shell` in addition to specifying them by using the `sdc_data` constraint. These commands can be specified only after the design database has been created. Therefore, if these commands are specified directly in `sg_shell` or by using the `read_file -type adc` command before the design read step, these commands are ignored and a message is generated in `sg_shell`. For example, if you specify the `create_clock` command through `read_file -type adc` before design read, `sg_shell` generates the following message:

```
create_clock: warning: Command can be specified only after
Design_read
```

These commands behave in a similar way because the commands are specified by using the `sdc_data` constraint.

Utility commands, namely `get_adc`, `report_adc`, and `remove_adc`, behave differently for these SDC-equivalent commands and other ADC commands. Please refer to each utility command for further details.

Currently, the following SDC-equivalent commands are a part of ADC command set:

Command	Description
<code>create_clock</code>	Creates the clock for the design
<code>create_clock_attribute</code>	Specifies the clock attributes
<code>create_generated_clock</code>	Creates generated clocks
<code>define_sgdc_severity_class</code>	Defines an SGDC severity class
<code>end_sgdc_severity_class</code>	Marks the end of an SGDC severity class
<code>set_annotated_transition</code>	Sets the transition time at a given pin
<code>set_case_analysis</code>	Sets a constant logic value 1 or 0 on a port or a pin for the SDC mode or multi-bit constant logic value for the SGDC mode
<code>set_dft_signal</code>	Specifies the DFT signal types for DRC and DFT insertion



## ADC Setup Commands

Command	Description
<i>set_dont_touch_network</i>	Sets the <i>dont_touch_network</i> attribute on clocks, pins, or ports in the current design to prevent cells and nets in the transitive fan-out of the <i>set_dont_touch_network</i> objects from being modified or replaced during optimization
<i>syn_set_dont_use</i>	Sets the <i>dont_use</i> attribute on library cells to exclude them from the target library during optimization
<i>set_driving_cell</i>	Sets attributes on input or inout ports of the current design, specifying that a library cell or output pin of a library cell drives the specified ports
<i>set_false_path</i>	Removes timing constraints from particular paths
<i>set_ideal_network</i>	Marks a set of ports or pins in the current design as sources of an ideal network. This disables the timing update and optimization of cells and nets in the transitive fan-out of the specified objects
<i>set_input_delay</i>	Sets an input delay on the ports relative to a clock
<i>set_load</i>	Sets the load attribute to a specified value on the specified ports and nets
<i>set_multicycle_path</i>	Modifies the single-cycle timing relationship of a constrained path
<i>set_output_delay</i>	Sets an output delay on the ports relative to a clock
<i>set_scan_group</i>	Specifies an unordered group of cells that are not yet connected, but should be kept together within a scan chain. It also identifies the existing logic in the current design that is to be designated as a scan segment.
<i>set_wire_load_mode</i>	Sets the <i>wire_load_model_mode</i> attribute on the current design, specifying how wire load models are to be used to calculate the wire capacitance in nets
<i>set_wire_load_model</i>	Sets the <i>wire_load_attach_name</i> attribute on designs, ports, hierarchical cells of current design, or the specified cluster of the current design, for selecting a wire load model to use in calculating the wire capacitance

The above commands are supported in *sg\_shell*. Some of these commands

have corresponding SGDC commands, but the constraint, field names, and semantic information differ in the two formats.

**NOTE:** *SDC equivalent commands specified directly on `sg_shell` are not considered if you have deleted the default `sdc_data` command, which is created by `sg_shell`, by using the `remove_adc` command.*

## create\_clock

**Creates the clock for the design**

### Syntax

```
create_clock
  -period period_value
  [-name clock_name]
  [-waveform egde_list]
  [-add] [port_pin_list]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *create\_clock* command creates the clock for the design. It is created in the current design and is applied to the specified value of source objects. If you do not specify a source object, but give a clock name, a virtual clock is created. This ADC command corresponds to the *clock* SGDC command.

This command can be specified only after the design database has been created.

SpyGlass reports the SDC\_219 violation if the user specifies the *create\_clock* command without the *-period* argument. SpyGlass adds 0.0 as the default period value in such cases.

### Product

SpyGlass Auto Verify, SpyGlass Constraints, SpyGlass DFT, SpyGlass DFT DSM, SpyGlass Power Estimate, SpyGlass ERC, SpyGlass Power Verify, and SpyGlass CDC

### Arguments

The *create\_clock* command has the following arguments:

**-period**

Use this argument to specify the clock field. This argument is mandatory.

**-name**

Use this argument to specify the clock name. This argument corresponds to the *-tag* field in the *clock* SGDC command.

**-waveform**

Use this argument to specify the rise and fall edge times of a clock waveform. This argument corresponds to the *-edge* field in the *clock* SGDC command.

**-add**

Use this argument to specify whether to add a clock to the existing clock or to overwrite the existing clock. Currently, *sg\_shell* ignores this argument.

**port\_pin\_list**

Use this argument to specify a list of ports and pins on which the clock needs to be set.

**NOTE:** *The create\_clock command cannot be applied on nets. It can only be applied on ports or pins.*

## Examples

### Example 1

The *create\_clock* command can be used as follows:

```
sg_shell> create_clock -name clk_tag -period 20 clk
```

### Example 2

The *create\_clock* command fails in the following case:

```
sg_shell> create_clock -name clk_tag -period 20 clk_typo  
create_clock: error: Rule `SGDC_clock01': 'clk_typo'[TopPort  
+ Net + HierTerminal] not found on/withinmodule 'clkdiv_15'
```

### Example 3

The *create\_clock* command also fails in the following case:

```
sg_shell> create_clock -period 5 clk1
create_clock: error: design database not yet created (that
is, design is neither flattened nor synthesized)
please perform compile_design or run a goal to create the
design
```

### See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#), [create\\_clock\\_attribute](#)

## create\_clock\_attribute

Specifies the clock attributes

### Syntax

```
create_clock_attribute
  -name clock_name
  [-domain domain_name]
  [-sysclock]
  [-testclock]
  [-value value]
  [-freq freq]
  [-fflimit fflimit]
  [-polarity polarity]
  [-switchingpin switching_pin_list]
  [-switching_value_list switching_value_list]
  [-atspeed]
  [-pll_reference]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *create\_clock\_attribute* command specifies the clock attributes that cannot be specified through the *create\_clock* command, but can only be specified through the *clock* SGDC command.

### Product

SpyGlass Auto Verify, SpyGlass Constraints, SpyGlass DFT, SpyGlass DFT DSM, SpyGlass Power Estimate, SpyGlass ERC, SpyGlass Power Verify, and SpyGlass CDC

## Arguments

The `create_clock_attribute` command has the following arguments:

### **-name**

Use this argument to specify the clock name. This argument corresponds to the `-tag` field in the `clock` SGDC command. This argument is mandatory.

**NOTE:** Refer to the `clock` SGDC command to view the description of other arguments of the `create_clock_attribute` command.

## Examples

### Example 1

The `create_clock_attribute` command can be used as follows:

```
sg_shell> create_clock -name clk_tag -period 20 clk
sg_shell> create_clock_attribute -name clk_tag -testclock
```

### Example 2

The `create_clock_attribute` command fails in the following cases:

```
sg_shell> create_clock_attribute -name clk_tag -testclock
sg_shell> create_clock_attribute -name clk_tag_typo
-testclock
```

### Example 3

The `create_clock_attribute` command can be used as follows:

```
sg_shell> create_clock {top.clk}
sg_shell> create_clock_attribute -name {top.clk} -atspeed
sg_shell> create_clock {top.clk2}
sg_shell> create_clock_attribute -name {top.clk2} -atspeed
```

### Example 4

The `create_clock_attribute` command fails in the following cases:

```
sg_shell> create_clock { {top.clk} {top.clk2} }
sg_shell> create_clock_attribute -name {top.clk} -atspeed
sg_shell> create_clock_attribute -name {top.clk2} -atspeed
```

## See Also

*[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#), [create\\_clock](#)*



## create\_generated\_clock

Creates generated clocks

### Syntax

```
create_generated_clock
  [-name clock_name]
  -source master_pin
  [-divide_by divide_factor | -multiply_by multiply_factor
  | -edges edge_list ]
  [-combinational]
  [-duty_cycle percent]
  [-invert]
  [-edge_shift edge_shift_list]
  [-add]
  [-master_clock clock]
  [-pll_output output_pin]
  [-pll_feedback feedback_pin]
  port_pin_list
```

### Scope

Project, Goal

### Return Value

None

### Description

The *create\_generated\_clock* command creates a generated clock object in the current design. This command defines a list of objects as generated clock sources in the current design. You can specify a pin or a port as a generated clock object. This command also specifies the clock source from which it is generated. This ADC command corresponds to the *clock* SGDC command.

### Product

SpyGlass Auto Verify, SpyGlass Constraints, SpyGlass DFT, SpyGlass DFT DSM, SpyGlass Power Estimate, SpyGlass ERC, SpyGlass Power Verify, and

## SpyGlass CDC

### Arguments

The *create\_generated\_clock* command has the following arguments:

#### **-name**

Use this argument to specify the clock name. This argument corresponds to the *-tag* field in the *clock* SGDC command.

#### **-source**

Use this argument to specify the master clock source from which the clock waveform needs to be derived.

#### **-edges**

Use this argument to specify a list of integers. These integers represent the edges from the source clock that form the edges of the generated clock.

#### **port\_pin\_list**

Use this argument to specify a list of ports and pins on which the clock needs to be set.

**NOTE:** *Currently, sg\_shell ignores other arguments of this command.*

### Examples

```
sg_shell> create_generated_clock -name u1 -source {\u2} u1
```

### See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## define\_sgdc\_severity\_class

Defines an SGDC severity class

### Syntax

```
define_sgdc_severity_class -value <option-value>
```

### Description

The *define\_sgdc\_severity\_class* command marks the start of an SGDC severity class.

For more details, refer to the *sgdc\_check\_severity* section in the *Atrenta Console Reference Guide*.

## end\_sgdc\_severity\_class

Marks the end of an SGDC severity class

### Syntax

```
end_sgdc_severity_class
```

### Description

The *end\_sgdc\_severity\_class* command marks the end of an SGDC severity class.

For more details, refer to the *sgdc\_check\_severity* section in the *Atrenta Console Reference Guide*.

## set\_annotated\_transition

Sets the transition time at a given pin

### Syntax

```
int set_annotated_transition
    [-rise | -fall]
    [-min]
    [-max]
    transition
    port_pin_list
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_annotated\_transition* command sets the transition time at a given pin.

### Product

SpyGlass Constraints

### Arguments

The *set\_annotated\_transition* command has the following arguments:

#### **-rise | -fall**

Use one of these arguments to specify whether the transition time is for data rise or data fall transition. If you do not specify *-rise* or *-fall*, both values are set.

#### **-min**

Use this argument to specify that the transition is used for minimum delay analysis. By default, the transition value is used for both

maximum and minimum delay analysis.

**-max**

Use this argument to specify that the transition is used for maximum delay analysis. By default, the transition value is used for both maximum and minimum delay analysis.

**transition**

Use this argument to specify the transition value at the pins supplied with the *port\_pin\_list* argument.

**port\_pin\_list**

Use this argument to specify a list of leaf-cell pins or top-level ports that are the end points of the timing arcs for which delays are to be annotated.

## Examples

### Example 1

The following example annotates a transition time of 20 units at the input pin A of the cell instance U1/U2/U3:

```
sg_shell> set_annotated_transition 20 U1/U2/U3/A
```

### Example 2

The following example annotates a rise transition of 1.4 units at the input pin U5/A:

```
sg_shell> set_annotated_transition -rise 1.4 U5/A
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_case\_analysis

Sets a constant logic value on a pin, port, or net

### Syntax

```
set_case_analysis
  <const-logic-value>
  <port-pin-net-list>
  [ -bc ]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_case\_analysis* command sets a constant logic value on a pin, port, or net. This ADC command supports two flavors of values to be applied on a pin, port, or net:

- **SDC-equivalent mode:** Applies the single bit value 1 or 0 on a pin, port, or net.
- **SGDC-equivalent mode:** Applies the following:
  - Multi-bit logic value for multi-cycle case analysis conditions.
  - Single bit 1 or 0 on a net.
  - Bit select on a pin, port, or net.
  - Part select on a pin, port, or net.

This ADC command corresponds to the [set\\_case\\_analysis](#) SGDC command.

### Product

SpyGlass Auto Verify, SpyGlass Power Verify, SpyGlass Power Estimate, SpyGlass ERC, SpyGlass CDC, SpyGlass latch, SpyGlass OpenMore, and SpyGlass STARC

## Arguments

The *set\_case\_analysis* command has the following arguments:

### <const-logic-value>

Use this argument to specify a constant logic value that needs to be set on a pin, port, or net.

### <port-pin-net-list>

Use this argument to specify a list of pins, ports, or nets on which the constant logic value should be set.

### -bc

(Optional) Use this argument if the command is being set on nets or if bit or part select is being used for a pin, port, or net.

## Examples

### Example 1

```
sg_shell> set_case_analysis {0} {Z}
sg_shell> set_case_analysis {1} {D}
sg_shell> report_adc [get_adc set_case_analysis]
```

```
*****
Report : ADC command(s)
*****
+++++
ID      Command
=====
0      set_case_analysis {0} {Z}
1      set_case_analysis {1} {D}
+++++
```

### Example 2

```
sg_shell> set_case_analysis 0 {temp3[1]} -bc
sg_shell> report_adc [get_adc set_case_analysis]
*****
```



## ADC Setup Commands

```

Report : ADC command(s)
*****
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ID      Command
=====
0       set_case_analysis {0} { {foo.temp3[1]} }
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```

**Example 3**

```

sg_shell> set_case_analysis {00} {temp3[1:0]} -bc
sg_shell> report_adc [get_adc set_case_analysis]
*****
Report : ADC command(s)
*****
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ID      Command
=====
0       set_case_analysis {00} { {foo.temp3[1:0]} }
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```

**See Also**

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_dft\_signal

Specifies the DFT signal types for DRC and DFT insertion

### Syntax

```
integer set_dft_signal
  -view existing_dft | spec
  -test_mode mode_name
  -type signal_type
  -port port_list
  -active_state active_state
  -timing timing
  -period period
  -hookup_pin hookup_pin
  -hookup_sense inverted | non_inverted
  -internal_clocks none | single | multi
  -ctrl_bits ctrl_bits_list
  -pll_clock pll_clock
  -ate_clock ate_clock
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_dft\_signal* command specifies one or more primary input or output ports as DFT signals.

### Product

SpyGlass Constraints

## Arguments

The *set\_dft\_signal* command has the following arguments:

**-view existing\_dft | spec**

Use this argument to indicate the view to which the specification applies.

**-test\_mode mode\_name**

Use this argument to specify the mode to which the specification applies. The default mode is `all_dft`.

**-type signal\_type**

Use this argument to specify the signal type.

**-port port\_list**

Use this argument to indicate the list of ports on which you want to apply the specifications.

**-active\_state active\_state**

Use this argument to specify the active states for the following signal types:

- ScanEnable
- Reset
- Constant
- TestMode

The active state can be 0 or 1. This argument specifies the active sense of the port (high or low), or an internal hookup. Only pin, no port, is associated with the hookup pin.

**-timing timing**

Use this argument to specify the rise time and fall time for clocks.

**-period period**

Use this argument to specify the period of the on-chip clocking

reference clock. The period value is a floating point number. The time unit is nanosecond.

**-hookup\_pin hookup\_pin**

Use this argument to identify a specific pin to which wires need to be connected.

**-hookup\_sense inverted | non\_inverted**

Use this argument to specify the hookup sense for the hookup pin.

**internal\_clocks none | single | multi**

Use this argument to specify the setting for an internal clock.

**-ctrl\_bits ctrl\_bits\_list**

Use this argument to list the triplets that specify the sequence of control bits needed to enable the propagation of the clock generator outputs.

**-pll\_clock pll\_clock**

Use this argument to specify the port name of the pll clock.

**-ate\_clock ate\_clock**

Use this argument to specify the port name of the ATE clock.

## Examples

```
set_dft_signal -hookup_pin {top.M1.in1} -ate_clock  
my_user_clock -port in3 -type Reset -ctrl_bits 01  
set_dft_signal -view existing_dft -type ScanClock -port in2  
-timing [list 45 55]
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_dont\_touch\_network

Sets the *dont\_touch\_network* attribute on clocks, pins, or ports in the current design to prevent cells and nets in the transitive fan-out of the *set\_dont\_touch\_network* objects from being modified or replaced during optimization

### Syntax

```
status set_dont_touch_network
      object_list
      [-no_propagate]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_dont\_touch\_network* command sets the *dont\_touch\_network* attribute on clocks, pins, or ports in the current design.

### Product

SpyGlass Constraints

### Arguments

The *set\_annotated\_transition* command has the following arguments:

#### **object\_list**

Use this argument to specify a list of clocks, pins, or ports in the current design on which you want to set the *dont\_touch\_network* attribute.

#### **-no\_propagate**

Use this argument to indicate that the *dont\_touch\_network* attribute is not propagated through logic gates.

## Examples

The following command adds a *dont\_touch\_network* attribute to the *clock\_in* port:

```
sg_shell> set_dont_touch_network clock_in
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## syn\_set\_dont\_use

Sets the *dont\_use* attribute on library cells to exclude them from the target library during optimization

### Syntax

```
int syn_set_dont_use  
    [-power] object_list
```

### Scope

Project, Goal

### Return Value

None

### Description

The *syn\_set\_dont\_use* command sets the *dont\_use* attribute on the specified library objects, so that they are not used.

### Product

SpyGlass Constraints

### Arguments

The *syn\_set\_dont\_use* command has the following arguments:

#### **-power**

Use this argument to specify that the list of objects passed should not be considered for power compiler clock gate mapping in addition to excluding them from the target library.

#### **object\_list**

Use this argument to specify a list of objects (lib\_cells, modules, or implementations), on which the *dont\_use* attribute needs to be set.

## Examples

The following command disables the lib\_cells G1 and G2 from the tech\_lib library:

```
sg_shell> syn_set_dont_use {tech_lib/G1 tech_lib/G2}
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)



## set\_driving\_cell

**Sets attributes on input or inout ports of the current design, specifying that a library cell or output pin of a library cell drives the specified ports**

### Syntax

```
int set_driving_cell
    [-lib_cell lib_cell_name]
    [-library lib]
    [-rise]
    [-fall]
    [-min]
    [-max]
    [-pin pin_name]
    [-from_pin from_pin_name]
    [-dont_scale]
    [-no_design_rule]
    [-none]
    [-input_transition_rise rtran]
    [-input_transition_fall ftran]
    [-multiply_by factor]
port_list
    [-cell obsolete_-_please_use_-lib_cell_instead]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_driving\_cell* command sets attributes on the specified input or inout ports in the current design to associate an external driving cell with the ports.

## Product

SpyGlass Constraints

## Arguments

The *set\_driving\_cell* command has the following arguments:

### **-lib\_cell lib\_cell\_name**

Use this argument to specify the name of the library cell to drive ports. When you use this option, you must also use the *-dont\_scale* and *-multiply\_by* options.

### **-library lib**

Use this argument to specify the library name or a collection of libraries in which you need to find the name of the library cell to drive ports.

### **-rise**

Use this argument to specify that the *lib\_cell\_name*, *lib*, *pin\_name*, and *from\_pin\_name* arguments correspond to the rising case. You can use this option with the *-fall* option to specify both the rising and falling cases.

### **-fall**

Use this argument to specify that the *lib\_cell\_name*, *lib*, *pin\_name*, and *from\_pin\_name* arguments correspond to the falling case. You can use this option with the *-rise* option to specify both the rising and falling cases.

### **-min**

Use this argument to set the driving cell information for analysis at the minimum operating condition only.

### **-max**

Use this argument to set the driving cell information for analysis at the maximum operating condition only.

### **-pin pin\_name**

Use this argument to specify the output pin on the driving cell to drive the ports.

## ADC Setup Commands

**-from\_pin from\_pin\_name**

Use this argument to specify the input pin on the driving cell while searching for a timing arc.

**-dont\_scale**

Use this argument to specify that the timing analyzer is not used to scale the drive capability of the ports according to the current operating conditions. You can use this option only with the *-lib\_cell* option.

**-no\_design\_rule**

Use this argument to indicate that the design rules associated with the driving cell are not to be applied to the driven port.

**-none**

Use this argument to remove the previous driving cell information.

**-input\_transition\_rise rtran**

Use this argument to specify the input rise transition time associated with the *-from\_pin* option.

**-input\_transition\_fall ftran**

Use this argument to specify the input fall transition time associated with the *-from\_pin* option. The default value is 0.

**-multiply\_by factor**

Use this argument to specify a factor to multiply the delay characteristics of the ports. You can use this option only with the *-lib\_cell* option.

**-port\_list**

Use this argument to specify a list of names of input or inout ports in the current design on which the driving cell attributes need to be placed.

**Examples**

The following example associates the drive capability of the AND2 library cell with the IN1 port:

```
sg_shell> set_driving_cell -lib_cell AND2 {IN1}
```

## See Also

*[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)*

## set\_false\_path

Removes timing constraints from particular paths

### Syntax

```
int set_false_path
    [-rise | -fall] [-setup | -hold]
    [-from from_list
    | -rise_from rise_from_list
    | -fall_from fall_from_list]
    [-through through_list]
    [-rise_through rise_through_list]
    [-fall_through fall_through_list]
    [-to to_list
    | -rise_to rise_to_list
    | -fall_to fall_to_list]
    [-reset_path]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_false\_path* command removes the timing constraints from the specified paths that do not affect the circuit operation.

### Product

SpyGlass Constraints

## Arguments

The *set\_false\_path* command has the following arguments:

### **-rise**

Use this argument to mark the rising delays as *false*, as measured on the path end point. If you do not specify either the *-rise* or *-fall* option, both rise and fall timing are marked as *false*.

### **-fall**

Use this argument to mark the falling delays as *false*, as measured on the path end point. If you do not specify either the *-rise* or *-fall* option, both rise and fall timing are marked as *false*.

### **-setup**

Use this argument to mark the setup (maximum) paths as *false*.

### **-hold**

Use this argument to mark the hold (minimum) paths as *false*.

### **-from from\_list**

Use this argument to specify the start points (clocks, ports, pins, or cells) of the disabled paths.

### **-rise\_from rise\_from\_list**

This argument is the same as the *-from* option, except that the path must rise from the specified objects.

### **-fall\_from fall\_from\_list**

This argument is the same as the *-from* option, except that the path must fall from the specified objects.

### **-through through\_list**

Use this argument to specify a list of path through-points (port, pin, or leaf cell names) of the current design.

## ADC Setup Commands

**-rise\_through rise\_through\_list**

This argument is the same as the *-through* option, but applies only to the paths with a rising transition at the specified objects.

**-fall\_through fall\_through\_list**

This argument is the same as the *-through* option, but applies only to the paths with a falling transition at the specified objects.

**-to to\_list**

Use this argument to specify the end points (clocks, ports, pins, or cells) of the disabled paths.

**-rise\_to rise\_to\_list**

This argument is the same as the *-to* option, but applies only to the paths rising at the end point.

**-fall\_to fall\_to\_list**

This argument is the same as the *-to* option, but applies only to the paths falling at the end point.

**-reset\_path**

This argument removes the existing point-to-point exception information on the specified paths.

**Examples**

The following example removes timing constraints on the paths from ff12 to ff34:

```
sg_shell> set_false_path -from {ff12} -to {ff34}
```

**See Also**

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_ideal\_network

Marks a set of ports or pins in the current design as sources of an ideal network. This disables the timing update and optimization of cells and nets in the transitive fan-out of the specified objects

### Syntax

```
integer set_ideal_network
    object_list
    [-dont_care_placement]
    [-no_propagate]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_ideal\_network* command marks a set of ports or pins in the current design as sources of an ideal network.

### Product

SpyGlass Constraints

### Arguments

The *set\_ideal\_network* command has the following arguments:

#### **object\_list**

Use this argument to mark a list of objects (ports, pins, or nets) as the sources of an ideal network.

#### **-dont\_care\_placement**

Use this argument to indicate that the ideal network is not considered in placement.



**-no\_propagate**

Use this argument to indicate that the ideal network is not propagated through logic gates, but it still propagates through hierarchies.

**Examples**

The following example creates an ideal network on the objects in the CLOCK\_GEN design:

```
sg_shell> current_design CLOCK_GEN  
sg_shell> set_ideal_network {port1 port2}
```

**See Also**

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_input\_delay

Sets an input delay on the ports relative to a clock

### Syntax

```
set_input_delay
  [-clock clock_name]
  [-reference_pin pin_port_name]
  [-clock_fall]
  [-level_sensitive]
  [-rise]
  [-fall]
  [-max]
  [-min]
  [-add_delay]
  [-network_latency_included]
  [-source_latency_included]
  delay_value
  port_pin_list
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_input\_delay* command sets an input delay on the ports relative to a clock. This ADC command corresponds to the *input* SGDC command.

### Product

SpyGlass CDC

## Arguments

The *set\_input\_delay* command has the following arguments:

### **-clock**

Use this argument to specify the clock to which the delay is related.

### **port\_pin\_list**

Use this argument to specify a list of ports and pins on which the input delay needs to be set.

**NOTE:** *Currently, sg\_shell ignores other arguments of this command.*

## Examples

### **Example 1**

The *set\_input\_delay* command can be used as follows:

```
sg_shell> set_input_delay -clock {clk1} {test.d1}
```

### **Example 2**

The *set\_input\_delay* command fails in the following case:

```
sg_shell> set_input_delay -clock {clk1} {test.d1_typo}
set_input_delay: error: Rule `checkSGDC_existence':
'test.d1_typo'[TopPort + Net + HierTerminal] not found on/
within module 'test'
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_load

**Sets the load attribute to a specified value on the specified ports and nets**

### Syntax

```
status set_load
    value
    objects
    [-subtract_pin_load]
    [-min]
    [-max]
    [[-pin_load] [-wire_load]]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_load* command sets the load attribute on ports and nets in the current design.

### Product

SpyGlass Constraints

### Arguments

The *set\_load* command has the following arguments:

#### value

Use this argument to set the value of the load attribute on the ports and nets contained in objects.

#### objects

Use this argument to specify a list of ports and nets in the current

## ADC Setup Commands

design whose loads are to be set.

**-subtract\_pin\_load**

Use this argument to indicate that the current pin capacitances of the net are to be subtracted from the value before the net load value is set.

**-min**

Use this argument to indicate that the load value is used for minimum delay analysis.

**-max**

Use this argument to indicate that the load value is used for maximum delay analysis.

**[-pin\_load] [-wire\_load]**

Use this argument to indicate whether the specified value on the port is treated as a pin load, as a wire load, or as both.

**Examples**

The following example sets a load of 2 units to the *the\_answer* port:

```
sg_shell> set_load 2 the_answer
```

**See Also**

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_multicycle\_path

Modifies the single-cycle timing relationship of a constrained path

### Syntax

```
integer set_multicycle_path
    path_multiplier
    [-rise | -fall]
    [-setup | -hold]
    [-start | -end]
    [-from from_list
        | -rise_from rise_from_list
        | -fall_from fall_from_list]
    [-through through_list]
    [-rise_through rise_through_list]
    [-fall_through fall_through_list]
    [-to to_list
        | -rise_to rise_to_list
        | -fall_to fall_to_list]
    [-reset_path]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_multicycle\_path* command specifies that the designated timing paths in the current design have non-default setup or hold relations.

### Product

SpyGlass Constraints

## Arguments

The *set\_multicycle\_path* command has the following arguments:

### **path\_multiplier**

Use this argument to specify the number of cycles that the data path must have for setup or hold relative to the start point or end point clock before the data is required at the end point.

### **-rise**

Use this argument to specify that the rising path delays are affected by the *path\_multiplier* argument. The default is that both rising and falling delays are affected.

### **-fall**

Use this argument to specify that the falling path delays are affected by the *path\_multiplier* argument. The default is that both rising and falling delays are affected.

### **-setup**

Use this argument to specify that the *path\_multiplier* argument is used for setup calculations.

### **-hold**

Use this argument to specify that the *path\_multiplier* argument is used for hold calculations.

### **-start | -end**

Use this argument to specify whether the multi-cycle information is relative to the period of the start clock or the end clock.

### **-from from\_list**

Use this argument to list the names of clocks, ports, pins, or cells to find the path start points.

### **-rise\_from rise\_from\_list**

This argument is same as the *-from* option, except that the path must

rise from the specified objects.

**-fall\_from fall\_from\_list**

This argument is the same as the *-from* option, except that the path must fall from the specified objects.

**-through through\_list**

Use this argument to list the path through-points (port, pin, or leaf cell names) of the current design.

**-rise\_through rise\_through\_list**

This argument is the same as the *-through* option, but applies only to the paths with a rising transition at the specified objects.

**-fall\_through fall\_through\_list**

This argument is the same as the *-through* option, but applies only to the paths with a falling transition at the specified objects.

**-to to\_list**

Use this argument to list the names of clocks, ports, pins, or cells to find path end points.

**-rise\_to rise\_to\_list**

This argument is the same as the *-to* option, but applies only to the paths rising at the end point.

**-fall\_to fall\_to\_list**

This argument is the same as the *-to* option, but applies only to the paths falling at the end point.

**-reset\_path**

Use this argument to remove the existing point-to-point exception information on the specified paths.



## Examples

The following example sets all the paths between `latch1b` and `latch2d` to two cycle paths for setup. Hold is measured at the previous edge of the clock at `latch2d`.

```
sg_shell> set_multicycle_path 2 -from {latch1b} -to  
{latch2d}
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_output\_delay

Sets an output delay on the ports relative to a clock

### Syntax

```
set_output_delay
  [-clock clock_name]
  [-reference_pin pin_port_name]
  [-clock_fall]
  [-level_sensitive]
  [-rise]
  [-fall]
  [-max]
  [-min]
  [-add_delay]
  [-network_latency_included]
  [-source_latency_included]
  [-group_path group_name]
  delay_value
  port_pin_list
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_output\_delay* command sets an output delay on the ports relative to a clock. This ADC command corresponds to the *output* SGDC command.

### Product

SpyGlass CDC

## Arguments

The *set\_output\_delay* command has the following arguments:

### **-clock**

Use this argument to specify the clock to which the delay is related.

### **port\_pin\_list**

Use this argument to specify a list of ports and pins on which the output delay needs to be set.

**NOTE:** *Currently, sg\_shell ignores other arguments of this command.*

## Examples

### **Example 1**

The *set\_output\_delay* command can be used as follows:

```
sg_shell> set_output_delay -clock {clk1} {test.d1}
```

### **Example 2**

The *set\_output\_delay* command fails in the following case:

```
sg_shell> set_output_delay -clock {clk1} {test.d1_typo}
set_output_delay: error: Rule `checkSGDC_existence':
'test.d1_typo'[TopPort + Net] not found on/withinmodule
'test'
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_scan\_group

**Specifies an unordered group of cells that are not yet connected, but should be kept together within a scan chain. It also identifies the existing logic in the current design that is to be designated as a scan segment**

### Syntax

```
int set_scan_group
    scan_group_name
    [-access signal_type_pin_pairs]
    [-include_elements member_list]
    [-serial_routed true | false]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_scan\_group* command identifies the existing logic in the current design that needs to be designated as a scan group.

### Product

SpyGlass Constraints

### Arguments

The *set\_scan\_group* command has the following arguments:

#### **scan\_group\_name**

Use this argument to specify the name of the scan group.

#### **-access signal\_type\_pin\_pairs**

Use this argument to list the ordered pairs, each consisting of a scan signal type and a design pin.

## ADC Setup Commands

**-include\_elements member\_list**

Use this argument to display an unordered list of scan group components. This list can include sequential cells, segment names, and design instances.

**-serial\_routed true | false**

Use this argument to identify whether the scan group is composed of the serially routed sequential cells or that you are specifying an unordered group of sequential cells. The default value is `false`.

**Examples**

The following example identifies an embedded shift register in the multiplexed flip-flop scan style. This specification can be completed in a single `set_scan_group` execution by including the identification of the ScanEnable access pin in the first `-access` option specification.

```
sg_shell> set_scan_group my_shift_reg \  
-access [list ScanDataIn P/B/U7/si ScanDataOut P/B/U9/QN] \  
-include_elements [list P/B/U7 P/B/U8 P/B/U9] \  
-serial_routed true
```

```
sg_shell> set_scan_group my_unconnected_group \  
-include_elements [list U1 U2]
```

```
sg_shell> set_scan_group virtual_scan_segment \  
-access [list ScanDataIn P/B/U7/si ScanDataOut P/B/U9/QN] \  
-segment_length 20 \  
-clock tck \  
-serial_routed true
```

**See Also**

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_wire\_load\_mode

Sets the *wire\_load\_model\_mode* attribute on the current design, specifying how wire load models are to be used to calculate the wire capacitance in nets

### Syntax

```
status set_wire_load_mode
      mode_name
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_wire\_load\_mode* command sets the *wire\_load\_model\_mode* attribute on the current design, specifying how hierarchical wire load models are to be used to calculate the wire capacitance of nets in the current design.

### Product

SpyGlass Constraints

### Arguments

The *set\_wire\_load\_mode* command has the following arguments:

#### **mode\_name**

Use this argument to specify the value of the *wire\_load\_model\_mode* attribute. This argument, therefore, specifies the mode to be used to handle hierarchical wire load models. The allowed values are as follows:

- The value of `top` (the default) specifies that the wire capacitance of all nets is calculated by using the wire load model set on the top-level design.

- The value of `enclosed` specifies that the wire capacitance of each net is calculated by using the wire load model set on the smallest subdesign that completely encloses that net. You must specify this mode to use the `set_wire_load_selection_group` and `set_wire_load_min_block_size` commands.
- The value of `segmented` specifies that for each net that crosses hierarchical subdesigns, the wire capacitance is calculated for each segment of the net based on the wire load model set on the subdesign that contains that segment. The total wire capacitance of the net is the sum of the wire capacitances of its segments.

## Examples

The following example sets the `wire_load_mode` attribute to be `enclosed` for the top-level design named `TOP`:

```
sg_shell> current_design TOP
sg_shell> set_wire_load_mode enclosed
```

## See Also

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## set\_wire\_load\_model

Sets the *wire\_load\_attach\_name* attribute on designs, ports, hierarchical cells of current design, or the specified cluster of the current design, for selecting a wire load model to use in calculating the wire capacitance

### Syntax

```
int set_wire_load_model
    -name model_name
    [-library lib]
    [-cluster cluster_name]
    [-min]
    [-max]
    [object_list]
```

### Scope

Project, Goal

### Return Value

None

### Description

The *set\_wire\_load\_model* command sets the *wire\_load\_attach\_name* attribute on the specified cluster in the current design, or on the ports, designs, and/or cells specified in the object list, or on the current design, for selecting a wire load model to be used in calculating the wire capacitance.

### Product

SpyGlass Constraints

### Arguments

The *set\_wire\_load\_model* command has the following arguments:

#### **-name model\_name**

Use this argument to specify the name of the wire load model to be



## ADC Setup Commands

used.

**-library lib**

Use this argument to specify the library that contains the desired wire load model.

**-cluster cluster\_name**

Use this argument to specify the cluster name to be used for the wire load model.

**-min**

Use this argument to indicate that the wire load model or selection group is used for minimum delay analysis only.

**-max**

Use this argument to indicate that the wire load model or selection group is used for maximum delay analysis only.

**object\_list**

Use this argument to specify a list of ports, designs, and/or cells to be assigned to this wire load model. If you use this option, the *-cluster* option is ignored. If neither *-cluster* nor *object\_list* is specified, the current design is assigned to the wire load model.

**Examples**

```
set_wire_load_model -name 20x20 -max -library lsi_10k
```

**See Also**

[report\\_adc](#), [get\\_adc](#), [remove\\_adc](#)

## Specifying Collection Objects in ADC Commands

You can specify Tcl collection objects, in place of design object names, in ADC commands.

For instance, the following specification uses the [get\\_ports](#) command to apply the [test\\_mode](#) command on a collection of design ports:

```
sg_shell> test_mode -name [ get_ports -filter { full_name =~
test.dioin[0] } ] -value 0
```

The [get\\_ports](#) command is a design query command whose output is a Tcl collection of ports. The above command, when expanded, applies the [test\\_mode](#) constraint on the `test.dioin[0]` design port, as shown below:

```
*****
Report : ADC command(s)
*****

+++++
ID      Command
=====
0      test_mode -name {test.dioin[0]} -value { {0} }
+++++
```

If an empty collection is passed to ADC commands, an appropriate message is generated on screen, which states that the empty collection is found and, therefore, the ADC constraint is ignored:

```
sg_shell> test_mode -name [ get_ports -filter { full_name =~
test.dioin_typo[0] } ] -value 0
test_mode: error: empty collection found in value for option
-name
```

## Utility Commands

Under this group, the following *utility commands* are available:

Command	Description
<a href="#"><i>get_adc</i></a>	Used to get a list of ADC commands on the basis of filtering criteria, if specified
<a href="#"><i>report_adc</i></a>	Reports user-specified ADC commands
<a href="#"><i>remove_adc</i></a>	Used for removing constraint command
<a href="#"><i>save_adc</i></a>	Saves the active constraints
<a href="#"><i>convert_sgdc2adc</i></a>	Converts SGDC commands specified in an input file to corresponding ADC commands in the specified output file

**NOTE:** Refer to `$SPYGLASS_HOME/examples/sg_shell/interactive_constraints` in your *SpyGlass* installation area for sample examples on how to effectively use these commands on *sg\_shell*.

## get\_adc

Used to get a list of ADC commands on the basis of filtering criteria, if specified

### Syntax

```
get_adc
    [<constr_name>]
    [-filter expression]
    [-exact]
```

### Scope

Project

### Return Value

Returns an empty string or a collection of ADC commands in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_adc* command is used to get a list of user-specified ADC commands on the basis of a filtering criterion, if specified.

This command will not return the *SDC-Equivalent Commands*, such as *create\_clock*, and so on, in the list.

### Arguments

The *get\_adc* command has the following arguments:

#### [constr\_name]

Use this argument to specify the constraint name to get ADC commands. Otherwise, ADC commands of all constraints are reported on the basis of a filtering criterion, if any.

#### [-filter expression]

Use this argument to filter a group with an expression. The supported

criteria in filter for this command are `file_name` and `current_design`. The following example illustrates the usage of this argument:

```
-filter file_name==s.adc
-filter current_design==c1
```

### **[-exact]**

Use this argument to consider wildcard characters as plain characters for constraint names.

## **Examples**

### **Example 1: Without -filter**

```
sg_shell> get_adc
_sggrp6
```

### **Example 2: With -filter**

```
sg_shell> get_adc -filter current_design==upper
_sggrp7
```

### **Example 3: get\_adc usage in report\_adc in a single step**

```
sg_shell> report_adc [get_adc -filter current_design==upper]
```

```
*****
Report : ADC command(s) for goal Design_Read
*****
```

```
+++++
ID      Command
=====
3      test_mode -name {upper.clk} -value {011}
7      test_mode -name {upper.tm} -value {1}
+++++
```

## See Also

*[report\\_adc](#), [remove\\_adc](#)*

## report\_adc

Reports user-specified ADC commands

### Syntax

```
report_adc  
  [<pattern>]  
  [-sdc]  
  [-verbose]  
  [-hide_id]  
  [-exact]
```

### Scope

Project

### Return Value

Returns an empty string if the command is successfully executed and nothing in case of any error

### Description

The *report\_adc* command displays the user-specified ADC commands.

**NOTE:** *The create\_clock command specified in sg\_shell is visible only after using the compile\_design or run\_goal command.*

### Arguments

The *report\_adc* command has the following arguments:

[<pattern>]

Use this argument to specify the output of the *get\_adc* command, which means this argument reports user-specified ADC commands that are filtered through the *get\_adc* command criteria. You can also specify a constraint name that can be an exact name or a wildcard pattern. If both are not specified, this argument reports all user-specified ADC commands.

**NOTE:** *The report\_adc command displays those commands only that are specified in the ADC file or the ones that are specified directly on the shell. It does not report the commands specified in the SGDC file.*

**[-sdc]**

Use this argument to report the *SDC-Equivalent Commands*, such as *create\_clock*, and so on. These commands will not be reported without using this switch.

**[-verbose]**

Use this argument to display verbose connection information in columns. These columns are ID, File, Line, CurrentDesign, Editable, and Command. The description of these columns is as follows:

Column	Description
ID	Unique ID for an ADC command
File	File name of a waiver command
Line	Line number of a waiver command
CurrentDesign	Current design name
Editable	A <code>yes</code> value means command can be edited. A <code>no</code> value means command cannot be edited
Command	Command string in Tcl format

**[-hide\_id]**

Use this argument to disable the listing of the ID field.

**[-exact]**

Use this argument to consider wildcard characters as plain characters for constraint name.

**Examples****Example 1: -hide\_id and -verbose not specified**

```
sg_shell> report_adc
```

```
*****
Report : ADC command(s) for goal Design_Read
*****
```



```

+++++
ID      Command
=====
0      test_mode -name {upper.clk} -value {011}
4      test_mode -name {upper.tm} -value {1}
+++++

```

### Example 2: -hide\_id specified and -verbose not specified

This case is useful when you want to redirect reported commands directly to a file.

```

sg_shell> test_mode -name {upper.clk} -value {011}
sg_shell> test_mode -name {upper.tm} -value {1}

```

### Example 3: -hide\_id not specified and -verbose specified

```

sg_shell> report_adc -verbose
*****
Report : ADC command(s) for goal Design_Read
*****

```

```

+++++
ID  File   Line  CurrentDesign  Editable  Command
=====
0   s1.sgdc  2     upper          yes       test_mode
                                   -name{upper.clk}
                                   -value {011}
+++++

```

### Example 4: -hide\_id and -verbose specified

```

sg_shell> report_adc -verbose -hide_id
*****
Report : ADC command(s) for goal Design_Read
*****

+++++
File      Line  CurrentDesign  Editable  Command
+++++

```

```

=====
s1.sgdc 2      upper      yes      test_mode -name
                                         {upper.clk}
                                         -value {011}
+++++

```

### Example 5: Using the get\_adc command in a single step

```

sg_shell> report_adc [get_adc -filter
                    current_design==upper]

```

```

*****
Report : ADC command(s) for goal Design_Read
*****

+++++
ID      Command
=====
0      test_mode -name {upper.clk} -value {011}
4      test_mode -name {upper.tm} -value {1}
+++++

```

### Example 6: Using the get\_adc command output in two passes

```

sg_shell> get_adc -filter current_design==upper
_sggrp2
sg_shell> report_adc _sggrp2
*****
Report : ADC command(s) for goal Design_Read
*****

+++++
ID      Command
=====
0      test_mode -name {upper.clk} -value {011}
4      test_mode -name {upper.tm} -value {1}
+++++

```

### Example 7: Using the `report_adc -sdc` command to report SDC equivalent command

```
*****
Report : ADC command(s)
*****

+++++
ID      Command
=====
0      sdc_data -type { {./new/top_default_sdc_file.sdc} }
1      test_mode -name {clka} -value { {1} }
3      create_clock -name {my_user_clock} -period {6}
        -waveform { {0.000000} {3.000000} } {clka}
3      create_clock_attribute -name {my_user_clock} -domain
        {my_user_clock}
+++++
```

If the `sd2sgdc` option is turned off after the SDC commands being applied, the SDC commands, other than the `create_clock` command, will be visible but will not be used until the `sd2sgdc` option is turned on again.

### See Also

[get\\_adc](#), [remove\\_adc](#)

## remove\_adc

Used for removing constraint command

### Syntax

```
remove_adc  
  [-id <idNum>]  
  [<collection>]
```

### Scope

Project

### Return Value

None

### Description

The *remove\_adc* command removes ADC commands for the given command ID or commands in the specified collection.

This command cannot be used to remove *SDC-Equivalent Commands*.

### Arguments

The *remove\_adc* command has the following arguments:

**<idNum>**

Use this argument to specify the ID of a constraint command in integers.

**<collection>**

Use this argument to report user-specified waiver commands that are filtered through the *get\_waiver* command criteria.

### Examples

#### Example 1

The following command removes the constraint command having ID as 1:

```
sg_shell> remove_adc -id 1
```

Use the [report\\_adc](#) command to view a list of applicable constraint commands and their IDs.

### **Example 2**

The following command removes ADC commands get from the [get\\_adc](#) command:

```
sg_shell> remove_adc [get_adc]
```

### **See Also**

[report\\_adc](#), [get\\_adc](#)

## save\_adc

**Saves the active constraints**

### Syntax

```
save_adc
```

### Scope

Project

### Return Value

None

### Description

The *save\_adc* command saves an active constraint used in the project in the ADC format. This command can be used to save the changes made to design constraints in *sg\_shell* through various commands, such as [remove\\_adc](#). This will save the set active constraints to a project.

All SDC constraints are saved in the *<top>.sdc* file. All ADC constraints are saved in the *default.adc* file.

**NOTE:** *If an ADC file has both SDC-equivalent commands, such as [create\\_clock](#), and non-SDC commands, such as [cdc\\_false\\_path](#), the file is rewritten during the [save\\_project](#) command. All SDC-equivalent commands are moved to the SDC file and the *sdc\_data* entry is added to the default ADC file. After rewriting, the original ADC file will have only non-SDC equivalent constraints.*

### Arguments

None

### Examples

```
sg_shell> save_adc
```

### See Also

[save\\_waiver](#), [save\\_project](#)

## convert\_sgdc2adc

Converts SGDC commands specified in an input file to corresponding ADC commands in the specified output file

### Syntax

```
convert_sgdc2adc  
  [<input_sgdc_file>]  
  [<output_adc_file>]
```

### Scope

Project

### Return Value

None

### Description

The *convert\_sgdc2adc* command converts SGDC commands from an input file to corresponding Atrenta Design Constraints, or ADC, commands in the specified output file.

### Arguments

The *convert\_sgdc2adc* command has the following arguments:

**[<input\_sgdc\_file>]**

Use this argument to specify the input SGDC file that is passed in batch run to SpyGlass.

**[<output\_adc\_file>]**

Use this argument to specify the name of the output file for capturing the created ADC commands.

### Examples

In the following example, the *test.adc* file is output that contains the corresponding ADC commands:

```
sg_shell> convert_sgdc2adc test.sgdc test.adc
```

## See Also

[\*convert\\_sw12awl\*](#)



## Reporting Commands

Reporting commands customize the message set, and generate custom, standard, or product-specific reports.

The following table describes the various reporting commands:

Command	Description
<a href="#"><i>define_filter</i></a>	Defines a criterion to filter a set of messages from the set of all generated non-waived messages
<a href="#"><i>define_view</i></a>	Defines how the selected output should be displayed
<a href="#"><i>define_report</i></a>	Defines a new report of the specified name
<a href="#"><i>write_report</i></a>	Generates the specified report
<a href="#"><i>write_aggregate_report</i></a>	Generates the specified aggregate report

**NOTE:** When the [\*save\\_project\*](#) command is run in *sg\_shell*, the [\*define\\_view\*](#), [\*define\\_filter\*](#), and [\*define\\_report\*](#) commands are not saved in a project file. If you are using a project file generated in SpyGlass 4.4.1 release that contains the [\*define\\_view\*](#), [\*define\\_filter\*](#), and [\*define\\_report\*](#) commands decompiled in it, move these commands to a Tcl file and source it in *sg\_shell*.

## define\_filter

**Defines a criterion to filter a set of messages from the set of all generated non-waived messages**

### Syntax

```
define_filter
  -name <filter-name>
  [ -filter <filter-list> ]
  [ -du <design-unit-list> ]
  [ -ip <ip-list> ]
  [ -instance <hierarchical-path-of-instance> ]
  [ -file <file-list> ]
  [ -file_line <file> <line> ]
  [ -file_lineblock <file> <start-line> <end-line> ]
  [ -severity <severity> ]
  [ -rules <rules> ]
  [ -msg <message> ]
  [ -except <list of rules/groups or keywords> ]
  [ -weight <weight> ]
  [ -weight_range <weight-start> <weight-end> ]
  [ -regexp ]
  [ -invert ]
  [-include_data]
```

### Description

The *define\_filter* command filters a set of messages from the pool of all non-waived messages. The filter can be used to define the schema of a report or used as an input to consequent filter definitions. It enables you to define a set of messages that you want to view in a specific report.

If there is no filtering criterion specified and just `define_filter -name <filter-name>` is specified, the specified *<filter-name>* holds all non-waived messages. Please note that the filtering works on non-waived messages only. If there are any messages already waived at the time of *run\_goal*, those are ignored by this command.

This command has many options that are identical to the *waive* command. For example, `-du`, `-ip`, `-file`, `-file_line`, and others behave in the

same manner as in the [waive](#) command. The only difference is that in the *define\_filter* command, these options allow you to get the message list to be viewed, whereas it is the opposite in context of the [waive](#) command.

sg\_shell does not perform any sanity check on various options of the *define\_filter* command. Therefore, you should recheck your specification if there is any mismatch in the filtered set.

**NOTE:** *Most of the arguments of this command are present in the SpyGlass waive command. Please refer to the Waiving Messages section in the Atrenta Console User Guide for more details.*

**NOTE:** *The "-instance" argument is supported only in the define\_filter command and not in the waive command. Also, this argument should only be specified when no other argument is specified in the define\_filter command, as it is mutually exclusive to the rest of the options supported in define\_filter command.*

## Arguments

This command has the following arguments:

**-name <filter-name>**

Specifies the name of the filter. If the filter with same name is already defined, it is overwritten by the current definition.

**-filter <filter-list>**

(Optional) Specifies the name of filters defined earlier by using the *define\_filter* commands. If you want to start from a filtered set, this argument accepts those previously defined filter names. If you specify multiple filter names, union of messages filtered by them is considered as the starting point for further filtering. All generated messages are considered if this argument is not specified.

**-du <design-unit-list>**

(Optional) Specifies space-separated lists of design unit names (module names for Verilog or architecture names in the format *<entity-name>*, for the entity and all its architectures, or in the *<entity-name>.<arch-name>* format for the entity and the specified architecture, package names, or configuration names for VHDL) or the logical library name of a precompiled Verilog or VHDL library. Use the *-du* argument to filter the rule messages for the specified design units or all design units in the specified library. Please

note that messages reported on the specified design units are filtered by this argument.

**-ip <ip-list>**

(Optional) Specifies space-separated lists of design unit names (module names for Verilog or architecture names in the *<entity-name>* format, for the entity and all its architectures, and in the *<entity-name>.<arch-name>* format for the entity and the specified architecture, package names, or configuration names for VHDL) or the logical library name of a precompiled Verilog or VHDL library. Use the *-ip* argument to filter the rule messages for the specified design units (IP blocks) or all design units in the specified IP library.

**-instance <hierarchical-path-of-instance>**

(Optional) Specifies space-separated list of hierarchical path of instances in the design. Use this argument to filter the rule messages for the specified hierarchical instances in the design. This argument cannot be specified with other arguments in the `define_filter` command.

**-file <file-list>**

(Optional) Specifies a space-separated list of source file names. Use this argument to filter all messages for the specified files.

**-file\_line <file> <line>**

(Optional) Specifies a space-separated pair of source file name and line number. Use this argument to filter the rule messages for a particular line of a source file.

**-file\_lineblock <file> <start-line> <end-line>**

(Optional) Specifies a space-separated tuple of source file name, start line number, and end line number. Use this argument to filter the rule messages for a block of lines in a source file.

**-severity <severity>**

(Optional) Specifies the actual severity label or severity class.

**-rules <rules>**

(Optional) Specifies a space-separated list of rule names, rule group names, or product mnemonics. Use this argument to filter messages of the specified rules, rule groups, products, or by rule type keywords. The following are allowed keywords:

- ALL
- ALL\_INFO
- ALL\_ELAB
- ALL\_SYNTHERR
- ALL\_SYNTHWRN
- ALL\_WRN

**-msg <message>**

(Optional) Specifies the actual rule message. Use this argument to filter a message. You should specify the message in this argument in curly brackets { } to prevent any Tcl interpretation being applied on it. For example, a bit-select, `top.i1.n1[0]`, would be taken as command "0", therefore, always enclose your message in { } for it to read verbatim.

**-except <rule-list>**

(Optional) Specifies a space-separated list of rule names, rule group names, or product mnemonics. Use this argument to exclude messages of the specified rules, rule groups, or products or by rule type keywords from the filtered set. The allowed keywords are same as in the *-rules* argument.

**-weight <weight>**

(Optional) Specifies the actual rule weight value. Use this argument to filter the messages of the rules with the specified weight.

**-weight\_range <weight-start> <weight-end>**

(Optional) Specifies a weight range for message filtering. Use this argument to filter messages of the rules with the weight within the specified range (both range values inclusive).

**-regexp**

(Optional) Turns on regular expression matching for various fields. This argument allows the use of regular expressions in many other arguments, such as `-du`, `-ip`, `-file`, `-file_line`, `-file_lineblock`, and `-msg`.

**-invert**

(Optional) Inverts the filtered selection. If the complete message set is 100 messages and set without `-invert` has 20 messages there would be 80 messages remaining in the filtered set if the `-invert` argument is specified.

**-include\_data**

Includes secondary messages.

**Examples**

```
sg_shell> define_filter -name f1 -msg {q/Module clcell is a
top level design unit/} -regexp
sg_shell> define_filter -name f2 -msg {m/Module clcell is a
top level.*}/} -regexp
sg_shell> define_filter -name f3 -du {cl.* sr.*} -regexp
sg_shell> define_filter -name f4 -severity {warning info} -
except {DetectTopDesignUnits checkSGDC_01}
sg_shell> define_filter -name f5 -file_lineblock test.v 3 5 -
file_lineblock test4.v 1 22 -file_lineblock a bc.v 4 5
sg_shell> define_filter -name f6 -file_line test4.v 22
-file_line test4.v 4
sg_shell> define_filter -name f7 -weight 2
sg_shell> define_filter -name f8 -weight_range 2 10
sg_shell> define_filter -name f9 -filter {f6 f8}
# union of messages filtered by filters f6 & f8

sg_shell> define_filter -name filter1 -rule {$rule} -
include_data
```

**See Also**

[define\\_view](#), [define\\_report](#), [write\\_report](#)

## define\_view

Defines how the selected output should be displayed

### Syntax

```
define_view
  -name <view-name>
  [ -header <header-string> ]
  [ -filter <filter-name> ]
  [ -display <field-list> ]
  [ -sort <field-list> ]
  [ -group <field-list> ]
```

### Scope

Any

### Return Value

None

### Description

The *define\_view* command defines the sorting and grouping criteria for a set of messages selected by the specified filter. This command also defines the fields to be displayed.

### Arguments

This command has the following arguments:

**-name <view-name>**

Specifies the name of the view. This name is then later used by the [define\\_report](#) command. If the view with the same name is already defined, it is overwritten by the current definition.

**-header <header-string>**

Specifies the header string of the view being created.

**-filter <filter-name>**

Specifies the name of the filter as created by using the [define\\_filter](#) command. Messages filtered by the specified filter are part of the current view. If you do not specify this argument, all messages are considered as part of the current view.

**-display <fields-list>**

Specifies the list of fields to be displayed. The following are the allowed values:

- policy
- rule
- alias
- severityclass
- severitylabel
- weight
- file
- line
- message
- du

The fields are displayed in the order specified in this argument. If you do not specify this argument, sg\_shell consider the default value as the following:

- rule
- alias
- severitylabel
- file
- line
- weight
- message

If there is a grouping criterion specified, you can skip displaying those fields, because all the messages of a group would have the same values for these grouped fields. In addition, these grouped field values are also



displayed as part of the group header.

**-sort <fields-list>**

(Optional) Specifies the list of fields to sort the output. The following are the allowed values:

- viol\_id
- builtin
- policy
- rulegroup
- rule
- alias
- severityclass
- severitylabel
- weight
- file
- line
- message
- du

By default, `sg_shell` does no sorting on the messages. For string values, sorting is done alphabetically ascending (in case-sensitive manner), whereas it is numerically ascending for numeric fields. However, for numeric field - `weight`, sorting is done in descending order of weight. In case of sorting based on severity class, the order is FATAL, ERROR, WARNING, and INFO. This difference is there for `weight` and severity class, so that more severe messages appear on the top after sorting.

If there is grouping done on any of the above fields, you can skip that field in the sorting criterion, since all messages of any group would have same value for that field.

**-group <fields-list>**

(Optional) Specifies the list of fields to perform grouping. Following are the allowed values:

- builtin

- ❑ policy
- ❑ rulegroup
- ❑ rule
- ❑ alias
- ❑ severityclass
- ❑ severitylabel
- ❑ weight
- ❑ file
- ❑ line
- ❑ message
- ❑ du
- ❑ goal
- ❑ sdcmode

The filtered messages are first grouped as per the grouping criterion and then each group is sorted as per the sorting criterion. If grouping is done based on multiple fields then each group has messages where those multiple fields have the same value for all messages. Further, each group has a header containing the grouped fields values for the current group.

## Examples

```
sg_shell> define_view -name v1 -filter f1 -display "rule
policy severitylabel message" -header "DISPLAY: rule policy
severitylabel message"
# display specified fields for messages in 'f1' filter
```

```
sg_shell> define_view -name v2 -filter f1 -display "rule
policy severityclass message" -sort "severityclass" -header
"SORT: severityclass"
# sort messages in 'f1' filter as per severityclass
```

```
sg_shell> define_view -name v3 -filter f2 -display "rule
policy severityclass message" -group "severityclass" -header
```

## Reporting Commands

```
"GROUP: severityclass"  
# group messages in 'f2' filter as per severityclass
```

```
sg_shell> define_view -name v4 -display "rule policy  
severityclass message" -group "severityclass"  
-header "GROUP: severityclass"  
# group all messages as per severityclass
```

**See Also**

[\*define\\_filter\*](#), [\*define\\_report\*](#), [\*write\\_report\*](#)

## define\_report

**Defines a new report of the specified name**

### Syntax

```
define_report
  -name <report-name>
  -view <view-list>
  [ -helpfile <help-file> ]
```

### Scope

Any

### Return Value

None

### Description

The *define\_report* command defines a new report of the specified name.

This report is basically an aggregation of the specified views. You can define your custom report by performing the following steps:

1. Define filter commands to isolate the message set that you want to capture in the final report. You can specify multiple *define\_filter* commands to achieve the final objective.
2. Once the filtered set is defined, you can create a layout for it by using the *define\_view* command.
3. Finally, you can hook-up various views in a single report by using the *define\_report* command.

Optionally, you can pass the path of the help file by using the *-helpfile* option, which is displayed by using the `help -report <report-name>` command.

It is recommended that you have complete report specification including *define\_filter*, *define\_view*, and *define\_report* in a separate Tcl file, which can be sourced as desired to define the report schema, and then generate this report for different projects.

**NOTE:** To view a list of all the valid reports in SpyGlass, refer to [Appendix C: SpyGlass Report Names](#).

## Arguments

This command has the following arguments:

**-name <report-name>**

Specifies the name of the report. If the report with same name is already defined, it is overwritten with the current definition.

**-view <view-list>**

Specifies a list of view names as defined by the [define\\_view](#) command. You should have previously defined the view names listed here by using the [define\\_view](#) command.

**-helpfile <help-file>**

Specifies the name of the help file containing the help of the report.

## Examples

```
sg_shell> define_report -name my_report -view v1  
# defines report 'my_report' using 'v1' view
```

```
sg_shell> define_report -name onlyFiltered -view v2 -helpfile  
./helpOnlyFiltered
```

## See Also

[define\\_filter](#), [define\\_view](#), [write\\_report](#)

## write\_report

**Generates the specified report**

### Syntax

```
write_report <report-name>
```

### Scope

Goal

### Return Value

None

### Description

The *write\_report* command generates the specified report. By default, the report is displayed on the screen. Optionally, you can redirect the output of the report to any file by using shell redirection operator, `>`, or capture that report in some file by using the *capture* command.

The report name specified by using this command can be any of the following:

- SpyGlass standard report, such as *moresimple*, *simple*, *waiver*, and so on. For further details on standard reports, refer to the *Reports Generated in Atrenta Console* section in the *Atrenta Console Reference Guide*.
- Product report, such as *Audit-Structure*, *CDC-report*, *CKSync01*, and so on.
- Any custom report defined by using the *define\_report* command
- The *goal\_summary* report that has information about all goals in the currently selected methodology
- The *goal\_setup* report that has information about rules and their parameters in the currently selected goal

**NOTE:** *The waived messages are not shown in any report except for the waiver report.*

You can view the list of all the reports available in the current scope by using the *help -report* command. For example, *goal\_setup* is available in goal scope only.

**NOTE:** To view a list of all the valid reports in SpyGlass, refer to [Appendix C: SpyGlass Report Names](#).

## Examples

```
sg_shell> write_report goal_summary  
# methodology goal summary report
```

```
sg_shell> current_goal initial_rtl/clock_reset_integrity/  
clock_reset_integrity
```

```
sg_shell> write_report goal_setup  
# goal setup report
```

```
sg_shell> write_report summary > summary.rpt  
# standard report
```

```
sg_shell> capture moresimple.rpt {write_report moresimple}
```

```
sg_shell> write_report onlyFiltered  
# custom report defined using define_report
```

```
sg_shell> write_report Clock-Reset-Detail  
# product specific report
```

## See Also

[define\\_filter](#), [define\\_view](#), [define\\_report](#), [write\\_aggregate\\_report](#)

## write\_aggregate\_report

Used to generate the specified aggregate report

### Syntax

```
write_aggregate_report <report_name>  
    [-config_file <file_path>]  
    [-reportdir <output_directory>]
```

### Scope

Any

### Return Value

None

### Description

The *write\_aggregate\_report* command generates the specified aggregate report.

The report name specified in the *write\_aggregate\_report* command can be one of the following:

- project\_summary
- DataSheet
- DashBoard

**NOTE:** *If SpyGlass Dashboard report generated using a project is executed through sg\_shell (TCL flow), the report shows the goal run status as Running even after the goal run is completed.*

### Arguments

This command has the following arguments:

**-config\_file <config\_file>**

Specifies a configuration file having project files path for which the aggregate report needs to be generated. If this argument is not specified, then by default, the aggregate report is generated for the current project.



## Reporting Commands

An error is displayed if you specify this argument when there is no current active project.

**-reportdir <output directory>**

The user can optionally redirect generated reports to a directory specified with this argument. By default, the report is generated in the <projectwdir>/<prj\_name>/<top\_name>/aggregated\_report directory. If the output directory is already present, it is overwritten. In addition, if this output directory is not present then only the leaf level directory is created.

**Examples**

```
# generate the 'datasheet' report for the current project -  
'project1.prj'
```

```
sg_shell> open_project project1.prj
```

```
sg_shell> write_aggregate_report datasheet
```

```
# generate the 'project_summary' report in the 'myreport'  
directory for the current project - 'project1.prj'
```

```
sg_shell> write_aggregate_report project_summary -reportdir  
myreport
```

## Waiver Commands

Interactive waiver commands provide the waiver intent directly on Tcl shell, referred as `sg_shell`. Before the introduction of interactive waiver commands, waivers were applied after running the goal. Now, the impact of waivers is visible just after using the `write_report` command. There is no need to run the `run_goal` command again to see the impact of waiver application. If there were messages present in standard or custom reports that you need to waive, you must specify additional waiver files and then generate these reports again by using the `write_report` command. Please refer to [Tcl Format Waiver Specification and Waiver Application](#) for more details.

Waiver commands can be supplied in Tcl shell in the following ways:

- **Directly specifying waiver commands in `sg_shell`:** The waiver content can be specified directly in `sg_shell`, as shown in the following examples:

```
sg_shell> waive -rule {Rule1}
sg_shell> waive -severity {warning}
sg_shell> waive -file_line {filename} {1}
```

Refer to the [waive](#) command for more details.

- **Specifying a new format waiver file (TCL compliant file):** The following command adds an AWL file that has waivers intent in Tcl format:

```
sg_shell> read_file -type awl <file_name>
```

In this way, AWL file behaves as if it has been sourced on the Tcl shell.

- **Sourcing a new format file:** The following command specifies a new format waiver file (TCL compliant) as if all commands in the file `<file_name>` have been directly specified in `sg_shell`:

```
sg_shell> source <file_name>
```

- **Specifying an old format waiver file (non-Tcl compliant file):** The old format waiver files can be read in `sg_shell`, as shown in the following example:

```
sg_shell> read_file -type waiver <file_name>
```

**NOTE:** Refer to `$SPYGLASS_HOME/examples/sg_shell/interactive_waivers` in your SpyGlass installation area for sample examples on how to effectively use waiver commands in `sg_shell`.

The following table describes the various waiver commands:

Command	Description
<i>waive</i>	Used for defining the criteria to waive a set of messages from the set of all generated messages
<i>get_waiver</i>	Used to get a list of waiver commands on the basis of filtering criteria, if specified
<i>report_waiver</i>	Reports user-specified waiver commands
<i>remove_waiver</i>	Used for removing waive commands
<i>save_waiver</i>	Saves the active waivers
<i>convert_sw12awl</i>	Converts waiver commands specified in an input file to corresponding AWL commands in the specified output file

## Tcl Format Waiver Specification and Waiver Application

### *Specification of Tcl Format Waiver Commands*

Waiver commands can be specified directly in `sg_shell` or in the new format waiver file, that is the AWL file (TCL compliant waiver file), as follows:

- For Boolean fields of waiver commands, such as *-invert* and *-regexp*, specify waiver commands directly in `sg_shell` without using curly brackets or double quotes, as shown in the following example:

```
sg_shell> waive -rule {R} -invert
```

- For waiver fields supporting single value, such as the *-msg* field, specify waiver commands as shown in the following example:

```
sg_shell> waive -msg {Design unit is top}
```

or

```
sg_shell> waive -msg "Design unit is top"
```

- For waiver fields supporting multiple values or fields that accept a list of values, such as *-rule*, *-severity*, *-file*, and *-du*, specify waiver commands as shown in the following example:

```
sg_shell> waive -rule { {R1} {R2} {R3} }
```

or

```
sg_shell> waive -rule { R1 R2 R3 }
```

or

```
sg_shell> waive -rule { "R1" "R2" "R3" }
```

It is mandatory to use double quotes, that is "", or curly brackets, that is {}, if list-based option values contain spaces, as shown in the following examples:

```
sg_shell> waive -du { "du name1" "du2" "du3" }
```

or

```
sg_shell> waive -du { {du name1} du2 du3 }
```

or

```
sg_shell> waive -du { {du name1} {du2} {du3} }
```

*but not like*

```
sg_shell> waive -du { du name1 {du2} {du3} }
```

The last example would act as if du and name1 are specified as separate names.

### ***Waiver Application***

The impact of waivers is visible just after using the [write\\_report](#) command. There is no need to run the [run\\_goal](#) command again to see the impact of waiver application.

The following examples demonstrate the simple command usage:

### **Adding waiver command**

```
sg_shell> waive -rule {R}
```

### **Reporting waiver commands**

```
sg_shell> report_waiver -verbose
```

### **Checking impact of waiver application**

Generating the *moresimple* file

```
sg_shell> write_report moresimple
```

Generating the *waiver* file

```
sg_shell> write_report waiver
```

## waive

Used for defining the criteria to waive a set of messages from the set of all generated messages

### Syntax

```

waive
  [-du <list of du> | <logical-lib-name>]
  [-ip <list of ip> | <logical-lib-name>]
  [-file <list of file>]
  [-file_line <file> <line>]
  [-file_lineblock <file> <start_line> <end_line>]
  [-severity <list of severity label/class>]
  [-rules <list of rules/groups or keywords>]
  [-msg <message>]
  [-except <list of rules/groups or keywords>]
  [-weight <weight>]
  [-weight_range <weight_start> <weight_end>]
  [-regexp | -exact]
  [-invert]
  [-comment]
  [-disable]
  [-import]
  [-ignore]

```

To know more about waiving messages during SpyGlass analysis in the Atrenta Console, refer to the *Waiving Messages by Using the waive Constraint* section in the *Atrenta Console User Guide*.

### Scope

Project

### Return Value

Returns a unique ID for every waiver command

### Description

The *waive* command is used to waive a set of messages from the generated messages. Sanity checking is done for the values that are

specified for the waive command options.

## Arguments

The *wave* command has the following arguments:

### **-du <list of du>**

This argument specifies a space-separated list of design unit names or the logical library name *<logical-lib-name>* of a precompiled Verilog or VHDL library. The design units can be any of the following types:

- <module-name>*: Module names for Verilog
- <entity-name>*: Entity names for the entity and all its architectures
- <entity-name>.<arch-name>*: Entity and the specified architecture
- <pkg-name>*: Package names
- <config-name>*: Configuration names for VHDL

Use this argument to waive the messages in the span of *duName* for the specified design units or for all design units in the specified library. The following examples illustrate the usage of this argument:

```
sg_shell> waive -du "duName"  
sg_shell> waive -du "du1 du2"  
sg_shell> waive -du {du1 du2}
```

**NOTE:** *If you want SpyGlass to consider schematic information to waive violations on design units, use the following command:*

```
set_option use_du_sch_hier yes
```

### **-ip <list of ip>**

This argument specifies a space-separated list of design unit names or the logical library name *<logical-lib-name>* of a precompiled Verilog or VHDL library. The design units can be of the following types:

- <module-name>*: Module names for Verilog
- <entity-name>*: Entity names for the entity and all its architectures

- ❑ `<entity-name>.<arch-name>`: Entity and the specified architecture
- ❑ `<pkg-name>`: Package names
- ❑ `<config-name>`: Configuration names for VHDL

Use this argument to waive the messages in the hierarchy of `ipName` for the specified design units (IP blocks) or all design units in the specified IP library. The following examples illustrate the usage of this argument:

```
sg_shell> waive -ip "ipName"
sg_shell> waive -ip ipName
sg_shell> waive -ip "ip1 ip2"
sg_shell> waive -ip {ip1 ip2}
```

**NOTE:** *If a module is instantiated in multiple IPs but you do not provide the waive -ip specification for each of these IPs, SpyGlass does not waive violations on such module instances when you specify the following command:*

```
set_option use_du_sch_hier yes
```

*However, if you set the above command to no, SpyGlass waives violations on only those module instances that are present in the IPs specified by the waive -ip specification.*

#### **-file <list of file>**

This argument specifies a space-separated list of source file names. Use this argument to waive all messages for the specified files. The following examples illustrate the usage of this argument:

```
sg_shell> waive -file fName
sg_shell> waive -file "fName"
sg_shell> waive -file "fName1 fName2"
sg_shell> waive -file {"fName1 " fName2"}
```

#### **-file\_line <file> <line>**

This argument specifies a space-separated pair of source file name and line number. Use this argument to waive the rule messages for a particular line of a source file. The following example illustrates the usage of this argument:

```
sg_shell> waive -file_line fName lineNumber
```

**NOTE:** *The -file\_line argument of the waive command does not support multiple files.*

**-file\_lineblock <file> <start\_line> <end\_line>**

This argument specifies a space-separated tuple of source file name, start line number, and end line number. Use this argument to waive the rule messages for a block of lines in a source file. The following example illustrates the usage of this argument:

```
sg_shell> waive -file_lineblock fName lineNum1 lineNum2
```

**NOTE:** *The -file\_lineblock argument of the waive command does not support multiple files.*

**-severity <list of severity label/class>**

This argument specifies the severity label or a severity class. Use this argument to waive the rule messages of the specified severity. The following example illustrates the usage of this argument:

```
sg_shell> waive -severity Info
```

**-rules <list of rules/groups or keywords>**

This argument specifies a space-separated list of rule names, rule group names, or product mnemonics. Use this argument to waive messages of the specified rules, rule groups, products, or by rule type keywords. The keywords that are allowed are as follows:

- ALL
- ALL\_INFO
- ALL\_WRN
- ALL\_ELAB
- ALL\_SYNTHERR
- ALL\_SYNTHWRN

The following examples illustrate the usage of this argument:

```
sg_shell> waive -rule R  
sg_shell> waive -rule "R"  
sg_shell> waive -rule "R1 R2"  
sg_shell> waive -rule {"R1" "R2"}
```

**NOTE:** *From the SpyGlass 5.3.1 release onwards, the -rule argument of the waive command has been made case sensitive to make it consistent with other arguments.*



**-msg <message>**

This argument specifies the rule message. Use this argument to waive a message. The *<message>* field should be specified within curly brackets { } to prevent any Tcl interpretation being applied on it. For example, a bit-select `top.i1.n1[0]` would be taken as command 0. Therefore, always enclose your message in { } for it to be read verbatim. The following examples illustrate the usage of this argument:

```
sg_shell> waive -msg message
sg_shell> waive -msg {message}
```

The *waivers\_translate\_generate\_names* option should be enabled while using a non escaped "generate block" name or an "instance array" name in the -msg field.

```
set_option waivers_translate_generate_names yes
```

**-except <list of rules/groups or keywords>**

This argument specifies a space-separated list of rule names, rule group names, or product mnemonics. Use this argument to exclude messages of the specified rules, rule groups, or products or by rule type keywords from the waived set. The keywords that are allowed are the same as those mentioned in the *-rules* argument. The following examples illustrate the usage of this argument:

```
sg_shell> waive -except R
sg_shell> waive -except "R"
sg_shell> waive -except "R1 R2"
sg_shell> waive -except {"R1" "R2"}
```

**-weight <weight>**

This argument specifies the rule weight value. Use this argument to waive the messages of the rules with the specified weight. The following example illustrates the usage of this argument:

```
sg_shell> waive -weight wt
```

**-weight\_range <weight\_start> <weight\_end>**

This argument specifies a weight range for message waiving. Use this argument to waive the messages of the rules with the weight within the specified range (both range values inclusive). The following example

illustrates the usage of this argument:

```
sg_shell> waive -weight_range wt1 wt2
```

### **-regex**

This is an optional argument that turns on regular expression matching for various fields. Use this argument to allow the use of regular expressions, such as `-du`, `-ip`, `-file`, `-file_line`, `-file_lineblock`, `-msg`, in other options. The following example illustrates the usage of this argument:

```
sg_shell> waive -msg msg1 -regex
```

### **-comment**

This argument is used to add comment as a single line text string enclosed in double quotes. The comment is printed in the waiver report and the sign\_off report. Comments can be specified to add some information about the *waive* command for readability. The following examples illustrate the usage of this argument:

```
sg_shell> waive -du du_name -comment "command to clean  
du_name based messages"
```

### **-disable**

This argument disables the *waive* command in which this option is specified. The following example illustrates the usage of this argument:

```
sg_shell> waive -rule R -disable
```

This means the command is disabled and will not be used.

When you specify the *-disable* argument, it acts on the current command only. Consider the following example:

```
sg_shell> ##command 1 waive -rule R ##command 2 waive  
-rule R -disable
```

This means command 2 is disabled, not command 1, because both these commands are separate commands.

### **-ignore**

This argument causes SpyGlass to list only waived message count in the Adjustments Waiver Report section of the waiver report and not the

actual waived messages.

### **-import**

This argument enables import of the waiver file, which is specified at the block level, to be used at the chip level. The following example illustrates the usage of this argument:

```
sg_shell> waive -import {{<block_name>}
<block_level_waiver_file>}
```

For more details, refer to the *Support for Hierarchical Waivers* section in the *Atrenta Console User Guide*.

The following are some exception:

- ❑ Waiver commands having `-import` argument are applied when the you are actually running a goal. If such waiver commands are specified after the `run_goal` command, these waiver commands are applied only during the consequent `run_goal` command. Therefore, if your standard/custom reports contain messages that you want to waive by using the `-import` command, re-run the goal and then generate these reports.
- ❑ You cannot edit or remove import-related commands in a block-level or a top-level file.
- ❑ Consider the following command.
 

```
waive -import {{<block-name>} <block-file>}
```

 In the above command, you can specify the block file `<block file>` only in the Tcl format.

**NOTE:** *The impact of waiver commands that use the `-import` argument will be seen after the `run_goal` command.*

### **-exact**

If the `-exact` argument is specified in the `waive` command, wildcard characters are considered as plain characters. The following example illustrates the usage of this argument:

```
sg_shell> waive -msg {message is this*} -exact
```

Here, `*` in the message field is treated as a literal character. If the `-exact` argument is not specified, `*` would act as a wildcard character.

**NOTE:** *The `-exact` argument can only be specified with TCL format waive command and not with non-TCL format waiver commands.*

**-invert**

This argument inverts the waiving selection. If a complete message set has 100 messages, and a set without *-invert* has 20 messages, the 80 remaining messages would be there in the waived set if *-invert* is specified. The following example illustrates the usage of this argument:

```
sg_shell> waive -msg msg1 -invert
```

**Examples**

```
sg_shell> waive -msg {Module clcell is a top level design  
unit} -regexp  
sg_shell> waive -du {cl.* sr.*} -regexp  
sg_shell> waive -severity {warning info} -except  
{DetectTopDesignUnits checkSGDC_01}  
sg_shell> waive -file_lineblock test.v 3 5 -file_lineblock  
test4.v 1 22 -file_lineblock abc.v 4 5  
sg_shell> waive -file_line test4.v 22 -file_line test4.v 4  
sg_shell> waive -weight 2  
sg_shell> waive -weight_range 2 10
```

**See Also**

[report\\_waiver](#), [remove\\_waiver](#), [get\\_waiver](#)

## get\_waiver

Used to get a list of waiver commands on the basis of filtering criteria, if specified

### Syntax

```
get_waiver  
  [-filter expression]
```

### Scope

Project

### Return Value

Returns an empty string or a collection of waiver commands in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_waiver* command is used to get a list of user-specified waiver commands on the basis of a filtering criterion, if specified.

### Arguments

The *get\_waiver* command has the following argument:

#### **[-filter expression]**

Use this argument to filter a group with an expression. The criterion supported in filter for this command is *file\_name*.

### Examples

#### **Example 1: Without -filter**

```
sg_shell> get_waiver  
_sggrp1
```

#### **Example 2: With -filter**

```
sg_shell> get_waiver -filter file_name==waiver1.awl
```

\_sggrp2

### Example 3: get\_waiver usage in report\_waiver in a single step (command nesting)

```
sg_shell> report_waiver [get_waiver -filter
                        file_name==waiver1.awl] -verbose
```

```
*****
Report : waive command(s) - user defined only
*****
```

```
+++++
ID  File           Line  Editable  Command
=====
0   waiver1.awl    1     yes      waive -rule {Reset_check07}
1   waiver1.awl    2     yes      waive -msg {msg}
+++++
```

### Example 4: get\_waiver usage in report\_waiver in two steps

```
sg_shell> set x [get_waiver -filter
                file_name==waiver1.awl]
```

\_sggrp5

```
sg_shell> report_waiver $x -verbose
```

```
*****
Report : waive command(s) - user defined only
*****
```

```
+++++
ID  File           Line  Editable  Command
=====
0   waiver1.awl    1     yes      waive -rule {Reset_check07}
1   waiver1.awl    2     yes      waive -msg {msg}
+++++
```

## See Also

[waive](#), [report\\_waiver](#), [remove\\_waiver](#), [report\\_adc](#), [get\\_adc](#)

## report\_waiver

Reports user-specified waiver commands

### Syntax

```
report_waiver
  [<collection>]
  [-verbose]
  [-hide_id]
```

### Scope

Project

### Return Value

Returns an empty string if the command is successfully executed and nothing in case of any error

### Description

The *report\_waiver* command displays the user-specified waiver commands. It displays the waiver commands that are provided in the AWL type files or specified on the `sg_shell`. It does not report the waiver commands specified in the SWL files.

### Arguments

The *report\_waiver* command has the following arguments:

#### [<collection>]

Collection used in the *report\_waiver* command is a set of waive commands that are filtered through the *get\_waiver* command. The *get\_waiver* command returns the name of a collection that internally contains a set of waive commands on the basis of the filtering criterion, if specified. If no filtering criterion is specified, collection contains a set of all user-defined waive commands. Collection can be supplied through either command nesting or storing value in a variable from the *get\_waiver* command and then using it in the next step for the *report\_waiver* command.

**[ -verbose ]**

Use this argument to display verbose connection information in columns. These columns are ID, File, Line, Editable, and Command. The description of these columns is as follows:

Column	Description
ID	Unique ID for a waiver command
File	File name of a waiver command
Line	Line number of a waiver command
Editable	A <code>yes</code> value means command can be edited. A <code>no</code> value means command cannot be edited. A <code>no</code> value is reported if <code>pragma2Constraint.sgdc</code> , which is formed due to waiver commands, is specified in RTL files through comments
Command	Waiver command in Tcl format

**[ -hide\_id ]**

Use this argument to disable the listing of the ID field.

**Examples****Example 1: -hide\_id and -verbose not specified**

```
sg_shell> report_waiver
*****
Report : waive command(s) - user defined only
*****

+++++
ID      Command
=====
0      waive -rule {Reset_check07}
1      waive -msg {msg}
2      waive -rule {R}
+++++
```



**Example 2: -hide\_id specified and -verbose not specified**

This case is useful when you want to redirect reported commands directly to a file.

```
sg_shell> report_waiver -hide_id
waive -rule {Reset_check07}
waive -msg {msg}
waive -rule {R}
```

**Example 3: -hide\_id not specified and -verbose specified**

```
sg_shell> report_waiver -verbose
*****
Report : waive command(s) - user defined only
*****

+++++
ID  File           Line Editable  Command
=====
0   waiver1.awl    1   yes       waive -rule {Reset_check07}
1   waiver1.awl    2   yes       waive -msg {msg}
2   waiver2.awl    1   yes       waive -rule {R}
+++++
```

**Example 4: -hide\_id and -verbose specified**

```
sg_shell> report_waiver -verbose -hide_id
*****
Report : waive command(s) - user defined only
*****

+++++
File           Line   Editable  Command
=====
waiver1.awl    1     yes       waive -rule {Reset_check07}
waiver1.awl    2     yes       waive -msg {msg}
+++++
```

```
waiver2.awl      1      yes      waive -rule {R}
+++++
```

### Example 5: Using the get\_waiver command

```
sg_shell> report_waiver [get_waiver -filter
file_name==waiver1.awl]
```

```
*****
Report : waive command(s) - user defined only
*****

+++++
ID      Command
=====
0      waive -rule {Reset_check07}
1      waive -msg {msg}
+++++
```

### Example 6: Using the get\_waiver command output in two passes

```
sg_shell> set x [get_waiver -filter
file_name==waiver1.awl]
_sggrp4
sg_shell> report_waiver $x
```

```
*****
Report : waive command(s) - user defined only
*****

+++++
ID      Command
=====
0      waive -rule {Reset_check07}
1      waive -msg {msg}
+++++
```

**See Also**

*[get\\_waiver](#), [waive](#), [remove\\_waiver](#)*

## remove\_waiver

Used for removing waive commands

### Syntax

```
remove_waiver  
  [-id <idNum>]  
  [<collection>]
```

### Scope

Project

### Return Value

None

### Description

The *remove\_waiver* command removes waiver commands on the basis of the specified criterion, which can be either the waiver command ID or collection. It removes the waivers provided in the AWL type files or the waiver commands directly given on the *sg\_shell*. There is no impact on the waivers present in the SWL files.

### Arguments

The *remove\_waiver* command has the following arguments:

**<idNum>**

Use this argument to specify the IDs of waiver commands in integers.

**<collection>**

Use this argument to report user-specified waiver commands that are filtered through the *get\_waiver* command.

### Examples

#### Example 1

In the following example, the *remove\_waiver* command removes the waiver command having ID as 3:

```
sg_shell> remove_waiver -id 3
```

### Example 2

In the following example, the *remove\_waiver* command removes the waiver commands having IDs as 1 and 2:

```
sg_shell> remove_waiver -id {1 2}
```

### Example 3

In the following example, the *remove\_waiver* command removes the waiver commands get from the *get\_waiver* command:

```
sg_shell> remove_waiver [get_waiver]
```

## See Also

[report\\_waiver](#), [waive](#), [get\\_waiver](#)

## save\_waiver

**Saves the active waivers**

### Syntax

```
save_waiver
```

### Scope

Project

### Return Value

None

### Description

The *save\_waiver* command saves an active waiver used in the project in Tcl format. This command can be used to save changes made to waivers in `sg_shell` through various commands, such as [remove\\_waiver](#). This will save the set active waivers to project.

### Arguments

None

### Examples

```
sg_shell> save_waiver
```

### See Also

[save\\_project](#), [save\\_adc](#)

## convert\_swl2awl

Converts waiver commands specified in an input file to corresponding AWL commands in the specified output file

### Syntax

```
convert_swl2awl  
  [<input_swl_file>]  
  [<output_awl_file>]
```

### Scope

Project

### Return Value

None

### Description

The `convert_swl2awl` command converts waiver commands in an input file to corresponding Atrenta Waiver Library, or AWL, commands in the specified output file.

**NOTE:** *If input file has "waive -import" command specified in it, you will need to separately convert the specified swl file to awl and then accordingly change the name of the converted file, in the input file.*

### Arguments

The `convert_swl2awl` command has the following arguments:

**<input\_swl\_file>**

Use this argument to specify the name of the input SWL file (non-Tcl format waiver file) that is passed in batch run to SpyGlass.

**<output\_awl\_file>**

Use this argument to specify the name of the output file for capturing the created AWL commands.

## Examples

### Example 1

In the following example, the test.awl file is output that contains the corresponding AWL commands:

```
sg_shell> convert_swl2awl test.swl test.awl
```

### Example 2

In the following example, the waiver.swl file contains the "waive - import" command. Hence, a message is reported suggesting to convert the specified swl file to awl file format and update the file name accordingly.

```
sg_shell> convert_swl2awl waiver.swl waiver.awl
```

```
Reading waiver file "waiver.swl" ...
```

```
convert_swl2awl: warning: block level waiver file(s) found.  
Please convert them to awl format and update the file name(s)  
in `waiver.awl'
```

## See Also

[\*convert\\_sgdc2adc\*](#)



## Debug Commands

Debug commands allow you to debug messages. Currently, you can open the Console UI to debug messages through schematic, waveform, and so on.

Debug commands can be categorized in the following groups:

- [General Debug Commands](#)
- [Design Query Commands](#)
- [Product Commands](#)
- [Built-in Attributes](#)
- [Product Attributes](#)

## General Debug Commands

The following table describes the various general debug commands:

Command	Description
<a href="#">gui_start</a>	Invokes SpyGlass Console
<a href="#">help</a>	Displays help for a particular command or item
<a href="#">report_design_status</a>	Reports design status as whether design has been linked, compiled, or flattened, and whether these views are up to date

## gui\_start

**Invokes Atrenta Console**

### Syntax

```
gui_start  
    [ -force ]
```

### Scope

Any

### Return Value

None

### Description

The *gui\_start* command invokes Atrenta Console with the currently active project. This command allows you to go back and forth between the shell environment and the GUI interface to interactively debug design issues.

Before launching the GUI, the current project is saved and closed. If the project could not be saved, this command fails to execute unless you specify the *-force* argument. After you exit Console GUI, *sg\_shell* reloads the last active project at the time of launching the GUI. If there are any changes done in the GUI environment and saved at the time of GUI exit, those changes are visible when control goes back to the *sg\_shell* prompt.

You can debug messages inside the GUI environment via schematic, waveform, and so on. You can also update project settings or fix RTL to resolve these messages, and come back to the *sg\_shell* prompt once the debugging is over.

### Arguments

This command has the following arguments:

#### **-force**

(Optional) Closes the current project and starts the GUI even if the project was not saved successfully.

## help

**Displays help for a particular command or item**

### Syntax

```
help
  [ -verbose ]
  [ -list ]
  [ -methodology | -goals | -rules | -params | -sgdc
    | -reports | -options | -preferences ]
  [ <pattern> ]
```

### Scope

Any

### Return Value

All the invocations, when done without a pattern but by using the *-list* argument, `sg_shell` returns an array of strings specifying the available values for the requested category, such as goals and methodology.

### Description

The *help* command displays the help for a particular command or item. This command is like the usual shell's help command. It is used to see the usage of any `sg_shell` command or item. The *help* command can be used for any of the following purposes:

- To show the list of commands under help
- To show the usage of a particular command
- To show the list of methodologies available
- To show the help of a particular methodology
- To show the list of goals available
- To show the help of a particular goal
- To show the list of rules available
- To show the help of a particular rule
- To show the list of parameters in current scope

- To show the help of a particular parameter
- To show the list of constraints in current scope
- To show the help of a particular constraint
- To show the list of reports available in current scope
- To show the help of a particular report
- To show the list of options in current scope
- To show the help of a particular option
- To show the list of recognized preference variables and help for any particular preference variable(s)

### Different Ways of Using the help Command

The following are the different ways to use the *help* command:

- If you specify the *help* command without specifying any arguments, all the *sg\_shell*-specific commands are displayed.
- If you specify the *help <command-pattern>* command, *sg\_shell* displays the help of all the commands matching the specified pattern.
- If you specify the command, *<command-name> -help*, *sg\_shell* displays the help of the specified command.

Other than the above ways, you can also invoke the *help* command in the following ways:

- `help [-list] -methodology [meth_pattern]`
- `help [-list] -goals [goal_pattern]`
- `help [-list] -rules [rule_pattern]`
- `help [-list] -params [param_pattern]`
- `help [-list] -sgdc [sgdc_pattern]`
- `help [-list] -reports [report_pattern]`
- `help [-list] -options [option_pattern]`
- `help [-list] -preferences [variable_pattern]`

**NOTE:** You can use the *'help -help'* command option on the *sg\_shell* prompt to view all the options accepted by the *help* command.

## Examples

```
sg_shell> help new_project
```

```
new_project          # starts a new project
```

Usage:

```
new_project <prj_name> [-projectwdir <path>] [-force]
```

```
sg_shell> help -verbose new_project
```

```
new_project          # starts a new project
```

Usage:

```
new_project <prj_name> [-projectwdir <path>] [-force]
```

Options:

```
-force                # do not ask user for confirmation
```

```
-projectwdir <path>  # directory to store all the project  
related data
```

```
<project>           # project file name for the new project
```

```
sg_shell> current_goal rtl_handoff/dft_readiness/dft_latches
```

```
sg_shell> help -rules TA_*
```

```
-- shows help for rules TA_01,TA_02,TA_06,TA_07,TA_08
```

## report\_design\_status

**Reports design status as whether design has been compiled or flattened, and whether these views are up to date**

### Syntax

```
report_design_status
```

### Scope

Project

### Return Value

A two-element list for each of compile and flat status of design

### Description

The *report\_design\_status* command reports the following:

- Whether your design has been successfully compiled and flattened
- Whether these views are present in memory and if yes, are they up to date

This command returns a two-element list, with each individual element a two element list in itself. First list reports whether synthesized design view is present in memory and whether it is up to date. Second list reports whether a flattened design view is present in memory and whether it is up to date.

### Arguments

None

### Examples

```
report_design_status
```

### See Also

[link\\_design](#), [compile\\_design](#), [run\\_goal](#)

## Design Query Commands

Design query commands allow you to run queries on your design. These commands can be used directly in `sg_shell` or can be included in Tcl scripts.

Tcl shell provides an access to *Library Object Model* through library commands. Library commands provide the flexibility to access different elements of SpyGlass Gates Object Model, or SGOM.

Tcl shell provides an access to synthesized netlist through netlist commands. The synthesized netlist can be accessed in three different forms: namely design unit view, flattened netlist view, or hierarchically flattened netlist view.

The design query commands related to various object models have *collection* as a way of communication between different commands. The collection encapsulates the objects of these object models, which are then exchanged among commands.

A variety of attributes is available that provides information about different object models. This information is captured in various data types, such as *int*, *float*, *string*, *boolean*, and so on.

Consider the following points pertaining to design query commands:

- Various meta-characters present in a regular expression have special meaning in Tcl shell. For example, if you want to use square brackets `[]` in a regular expression, Tcl interprets it as a command execution directive (in Tcl shell, anything specified inside `[]` is executed as a command). Therefore, it is recommended that you give regular expression patterns inside brackets `{ }`. Tcl shell takes anything specified inside brackets `{ }` as a pattern only and does not interpret any special meaning of the characters.
- It is recommended that you set the `enable_save_restore` option to `yes` for saving design database. This setting allows faster goal runs due to the reuse of design database across goal runs. It also ensures that design exploration inside `sg_shell` works seamlessly across different `sg_shell` sessions. If this option is set to `no`, the design created in the last `sg_shell` session is not available during subsequent `sg_shell` invocation, and needs to be recreated through the `compile_design` or `run_goal` commands.

- For multidimensional signals, only part select in the least significant position is allowed. For example, if you want to search a [1:2] [1:3] in a vector net, a [0:2] [0:4], you must specify a [1] [1:3] and a [2] [1:3] separately.

**NOTE:** Refer to `$SPYGLASS_HOME/examples/sg_shell/design_query` in your SpyGlass installation area for sample examples on how to effectively use design query commands in `sg_shell`.

The design query commands can be categorized in the following groups:

- *Library Commands*
- *Netlist Commands*
- *Collection Commands*
- *Attribute Commands*

## Library Commands

Tcl shell provides an access to *Library Object Model* through library commands. The Library Object Model is basically populated from `.sglib` that is specified as part of design inputs. Also known as SpyGlass Gates Object Model, or SGOM, this model has information about different attributes defined in `.lib`, and other derived information in terms of cell type, pin functionality, and so on.

Library commands provide the flexibility to access different elements of SGOM. These elements include the following:

- **lib:** SGOM library node
- **lib\_cell:** SGOM library cell that is a part of the *lib* node
- **lib\_pin:** SGOM cell pin that is a part of the *lib\_cell* node
- **lib\_timing\_arcs:** SGOM timing arcs from one *lib\_pin* node to another *lib\_pin* node

You can access these nodes through name, parent or child relationship, or by passing corresponding objects of netlist object models. There are various application attributes defined on each of these SGOM objects that can be fetched using the attribute-related commands.

To specify wildcard or regular expression in library commands, the pattern should start from the library and mention every subsequent hierarchy till



the end. You can specify patterns where library or other hierarchies are implied. Consider the following examples:

- All pins of a cell `AND2` of a library `NEW` can be referred as `NEW.AND2.*`
- All pins of all cells of a library `NEW` can be referred as `NEW.*.*`
- All pins of all cells of all libraries can be referred as `*.*.*`
- All pins named `A` of all cells of all libraries can be referred as `*.*.A`
- All pins of all cells named `AND2` of all libraries can be referred as `*.AND2.*`

The following table describes the various library commands:

<b>Command</b>	<b>Description</b>
<a href="#"><i>get_libs</i></a>	Used to get a list of libraries currently loaded in <code>sg_shell</code>
<a href="#"><i>get_lib_cells</i></a>	Used to get a list of library cells currently loaded in <code>sg_shell</code>
<a href="#"><i>get_lib_pins</i></a>	Used to get a list of library pins currently loaded in <code>sg_shell</code>
<a href="#"><i>get_lib_timing_arcs</i></a>	Used to get a list of library timing arcs from libraries currently loaded in <code>sg_shell</code>

## get\_libs

Used to get a list of libraries currently loaded in `sg_shell`

### Syntax

```
get_libs
  [-filter expression]
  [-regex | -exact]
  [patterns | -of_objects objects]
```

### Scope

Project

### Return Value

Returns an empty string or a collection of libraries in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the `catch` command.

### Description

The `get_libs` command is used to get a list of libraries that are currently loaded in `sg_shell` and that match certain criteria. This command returns a collection if any library matches the `patterns` and passes the filtering criterion, if specified.

### Arguments

The `get_libs` command has the following arguments:

#### <patterns>

Use this argument to match library names against patterns. Patterns can include wildcard characters `*` and `?` or regular expressions, based on the `-regex` argument. Patterns can also include collections of type `lib`. The `patterns` and `-of_objects` arguments are mutually exclusive. You can specify only one at a time.

#### <-of\_objects objects>

Use this argument to create a collection of libraries that contain the

specified objects. In this case, each object is either a named library cell or a library cell collection. The *-of\_objects* and *patterns* arguments are mutually exclusive. You can specify only one at a time.

### **[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search objects that contain the *\** and *?* wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can specify only one at a time.

### **[-filter expression]**

Use this argument to filter collection with an expression. For any library that matches *patterns* (or *objects*), the expression is evaluated on the basis of the library's attributes. If the expression evaluates to `true`, the library is included in the result.

## Examples

```
sg_shell> get_libs
{"mylib_20c", "t*ypical", "fast", "tlib"}

sg_shell> get_libs t*
{"t*ypical", "tlib"}

sg_shell> get_libs -regexp t.*
{"t*ypical", "fast", "tlib"}

sg_shell> get_libs -exact t*ypical
{"t*ypical"}

sg_shell> get_libs -of_objects {mylib_20c.AN2
fast.ACCSHCINX2}
{"mylib_20c", "fast"}

sg_shell> get_libs -regexp -of_objects {{mylib_20c\.*}}
{"mylib_20c"}

sg_shell> define_user_attribute -class lib -type int intAttr
define_user_attribute: info: defining new attribute
'intAttr' of type 'int'
```

```
sg_shell> set_user_attribute -class lib {fast tlib} intAttr
10
sg_shell> set_user_attribute -class lib {mylib_20c t*ypical}
intAttr 55
sg_shell> get_libs * -filter "intAttr == 10"
{"fast", "tlib"}
sg_shell> foreach_in_collection i [get_libs] {
  puts "lib_name = [get_attribute $i base_name]
    (sglib_name = [get_attribute $i sglib_name],
    technology = [get_attribute $i technology])"
}
lib_name = mylib_20c (sglib_name = new/autogenerated_sglib/
aggregate.sglib, technology = cmos)
lib_name = t*ypical (sglib_name = new/autogenerated_sglib/
aggregate.sglib, technology = cmos)
lib_name = fast (sglib_name = new/autogenerated_sglib/
aggregate.sglib, technology = cmos)
lib_name = tsmc (sglib_name = new/autogenerated_sglib/
aggregate.sglib, technology = cmos)
```

## See Also

[get\\_lib\\_cells](#), [get\\_lib\\_pins](#), [get\\_lib\\_timing\\_arcs](#), [filter\\_collection](#)

## get\_lib\_cells

Used to get a list of library cells currently loaded in `sg_shell`

### Syntax

```
get_lib_cells
  [-filter expression]
  [-regexp | -exact]
  <patterns | -of_objects objects>
```

### Scope

Project

### Return Value

Returns an empty string or a collection of library cells in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the `catch` command.

### Description

The `get_lib_cells` command is used to get a list of library cells that are currently loaded in `sg_shell` and that match certain criteria. This command returns a collection if any library cell matches the *patterns* or *objects* and passes the filtering criterion, if specified.

### Arguments

The `get_lib_cells` command has the following arguments:

#### <patterns>

Use this argument to match library cell names against patterns. Patterns can include wildcard characters `*` and `?` or regular expressions, based on the `-regexp` argument. Patterns can also include collections of type `lib_cell`. The *patterns* and `-of_objects` are mutually exclusive. You can specify only one at a time.

**<-of\_objects objects>**

Use this argument to create a collection of library cells that contain the specified objects. In this case, each object can be a named library, a library pin, a netlist cell, a library pin collection, or a netlist cell collection. The *-of\_objects* and *patterns* are mutually exclusive. You can specify only one at a time.

**[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search objects that contain the *\** and *?* wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can specify only one at a time.

**[-filter expression]**

Use this argument to filter collection with an expression. For any library cell that matches *patterns* (or *objects*), the expression is evaluated on the basis of the library cell's attributes. If the expression evaluates to *true*, the library cell is included in the result.

**Examples**

```
sg_shell> get_lib_cells mylib_20c.AN*
{"mylib_20c.AN2", "mylib_20c.AN3", "mylib_20c.AN2P",
"mylib_20c.AN3P", "mylib_20c.AN4", "mylib_20c.AN4P"}

sg_shell> get_lib_cells -regexp {mylib.*\AN[0-9]}
{"mylib_20c.AN2", "mylib_20c.AN3", "mylib_20c.AN2P",
"mylib_20c.AN3P", "mylib_20c.AN4", "mylib_20c.AN4P"}

sg_shell> get_lib_cells -exact t*ypical.ACHCI~NX2
{"t*ypical.ACHCI~NX2"}

sg_shell> get_lib_cells -of_objects mylib_20c.AN2.A
{"mylib_20c.AN2"}

sg_shell> get_lib_cells -of_objects U*
{"mylib_20c.AN2", "mylib_20c.OR2", "mylib_20c.AN3",
"mylib_20c.OR3"}
```

```

sg_shell> define_user_attribute -class lib_cell -type string
cellAttr
define_user_attribute: info: defining new attribute
'cellAttr' of type 'string'

sg_shell> set_user_attribute -class lib_cell [get_lib_cells
mylib_20c.AN*] cellAttr attr1

sg_shell> set_user_attribute -class lib_cell [get_lib_cells
mylib_20c.AO*] cellAttr attr6

sg_shell> get_lib_cells mylib_20c.A* -filter
"cellAttr == attr1"
{"mylib_20c.AN2", "mylib_20c.AN3", "mylib_20c.AN2P",
"mylib_20c.AN3P", "mylib_20c.AN4", "mylib_20c.AN4P"}

sg_shell> foreach_in_collection i [get_lib_cells -of_objects
[get_libs fast] -filter {is_sequential==true}] {
  puts "cell_name = [get_attribute $i base_name]
(area = [get_attribute $i area],
no. of pins = [get_attribute $i number_of_pins])"
}
cell_name = DFFHQX1 (area = 35.64540100097656,
no. of pins = 3)
cell_name = DFFHQX2 (area = 35.64540100097656,
no. of pins = 3)
cell_name = DFFHQX4 (area = 45.82979965209961,
no. of pins = 3)
cell_name = DFFHQX8 (area = 49.22460174560547,
no. of pins = 3)
cell_name = DFFNSRX1 (area = 37.34280014038086,
no. of pins = 6)
cell_name = DFFNSRX2 (area = 40.73759841918945,
no. of pins = 6)
cell_name = DFFNSRX4 (area = 56.014198303222656,
no. of pins = 6)
cell_name = DFFNSRXL (area = 37.34280014038086,
no. of pins = 6)

```

## See Also

*[get\\_libs](#), [get\\_lib\\_pins](#), [get\\_lib\\_timing\\_arcs](#), [filter\\_collection](#)*



## get\_lib\_pins

Used to get a list of library pins currently loaded in `sg_shell`

### Syntax

```
get_lib_pins
  [-filter expression]
  [-regexp | -exact]
  < patterns | -of_objects objects >
```

### Scope

Project

### Return Value

Returns an empty string or a collection of library pins in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the `catch` command.

### Description

The `get_lib_pins` command is used to get a list of library pins that are currently loaded in `sg_shell` and that match certain criteria. This command returns a collection if any library cell pin matches the `patterns` or `objects` and passes the filtering criterion, if specified.

### Arguments

The `get_lib_pins` command has the following arguments:

#### <patterns>

Use this argument to match library pin names against patterns. Patterns can include wildcard characters `*` and `?` or regular expressions, based on the `-regexp` argument. Patterns can also include collections of type `lib_pin`. The `patterns` and `-of_objects` arguments are mutually exclusive. You can specify only one at a time.

**<-of\_objects objects>**

Use this argument to create a collection of library pins that contain the specified objects. In this case, each object can be either a named library cell, netlist pin, a library cell collection, or a netlist pin collection. The *-of\_objects* and *patterns* arguments are mutually exclusive. You can specify only one at a time.

**[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search objects that contain the *\** and *?* wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can use only one at a time.

**[-filter expression]**

Use this argument to filter collection with an expression. For any library pin that matches *patterns* (or *objects*), the expression is evaluated on the basis of the library pin's attributes. If the expression evaluates to *true*, the library pin is included in the result.

**Examples**

```
sg_shell> get_lib_pins mylib_20c.AN2.*
{"mylib_20c.AN2.A", "mylib_20c.AN2.B", "mylib_20c.AN2.Z"}

sg_shell> get_lib_pins -regexp {mylib.*\.[AN][0-9]\.Z}
{"mylib_20c.AN2.Z", "mylib_20c.AN3.Z", "mylib_20c.AN2P.Z",
"mylib_20c.AN3P.Z", "mylib_20c.AN4.Z", "mylib_20c.AN4P.Z"}

sg_shell> get_lib_pins -exact t*typical.ACHCI~NX2.B%B
{"t*typical.ACHCI~NX2.B%B"}

sg_shell> get_lib_pins -of_objects mylib_20c.AN2
{"mylib_20c.AN2.A", "mylib_20c.AN2.B", "mylib_20c.AN2.Z"}

sg_shell> get_lib_pins -of_objects U1.A
{"mylib_20c.OR2.A"}

sg_shell> define_user_attribute -class lib_pin -type float
pinAttr
```

## Debug Commands

```

define_user_attribute: info: defining new attribute
'pinAttr' of type 'float'

sg_shell> set_user_attribute -class lib_pin [get_lib_pins
mylib_20c.AN*.A] pinAttr 5.24

sg_shell> set_user_attribute -class lib_pin [get_lib_pins
mylib_20c.OR*.A] pinAttr 2.22

sg_shell> get_lib_pins mylib_20c.*.* -filter
"pinAttr == 5.24"
{"mylib_20c.AN2", "mylib_20c.AN3", "mylib_20c.AN2P",
"mylib_20c.AN3P", "mylib_20c.AN4", "mylib_20c.AN4P"}

sg_shell> foreach_in_collection i [get_lib_pins -of_objects
[get_lib_cells mylib_20c.AN2*]] {
    puts "pin_name = [get_attribute $i full_name]
        (direction = [get_attribute $i direction])"
}
pin_name = mylib_20c.AN2.A (direction = input)
pin_name = mylib_20c.AN2.B (direction = input)
pin_name = mylib_20c.AN2.Z (direction = output)
pin_name = mylib_20c.AN2P.A (direction = input)
pin_name = mylib_20c.AN2P.B (direction = input)
pin_name = mylib_20c.AN2P.Z (direction = output)

```

**See Also**

[get\\_libs](#), [get\\_lib\\_cells](#), [get\\_lib\\_timing\\_arcs](#), [filter\\_collection](#)

## get\_lib\_timing\_arcs

Used to get a list of library timing arcs from the libraries currently loaded in `sg_shell`

### Syntax

```
get_lib_timing_arcs
  [-filter expression]
  [-regexp | -exact]
  < -of_objects objects | -from from_pin | -to to_pin >
```

### Scope

Project

### Return Value

Returns an empty string or a collection of library timing arcs in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the `catch` command.

### Description

The `get_lib_timing_arcs` command is used to get a list of library timing arcs from libraries that are currently loaded in `sg_shell` and that match certain criteria. This command returns a collection of timing arcs related to given pins or cells and passes the filtering criterion, if specified.

### Arguments

The `get_lib_timing_arcs` command has the following arguments:

#### <-of\_objects objects>

Use this argument to specify library cells. All library cell arcs of that cell are considered.

#### <-from from\_pin>

Use this argument to specify the `from` library pins or netlist terminals. All forward library arcs from the specified library pins or corresponding netlist terminals are considered.

**<-to to\_pin>**

Use this argument to specify the `to` library pins or netlist terminals. All backward library arcs from the specified library pins or corresponding netlist terminals are considered.

**[-regexp | -exact]**

If the `-regexp` switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the `-exact` switch is provided, simple pattern matching is disabled. This is used to search objects that contain the `*` and `?` wildcard characters. The `-exact` and `-regexp` arguments are mutually exclusive. You can use only one at a time.

**[-filter expression]**

Use this argument to filter collection with an expression. For any library timing arc that matches patterns or objects, the expression is evaluated on the basis of library timing arc's attributes. If the expression evaluates to `true`, the library timing arc is included in the result.

**Examples**

The `get_lib_timing_arcs` command returns only a collection. You have to write a proc in Tcl to see the results data included in the collection. You may use the following proc to see the results in `sg_shell`:

```
proc report_lib_timing_arcs {args} {
    set lib_arcs [eval [concat get_lib_timing_arcs $args]]
    puts [format "%15s %-15s %18s %18s" "from_lib_pin"
        "to_lib_pin" \
        "timing_type" "timing_sense"]
    puts [format "%15s %-15s %15s %18s" "-----"
        "-----" \ "-----" "-----"]
    foreach_in_collection lib_arc $lib_arcs {
        set fpin [get_attribute $lib_arc from_lib_pin]
        set fname [get_attribute $fpin base_name]
        set tpin [get_attribute $lib_arc to_lib_pin]
        set tname [get_attribute $tpin base_name]
        set type [get_attribute $lib_arc timing_type]
        set sense [get_attribute $lib_arc timing_sense]
```

```

    puts [format "%15s -> %-15s %15s %18s" $fname $tname
$type $sense]
    }
}

```

```

sg_shell> report_lib_timing_arcs -from mylib_20c.AN2.A
from_lib_pin to_lib_pin      timing_type      timing_sense
-----
          A -> Z              timing_type_undef  timing_sense_undef

```

```

sg_shell> report_lib_timing_arcs -regexp -to
{mylib.*\AN[0-9]\.Z}
from_lib_pin to_lib_pin      timing_type      timing_sense
-----
          A -> Z              timing_type_undef  timing_sense_undef
          B -> Z              timing_type_undef  timing_sense_undef
          A -> Z              timing_type_undef  timing_sense_undef
          B -> Z              timing_type_undef  timing_sense_undef
          C -> Z              timing_type_undef  timing_sense_undef
          B -> Z              timing_type_undef  timing_sense_undef
          C -> Z              timing_type_undef  timing_sense_undef
          D -> Z              timing_type_undef  timing_sense_undef
          A -> Z              timing_type_undef  timing_sense_undef
          A -> Z              timing_type_undef  timing_sense_undef
          B -> Z              timing_type_undef  timing_sense_undef
          C -> Z              timing_type_undef  timing_sense_undef
          B -> Z              timing_type_undef  timing_sense_undef
          A -> Z              timing_type_undef  timing_sense_undef
          A -> Z              timing_type_undef  timing_sense_undef
          B -> Z              timing_type_undef  timing_sense_undef
          C -> Z              timing_type_undef  timing_sense_undef
          D -> Z              timing_type_undef  timing_sense_undef

```

```

sg_shell> report_lib_timing_arcs -exact -from
t*typical.ACHCI~NX2.B%B
from_lib_pin to_lib_pin      timing_type      timing_sense
-----

```

## Debug Commands

```

B%B -> CO          timing_type_undef    positive_unate
B%B -> CO          timing_type_undef    positive_unate
B%B -> CO          timing_type_undef    positive_unate

```

```

sg_shell> report_lib_timing_arcs -from [get_pins U*.A]
from_lib_pin to_lib_pin      timing_type      timing_sense
-----
A -> Z          timing_type_undef    timing_sense_undef
A -> Z          timing_type_undef    timing_sense_undef
A -> Z          timing_type_undef    timing_sense_undef
A -> Z          timing_type_undef    timing_sense_undef

```

```

sg_shell> report_lib_timing_arcs -of_objects mylib_20c.AN2
from_lib_pin to_lib_pin      timing_type      timing_sense
-----
A -> Z          timing_type_undef    timing_sense_undef
B -> Z          timing_type_undef    timing_sense_undef

```

```

sg_shell> define_user_attribute -class lib_timing_arcs -type
int timingAttr

```

Defining new attribute 'timingAttr' of type 'int'

```

sg_shell> set_user_attribute -class lib_timing_arcs
[get_lib_timing_arcs -from mylib_20c.AN*.A] timingAttr 67

```

```

sg_shell> set_user_attribute -class lib_timing_arcs
[get_lib_timing_arcs -to mylib_20c.OR*.Z] timingAttr 11

```

```

sg_shell> report_lib_timing_arcs mylib_20c.A* -filter
"timingAttr == 11"
from_lib_pin to_lib_pin      timing_type      timing_sense
-----
A -> Z          timing_type_undef    timing_sense_undef
B -> Z          timing_type_undef    timing_sense_undef
A -> Z          timing_type_undef    timing_sense_undef
A -> Z          timing_type_undef    timing_sense_undef
B -> Z          timing_type_undef    timing_sense_undef
B -> Z          timing_type_undef    timing_sense_undef
C -> Z          timing_type_undef    timing_sense_undef

```

A -> Z	timing_type_undef	timing_sense_undef
B -> Z	timing_type_undef	timing_sense_undef
C -> Z	timing_type_undef	timing_sense_undef
A -> Z	timing_type_undef	timing_sense_undef
C -> Z	timing_type_undef	timing_sense_undef
D -> Z	timing_type_undef	timing_sense_undef
A -> Z	timing_type_undef	timing_sense_undef
B -> Z	timing_type_undef	timing_sense_undef
C -> Z	timing_type_undef	timing_sense_undef
D -> Z	timing_type_undef	timing_sense_undef
B -> Z	timing_type_undef	timing_sense_undef

## See Also

[get\\_libs](#), [get\\_lib\\_cells](#), [get\\_lib\\_pins](#), [filter\\_collection](#)



## Netlist Commands

Tcl shell provides an access to synthesized netlist through netlist commands. The synthesized netlist can be accessed in three different forms: namely design unit, flattened netlist, or hierarchically flattened netlist view. Decide and set the appropriate view by using the `set_pref dq_design_view_type <du/flat/hier_flat>` command. To hierarchically flatten your design, compile your design or run a goal by using the [enable\\_hier\\_flattening](#) option.

In the `du` view, all the design query commands work relative to the current design. Therefore, you cannot cross the boundaries of the current design while working in the `du` view. If you want to traverse across boundaries, current design should be changed explicitly. If current design is `top`, it essentially means that `du_cell/du_net/du_port` can be specified as `top.X` or `X` whereas `du_pin` can be specified as `top.I.P` or `I.P`, where `X` is a cell, net, or port pattern, and `P` is a pin pattern.

In the `flat` view, design query works relative to the currently set top design unit, which is essentially the current design itself.

In the `hier_flat` view, all the design query commands work relative to the [current\\_instance](#). By default, the [current\\_instance](#) is set to `top`. Therefore, all queries work relative to `top`. When the [current\\_instance](#) is:

- **Not set or reset:** A cell, net, or port can be specified as `top.X`, `X`, `top.U.X`, or `U.X`, whereas a pin can be specified as `top.I.P`, `I.P`, `top.U.I.P`, or `U.I.P`, where `X` is a cell, net, or port pattern, `P` is a pin pattern, and `U` is a top-level hierarchical instance pattern.

**NOTE:** *Using `*` at the beginning of the pattern matches it with `top` and the rest accordingly. It means `*.P` cannot match any pin of a top-level instance, but can match a top-level instance, port, or net named `P`.*

- **Set to a hierarchical instance:** A cell and net can be specified as `X` or `U.X`, whereas a pin can be specified as `I.P` or `U.I.P`, where `X` is a cell, net, or port pattern, `P` is a pin pattern, and `U` is a hierarchical instance pattern inside the current instance.

**NOTE:** *The provided pattern should not include the current instance.*

The basic elements of a netlist view are as follows:

- **du\_cell/flat\_cell**: Design unit (du) or flattened instance
- **du\_pin/flat\_pin**: Terminal corresponding to du\_cell or flat\_cell
- **du\_net/flat\_net**: Net in design unit (du) or flattened view
- **du\_port/flat\_port**: Ports as given for a design
- **design**: Current design, that is, module in design unit (du) view, and top-level module in flattened (flat) view

**NOTE:** *Netlist commands in the hier\_flat view also returns flat\_cell, flat\_net, flat\_pin, and flat\_port objects. But, unlike the flat view, the objects returned may be a hierarchical cell, net, or pin.*

Similar to library commands, netlist commands support different ways to get handle for the above objects through name, parent or child relationship, or by connectivity pattern among different object types. Use application attributes, which are additionally provided for guided traversal and decision making, as you are building queries on object models.

To specify exact, wildcard, or regular expression in netlist commands, the pattern may or may not start from the top-level module of the design, but it should mention every subsequent hierarchy till the end. You can specify patterns where the top-level module is implied. In netlist commands, exact, wildcard, or regular expression is not allowed in the name field of the top-level module. You can specify exact, wildcard, or regular expression in any hierarchy other than the top-level module of the design.

### Examples

Let us assume that *top* is the top-level design unit set as both top and current design. There are a number of cells present in hierarchy, such as I1, I2, and so on. Here, I1 is a cell present at the first level of hierarchy after *top*, I2 is present at the second level of hierarchy, and so on.

#### In the du view

All the examples given below are related to du cell, net, or port. As already described, one more level of hierarchy is accepted for the du pin.

- `top.I1` or `I1` returns the `top.I1` cell, where *top* was implied in the latter pattern.
- `*` matches `top.I1`, `top.J1`, `top.K1`, and so on. It indicates all cells that are present at the first level of hierarchy after *top*.

- `top.I1.I2` does not match anything because you are specifying more than two hierarchies.
- `p*` matches `top.p1`, `top.pp1`, and so on. It indicates that the pattern matches with the cells present at the first level of hierarchy after *top*.
- `t*.p*` does not match anything because `t*` does not match *top*. Here, you can explicitly specify *top*; otherwise, it is implied.
- `*.I1` or `t*.I1` does not match anything because patterns are not allowed in *top* and specifying more than two hierarchies is not possible in the `du` view.

### In the `flat` view

All the examples given below are related to the `flat` cell.

- `top.I1` or `I1` returns the `top.I1` cell, where *top* was implied in the latter pattern.
- `*` matches all cells in the design.
- `p*` matches `top.p1`, `top.p2`, and so on. It indicates the pattern matches with the cells present at the first level of hierarchy after *top*.
- `t*.*` matches `top.t1`, `top.t2`, and so on.
- `t*.*.*` matches `top.t1.tt1`, `top.t1.tt2`, `top.t2.tt3`, `top.t1.tt4`, and so on.
- `t*.I1.I2` matches `top.I1.I2`.

In summary, if a pattern does not start with the currently selected `<current_design>`, the `<current_design>` is implicitly assumed, and the pattern is matched in the selected `<current_design>`. This holds true for exact, wildcard, or regular expression pattern specifications.

### In the `hier_flat` view

Consider a design `top` containing hierarchical instances `H11`, `H12`, and `H13`, and top-level leaf instances `L11`, `L12`, and `L13`. The top-level hierarchical instance `H12` contains `H21`, `H22`, `L21`, and `L22`. `H21` further contains `L31` and `L32`.

- The `get_cells` command from `top` returns `H11`, `H12`, `H13`, `L11`, `L12`, `L13`.

- The `get_cells` command from H12 (i.e., the `current_instance` is set to `top.H12`) returns H21, H22, L21, L22.
- The `get_cells` command from H21 (i.e., the `current_instance` is set to `top.H12.H21`) returns L31, L32.
- `get_cells top.H*.H*` or `get_cells H*.H*` from top returns H21, H22 (i.e., top may or may not be mentioned if the `current_instance` is reset to top). But, top is always matched first. It means if the starting pattern is `*`, that is, `*.H*`, it always matches the first `*` with top and return H11, H12, H13.
- If the `current_instance` is set, the input pattern should not contain the current instance. In other words, if the `current_instance` is set to H12, the `get_cells H12.*` command does not match anything.
- The above properties are also applicable while searching pins and nets in the `hier_flat` view.

### List of Netlist Commands

The following table describes various netlist commands:

Command	Description
<code>current_instance</code>	Used to select a scope (instance) for design query on hierarchically flattened netlist
<code>current_design</code>	Used to select a current design for interactive constraint and design query commands
<code>get_cells</code>	Creates a list of cells in the current design that match certain criteria
<code>get_nets</code>	Creates a list of nets in the current design that match certain criteria
<code>get_pins</code>	Creates a list of pins in the current design that match certain criteria
<code>get_ports</code>	Creates a list of ports in the current design that match certain criteria
<code>report_cell</code>	Used to display information and statistics about cells in the current instance or current design
<code>get_fanin_pins</code>	Creates a list of fan-in pins in the design that match certain criteria

Command	Description
<a href="#"><i>get_fanin_ports</i></a>	Creates a list of fan-in ports in the design that match certain criteria
<a href="#"><i>get_fanout_pins</i></a>	Creates a list of fan-out pins in the design that match certain criteria
<a href="#"><i>get_fanout_ports</i></a>	Creates a list of fan-out ports in the design that match certain criteria
<a href="#"><i>get_master</i></a>	Returns the master module of the specified instance
<a href="#"><i>get_parent</i></a>	Returns the parent node of the specified object
<a href="#"><i>get_clocks</i></a>	Creates a list of user-defined clocks in the current design
<a href="#"><i>report_clocks</i></a>	Reports properties of user-specified clocks in current design
<a href="#"><i>get_registers</i></a>	Used to get a list of cells driven by specified clocks/resets
<a href="#"><i>get_resets</i></a>	Creates a list of user defined resets in current design
<a href="#"><i>get_value</i></a>	Used to get simulation value of specified design object (port, pin, or net) in last cycle
<a href="#"><i>propagate_clocks</i></a>	Propagates the user-defined clocks
<a href="#"><i>propagate_resets</i></a>	Propagates the user-defined resets
<a href="#"><i>get_domains</i></a>	Creates a list of domains of the user-defined clocks in the current design
<a href="#"><i>report_domains</i></a>	Reports the list of clocks of the specified domains
<a href="#"><i>report_resets</i></a>	Reports properties of user specified resets in current design

**NOTE:** *The following Netlist commands and their option usages may change across releases, with proper notification, because the use model evolves to improve productivity and effectiveness.*

- [\*get\\_clocks\*](#)
- [\*report\\_clocks\*](#)
- [\*get\\_registers\*](#)
- [\*get\\_value\*](#)

For scripting purpose, use your own Tcl wrappers for these commands to enable easy adoption to command changes.

## current\_instance

Used to select a scope (instance) for design query on hierarchically flattened netlist

### Syntax

#### Usage 1

```
current_instance <pattern>
```

#### Usage 2

```
current_instance <collection>
```

#### Usage 3

```
current_instance .
```

#### Usage 4

```
current_instance
```

### Scope

Project

### Return Value

**Usage 1, 2, and 3:** Returns a new single object collection, which contains the *current\_instance* object, in case of successful execution. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

**Usage 4:** Returns nothing.

### Description

The *current\_instance* command sets the scope of the design query in the *hier\_flat* (*dq\_design\_view\_type*) mode. The command enables all design query commands to be used relative to the current instance in the design hierarchy.

This command can be specified in the following four modes:

- **current\_instance <pattern>:** Use this mode to set the current instance that matches the provided pattern. The number of matches should be 1. The matched instance should be a hierarchical instance.

Otherwise, it results in an error. If the pattern is an empty string (""), the current instance is reset and the scope is set to `top`.

- `current_instance <collection>`: Use this mode to set the current instance with the hierarchical cell object contained in the argument collection. The collection should be a single object collection. The object contained should be a hierarchical cell. Otherwise, it results in an error. If the collection is empty (""), the current instance is reset and the scope is set to `top`.
- `current_instance .:` It is known as the query mode of operation, which returns the current instance that is set in the form of a single object collection. This command returns nothing if the scope is set to `top`.
- `current_instance`: Use this mode to reset the current instance. The scope of query is set to `top` and returns nothing.

**NOTE:** Setting the [current\\_design](#) or changing the [dq\\_design\\_view\\_type](#) preference variable resets the current instance.

## Arguments

The `current_instance` command has the following arguments:

### <pattern>

Use this argument to set the current instance that matches the provided pattern.

### <collection>

Use this argument to set the current instance with the hierarchical cell object contained in the argument collection.

## Examples

```
sg_shell> current_instance
current_instance: info: current instance is the top-level of
design 'SISO'.
sg_shell> current_instance sf_inst_1
{"SISO.sf_inst_1"}
sg_shell> current_instance .
```

```
 {"SISO.sf_inst_1"}
sg_shell> get_cells
 {"SISO.sf_inst_1.bbox_inst", "SISO.sf_inst_1.flop_inst_sf",
 "SISO.sf_inst_1.mux_inst_sf"}
sg_shell> current_instance mux_inst_sf
 {"SISO.sf_inst_1.mux_inst_sf"}
sg_shell> get_cells
 {"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst"}
sg_shell> current_instance
current_instance: info: current instance is the top-level of
design 'SISO'.
sg_shell> current_instance comb_cloud_inst_1
current_instance: error: cannot set current instance to leaf
cell 'SISO.comb_cloud_inst_1'
sg_shell> set cell [get_cells ic*]
 {"SISO.icgc_inst"}
sg_shell> current_instance $cell
 {"SISO.icgc_inst"}
sg_shell> current_instance .
 {"SISO.icgc_inst"}
sg_shell> get_cells
 {"SISO.icgc_inst.and_inst_icgc",
 "SISO.icgc_inst.latch_inst_icgc"}
```

## See Also

[current\\_design](#), [get\\_cells](#)



## current\_design

Used to select a current design for interactive constraint and design query commands

### Syntax

#### Usage 1

```
current_design <current_design_name>
```

#### Usage 2

```
current_design
```

To know more about defining scope of constraints during SpyGlass analysis in the Atrenta Console, refer to the *Defining a Scope for Constraints* section in the *Atrenta Console User Guide*.

### Scope

Project

### Return Value

**Usage 1:** Returns nothing

**Usage 2:** Returns design unit name

### Description

The *current\_design* command selects a design unit to be used for subsequent interactive constraint and design query commands. This command can be specified in the following two modes:

- **current\_design with specific design unit name:** Use this mode to set the *current\_design* command.
- **current\_design without any argument:** Use this mode to return currently selected design unit in the *current\_design* command.

**NOTE:** The *current\_design* command is set in *sg\_shell* only when it is specified directly in *sg\_shell* or when an ADC file is read through the *read\_file -type adc* command. The *current\_design SGDC* command, which is a part of the *SGDC* file, cannot set the *current\_design sg\_shell* command. The scope of the *current\_design SGDC* command is limited to the *SGDC* file in which it is specified.

## Arguments

The *current\_design* command has the following arguments:

### <current\_design\_name>

Use this argument to specify either a design unit name or a collection having a single design unit.

## Examples

```
# specify current_design for verilog design unit
sg_shell> current_design test
{"test"}
```

```
# specify current_design for vhdl design unit test_state.rtl
sg_shell> current_design test_state.rtl
{"test_state"}
```

```
# run without current_design specification
sg_shell> current_design
{"test_state"}
```

```
# error scenario when wrong current_design set
sg_shell> current_design test_stat.rtl
current_design: error: can't find design `test_stat.rtl'
sg_shell> get_master mod1_inst
{"mod1"}
```

```
sg_shell> current_design [get_master mod1_inst]
{"mod1"}
```

## See Also

[get\\_adc](#)

## get\_cells

**Creates a list of cells in the current design that match certain criteria**

### Syntax

```
get_cells
  [-filter expression]
  [-regexp | -exact]
  < patterns | -of_objects objects >
```

### Scope

Project

### Return Value

Returns an empty string or a collection of cells in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_cells* command creates a collection of cells in the current design or current instance that match certain criteria. This command returns a collection if any cell matches the *patterns* or *-of\_objects* specification and also passes the filtering criterion, if specified. If no object matches the criterion, an empty string is returned.

### Arguments

The *get\_cells* command has the following arguments:

#### <patterns>

Use this argument to match cell names against patterns. Patterns can include wildcard characters \* and ? or regular expressions, based on the *-regexp* argument. Patterns can also include a collection of cell types. The *patterns* and *-of\_objects* arguments are mutually exclusive.

**<-of\_objects objects>**

Use this argument to create a collection of cells connected to the specified objects. Objects that can be specified include pin names, collection of pins, net names, or a collection of nets.

**[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search the objects that contain the \* and ? wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can use only one at a time.

**[-filter expression]**

Use this argument to filter a collection with an expression. For any cell that matches *patterns* or *objects*, the expression is evaluated on the basis of the cell's attributes. If the expression evaluates to true, the cell is included in the result.

**Examples**

```
sg_shell> set_pref dq_design_view_type du
du
sg_shell> get_cells
{"inst_X", "inst_Y", "inst_Z"}
sg_shell> get_cells inst_*
{"inst_X", "inst_Y", "inst_Z"}
sg_shell> get_cells -regexp {inst.*}
{"inst_X", "inst_Y", "inst_Z"}
sg_shell> get_cells -of_objects "inst_X.A"
{"inst_X"}
sg_shell> get_cells -regexp -of_objects "inst_.*\.A"
{"inst_X", "inst_Y", "inst_Z"}
sg_shell> get_cells -regexp -of_objects [get_nets]
{"inst_X", "inst_Y", "inst_Z"}
sg_shell> get_cells -regexp -of_objects [get_pins inst_X.A]
{"inst_X"}
sg_shell> define_user_attribute -class du_cell -type int
```

```

intAttr
sg_shell> set_user_attribute [get_cells "inst_Y"] intAttr 10
set_user_attribute: info: setting attribute 'intAttr' on
object 'inst_Y' with value '10'
sg_shell> get_cells -filter {intAttr == 10}
{"inst_Y"}
sg_shell> get_cells -filter {is_sequential==true}
{"inst_Z"}
sg_shell> get_cells {{\cstate_r_reg[0] } {\cstate_r_reg[1] }
U6} -filter {is_blackbox==false}
{\cstate_r_reg[0] ", "\cstate_r_reg[1] ", "U6"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> get_cells {{top.u_ctrl.U_CT.\cstate_r_reg[0] }}
{"top.u_ctrl.U_CT.\cstate_r_reg[0] "}
sg_shell> foreach_in_collection i [get_cells -filter
{is_sequential == true}] {
puts "cell_name = [get_attribute $i full_name] (file:
[get_attribute $i file_name], line: [get_attribute $i
line_num])"
}
cell_name = {top.u_ctrl.U_CT.\cstate_r_reg[0] } (file:
netlist.v, line: 47)
cell_name = {top.u_ctrl.U_CT.\cstate_r_reg[1] } (file:
netlist.v, line: 48)
cell_name = top.u_ctrl.out_dvll_reg (file: netlist.v, line:
29)
cell_name = top.u_ctrl.out_dvld_reg (file: netlist.v, line:
31)
cell_name = top.u_ctrl.U_CT.U9 (file: netlist.v, line: 55)
sg_shell> set_pref dq_design_view_type hier_flat
hier_flat
sg_shell> current_instance .
current_instance: info: current instance is the top-level of
design 'SISO'.
sg_shell> get_cells
{"SISO.comb_cloud_inst_2", "SISO.comb_cloud_inst_1",
"SISO.icgc_inst", "SISO.sf_inst_2", "SISO.sf_inst_1"}

```

```
sg_shell> get_cells SISO.*.*
{"SISO.sf_inst_1.bbox_inst", "SISO.sf_inst_1.flop_inst_sf",
 "SISO.sf_inst_1.mux_inst_sf", "SISO.sf_inst_2.bbox_inst",
 "SISO.sf_inst_2.flop_inst_sf", "SISO.sf_inst_2.mux_inst_sf",
 "SISO.icgc_inst.and_inst_icgc",
 "SISO.icgc_inst.latch_inst_icgc"}
sg_shell> get_cells SISO.*.*.*
{"SISO.sf_inst_2.mux_inst_sf.leaf_mux_inst",
 "SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst"}
sg_shell> current_instance sf_inst_1
{"SISO.sf_inst_1"}
sg_shell> get_cells *
{"SISO.sf_inst_1.bbox_inst", "SISO.sf_inst_1.flop_inst_sf",
 "SISO.sf_inst_1.mux_inst_sf"}
sg_shell> get_cells *.*
{"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst"}
```

## See Also

[\*get\\_nets\*](#), [\*get\\_ports\*](#), [\*filter\\_collection\*](#)

## get\_nets

**Creates a list of nets in the current design that match certain criteria**

### Syntax

```
get_nets
  [-filter expression]
  [-regexp | -exact]
  < patterns | -of_objects objects>
  [ -boundary_type btype ]
```

### Scope

Project

### Return Value

Returns an empty string or a collection of nets in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_nets* command creates a collection of nets in the current design or current instance that match certain criteria. This command returns a collection if any net matches the *patterns* or *-of\_objects* specification and also passes the filtering criterion, if specified. If no object matches the criterion, an empty string is returned.

### Arguments

The *get\_nets* command has the following arguments:

#### <patterns>

Use this argument to match net names against patterns. Patterns can include the wildcard characters \* and ? or regular expressions, based on the *-regexp* argument. Patterns can also include a collection of net type. The *patterns* and *-of\_objects* arguments are mutually exclusive.

**<-of\_objects objects>**

Use this argument to create a collection of nets connected to the specified objects. Objects that can be specified include pin names, collection of pins, port names, or collection of ports.

**[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search objects that contain the \* and ? wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can use only one at a time.

**[-filter expression]**

Use this argument to filter a collection with an expression. For any net that matches *patterns* or *objects*, the expression is evaluated on the basis of net's attributes. If the expression evaluates to `true`, the net is included in the result.

**[-boundary\_type btype]**

Use this argument to select the hierarchy of the connected net corresponding to hierarchical pins. This option requires the *-of\_objects* option. The allowed values are `lower`, `upper`, and `both`, which means the net inside the hierarchical block, the net outside the hierarchical block, or both nets, respectively. The option has no meaning for non-hierarchical pins. The default value is `upper`.

## Examples

```
sg_shell> set_pref dq_design_view_type du
du
sg_shell> get_nets
{"N1", "N2", "N3", "N4"}
sg_shell> get_nets *
{"N1", "N2", "N3", "N4"}
sg_shell> get_nets -regexp {N[0-2].*}
{"N1", "N2"}
sg_shell> get_nets -of_objects [get_ports]
```



```

{"N1", "N2", "N3", "N4"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> get_nets -of_objects [get_pins -of_objects
{{top.u_ctrl.U_CT.\cstate_r_reg[0] }}]
{"top.u_ctrl.U_CT.\cstate_r[0] ", "top.u_ctrl.U_CT.n9",
"top.u_ctrl.U_CT.N3", "top.clk_1"}
sg_shell> foreach_in_collection i [get_nets -of_objects
[get_pins -of_objects {{top.u_ctrl.U_CT.\cstate_r_reg[0]
}}]] {
puts "net_name = [get_attribute $i full_name] (file:
[get_attribute $i file_name], line: [get_attribute $i
line_num])"
}
net_name = {top.u_ctrl.U_CT.\cstate_r[0] } (file: netlist.v,
line: 47)
net_name = top.u_ctrl.U_CT.n9 (file: netlist.v, line: 47)
net_name = top.u_ctrl.U_CT.N3 (file: netlist.v, line: 52)
net_name = top.clk_1 (file: netlist.v, line: 1)
sg_shell> set_pref dq_design_view_type hier_flat
hier_flat
sg_shell> set n [get_nets -filter {direction == output}]
{"SISO.DATA_OUT"}
sg_shell> set p [get_pins -of_objects $n]
{"SISO.sf_inst_2.Q"}
sg_shell> get_nets -of_objects $p
{"SISO.DATA_OUT"}
sg_shell> get_nets -of_objects $p -boundary_type upper
{"SISO.DATA_OUT"}
sg_shell> get_nets -of_objects $p -boundary_type lower
{"SISO.sf_inst_2.Q"}
sg_shell> get_nets -of_objects $p -boundary_type both
{"SISO.DATA_OUT", "SISO.sf_inst_2.Q"}
sg_shell> current_instance sf_inst_1
{"SISO.sf_inst_1"}
sg_shell> get_nets
{"SISO.sf_inst_1.D", "SISO.sf_inst_1.SI",
"SISO.sf_inst_1.SE", "SISO.sf_inst_1.CLK",

```

```
"SISO.sf_inst_1.Q", "SISO.sf_inst_1.mux_out"}  
sg_shell> current_instance mux_inst_sf  
{ "SISO.sf_inst_1.mux_inst_sf" }  
sg_shell> get_nets  
{ "SISO.sf_inst_1.mux_inst_sf.A",  
  "SISO.sf_inst_1.mux_inst_sf.B",  
  "SISO.sf_inst_1.mux_inst_sf.S",  
  "SISO.sf_inst_1.mux_inst_sf.Z" }
```

## See Also

[get\\_pins](#), [get\\_ports](#), [current\\_instance](#), [filter\\_collection](#)

## get\_pins

**Creates a list of pins in the current design that match certain criteria**

### Syntax

```
get_pins
  [-filter expression]
  [-regexp | -exact]
  < patterns | -of_objects objects >
```

### Scope

Project

### Return Value

Returns an empty string or a collection of pins in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_pins* command creates a collection of pins in the current design or current instance that match certain criteria. This command returns a collection if any pin matches the *patterns* or *-of\_objects* specification and also passes the filtering criterion, if specified. If no object matches the criterion, an empty string is returned.

### Arguments

The *get\_pins* command has the following arguments:

#### <patterns>

Use this argument to match pin names against patterns. Patterns can include the wildcard characters \* and ? or regular expressions, based on the *-regexp* argument. Patterns can also include a collection of pin type. The *patterns* and *-of\_objects* arguments are mutually exclusive.

**<-of\_objects objects>**

Use this argument to create a collection of pins connected to the specified objects. Objects that can be specified include net names, collection of nets, cell names, or collection of cells.

**[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search objects that contain the \* and ? wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can use only one at a time.

**[-filter expression]**

Use this argument to filter collection with an expression. For any pin that matches *patterns* or *objects*, the expression is evaluated on the basis of the pin's attributes. If the expression evaluates to true, the pin is included in the result.

**Examples**

```
sg_shell> set_pref dq_design_view_type du
du
sg_shell> get_pins
{"inst_A.A", "inst_A.B", "inst_A.Z", "inst_B.A", "inst_B.B",
"inst_B.Z"}
sg_shell> get_pins *
{"inst_A.A", "inst_A.B", "inst_A.Z", "inst_B.A", "inst_B.B",
"inst_B.Z"}
sg_shell> get_pins -regexp "inst_A.*\.A.*"
{"inst_A.A"}
sg_shell> get_pins -of_objects [get_cells inst_A]
{"inst_A.A", "inst_A.B", "inst_A.Z"}
sg_shell> define_user_attribute -class du_pin -type int
intAttr
sg_shell> set_user_attribute [get_pin "inst_A.B inst_B.A"]
intAttr 10
set_user_attribute: info: setting attribute 'intAttr' on
```

```

object 'inst_A.B' with value '10'
set_user_attribute: info: setting attribute 'intAttr' on
object 'inst_B.A' with value '10'
sg_shell> get_pins * -filter "intAttr == 10"
{"inst_A.B", "inst_B.A"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> get_pins -of_objects
{{top.u_ctrl.U_CT.\cstate_r_reg[0] }}
{"top.u_ctrl.U_CT.\cstate_r_reg[0] .Q",
"top.u_ctrl.U_CT.\cstate_r_reg[0] .QN",
"top.u_ctrl.U_CT.\cstate_r_reg[0] .D",
"top.u_ctrl.U_CT.\cstate_r_reg[0] .CP"}
sg_shell> foreach_in_collection i [get_pins -of_objects
{{top.u_ctrl.U_CT.\cstate_r_reg[0] }} -filter
{direction==input}] {
puts "pin_name = [get_attribute $i full_name] (file:
[get_attribute $i file_name], line: [get_attribute $i
line_num])"
}
pin_name = {top.u_ctrl.U_CT.\cstate_r_reg[0] .D} (file:
netlist.v, line: 47)
pin_name = {top.u_ctrl.U_CT.\cstate_r_reg[0] .CP} (file:
netlist.v, line: 47)
sg_shell> set_pref dq_design_view_type hier_flat
hier_flat
sg_shell> current_instance icgc_inst
{"SISO.icgc_inst"}
sg_shell> get_pins *.Z
{"SISO.icgc_inst.and_inst_icgc.Z"}
sg_shell> get_pins *.Q*
{"SISO.icgc_inst.latch_inst_icgc.Q",
"SISO.icgc_inst.latch_inst_icgc.QN"}
sg_shell> get_pins -filter {is_enable_pin == true}
{"SISO.icgc_inst.latch_inst_icgc.G"}
sg_shell> current_instance
current_instance: info: current instance is the top-level of
design 'SISO'.

```

```
sg_shell> current_instance sf_inst_1.mux_inst_sf  
{"SISO.sf_inst_1.mux_inst_sf"}  
sg_shell> get_pins -filter {is_mux_select_pin == true}  
{"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst.S"}
```

## See Also

[get\\_nets](#), [get\\_cells](#), [current\\_instance](#), [filter\\_collection](#)

## get\_ports

**Creates a list of ports in the current design that match certain criteria**

### Syntax

```
get_ports
  [-filter expression]
  [-regexp | -exact]
  < patterns | -of_objects objects>
```

### Scope

Project

### Return Value

Returns an empty string or a collection of ports in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_ports* command creates a collection of ports in the current design that match certain criteria. This command returns a collection if any port matches the *patterns* or *-of\_objects* specification and also passes the filtering criterion, if specified. If no object matches the criterion, an empty string is returned.

### Arguments

The *get\_ports* command has the following arguments:

#### <patterns>

Use this argument to match port names against patterns. Patterns can include the wildcard characters \* and ? or regular expressions, based on the *-regexp* argument. Patterns can also include a collection of port type. The *patterns* and *-of\_objects* arguments are mutually exclusive.

**<-of\_objects objects>**

Use this argument to create a collection of ports connected to the specified objects. Objects that can be specified include net names or a collection of nets.

**[-regexp | -exact]**

If the *-regexp* switch is specified, patterns are seen as real regular expressions rather than simple wildcard patterns. If the *-exact* switch is specified, simple pattern matching is disabled. This is used to search objects that contain the \* and ? wildcard characters. The *-exact* and *-regexp* arguments are mutually exclusive. You can use only one at a time.

**[-filter expression]**

Use this argument to filter collection with an expression. For any port that matches *patterns* or *objects*, the expression is evaluated on the basis of port's attributes. If the expression evaluates to true, the port is included in the result.

**Examples**

```
sg_shell> set_pref dq_design_view_type du
du

sg_shell> get_ports
{"P1", "P2", "P3", "P4"}

sg_shell> get_ports P*
{"P1", "P2", "P3", "P4"}

sg_shell> get_ports -regexp {P[0-1].*}
{"P1"}

sg_shell> get_ports -of_objects [get_nets]
{"P1", "P2", "P3", "P4"}

sg_shell> define_user_attribute -class du_port -type int
intAttr

sg_shell> set_user_attribute [get_ports "P2 P4"] intAttr 10
set_user_attribute: info: setting attribute 'intAttr' on
object 'B' with value '10'
```



```
set_user_attribute: info: setting attribute 'intAttr' on
object 'C' with value '10'
```

```
sg_shell> get_ports * -filter "intAttr == 10"
{"P2", "P4"}
```

```
sg_shell> foreach_in_collection i [get_ports -filter
{direction==output}] {
    puts "port_name = [get_attribute $i full_name]
        (object_class = [get_attribute $i object_class])"
}
```

```
port_name = dload (object_class = du_port)
port_name = dclear (object_class = du_port)
port_name = cload (object_class = du_port)
port_name = cclear (object_class = du_port)
port_name = iload (object_class = du_port)
port_name = iclear (object_class = du_port)
port_name = readf (object_class = du_port)
port_name = chk_crc (object_class = du_port)
```

```
sg_shell> set_pref dq_design_view_type flat
flat
```

```
sg_shell> get_ports clk*
{"top.clk_1", "top.clk_2"}
```

```
sg_shell> get_ports -regexp {{top\..*data}}
{"top.data_in[0]", "top.data_in[1]", "top.data_in[2]",
"top.data_in[3]", "top.data_in[4]", "top.data_in[5]",
"top.data_in[6]", "top.data_in[7]", "top.sync_data[0]",
"top.sync_data[1]", "top.sync_data[2]", "top.sync_data[3]",
"top.sync_data[4]", "top.sync_data[5]", "top.sync_data[6]",
"top.sync_data[7]", "top.data_out[0]", "top.data_out[1]",
"top.data_out[2]", "top.data_out[3]", "top.data_out[4]",
"top.data_out[5]", "top.data_out[6]", "top.data_out[7]"}
```

```
sg_shell> get_ports -regexp {{top\.data_in\[[0-5]\]}}
{"top.data_in[0]", "top.data_in[1]", "top.data_in[2]",
"top.data_in[3]", "top.data_in[4]", "top.data_in[5]"}
```

```
sg_shell> get_ports -regexp -of_objects {{top\.data_in\[[0-5]\]}}
```

```
{ "top.data_in[0]", "top.data_in[1]", "top.data_in[2]",  
  "top.data_in[3]", "top.data_in[4]", "top.data_in[5]" }
```

## See Also

[get\\_nets](#), [get\\_ports](#), [filter\\_collection](#)

## report\_cell

Used to display information and statistics about cells in the current instance or current design

### Syntax

```
report_cell  
[-connections [-verbose]]  
[cell_names]
```

### Scope

Project

### Return Value

Returns an empty string if the command is successfully executed and nothing in case of any error

### Description

The *report\_cell* command displays information and statistics about cells in the current instance or current design. If the current instance is set, the report is generated for the design of that instance. Otherwise, the report is generated for the current design.

### Arguments

The *report\_cell* command has the following arguments:

#### [cell\_names]

Use this argument to specify the collection or names of cells to be reported. If this argument is not specified, all the cells in the current instance are reported.

#### [-connections]

Use this argument to display the pins and the nets to which connections are made.

#### [-verbose]

Use this argument to display verbose connection information. For each

input pin, it displays the net and the driver pins on that net. For each output pin, it displays the net and the load pins on that net. This argument is used in conjunction with the *-connections* option.

## Examples

```
sg_shell> report_cell
*****
Report : cell
Design : design1
*****

          h - hierarchical
          l - leaf level instance
          b - blackbox

Cell      Master      Library      Attributes
-----
U0        AN2         mylib_20c   1
U1        OR2         mylib_20c   1
U2        AN2         mylib_20c   1
U3        OR3         mylib_20c   1
U4        AN3         mylib_20c   1
A1        AOI
AU0       AN2         mylib_20c   1
OU        AN2P        mylib_20c   1
```

## See Also

[get\\_cells](#), [get\\_lib\\_cells](#)

## get\_fanin\_pins

Creates a list of fan-in pins in the design that match certain criteria

### Syntax

```
get_fanin_pins
  [-timing_type <timing_type>]
  [-boundary_type btype]
  < objects >
```

### Scope

Project

### Return Value

Returns a collection of fan-in pins of the specified net, pin, or port object. If nothing matches the specified object, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_fanin\_pins* command creates a collection of pins in the flattened design that match certain criteria. This command returns a collection of pins found in the fan-in cone of the specified net, pin, or port. If no object matches the criterion, an empty collection is returned.

### Arguments

The *get\_fanin\_pins* command has the following arguments:

#### **[-timing\_type]**

Pins in the fan-in cone of a flat pin, port, or net, which have *timing\_type* as the timing arc type, are included in the final result. Timing type can be combinational, combinational\_rise, combinational\_fall, three\_state\_disable, three\_state\_disable\_rise, three\_state\_disable\_fall, three\_state\_enable, three\_state\_enable\_rise, three\_state\_enable\_fall, rising\_edge, falling\_edge, preset, clear, hold\_rising, hold\_falling, setup\_rising, setup\_falling, recovery\_rising, recovery\_falling, skew\_rising, skew\_falling, removal\_rising, removal\_falling, min\_pulse\_width, minimum\_period, max\_clock\_tree\_path,

min\_clock\_tree\_path, non\_seq\_setup\_rising, non\_seq\_setup\_falling, non\_seq\_hold\_rising, non\_seq\_hold\_falling, nochange\_high\_high, nochange\_high\_low, nochange\_low\_high, or nochange\_low\_low.

### **[-boundary\_type btype]**

When *<objects>* is an inout hierarchical pin, it is mandatory to use this argument. Depending on its value (upper or lower), the command returns fan-in inside or outside the hierarchy of *<objects>*.

### **<object>**

Use this argument to specify an object, which can be a flattened pin, port, or net.

## **Examples**

```
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> get_fanin_pins top.inst_mid.AN2.Z
{"top.inst_mid.AN2.A", "top.inst_mid.AN2.B"}
sg_shell> foreach_in_collection i [get_pins -filter
{direction==output}] {
if [catch {set var [get_fanin_pins $i]} msg] {
puts "pin_name = [get_attribute $i full_name] fanin_pins =
(no fanin pin)"
} else {
puts "pin_name = [get_attribute $i full_name] fanin_pins =
([get_attribute $var full_name])" }
}
pin_name = {top.u_ctrl.U_CT.\cstate_r_reg[0] .Q} fanin_pins
= ({top.u_ctrl.U_CT.\cstate_r_reg[0] .CP})
pin_name = {top.u_ctrl.U_CT.\cstate_r_reg[0] .QN}
fanin_pins = ({top.u_ctrl.U_CT.\cstate_r_reg[0] .CP})
pin_name = {top.u_ctrl.U_CT.\cstate_r_reg[1] .Q} fanin_pins
= ({top.u_ctrl.U_CT.\cstate_r_reg[1] .CP})
pin_name = {top.u_ctrl.U_CT.\cstate_r_reg[1] .QN}
fanin_pins = ({top.u_ctrl.U_CT.\cstate_r_reg[1] .CP})
pin_name = top.u_ctrl.out_dvll_reg.Q fanin_pins =
(top.u_ctrl.out_dvll_reg.CP)
```

```

pin_name = top.u_ctrl.out_dvll_reg.QN fanin_pins =
(top.u_ctrl.out_dvll_reg.CP)
pin_name = top.u_ctrl.out_dvld_reg.Q fanin_pins =
(top.u_ctrl.out_dvld_reg.CP)
pin_name = top.u_ctrl.out_dvld_reg.QN fanin_pins =
(top.u_ctrl.out_dvld_reg.CP)
pin_name = top.u_ctrl.U_CT.U9.Q fanin_pins =
(top.u_ctrl.U_CT.U9.C2 top.u_ctrl.U_CT.U9.C1)
pin_name = top.u_ctrl.U_CT.U9.QN fanin_pins =
(top.u_ctrl.U_CT.U9.C2 top.u_ctrl.U_CT.U9.C1)
sg_shell> set_pref dq_design_view_type hier_flat
hier_flat
sg_shell> set point [get_ports -filter {direction ==
output}]
{"SISO.DATA_OUT"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.sf_inst_2.Q"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.sf_inst_2.flop_inst_sf.Q"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.sf_inst_2.flop_inst_sf.CP"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.sf_inst_2.CLK"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.icgc_inst.CLKOUT"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.icgc_inst.and_inst_icgc.Z"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.icgc_inst.and_inst_icgc.B",
"SISO.icgc_inst.and_inst_icgc.A"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> set point [get_ports -filter {direction ==
output}]
{"SISO.DATA_OUT"}
sg_shell> set point [get_fanin_pins $point]
{"SISO.sf_inst_2.flop_inst_sf.Q"}
sg_shell> set point [get_fanin_pins $point]

```

```
 {"SISO.sf_inst_2.flop_inst_sf.CP"}  
sg_shell> set point [get_fanin_pins $point]  
 {"SISO.icgc_inst.and_inst_icgc.Z"}  
sg_shell> set point [get_fanin_pins $point]  
 {"SISO.icgc_inst.and_inst_icgc.B",  
 "SISO.icgc_inst.and_inst_icgc.A"}
```

## See Also

[get\\_fanin\\_ports](#), [get\\_fanout\\_pins](#), [get\\_fanout\\_ports](#), [get\\_cells](#), [get\\_nets](#),  
[get\\_pins](#), [get\\_ports](#)



## get\_fanin\_ports

Creates a list of fan-in ports in the design that match certain criteria

### Syntax

```
get_fanin_ports < objects >
```

### Scope

Project

### Return Value

Returns a collection of fan-in ports of specified net, pin, or port object. In case nothing matches the specified object, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_fanin\_ports* command creates a collection of ports found in fan-in cone of the net, pin, or port specified. If no object matches the criterion, an empty collection is returned.

### Arguments

The *get\_fanin\_ports* command has the following argument:

#### <objects>

Use this argument to specify an object, which can be a flattened pin, port, or net.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
flat
```

```
sg_shell> get_nets
{"top.A", "top.B", "top.C", "top.Z"}
```

```
sg_shell> get_fanin_ports top.C
{"top.C"}
```

```
sg_shell> foreach_in_collection i [get_nets -filter
{direction==input}] {
    if [catch {set var [get_fanin_ports $i]} msg] {
        puts "net_name = [get_attribute $i full_name]
fanin_ports = (no fanin port)"
    } else {
        puts "net_name = [get_attribute $i full_name]
fanin_ports = ([get_attribute $var full_name])" }
    }
net_name = top.clk_1 fanin_ports = (top.clk_1)
net_name = top.clk_2 fanin_ports = (top.clk_2)
net_name = top.rst_n fanin_ports = (top.rst_n)
net_name = {top.data_in[0]} fanin_ports = ({top.data_in[0]})
net_name = {top.data_in[1]} fanin_ports = ({top.data_in[1]})
net_name = {top.data_in[2]} fanin_ports = ({top.data_in[2]})
net_name = {top.data_in[3]} fanin_ports = ({top.data_in[3]})
net_name = {top.data_in[4]} fanin_ports = ({top.data_in[4]})
net_name = {top.data_in[5]} fanin_ports = ({top.data_in[5]})
net_name = {top.data_in[6]} fanin_ports = ({top.data_in[6]})
net_name = {top.data_in[7]} fanin_ports = ({top.data_in[7]})
net_name = top.write_n fanin_ports = (top.write_n)
```

## See Also

[get\\_fanin\\_pins](#), [get\\_fanout\\_pins](#), [get\\_fanout\\_ports](#), [get\\_cells](#), [get\\_nets](#), [get\\_pins](#),  
[get\\_ports](#)

## get\_fanout\_pins

**Creates a list of fan-out pins in the design that match certain criteria**

### Syntax

```
get_fanout_pins
  [-timing_type <timing_type>]
  [-boundary_type btype]
  < objects >
```

### Scope

Project

### Return Value

Returns a collection of fan-out pins of the specified net, pin, or port object. If nothing matches the specified object, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_fanout\_pins* command creates a collection of pins in the flattened design that match certain criteria. This command returns a collection of pins found in the fan-out cone of the specified net, pin, or port. If no object matches the criterion, an empty collection is returned.

### Arguments

The *get\_fanout\_pins* command has the following arguments:

#### **[-timing\_type]**

Pins in the fan-out cone of a flat pin, port, or net that have *timing\_type* as the timing arc type are included in the final result. Timing type can be combinational, combinational\_rise, combinational\_fall, three\_state\_disable, three\_state\_disable\_rise, three\_state\_disable\_fall, three\_state\_enable, three\_state\_enable\_rise, three\_state\_enable\_fall, rising\_edge, falling\_edge, preset, clear, hold\_rising, hold\_falling, setup\_rising, setup\_falling, recovery\_rising, recovery\_falling, skew\_rising, skew\_falling, removal\_rising, removal\_falling,

min\_pulse\_width, minimum\_period, max\_clock\_tree\_path,  
min\_clock\_tree\_path, non\_seq\_setup\_rising, non\_seq\_setup\_falling,  
non\_seq\_hold\_rising, non\_seq\_hold\_falling, nochange\_high\_high,  
nochange\_high\_low, nochange\_low\_high, or nochange\_low\_low.

### **[-boundary\_type btype]**

When *<objects>* is an inout hierarchical pin, it is mandatory to use this argument. Depending on its value (upper or lower), the command returns fan-out inside or outside the hierarchy of *<objects>*.

### **<object>**

Use this argument to specify an object, which can be a flattened pin, port, or net.

## **Examples**

```
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> get_fanout_pins top.inst_mid.AN2.A
{"top.inst_mid.AN2.Z"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> foreach_in_collection i [get_pins -of_objects
[get_cells top.u_ctrl.U_CT.U*] -filter {direction==output}]
{
if [catch {set var [get_fanout_pins $i]} msg] {
puts "pin_name = [get_attribute $i full_name] fanout_pins =
(no fanout pin)"
} else {
puts "pin_name = [get_attribute $i full_name] fanout_pins =
([get_attribute $var full_name])" }
}
pin_name = top.u_ctrl.U_CT.U12.Z fanout_pins =
(top.u_ctrl.out_dvll_reg.D)
pin_name = top.u_ctrl.U_CT.U11.Z fanout_pins =
(top.u_ctrl.U_CT.U6.C)
pin_name = top.u_ctrl.U_CT.U10.Z fanout_pins =
(top.u_ctrl.U_CT.U4.B top.u_ctrl.U_CT.U12.A)
```

## Debug Commands

```
pin_name = top.u_ctrl.U_CT.U9.Q fanout_pins =
(top.u_ctrl.U_CT.U5.B)
pin_name = top.u_ctrl.U_CT.U9.QN fanout_pins = (no fanout
pin)
pin_name = top.u_ctrl.U_CT.U8.Z fanout_pins =
(top.u_ctrl.U_CT.U5.A top.u_ctrl.U_CT.U6.B)
pin_name = top.u_ctrl.U_CT.U7.Z fanout_pins =
(top.u_ctrl.U_CT.U8.C)
pin_name = top.u_ctrl.U_CT.U6.Z fanout_pins =
({top.u_ctrl.U_CT.\cstate_r_reg[0] .D})
pin_name = top.u_ctrl.U_CT.U5.Z fanout_pins =
({top.u_ctrl.U_CT.\cstate_r_reg[1] .D})
sg_shell> set_pref dq_design_view_type hier_flat
hier_flat
sg_shell> set point [get_ports SISO.DATA_IN]
{"SISO.DATA_IN"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.comb_cloud_inst_1.A"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.comb_cloud_inst_1.Z"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.D"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.mux_inst_sf.A"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst.A"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst.Z"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.mux_inst_sf.Z"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.flop_inst_sf.D"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> set point [get_ports SISO.DATA_IN]
{"SISO.DATA_IN"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.comb_cloud_inst_1.A"}
```

```
sg_shell> set point [get_fanout_pins $point]
{"SISO.comb_cloud_inst_1.Z"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst.A"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst.Z"}
sg_shell> set point [get_fanout_pins $point]
{"SISO.sf_inst_1.flop_inst_sf.D"}
```

## See Also

[get\\_fanin\\_pins](#), [get\\_fanin\\_ports](#), [get\\_fanout\\_ports](#), [get\\_cells](#), [get\\_nets](#), [get\\_pins](#), [get\\_ports](#)

## get\_fanout\_ports

Creates a list of fan-out ports in the design that match certain criteria

### Syntax

```
get_fanout_ports  
  <objects>
```

### Scope

Project

### Return Value

Returns a collection of fan-out ports of a specified net, pin, or port object. In case nothing matches the specified object, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_fanout\_ports* command creates a collection of ports found in fan-out cone of the net, pin, or port specified. If no object matches the criterion, an empty collection is returned.

### Arguments

The *get\_fanout\_ports* command has the following argument:

**<objects>**

Use this argument to specify an object, which can be a flattened pin, port, or net.

### Examples

```
sg_shell> set_pref dq_design_view_type flat  
flat  
  
sg_shell> get_nets  
{ "top.A", "top.B", "top.C", "top.Z" }  
  
sg_shell> get_fanout_ports top.Z  
{ "top.Z" }
```

```
sg_shell> foreach_in_collection i [get_nets {top.crc*
top.u_ctrl.U_CT.*}] {
    if [catch {set var [get_fanout_ports $i]} msg] {
        puts "net_name = [get_attribute $i full_name]
fanout_ports = (no fanout port)"
    } else {
        puts "net_name = [get_attribute $i full_name]
fanout_ports = ([get_attribute $var full_name])"
    }
}
net_name = {top.crc_out[7]}
fanout_ports = ({top.crc_out[7]})
net_name = {top.crc_out[6]}
fanout_ports = ({top.crc_out[6]})
net_name = {top.crc_out[5]}
fanout_ports = ({top.crc_out[5]})
net_name = {top.crc_out[4]}
fanout_ports = ({top.crc_out[4]})
net_name = {top.crc_out[3]}
fanout_ports = ({top.crc_out[3]})
net_name = {top.crc_out[2]}
fanout_ports = ({top.crc_out[2]})
net_name = {top.crc_out[1]}
fanout_ports = ({top.crc_out[1]})
net_name = {top.crc_out[0]}
fanout_ports = ({top.crc_out[0]})
net_name = top.clk_1 fanout_ports = (no fanout port)
net_name = top.rst_n fanout_ports = (no fanout port)
net_name = top.pkt_avl fanout_ports = (no fanout port)
net_name = top.loop_cmp fanout_ports = (no fanout port)
net_name = top.empty fanout_ports = (no fanout port)
net_name = top.dload fanout_ports = (no fanout port)
net_name = top.u_ctrl.U_CT.dclear fanout_ports = (no fanout
port)
net_name = top.cload fanout_ports = (no fanout port)
net_name = top.cclear fanout_ports = (no fanout port)
net_name = top.iload fanout_ports = (no fanout port)
net_name = top.iclear fanout_ports = (no fanout port)
```



## Debug Commands

```
net_name = top.read_n fanout_ports = (no fanout port)
net_name = top.u_ctrl.out_dvl2 fanout_ports = (no fanout
port)
net_name = {top.u_ctrl.U_CT.\cstate_r[1] } fanout_ports = (no
fanout port)
net_name = {top.u_ctrl.U_CT.\cstate_r[0] } fanout_ports = (no
fanout port)
net_name = top.u_ctrl.U_CT.N3 fanout_ports = (no fanout port)
net_name = top.u_ctrl.U_CT.N4 fanout_ports = (no fanout port)
net_name = top.u_ctrl.U_CT.n9 fanout_ports = (no fanout port)
net_name = top.u_ctrl.U_CT.n10 fanout_ports = (no fanout
port)
net_name = top.u_ctrl.U_CT.n11 fanout_ports = (no fanout
port)
net_name = top.u_ctrl.U_CT.n12 fanout_ports = (no fanout
port)
net_name = top.u_ctrl.U_CT.n13 fanout_ports = (no fanout
port)
net_name = top.u_ctrl.U_CT.n14 fanout_ports = (no fanout
port)
net_name = top.u_ctrl.U_CT.n15 fanout_ports = (no fanout
port)
net_name = top.u_ctrl.U_CT.VCC fanout_ports = (no fanout
port)
```

**See Also**

[get\\_fanin\\_pins](#), [get\\_fanin\\_ports](#), [get\\_fanout\\_pins](#), [get\\_cells](#), [get\\_nets](#), [get\\_pins](#),  
[get\\_ports](#)

## get\_master

Returns the master module of the specified instance

### Syntax

```
get_master <objects>
```

### Scope

Project

### Return Value

Returns a collection of master nodes of the specified object. In case nothing matches the specified object, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_master* command creates a collection of master modules of the specified instance object. Objects can be a named instance or an instance collection.

### Arguments

The *get\_master* command has the following arguments:

#### <objects>

Use this argument to create a collection of master modules of the specified instance objects.

### Examples

```
sg_shell> set_pref dq_design_view_type du
du
```

```
sg_shell> get_master inst_X
{"master_inst_X"}
```

```
sg_shell> get_cells
{"inst_X", "inst_Y", "inst_Z", "inst_W"}
```

```
sg_shell> get_master [get_cells]
```

```
{ "master_inst_X", "master_inst_Y", "master_inst_Z",
  "master_inst_W" }

sg_shell> get_master inst_*
{"master_inst_X", "master_inst_Y", "master_inst_Z",
 "master_inst_W"}

sg_shell> current_design [get_master inst_X]
{"master_inst_X"}

sg_shell> set_pref dq_design_view_type flat
flat

sg_shell> foreach_in_collection i [get_cells
top.u_ctrl.U_CT.U*] {
  puts "cell_name = [get_attribute $i base_name]
(master_name = [get_attribute [get_master $i] base_name])"
}
cell_name = U001 (master_name = TC4GTDSDLANDYECLXP)
cell_name = U14 (master_name = PCI33DGZ)
cell_name = U13 (master_name = BBOX)
cell_name = U7 (master_name = IV)
cell_name = U6 (master_name = BTS4P)
cell_name = U4 (master_name = ISOAND)
```

## See Also

[\*get\\_parent\*](#)

## get\_parent

Returns the parent node of the specified object

### Syntax

```
get_parent <objects>
```

### Scope

Project

### Return Value

Returns a collection of parent nodes of the specified object. In case nothing matches the specified object, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_parent* command creates a collection of parent nodes of the specified object. Objects can be an instance, pin, port, net names, or collection comprising either of these.

### Arguments

The *get\_parent* command has the following arguments:

#### <objects>

Use this argument to create a collection of parent modules of the specified instance, port, or net.

### Examples

```
sg_shell> current_design
{"top"}
sg_shell> get_cells
{"inst_X", "inst_Y", "inst_Z", "inst_W"}
sg_shell> get_parent inst_X
{"top"}
sg_shell> get_pins inst_X
{"inst_X.A", "inst_X.B", "inst_X.Z"}
```

```
sg_shell> get_parent inst_X.B
{"inst_X"}
sg_shell> current_design [get_master inst_X]
{"master_inst_X"}
sg_shell> foreach_in_collection i [get_pins -of_objects
[get_cells U*]] {
puts "pin_name = [get_attribute $i base_name]
(cell_name = [get_attribute [get_parent $i] full_name],
direction = [get_attribute $i direction])"
}

pin_name = A (cell_name = U0, direction = input)
pin_name = B (cell_name = U0, direction = input)
pin_name = Z (cell_name = U0, direction = output)
pin_name = A (cell_name = U1, direction = input)
pin_name = B (cell_name = U1, direction = input)
pin_name = Z (cell_name = U1, direction = output)
pin_name = A (cell_name = U2, direction = input)
pin_name = B (cell_name = U2, direction = input)
pin_name = Z (cell_name = U2, direction = output)
```

## See Also

[get\\_master](#), [get\\_cells](#), [get\\_nets](#), [get\\_pins](#), [get\\_ports](#)

## get\_clocks

**Creates a list of user-defined clocks in the current design**

### Syntax

```
get_clocks
  [<clkName>]
  [-filter <filter_expression>]
  [-of_objects <obj>]
```

### Scope

Project

### Return Value

Returns a collection of user-defined clocks in the current design. This collection can be used as an input to other commands, such as [report\\_clocks](#), [get\\_registers](#), and [get\\_domains](#). If no user-defined clocks are found, an empty collection is returned.

### Description

The *get\_clocks* command creates a collection of user-defined clocks in the current design matching specified criteria.

#### Limitations

The *get\_clocks* command fails in the following cases:

- Selected design query view type is not flat.
- Flattened design view is not present.

### Arguments

The *get\_clocks* command has the following arguments:

[ <clkName> ]

Use this argument to specify the tag names of clocks in the design. Collection of all clocks having tag name as specified along with this option are returned.

**[-of\_objects <obj>]**

Use this argument to specify a collection of flat cells, flat nets, flat pins, or domains. Collection of clocks driving specified flat objects are returned.

**[-filter <filter\_expression>]**

Use this argument to specify a filter expression over the named attributes of clocks, such as `clk_name` (tag name of a clock), `file_name` (source file where a clock is defined), `period` (period of clocks), `edgelist` (edgelist of a clock), and `clock_type` (type of the clock).

**Examples**

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> get_clocks
# returns a collection of all user-defined clocks in the
current design
sg_shell> get_clocks tag1
# returns a collection of user-defined clocks that have the
tag name as 'tag1'
sg_shell> get_clocks -of_objects [get_pins expr]
# returns a collection of user-defined clocks that are
driving flat pins matching 'expr'
sg_shell> get_clocks -of_objects [get_cells expr]
# returns a collection of user-defined clocks that are
driving flat cells matching 'expr'
sg_shell> get_clocks -of_objects [get_domains expr]
# returns a collection of user-defined clocks of domain
returned by the get_domains command
```

**See Also**

[report\\_clocks](#), [get\\_registers](#), [propagate\\_clocks](#), [get\\_domains](#), [report\\_domains](#)

## get\_clock\_relation

Returns a collection of clock objects for given names

### Syntax

```
get_clock_relation [<clock-names>]
```

### Scope

Project

### Return Value

Returns a collection of clock objects.

### Description

The *get\_clock\_relation* command returns a collection of clock objects for the given list of clock names. For virtual clocks, specify tag names instead of clock names. If clock name or tag name is not provided, the command returns all clock objects in the design.

### Arguments

The *get\_clock\_relation* command has the following arguments:

#### [<clock-names>]

Use this argument to specify a list of clock names (tag names for virtual clocks) defined in the sgdc file.

### Examples

```
sg_shell> get_clock_relation
//returns all objects corresponding to clocks in design

sg_shell> get_clock_relation clk1 clk2
//returns object correspond to clk1 and clk2
```

### See Also

[report\\_clocks](#), [get\\_registers](#), [propagate\\_clocks](#), [get\\_domains](#), [report\\_domains](#), [report\\_clock\\_relation](#)



## report\_clock\_relation

Returns synchronous and asynchronous relationship between clocks of the design in matrix format

### Syntax

```
report_clock_relation [<clock-objects>]
```

### Scope

Project

### Return Value

None

### Description

The *report\_clock\_relation* command reports synchronous and asynchronous relationship between clocks in a matrix. In the matrix, "S" indicates synchronous clocks and "A" indicates asynchronous clocks.

### Arguments

The *report\_clock\_relation* command has the following arguments:

#### [<clock-objects>]

Use this argument to specify a collection of clock objects returned by the *get\_clock\_relation* command. If this argument is not specified, the *report\_clock\_relation* command returns the relationships between all clocks in the design.

### Examples

```
sg_shell> report_clock_relation  
//relationships between all clocks in the design
```

The clock matrix reported is shown in the following figure.

```

+++++
Clock matrix          top.clk1(SG_AUTO_TAG_1)  top.clk2(SG_AUTO_TAG_2)  top.clk3(SG_AUTO_TAG_3)
=====
top.clk1(SG_AUTO_TAG_1)  NA              A              A
top.clk2(SG_AUTO_TAG_2)  A              NA             A
top.clk3(SG_AUTO_TAG_3)  A              A              NA
+++++

```

**FIGURE 1.** Sample clock matrix

```

sg_shell> report_clock_relation [get_clock_relation clk1
clk2]
//relationship between clk1 and clk2

```

The clock matrix reported is shown in the following figure.

```

+++++
Clock matrix          top.clk2(SG_AUTO_TAG_2)  top.clk1(SG_AUTO_TAG_1)
=====
top.clk2(SG_AUTO_TAG_2)  NA              A
top.clk1(SG_AUTO_TAG_1)  A              NA
+++++

```

**FIGURE 2.** Sample clock matrix

## See Also

[report\\_clocks](#), [get\\_registers](#), [propagate\\_clocks](#), [get\\_domains](#), [report\\_domains](#), [get\\_clock\\_relation](#)

## report\_clocks

**Reports properties of user-specified clocks in current design**

### Syntax

```
report_clocks  
  [<clocks>]  
  [-verbose]
```

### Scope

Project

### Return Value

None

### Description

The *report\_clocks* command reports the properties of the user-specified clocks in the current design. The properties of clocks are reported in a tabular format.

#### Limitations

The *report\_clocks* command fails in the following cases:

- Selected design query view type is not flat.
- Flattened design view is not present.
- An improper collection of objects, which does not contain clocks returned by the *get\_clocks* command, has been provided as an input to the *report\_clocks* command.

### Arguments

The *report\_clocks* command has the following arguments, none of which is mandatory.

#### [<clocks>]

Collection used in the *report\_clocks* command is a set of clocks returned by the *get\_clocks* command.

**[ -verbose ]**

Use this argument to display the verbose information about clocks. If specified, this argument reports the following properties of clocks:

- Clock name
- Clock type
- Domain name
- Period (if defined)
- Edgelist of a clock (if defined)
- Backref of a clock
- Tag associated with a clock (if defined)
- List of flip-flops being driven by the positive edge of a clock (if at least one exists)
- List of flip-flops being driven by the negative edge of a clock (if at least one exists)
- List of latches being driven by a clock (if at least one exists)

If the *-verbose* argument is not specified, the following properties are reported:

- Clock name
- Clock type
- Domain name
- Period (if defined)
- Edgelist of a clock (if defined)
- Backref of a clock
- Tag associated with a clock (if defined)

**Examples**

```
sg_shell> report_clocks
# reports properties of all user defined clocks in current
design

sg_shell> report_clocks [get_clocks tagName] -verbose
# reports verbose properties of clocks in current design for
```

which clock tag is defined as 'tagName'

```
sg_shell> report_clocks [get_clocks -of_objects [get_cells  
top.I1.I2.bot]] -verbose  
# reports verbose properties of clocks in current design  
which are driving the flat cell 'top.I1.I2.bot'
```

## See Also

[get\\_clocks](#), [get\\_registers](#), [propagate\\_clocks](#), [get\\_domains](#), [report\\_domains](#)

## get\_registers

Used to get a list of cells driven by specified clocks/resets

### Syntax

```
get_registers
  <patterns>
  [-all ]
  [ -edge_triggered | -level_sensitive | -posedge_triggered |
    -negedge_triggered ]
```

### Scope

Project

### Return Value

Returns a collection of cells driven by the specified clocks/resets in case of successful execution. An empty collection is returned if no cells are driven by the specified clocks/resets. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_registers* command is used to get a list of cells driven by the specified clocks/resets.

#### Limitations

The *get\_registers* command fails in the following cases:

- Selected design query view type is not flat.
- Flattened design view is not present.

### Arguments

The *get\_registers* command has the following arguments:

#### <patterns>

Use this argument to specify a collection of clocks/resets returned by the *get\_clocks/get\_resets* command respectively. Use this argument to specify a collection of domains returned by the *get\_domains* command.

**[-all ]**

Use this argument to report all registers, in and below the current design hierarchy, driven by the specified clocks/resets. In the absence of this switch, all registers, only in current design hierarchy, driven by the specified clocks/resets are reported.

**[-edge\_triggered | -level\_sensitive | -posedge\_triggered |  
-negedge\_triggered ]**

- If you specify the *-edge\_triggered* switch, only flip-flops that are driven by the specified clocks/resets are reported.
- If you specify the *-posedge\_triggered* switch, only flip-flops that are driven by the positive edge of the specified clocks are reported.
- If you specify the *-negedge\_triggered* switch, only flip-flops that are driven by the negative edge of the specified clocks are reported.
- If you specify the *-level\_sensitive* switch, only latches that are driven by the specified clocks/resets are reported.

**NOTE:** *The -posedge\_triggered and -negedge\_triggered arguments are applicable only for clocks.*

## Examples

```

sg_shell> get_registers [get_clocks expr]
# Lists cells (both flip-flops and latches) in current design
hierarchy driven by specified clocks

sg_shell> get_registers [get_domains expr] -all
# Lists all cells (both flip-flops and latches) driven by
clocks of specified domains

sg_shell> get_registers [get_clocks expr] -all
-edge_triggered
# Lists all flip-flops driven by specified clocks

sg_shell> get_registers [get_domains expr] -edge_triggered
# Lists flip-flops in current design hierarchy driven by
clocks of specified domains

sg_shell> get_registers [get_domains expr]
-posedge_triggered
# Lists flip-flops in current design hierarchy driven by

```

positive edge of clocks of specified domains

```
sg_shell> get_registers [get_domains expr]
-negedge_triggered
# Lists flip-flops in current design hierarchy driven by
negative edge of clocks of specified domains

sg_shell> get_registers [get_clocks expr] -all
-level_sensitive
# Lists all latches driven by clock

sg_shell> get_registers <resetName>
# Returns all the flops/latches where the reset <resetName>
is reaching

sg_shell> get_registers [get_resets -of_object <get_cells/
get_pins/get_nets> <obj Name>]
# Returns all the registers where the resets, that are
returned by get_resets command, are reaching.

sg_shell> get_registers <resetName> -edge_triggered
# Returns all the flops that are receiving the reset
<resetName>

sg_shell> get_registers <resetName> -level_sensitive
#Returns all the latches that are receiving the reset
<resetName>
```

## See Also

[get\\_clocks](#), [report\\_clocks](#), [propagate\\_clocks](#), [get\\_domains](#), [report\\_domains](#),  
[get\\_resets](#), [report\\_resets](#)



## get\_resets

Creates a list of user defined resets in current design

### Syntax

```
get_resets [<resetName>] [-filter <filter_expression>] [-of_objects <obj>]
```

### Scope

Project

### Return Value

Returns a collection of user defined resets in current design. This collection may be used as input to other commands, for example, `report_resets` and `get_registers`. In case if no user defined resets are found then an empty collection is returned.

### Description

This command creates a collection of user defined resets in current design matching specified criteria.

The `get_resets` command fails when:

- Design query view type selected is not flat.
- Flattened view of design is not present.

### Arguments

The `get_resets` command has the following arguments:

#### <-resetName>

Use this argument as name of resets. Collection of all resets having name as specified along with this option are returned.

#### [-filter]

Filter expression can be specified over named attributes of resets. For example, use `reset_name` (for name of reset), `reset_type` (for type of the reset, which can be sync or async), `file_name` (for the source file

where a reset is defined), `active_value` [high or low], `is_soft` (soft resets are returned).

### **[ -of\_objects ]**

It can be specified as collection of flat instances, flat nets, or flat pins. A collection of resets driving the specified flat objects are returned.

## **Examples**

Consider the following commands:

```
sg_shell> set_pref dq_design_view_type flat
```

```
sg_shell> get_resets # returns collection of all user  
defined resets in current design
```

```
sg_shell> get_resets -of_objects [get_pins expr] # returns  
collection of user defined resets which are driving flat pins  
which match expr
```

## **See Also**

[report\\_resets](#), [get\\_registers](#)

## get\_value

Used to get simulation value of specified design object (port, pin, or net) in last cycle

### Syntax

```
get_value <design_objects>
```

### Scope

Project

### Return Value

Returns a list of simulation values of the specified design objects

### Description

The *get\_value* command is used to get a simulation value of the specified design object (port, pin, or net) in the last simulation cycle.

In interactive environment, whenever the [set\\_case\\_analysis](#) design constraints are applied on terminal, ports, or nets, its impact on the *get\_value* command is observed only after the flattened view of a design is recreated. The flattened design view is created by the [compile\\_design](#) command. In addition, you can use the [run\\_goal](#) command if there are rules that require a flattened view.

### Limitations

The *get\_value* command fails in the following cases:

- Selected design query view type is not flat.
- Flattened design view is not present.
- A collection of objects has been provided as an input to this command, and those objects are not of the *flat\_net*, *flat\_port*, or *flat\_term* type.
- Specified design object is not found in the current design.

## Arguments

The *get\_value* command has the following argument:

### <design\_objects>

Use this argument to specify the design object for which a simulation value needs to be found. It can be specified as a collection of nets, ports, or pins matching the user-defined criteria, as returned from the [get\\_nets](#), [get\\_ports](#), or [get\\_pins](#) command executed on the flat design view. It can also be specified as space-separated hierarchical path names of nets, ports, or terminals for which a simulation value needs to be known.

## Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> get_value top.clk
#Prints simulation value on top.clk
sg_shell> get_value [get_nets top.CDC04b.*.*]
#Prints simulation value of all flat nets matching this
criteria
sg_shell> get_value [get_ports top.*]
#Prints simulation value of all flat ports matching this
criteria
sg_shell> get_value [get_pins top.CDC0b.inst.*.*]
#Prints simulation value of all flat terms matching this
criteria
```

## See Also

[get\\_ports](#), [get\\_pins](#), [get\\_nets](#)

## propagate\_clocks

Propagates the user-defined clocks

### Syntax

```
propagate_clocks
```

### Scope

Project

### Return Value

None

### Description

The *propagate\_clocks* command propagates the user-defined clocks. The clocks specified in the SGDC constraints file are propagated across the design.

### Arguments

None

### Examples

```
propagate_clocks
```

### See Also

[get\\_clocks](#), [report\\_clocks](#), [get\\_domains](#), [report\\_domains](#), [get\\_registers](#)

## propagate\_resets

Propagates the user-defined resets

### Syntax

```
propagate_resets
```

### Scope

Project

### Return Value

None

### Description

The *propagate\_resets* command propagates the user-defined resets. The resets specified in the SGDC constraints file are propagated across the design.

### Arguments

None

### Examples

```
propagate_resets
```

### See Also

[get\\_resets](#), [report\\_resets](#), [get\\_registers](#)

## get\_domains

**Creates a list of domains of the user-defined clocks in the current design**

### Syntax

```
get_domains  
  [<dName>]  
  [-of_objects <obj>]
```

### Scope

Project

### Return Value

Returns a collection of domains of the user-defined clocks in the current design. This collection can be used as an input to other commands, such as [report\\_domains](#), [get\\_registers](#), and [get\\_clocks](#). If no user-defined domains are found, an empty collection is returned.

### Description

The *get\_domains* command creates a collection of domains of the user-defined clocks in the current design.

#### Limitations

The *get\_domains* command fails in the following cases:

- Selected design query view type is not flat.
- Flattened design view is not present.

### Arguments

The *get\_domains* command has the following arguments:

[ <dName> ]

Use this argument to specify the clock domain name. If this input is provided, domains that have a domain name as <dName> are returned.

**[-of\_objects <obj>]**

Use this argument to specify a collection of clocks, flat cells, flat pins, or flat nets. If this input is specified as a collection of clocks, the domains of given clocks are returned. If this input is specified as a collection of flat cells, flat pins, or flat nets, the domains of clocks that are driving the specified flat cells, flat pins, or flat nets are returned.

**Examples**

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> get_domains
# returns collection of domains of all user defined clocks in
current design
sg_shell> get_domains dName
# returns collection of domains named as 'dName'
sg_shell> get_domains -of_objects [get_pins expr]
# returns collection of domains of clocks that are driving
flat pins matching 'expr'
sg_shell> get_domains -of_objects [get_cells expr]
# returns collection of domains of clocks that are driving
flat cells matching 'expr'
sg_shell> get_domains -of_objects [get_clocks expr]
# returns collection of domains of clocks matching 'expr'
```

**See Also**

[get\\_clocks](#), [report\\_clocks](#), [propagate\\_clocks](#), [report\\_domains](#), [get\\_registers](#)



## report\_domains

**Reports the list of clocks of the specified domains**

### Syntax

```
report_domains  
  [<domains>]
```

### Scope

Project

### Return Value

None

### Description

The *report\_domains* command reports a list of clocks of the specified domains in a tabular format.

#### Limitations

The *report\_domains* command fails in the following cases:

- Selected design query view type is not flat.
- Flattened design view is not present.
- An improper collection of objects, which does not contain the domains returned by the *get\_domains* command, is provided as an input to this command.

### Arguments

The *report\_domains* command has the following argument:

[<domains>]

Use this argument to specify a collection of domains returned by the *get\_domains* command.

## Examples

```
sg_shell> report_domains  
# reports list of clocks of all user defined domains  
sg_shell> report_domains [get_domains expr]  
# reports list of clocks of domains which match 'expr'
```

## See Also

[get\\_clocks](#), [report\\_clocks](#), [get\\_domains](#), [propagate\\_clocks](#), [get\\_registers](#)

## report\_resets

**Reports properties of user specified resets in current design**

### Syntax

```
report_resets [<resets>] [-verbose]
```

### Scope

Project

### Return Value

None.

### Description

This command is used to report properties of user specified resets in current design.

It is mandatory to run the `propagate_resets` command before `reports_resets`, so that flops/latches driven by resets are reported.

The `get_resets` command fails when:

- Design query view type selected is not flat.
- Flattened design view is not present.
- An improper collection (a collection that is not of resets as returned by the `get_resets` command) of objects has been provided as an input to the `report_resets` command.

### Arguments

The `report_resets` command has the following arguments:

#### <resets>

Collection of resets returned from the `get_resets` command. It reports properties of resets in tabular format.

#### [-verbose]

Displays verbose information about resets. If specified, it reports the

following properties of resets:

- Reset name
- Backref of reset
- List of latches/flops being driven by reset, if at least one exists.

## Examples

Consider the following commands:

```
sg_shell> report_resets # reports properties of all user  
defined resets in current design
```

```
sg_shell> report_resets [get_resets resetName] -verbose #  
reports verbose properties of resets in current design for  
which reset name is defined as resetName
```

## See Also

[get\\_resets](#), [get\\_registers](#)

## Collection Commands

The design query commands related to various object models have *collection* as a way of communication between different commands. The collection encapsulates the objects of these object models, which are then exchanged among commands. A collection can have single or multiple objects depending on the result of a given command.

The table below describes the various collection commands. You use these commands to traverse or filter objects and to get a count of objects in a given collection.

Command	Description
<i>add_to_collection</i>	Add objects to a base collection and form a new collection. The base collection remains unchanged.
<i>append_to_collection</i>	Add objects to a collection, modifying the variable containing it
<i>compare_collections</i>	Used to compare two collections, returning 0 if they match
<i>filter_collection</i>	Used to filter a given base collection with some specific criteria
<i>foreach_in_collection</i>	Used to iterate over the objects of a collection
<i>index_collection</i>	Used to extract a single object from a collection based on its index
<i>query_objects</i>	Used to display objects in the argument collection
<i>remove_from_collection</i>	Remove objects from a base collection and form a new collection
<i>sizeof_collection</i>	Used to determine the number of objects in a collection

## add\_to\_collection

**Add objects to a base collection and form a new collection. The base collection remains unchanged**

### Syntax

```
add_to_collection
  [-unique]
  <base_collection>
  <objects>
```

### Scope

Project

### Return Value

Returns a new collection containing objects in the *<objects>* argument added to the objects in the *<base\_collection>* argument. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *add\_to\_collection* command is used to add elements to a collection.

All objects from the *<base\_collection>* argument are added to the resulting collection. Depending on the nature of the *<base\_collection>* argument, the following actions are taken on the *<objects>* argument:

- **Case 1:** The *<base\_collection>* argument is homogeneous, that is, all objects in the collection are of the same object class
  - If the *<objects>* argument is a collection, the *add\_to\_collection* command adds the objects from the *<objects>* argument to the result collection. This operation may result in a heterogeneous collection if the *<base\_collection>* and *<objects>* arguments are of different object class.
  - If the *<objects>* argument is a pattern, the *add\_to\_collection* command searches for the design object's name patterns by using

the object class of the base collection and then adds all the matched objects in the resulting collection.

- **Case 2:** The `<base_collection>` argument is heterogeneous, that is, not all objects in the collection are of the same object class
  - If the `<objects>` argument is a collection, the `add_to_collection` command adds the objects from the `<objects>` argument to the result collection.
  - If the `<objects>` argument is a pattern, the `add_to_collection` command ignores all implicit object patterns.
- **Case 3:** The `<base_collection>` argument is empty, that is, ""
  - There must be at least one homogeneous collection in the `<objects>` argument list. The position of the collection in the list does not matter. The first homogeneous collection in the `<objects>` argument list becomes the base collection. The implicit object name patterns in the `<objects>` argument list are searched by using this object class. The order of elements in the result collection remains same to the order of elements in the `<objects>` argument.
- **Case 4:** The `<objects>` argument is empty, that is, ""
  - The result is a copy of the base collection.
- **Case 5:** The `-unique` argument is provided
  - Duplicate objects are removed from the result collection.

## Arguments

The `add_to_collection` command has the following arguments:

### `[-unique]`

Use this argument to remove duplicate objects from the resulting collection. By default, the `add_to_collection` command does not remove duplicate objects.

### `<base_collection>`

Use this argument to specify the base collection in which objects are to be added. This option can be an empty collection (empty string).

## <objects>

Use this argument to specify a list of named objects or collections to be added to the base collection.

## Examples

```
sg_shell> set cells [get_cells U*]
{"U0", "U1", "U2"}

sg_shell> set nets [get_nets N*]
{"N0", "N1", "N2"}

sg_shell> add_to_collection $cells *1
{"U0", "U1", "U2", "I1", "U1"}

sg_shell> set mix [add_to_collection $cells $nets]
{"U0", "U1", "U2", "N0", "N1", "N2"}

sg_shell> add_to_collection $mix N*
add_to_collection: warning: nothing implicitly matched 'N*'
{"U0", "U1", "U2", "N0", "N1", "N2"}

sg_shell> add_to_collection $mix [get_ports P*]
{"U0", "U1", "U2", "N0", "N1", "N2", "P0", "P1", "P2"}

sg_shell> add_to_collection "" *2 $cells
{"I2", "U2", "U0", "U1", "U2"}

sg_shell> add_to_collection "" "U* N*"
add_to_collection: error: nothing implicitly matched
object(s)

sg_shell> add_to_collection $cells $cells
{"U0", "U1", "U2", "U0", "U1", "U2"}

sg_shell> add_to_collection $nets $nets -unique
{"N0", "N1", "N2"}
```

## See Also

[append\\_to\\_collection](#), [remove\\_from\\_collection](#), [sizeof\\_collection](#), [filter\\_collection](#)



## append\_to\_collection

Add objects to a collection, modifying the variable containing it

### Syntax

```
append_to_collection  
    [-unique]  
    <var_name>  
    <objects>
```

### Scope

Project

### Return Value

Returns a collection containing the objects in the *<objects>* argument added to the objects in the collection referenced by a variable specified in the *<var\_name>* argument. If the variable *<var\_name>* does not exist, it is created. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *append\_to\_collection* command is used to add elements to a collection and simultaneously modify the variable containing the collection. This command treats the variable name, specified by using the *<var\_name>* argument, as a collection, and it appends all the objects in the *<objects>* argument to that collection. If the variable name does not exist, it is created as a collection of objects, which are specified by using the *<objects>* argument, as its value. If a variable exists and it does not hold a collection, an error is reported.

Consider the following examples to compare the usage of the *add\_to\_collection* command with the *append\_to\_collection* command.

The following example illustrates the usage of the *add\_to\_collection* command:

```
sg_shell> set base_col [add_to_collection $base_col  
$new_objs]
```

The following example illustrates the usage of the *append\_to\_collection* command:

```
sg_shell> append_to_collection base_col $new_objs
```

Please note that `base_col`, not `$base_col`, is used in the *append\_to\_collection* command.

**NOTE:** *The semantics of the `append_to_collection` command is similar to that of the `add_to_collection` command. However, it is recommended that you use the `append_to_collection` command for building up a collection in iterations.*

## Arguments

The *append\_to\_collection* command has the following arguments:

### `[-unique]`

Use this argument to remove duplicate objects from the resulting collection. By default, the *append\_to\_collection* command does not remove duplicate objects.

### `<var_name>`

Use this argument to specify a variable name. The design objects matching the *<objects>* argument are added into the collection referenced by this variable. This variable can point to an empty collection (empty string) or a new variable. If the variable is nonexistent or contains an empty string, it is subject to the same restrictions as in the *add\_to\_collection* command.

### `<objects>`

Use this argument to specify a list of named objects or collections to be added to the specified variable *<var\_name>*.

**NOTE:** *The arguments of the `append_to_collection` command have the same restrictions and semantics as the `add_to_collection` command.*

## Examples

```
sg_shell> append_to_collection objs [get_cells U*]  
{ "U0", "U1", "U2" }
```

```
sg_shell> append_to_collection objs [get_nets N*]  
{ "U0", "U1", "U2", "N0", "N1", "N2" }
```

```
sg_shell> foreach_in_collection cell [get_lib_cells *.*] {
    if { [get_attribute $cell is_sequential] } {
        append_to_collection seq_cell_list $cell
    } else {
        append_to_collection comb_cell_list $cell
    }
}

sg_shell> set var 10
sg_shell> append_to_collection var [get_ports P*]
append_to_collection: error: variable 'var' does not hold a
collection.

sg_shell> append_to_collection nets [get_nets N*]
{"N0", "N1", "N2"}
sg_shell> append_to_collection nets [get_nets N*]
{"N0", "N1", "N2", "N0", "N1", "N2"}
sg_shell> append_to_collection nets "" -unique
{"N0", "N1", "N2"}
```

## See Also

[add\\_to\\_collection](#), [remove\\_from\\_collection](#), [sizeof\\_collection](#), [filter\\_collection](#)

## compare\_collections

Used to compare two collections, returning 0 if they match

### Syntax

```
compare_collections  
    [-order_dependent]  
    <collection1>  
    <collection2>
```

### Scope

Project

### Return Value

Returns an integer value with 0 indicating a match and any other integer indicating a mismatch. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *compare\_collections* command is used to compare the contents of two collections. The behavior of this command is similar to the string compare function, *strcmp*, in a way that it returns 0 if *<collection1>* and *<collection2>* match and any other integer in case of mismatch.

By default, the order of objects in *<collection1>* and *<collection2>* does not matter. But, if you use the *[-order\_dependent]* option, the order of objects in both collections is also matched.

An empty string (" ") is equivalent to an empty collection, that is, collection of size 0 with no objects. Therefore, if two empty collections or strings are compared, the comparison is successful, that is, the result is 0.

### Arguments

The *compare\_collections* command has the following arguments:

#### **[-order\_dependent]**

Use this argument to specify the order of objects in the collections that need to be considered during comparison.

## Debug Commands

**<collection1>**

Use this argument to specify the first collection to be compared.

**<collection2>**

Use this argument to specify the second collection to be compared.

**Examples**

```
sg_shell> append_to_collection obj1 [get_cells U*]
{"U0", "U1", "U2", "U3"}

sg_shell> append_to_collection obj1 [get_nets N*]
{"U0", "U1", "U2", "U3", "N0", "N1", "N2", "N3"}

sg_shell> append_to_collection obj2 [get_nets N*]
{"N0", "N1", "N2", "N3"}

sg_shell> append_to_collection obj2 [get_cells U*]
{"N0", "N1", "N2", "N3", "U0", "U1", "U2", "U3"}

sg_shell> compare_collections $obj1 $obj2
0

sg_shell> compare_collections $obj1 $obj2 -order_dependent
-4

sg_shell> compare_collections "" $obj2
-1

sg_shell> compare_collections "" ""
0
```

**See Also**

[foreach\\_in\\_collection](#), [sizeof\\_collection](#), [append\\_to\\_collection](#)

## filter\_collection

Used to filter a given base collection with some specific criteria

### Syntax

```
filter_collection base_collection expression [-regexp]
```

### Scope

Project

### Return Value

Returns a collection with matched objects as per the expression. An empty collection is returned when no match is found as per the expression. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *filter\_collection* command filters a given base collection with specific criteria. After filtering, a new collection is returned and the base collection remains unchanged. This command is most useful if you plan to filter the same large collection many times using different criteria. You can give more than one condition joined together with AND or OR operator. Parentheses () are also supported.

### Arguments

The *filter\_collection* command has the following arguments:

#### **base\_collection**

Use this argument to specify the base collection to be filtered. Objects from this collection are copied to the result collection as per the given condition. The given conditional expression is evaluated on the attribute of the objects. Objects that satisfy the condition, that is returning `true`, are copied to the resulting collection.

#### **expression**

Use this argument to specify an expression with which base collection needs to be filtered.

**[-regexp]**

Use this argument to specify that the `=~` and `!~` filter operators use a real regular expressions. By default, the `=~` and `!~` filter operators use simple wildcard pattern matching with the `*` and `?` wildcard characters. The relational operators that can be used are as follows:

- `==` Equal
- `!=` Not equal
- `>` Greater than
- `<` Less than
- `>=` Greater than or equal to
- `<=` Less than or equal to
- `=~` Matches pattern
- `!~` Does not match pattern

The existence operators that can be used are as follows:

- `defined`
- `undefined`

**Examples**

```
sg_shell> define_user_attribute -class lib_cell -type string
srtattr
define_user_attribute: info: defining new attribute
'srtattr' of type 'string'
sg_shell> set_user_attribute [get_lib_cells mylib_20c.AN*]
srtattr a
sg_shell> set_user_attribute [get_lib_cells mylib_20c.FD*]
srtattr h
sg_shell> set_user_attribute [get_lib_cells mylib_20c.FJK*]
srtattr x
sg_shell> set A [get_lib_cells -of_objects [get_libs
mylib_20c]]
sg_shell> filter_collection -regexp $A {srtattr=~[a-m]}

{"mylib_20c.AN2", "mylib_20c.OR2", "mylib_20c.AN3",
```

```
"mylib_20c.OR3", "mylib_20c.AN2P", "mylib_20c.AN3P",  
"mylib_20c.AN4", "mylib_20c.AN4P", "mylib_20c.OR2P",  
"mylib_20c.OR3P", "mylib_20c.OR4", "mylib_20c.OR4P",  
"mylib_20c.FD1", "mylib_20c.FD1P", "mylib_20c.FD1S",  
"mylib_20c.FD1SP", "mylib_20c.FD2", "mylib_20c.FD2P",  
"mylib_20c.FD2S", "mylib_20c.FD2SP", "mylib_20c.FD3",  
"mylib_20c.FD3P", "mylib_20c.FD3S", "mylib_20c.FD3SP",  
"mylib_20c.FD4", "mylib_20c.FD4P", "mylib_20c.FD4S",  
"mylib_20c.FD4SP", "mylib_20c.FDS2", "mylib_20c.FDS2L",  
"mylib_20c.FDS2LP", "mylib_20c.FDS2P", "mylib_20c.FD2TS",  
"mylib_20c.FD2TSP"}
```

```
sg_shell> filter_collection -regexp $A {srtattr==x}
```

```
{"mylib_20c.FJK1", "mylib_20c.FJK1P", "mylib_20c.FJK1S",  
"mylib_20c.FJK1SP", "mylib_20c.FJK2", "mylib_20c.FJK2P",  
"mylib_20c.FJK2S", "mylib_20c.FJK2SP", "mylib_20c.FJK3",  
"mylib_20c.FJK3P", "mylib_20c.FJK3S", "mylib_20c.FJK3SP"}
```

```
sg_shell> filter_collection $A {is_combinational==true &&  
area > 2}
```

```
{"mylib_20c.AN2", "mylib_20c.OR2", "mylib_20c.AN3",  
"mylib_20c.OR3", "mylib_20c.AN4", "mylib_20c.AN4P",  
"mylib_20c.OR3P", "mylib_20c.OR4", "mylib_20c.OR4P"}
```

## See Also

[get\\_attribute](#), [set\\_user\\_attribute](#), [define\\_user\\_attribute](#), [remove\\_user\\_attribute](#),  
[list\\_attributes](#)



## foreach\_in\_collection

Used to iterate over the objects of a collection

### Syntax

```
foreach_in_collection <varname> <collection> <body>
```

### Scope

Project

### Return Value

None

### Description

The *foreach\_in\_collection* command iterates over the objects of a collection. The provided *sg\_shell* script is applied on each object. The iterator variable holds the object and it is of the collection type. Therefore, commands that accept collection as an argument can accept the iterator variable.

**NOTE:** *The Tcl shell foreach command cannot be used for collections because it needs a list and collection is an internal structure only understood by sg\_shell.*

### Arguments

The *foreach\_in\_collection* command has the following arguments:

**<varname>**

Use this argument to specify the name of an iterator variable.

**<collection>**

Use this argument to specify a collection or a list of collections.

**<body>**

Use this argument to specify *sg\_shell* script that needs to be executed for each element.

## Examples

```
sg_shell> foreach_in_collection i [get_lib_cells  
mylib_20c.OR*] {  
    puts "[get_attribute $i base_name]"  
}
```

```
OR2  
OR3  
OR2P  
OR3P  
OR4  
OR4P
```

## See Also

[sizeof\\_collection](#), [filter\\_collection](#)

## index\_collection

Used to extract a single object from a collection based on its index

### Syntax

```
index_collection  
  <collection>  
  <index>
```

### Scope

Project

### Return Value

Returns a single object collection containing the extracted object from the *<collection>* argument based on the *<index>* argument. The base collection *<collection>* remains unchanged. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *index\_collection* command is used to extract a single object from a collection based on its index. The index should be within the range of 0 to `[sizeof_collection $collection] - 1`. Any index outside the range results in an error. A successful execution of this command results in a new collection that contains only a single object.

You can use the empty string in the *<collection>* argument. However, an empty string means an empty collection. Therefore, any index into the empty collection is invalid. In other words, if you use the *index\_collection* command with an empty string, it will always result in an error.

**NOTE:** *The index\_collection command presently does not support a constant time algorithm. Therefore, this command should be used judiciously when working with large indices.*

## Arguments

The *index\_collection* command has the following arguments:

### <collection>

Use this argument to specify the collection from which an object needs to be extracted.

### <index>

Use this argument to specify the index of the collection. This argument accepts integer values from 0 to [sizeof\_collection \$collection] - 1.

## Examples

```
sg_shell> set or_cells [get_lib_cells mylib_20c.OR*]
{"mylib_20c.OR2", "mylib_20c.OR3", "mylib_20c.OR2P",
"mylib_20c.OR3P", "mylib_20c.OR4", "mylib_20c.OR4P"}
```

```
sg_shell> index_collection $or_cells 2
{"mylib_20c.OR2P"}
```

```
sg_shell> index_collection "" 4
index_collection: error: Invalid index 4 for collection ''
```

## See Also

[foreach\\_in\\_collection](#), [sizeof\\_collection](#)

## query\_objects

Used to display objects in the argument collection

### Syntax

```
query_objects
  [-verbose]
  [-truncate elem_count]
  <collection>
```

### Scope

Project

### Return Value

Returns an empty string in case of success. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *query\_objects* command is used to display objects in the argument collection. The command does not have a return value. It only prints the object names contained in the collection. An error is returned if the argument collection contains non-displayable objects, such as *lib\_timing\_arcs*.

**NOTE:** *The output from the query\_objects command is similar to the output from any command that returns a collection. However, the result of the query\_objects command is always an empty string.*

### Arguments

The *query\_objects* command has the following arguments:

#### **[-verbose]**

Use this argument to display the object class of each object. By default, the *query\_objects* command only prints the object names. When you use this argument, the command also prints the object class along with the object names.

**[-truncate elem\_count]**

Use this argument to truncate display to the `elem_count` objects. By default, all objects of a collection are printed. If the display is truncated, you will see the ellipsis (...) as the last element.

**<collection>**

Use this argument to specify the collection whose object names need to be displayed.

**Examples**

```
sg_shell> query_objects [get_cells U*]
{"U0", "U1", "U2", "U3"}

sg_shell> query_objects [get_cells U*] -truncate 2
{"U0", "U1",...}

sg_shell> append_to_collection objs [get_cells U*]
{"U0", "U1", "U2", "U3"}

sg_shell> append_to_collection objs [get_nets N*]
{"U0", "U1", "U2", "U3", "N0", "N1", "N2", "N3"}

sg_shell> query_objects $objs
{"flat_cell:U0", "flat_cell:U1", "flat_cell:U2",
"flat_cell:U3", "flat_net:N0", "flat_net:N1", "flat_net:N2",
"flat_net:N3"}

sg_shell> query_objects [get_lib_timing_arcs -of_objects
*.*]
query_objects: error: Collection '_sggrp12' contains
object(s) of type 'lib_timing_arcs', which cannot be
displayed
```

**See Also**

[collection\\_display\\_limit](#), [get\\_cells](#), [get\\_lib\\_cells](#), [get\\_lib\\_pins](#), [get\\_libs](#), [get\\_nets](#), [get\\_pins](#), [get\\_ports](#)

## remove\_from\_collection

Remove objects from a base collection and form a new collection

### Syntax

```
remove_from_collection  
    [-intersect]  
    <base_collection>  
    <objects>
```

### Scope

Project

### Return Value

Returns a new collection with the objects in the *<objects>* argument removed or retained from the *<base\_collection>* argument. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *remove\_from\_collection* command is used to remove elements from a collection, creating a new collection.

By default, this command removes all elements, which are specified in the *<objects>* argument and are also present in the *<base\_collection>* argument, from the result collection. If you specify the *-intersect* argument, all elements in the *<objects>* argument, which are also present in the *<base\_collection>* argument, are retained while others are removed.

Depending on the nature of the *<base\_collection>* argument, the following actions are taken on the *<objects>* argument:

- **Case 1:** The *<base\_collection>* argument is homogeneous, that is, all objects in the collection are of the same object class
  - If the *<objects>* argument is a collection, the *remove\_from\_collection* command removes or retains the objects in

the *<objects>* argument from the *<base\_collection>* argument to the result collection.

- If the *<objects>* argument is a pattern, the *remove\_from\_collection* command searches for the design object's name patterns by using the object class of the base collection and then removes or retains all the matched objects in the resulting collection.
- **Case 2:** The *<base\_collection>* argument is heterogeneous, that is, not all objects in the collection are of the same object class
  - If the *<objects>* argument is a collection, the *remove\_from\_collection* command removes or retains the objects from the *<objects>* argument to the result collection.
  - If the *<objects>* argument is a pattern, the *remove\_from\_collection* command ignores all implicit object patterns.
- **Case 3:** The *<objects>* argument is empty, that is, ""
  - If the *-intersect* argument is provided, the result is also empty.
  - If the *-intersect* argument is not provided, the result is a copy of the base collection.

## Arguments

The *remove\_from\_collection* command has the following arguments:

### **[ -intersect ]**

Use this argument to indicate that objects in the *<objects>* argument, which are also present in the *<base\_collection>* argument, are to be retained in the result collection. By default, these objects are removed from the result collection.

### **<base\_collection>**

Use this argument to specify the base collection from which objects need to be removed or retained.



**<objects>**

Use this argument to specify a list of named objects or collections to be removed or retained from the base collection in the result collection.

**Examples**

```
sg_shell> set cells [get_cells U*]
{"U0", "U1", "U2"}

sg_shell> set nets [get_nets N*]
{"N0", "N1", "N2"}

sg_shell> remove_from_collection $cells U1
{"U0", "U2"}

sg_shell> remove_from_collection $nets [get_nets N1]
{"N0", "N2"}

sg_shell> set mix [add_to_collection $cells $nets]
{"U0", "U1", "U2", "N0", "N1", "N2"}

sg_shell> remove_from_collection $mix N*
remove_from_collection: warning: nothing implicitly matched
'N*'
{"U0", "U1", "U2", "N0", "N1", "N2"}

sg_shell> remove_from_collection $mix [get_nets N2]
{"U0", "U1", "U2", "N0", "N1"}

sg_shell> remove_from_collection $cells ""
{"U0", "U1", "U2"}

sg_shell> remove_from_collection $cells "" -intersect

sg_shell> remove_from_collection $mix [get_cells]
{"N0", "N1", "N2"}

sg_shell> remove_from_collection $mix [get_nets] -intersect
{"N0", "N1", "N2"}
```

**See Also**

[append\\_to\\_collection](#), [add\\_to\\_collection](#), [sizeof\\_collection](#), [filter\\_collection](#)

## sizeof\_collection

Used to determine the number of objects in a collection

### Syntax

```
sizeof_collection <collection>
```

### Scope

Project

### Return Value

Returns the size of a given collection if the collection exists. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *sizeof\_collection* command determines the number of objects in a collection.

### Arguments

The *sizeof\_collection* command has the following argument:

**<collection>**

Use this argument to specify the collection for which the total number of objects needs to be determined.

### Examples

```
sg_shell> set X [get_lib_cells mylib_20c.OR*]  
{ "mylib_20c.OR2", "mylib_20c.OR3", "mylib_20c.OR2P",  
  "mylib_20c.OR3P", "mylib_20c.OR4", "mylib_20c.OR4P" }
```

```
sg_shell> sizeof_collection $X
```

```
6
```

### See Also

[foreach\\_in\\_collection](#), [filter\\_collection](#)

## Attribute Commands

An attribute is a value associated with a design or a library object that carries some information about that object. This information is captured in various data types, such as *int*, *float*, *string*, *boolean*, and so on. You can write procedures in Tcl to fetch the attribute information and generate custom reports on the design or library.

Attributes can be of the following types:

- **Application attributes:** Application attributes are defined and used internally by the tool. The values of these attributes are inferred and set during design read operation. You can fetch the values of these attributes on design or library objects by using attribute commands. These attributes are read-only. You cannot set, modify, remove, or redefine these attributes.

Application attributes can be categorized in the following groups:

- Built-in Attributes*
- Product Attributes*

Refer to [Appendix B: Application Attributes](#) for the complete list of application attributes defined in SpyGlass.

- **User-defined attributes:** You can define your own attributes. These attributes can be set, fetched, modified, and removed. SpyGlass does not use these attributes internally.

In addition, these attributes are not persistent across different *sg\_shell* sessions and are lost once the project is closed. Even if you reopen the same project, the attributes are not restored. You have to redefine these attributes for further usage.

Attribute support is available for all object models, which include the following:

- **Library Object Model:** Attributes classified as *lib*, *lib\_cell*, *lib\_pin*, *lib\_timing\_arcs*, and so on.
- **Hierarchical Netlist Object Model:** Attributes classified as *du\_cell*, *du\_pin*, *du\_port*, *du\_net*, and so on.
- **Flat Netlist Object Model:** Attributes classified as *flat\_cell*, *flat\_pin*, *flat\_port*, *flat\_net*, and so on.
- **Atrenta Design Constraints Object Model:** Attributes tagged as *adc\_node*.

These commands perform various operations on attributes, which include *define*, *set*, *get*, *list*, *remove*, and *destroy*. The following table describes the various attribute commands:

<b>Command</b>	<b>Description</b>
<i>define_user_attribute</i>	Used to define a new user-defined attribute
<i>set_user_attribute</i>	Used to set a user attribute to a specified value on an object
<i>get_attribute</i>	Used to retrieve the value of an attribute on an object
<i>list_attributes</i>	Used to display a list of currently defined attributes
<i>remove_user_attribute</i>	Used to remove attributes set with <i>set_user_attribute</i> command
<i>destroy_user_attribute</i>	Used to destroy an attribute

## define\_user\_attribute

Used to define a new user-defined attribute

### Syntax

```
define_user_attribute -type data_type -classes class_list  
attr_name
```

### Scope

Any

### Return Value

Returns an empty string when the attribute is defined successfully. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *define\_user\_attribute* command defines a new user-defined attribute. A user-defined attribute is any attribute that SpyGlass does not understand by default. User-defined attributes can be applied to most object classes in SpyGlass. SpyGlass cannot use these attributes, but a user can use them in scripts, procedures, and so on.

**NOTE:** A user can list the user-defined attributes by using the [list\\_attributes](#) command.

### Arguments

The *define\_user\_attribute* command has the following arguments:

#### **-type data\_type**

Use this argument to specify the data type of the attribute. The supported data types are string, int, float, and boolean.

#### **-classes class\_list**

Use this argument to define the attribute for one or more classes. The valid object classes are lib, lib\_cell, lib\_pin, lib\_timing\_arcs, design, du\_cell, du\_port, du\_net, du\_pin, flat\_pin, flat\_port, flat\_cell, and flat\_net. While providing multiple class names, it should be specified within brackets or quotes and using space as delimiter.

**attr\_name**

Use this argument to specify the attribute name.

**Examples**

```
sg_shell> define_user_attribute -class lib_cell -type string
cellAttr
define_user_attribute: info: defining new attribute
'cellAttr' of type 'string'
sg_shell> set_user_attribute [get_lib_cells mylib_20c.AN*]
cellAttr attr1
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN2' with value 'attr1'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN3' with value 'attr1'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN2P' with value 'attr1'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN3P' with value 'attr1'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN4' with value 'attr1'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN4P' with value 'attr1'
sg_shell> get_lib_cells mylib_20c.A* -filter {cellAttr ==
attr1}
{"mylib_20c.AN2", "mylib_20c.AN3", "mylib_20c.AN2P",
"mylib_20c.AN3P", "mylib_20c.AN4", "mylib_20c.AN4P"}
```

**See Also**

[get\\_attribute](#), [set\\_user\\_attribute](#), [remove\\_user\\_attribute](#), [destroy\\_user\\_attribute](#),  
[list\\_attributes](#), [filter\\_collection](#)

## set\_user\_attribute

Used to set a user attribute to a specified value on an object

### Syntax

```
set_user_attribute [-class class_name] object attr_name  
value
```

### Scope

Project

### Return Value

Returns an empty string when the attribute is set successfully. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *set\_user\_attribute* command sets a user attribute to a specified value on an object.

### Arguments

The *set\_user\_attribute* command has the following arguments:

#### object

Use this argument to specify the object from which the attribute value needs to be set. Object must be either a collection or a named object with its *class\_name*. If name (string) is used, only single string is allowed. Using wildcard characters is the only way to specify more than one string.

#### [-class class\_name]

Use this argument to specify the class name of an object, if object is a name. Valid classes are lib, lib\_cell, lib\_pin, design, du\_cell, du\_port, du\_net, du\_pin, flat\_pin, flat\_port, flat\_cell, and flat\_net.

**attr\_name**

Use this argument to specify the attribute name whose value needs to be set on the given object.

**value**

The *value* of the attribute is used to set on the object. The value should conform to the type of the attribute, that is, int, float, string, and so on.

**Examples**

```
sg_shell> define_user_attribute -class lib_cell -type string
cellAttr
define_user_attribute: info: defining new attribute
'cellAttr' of type 'string'
sg_shell> set_user_attribute -class lib_cell mylib_20c.AN2
cellAttr attr1
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN2' with value 'attr1'
sg_shell> set_user_attribute [get_lib_cells mylib_20c.A04*]
cellAttr attr6
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.A04' with value 'attr6'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.A04P' with value 'attr6'
sg_shell> get_lib_cells mylib_20c.A* -filter "cellAttr ==
attr1"
{"mylib_20c.AN2"}
```

**See Also**

[get\\_attribute](#), [define\\_user\\_attribute](#), [remove\\_user\\_attribute](#),  
[destroy\\_user\\_attribute](#), [list\\_attributes](#), [filter\\_collection](#)



## get\_attribute

Used to retrieve the value of an attribute on an object

### Syntax

```
get_attribute [-class class_name] object attr_name value
```

### Scope

Project

### Return Value

Returns the value of attribute of the object(s). In case the attribute is not set on the object(s), an empty string is returned. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_attribute* command retrieves the value of an attribute on an object. The object is either a collection or a named object. If it is a name, the *-class* option is required.

### Arguments

The *get\_attribute* command has the following arguments:

#### object

Use this argument to specify the object from which the attribute value needs to be retrieved. Object must be either a collection or a named object with its *class\_name*. If name (string) is used, only single string is allowed. Using wildcard characters is the only way to specify more than one string.

#### [-class class\_name]

Use this argument to specify the class name of an object, if object is a name. Valid classes are lib, lib\_cell, lib\_pin, design, du\_cell, du\_port, du\_net, du\_pin, flat\_pin, flat\_port, flat\_cell, and flat\_net.

**attr\_name**

Use this argument to specify the name of the attribute whose value needs to be retrieved.

**Examples**

```
sg_shell> define_user_attribute -class lib_cell -type string
cellAttr
define_user_attribute: info: defining new attribute
'cellAttr' of type 'string'
sg_shell> set_user_attribute -class lib_cell mylib_20c.AN2
cellAttr attr1
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.AN2' with value 'attr1'
sg_shell> set_user_attribute [get_lib_cells mylib_20c.A04*]
cellAttr attr6
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.A04' with value 'attr6'
set_user_attribute: info: setting attribute 'cellAttr' on
object 'mylib_20c.A04P' with value 'attr6'
sg_shell> get_attribute [get_lib_cells mylib_20c.AN2]
cellAttr
attr1
sg_shell> get_attribute [get_lib_cells mylib_20c.A04P]
cellAttr
attr6
```

**See Also**

[set\\_user\\_attribute](#), [define\\_user\\_attribute](#), [remove\\_user\\_attribute](#),  
[destroy\\_user\\_attribute](#), [list\\_attributes](#), [filter\\_collection](#)

## list\_attributes

Used to display a list of currently defined attributes

### Syntax

```
list_attributes
  [-application]
  [-class class_name | attr_pattern]
  [-verbose]
```

### Scope

Any

### Return Value

None

### Description

The *list\_attributes* command displays a list of currently defined attributes. *sg\_shell* attributes are divided into the following two categories:

- Application-defined attributes
- User-defined attributes

There are a number of application attributes. It is often useful to limit the listing to the following:

- A specific object class by using the *class\_name* argument
- A specific attribute name by using the *attr\_pattern* argument

### Arguments

The *list\_attributes* command has the following arguments:

#### **[-application]**

By default, the *list\_attributes* command lists all user-defined attributes. Use this argument to add all application attributes to the listing.

#### **[-class class\_name]**

Use this argument to limit the listing to attributes of a single class. Valid

classes are `adc_node`, `cdc_conv_node`, `cdc_glitch_node`, `cdc_node`, `clock`, `clock_domain`, `design`, `du_cell`, `du_net`, `du_pin`, `du_port`, `flat_cell`, `flat_net`, `flat_pin`, `flat_port`, `lib`, `lib_cell`, `lib_pin`, `lib_timing_arcs`, `message`, `pwr_intent_node`, `pwr_retention_node`, `pwr_isolation_node`, `pwr_level_shift_node`, `pwr_psw_node`, `pwr_supply_node`, `reset_flop_node`, `reset_sync_node`, `reset`, `paths_node`, `cdc_source_node`, `cdc_conv_signal_node`, `cdc_glitch_source_node`, `rule`, and `sd_c_node`.

#### [`attr_pattern`]

Use this argument to limit the listing to attributes that match the `attr_pattern` argument. The short help includes the possible values of the attributes supported by the SpyGlass CDC product. You can only specify wildcard patterns in this argument. The `attr_pattern` and `class_name` arguments are mutually exclusive.

#### [`-verbose`]

Use this argument to display the short help of the application attributes as well. The short help includes the possible values of the attributes supported by the SpyGlass CDC product. This argument has no significance if the `-application` argument is not specified.

## Examples

```
sg_shell> list_attributes -class lib -application
*****
Report : List of Attribute Definitions
*****
Attribute Name                               Object      Type      User-Def
-----
full_name                                    lib         string    builtin
base_name                                    lib         string    builtin
default_cell_leakage_power                   lib         float     builtin
default_fanout_load                           lib         float     builtin
```

## Debug Commands

default_inout_pin_cap	lib	float	builtin
default_input_pin_cap	lib	float	builtin
default_leakage_power_density	lib	float	builtin
default_max_capacitance	lib	float	builtin
default_max_fanout	lib	float	builtin
default_max_transition	lib	float	builtin
default_output_pin_cap	lib	float	builtin
default_wire_load_area	lib	float	builtin
default_wire_load_capacitance	lib	float	builtin
default_wire_load_resistance	lib	float	builtin
nom_process	lib	float	builtin
nom_temperature	lib	float	builtin
nom_voltage	lib	float	builtin
voltage_unit	lib	string	builtin
time_unit	lib	string	builtin
current_unit	lib	string	builtin
capacitive_load_unit	lib	string	builtin
pulling_resistance_unit	lib	string	builtin
leakage_power_unit	lib	string	builtin
default_connection_class	lib	string	builtin
default_operating_conditions	lib	string	builtin
default_power_rail	lib	string	builtin
default_threshold_voltage_group	lib	string	builtin
default_wire_load	lib	string	builtin
default_wire_load_mode	lib	string	builtin
default_wire_load_selection	lib	string	builtin
define_cell_area	lib	string	builtin
delay_model	lib	string	builtin
technology	lib	string	builtin

```
sg_shell> list_attributes -verbose -application default_m*
```

```
*****
Report : List of Attribute Definitions
*****
```

```
default_max_capacitance
```

```
-----  
Object(s) : lib  
Type      : float  
Product   : builtin  
Help      : Default maximum capacitance as defined in the  
           library
```

default\_max\_fanout

```
-----  
Object(s) : lib  
Type      : float  
Product   : builtin  
Help      : Default maximum fan-out as defined in the  
           library
```

default\_max\_transition

```
-----  
Object(s) : lib  
Type      : float  
Product   : builtin  
Help      : Default maximum transition as defined in the  
           library
```

## See Also

[get\\_attribute](#), [set\\_user\\_attribute](#), [define\\_user\\_attribute](#), [remove\\_user\\_attribute](#),  
[destroy\\_user\\_attribute](#), [filter\\_collection](#)

## remove\_user\_attribute

Used to remove attributes set with `set_user_attribute` command

### Syntax

```
remove_user_attribute -class class_name object attr_name
```

### Scope

Project

### Return Value

Returns an empty string when an attribute is removed from the objects successfully. In case of unsuccessful execution, an error is returned that can be trapped by using the `catch` command.

### Description

The `remove_user_attribute` command removes the attributes set with the `set_user_attribute` command. Application attributes cannot be removed by this command.

**NOTE:** *This command only removes the attribute value. The attribute is not deleted from the attribute list.*

### Arguments

The `remove_user_attribute` command has the following arguments:

#### object

Use this argument to specify the object from which the attribute value needs to be removed. Object must be either a collection or a named object with its `class_name`. If name (string) is used, only single string is allowed. Using wildcards is the only way to specify more than one string.

#### [-class class\_name]

Use this argument to specify the class name of an object, if object is a name. Valid classes are `lib`, `lib_cell`, `lib_pin`, `design`, `du_cell`, `du_port`, `du_net`, `du_pin`, `flat_pin`, `flat_port`, `flat_cell`, and `flat_net`.

**attr\_name**

Use this argument to provide the name of the attribute.

**Examples**

```
sg_shell> remove_user_attribute -class lib_cell
mylib_20c.A012P cellAttr
remove_user_attribute: info: removing attribute 'cellAttr'
from object 'mylib_20c.A012P'
sg_shell> remove_user_attribute [get_lib_cells
mylib_20c.AN2*] cellAttr
remove_user_attribute: info: removing attribute 'cellAttr'
from object 'mylib_20c.AN2'
remove_user_attribute: info: removing attribute 'cellAttr'
from object 'mylib_20c.AN2P'
```

**See Also**

[get\\_attribute](#), [set\\_user\\_attribute](#), [define\\_user\\_attribute](#), [destroy\\_user\\_attribute](#),  
[list\\_attributes](#), [filter\\_collection](#)



## destroy\_user\_attribute

Used to destroy an attribute

### Syntax

```
destroy_user_attribute attr_name
```

### Scope

Any

### Return Value

Returns an empty string when the attribute is destroyed successfully. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *destroy\_user\_attribute* command destroys the *attr\_name* attribute defined by using the [define\\_user\\_attribute](#) command. Any further query about this attribute generates an unknown attribute error.

### Arguments

The *destroy\_user\_attribute* command has the following argument:

#### **attr\_name**

Use this argument to specify the name of the attribute that needs to be destroyed.

### Examples

```
sg_shell> destroy_user_attribute srtattr
destroy_user_attribute: info: destroying user attribute
'srtattr'
```

### See Also

[get\\_attribute](#), [set\\_user\\_attribute](#), [define\\_user\\_attribute](#), [remove\\_user\\_attribute](#), [list\\_attributes](#), [filter\\_collection](#)

## Product Commands

Product commands can be further categorized in the following groups:

- *SpyGlass Base Commands*
- *SpyGlass Lint Turbo Commands*
- *SpyGlass Constraints Commands*
- *SpyGlass CDC Commands*
- *SpyGlass DFT Commands*
- *SpyGlass Power Verify Commands*
- *SpyGlass Power Estimate and Reduce Commands*

The information provided by the product commands is computed as part of the design analysis done during the execution of the *run\_goal* command. So, it is required that you execute the *run\_goal* command in the currently selected goal for these product commands to be functional.

## SpyGlass Base Commands

The following table describes the Tcl commands that are a part of the SpyGlass Base product:

Command	Description
<a href="#"><i>get_combloop</i></a>	Creates a list of collection of combinational loop in the current design that match certain criteria

## get\_combloop

**Creates a list of combinational loop in the current design that match certain criteria**

### Syntax

```
get_combloop [-from <from_pattern>]
```

### Scope

Goal

### Return Value

Returns an empty string or list of combinational loops in a design. The empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped using the *catch* command.

### Description

The *get\_combloop* command creates a list of combinational loop in a design that match certain criteria. To get the list of combinational loop in a design, the user has to use the *CombLoop* rule of the *openmore* policy.

This command fails in case of the situations if:

- Flattened design view does not exist
- No data is available from the CombLoop rule

### Arguments

None

### Examples

```
sg_shell> get_combloop -from top.q_syncl //returns list of  
collection of combinational loop in which net is present  
which passed through -from  
sg_shell> get_combloop //returns list of collection of all  
combinational loop in the design
```

---

Debug Commands

## See Also

[\*get\\_nets\*](#)

## SpyGlass Lint Turbo Commands

The following table describes the Tcl commands that are a part of the SpyGlass Lint Turbo product:

<b>Command</b>	<b>Description</b>
<i><a href="#">get_lint_formal_results</a></i>	Gets a list of violations of lint turbo rules in the current design
<i><a href="#">report_lint_formal_results</a></i>	Reports properties of violations of lint turbo rules in the current design

## get\_lint\_formal\_results

**Gets a list of violations of lint turbo rules in the current design**

### Syntax

```
get_lint_formal_results
  [-rules <formal-Lint-rule-names >]
  [-modules <List-of-module>]
  [ -filter expression]
```

### Scope

Project

### Return Value

Returns a collection of violations of lint turbo rules in the current design. This collection can be used as an input to other commands, such as the *report\_lint\_formal\_results* command. If no violations are found, an empty collection is returned.

### Description

The *get\_lint\_formal\_results* command creates a collection of violations of lint turbo rules in the current design.

### Arguments

#### <-rules>

Use this argument to specify the rules for which violations need to be reported.

Possible options are *AV\_WIDTH\_MISMATCH\_ASSIGN*, *AV\_WIDTH\_MISMATCH\_EXPR*, *AV\_WIDTH\_MISMATCH\_EXPR02*, *AV\_WIDTH\_MISMATCH\_EXPR03*, *AV\_DONTCARE\_MISMATCH*, *AV\_CASE\_DEFAULT\_MISSING*, *AV\_WIDTH\_MISMATCH\_FUNCTION*, *AV\_SIGNED\_UNSIGNED\_MISMATCH*, *AV\_WIDTH\_MISMATCH\_CASE*, *AV\_WIDTH\_MISMATCH\_PORT*, and *AV\_CASE\_DEFAULT\_REDUNDANT*.

#### [-modules]

Use this argument to specify the modules in which violations need to

debug.

### **[-filter expression]**

Use this argument to specify a filter expression over the property status attributes of formal properties such as PROVED. Possible options are *NOT\_ANALYZED*, *PROVED*, *FAILED*, *CONS\_UNSAT*, *ANALYZED*, *INT\_ERR*.

Use following format for the expression.

```
[av_property_status==NOT_ANALYZED]
```

## **Examples**

### **Example 1**

In the following example, the *get\_lint\_formal\_results* command returns all the violations for lint turbo rules in the design.

```
sg_shell> get_lint_formal_results
```

### **Example 2**

In the following example, the command returns all the violations for lint turbo rules in the design unit (module) 'sub'.

```
sg_shell> get_lint_formal_results -modules sub
```

### **Example 3**

In the following example, the command returns all the violations for the lint turbo *Av\_width\_mismatch\_assign* rule in the design.

```
sg_shell> get_lint_formal_results -rules  
AV_WIDTH_MISMATCH_ASSIGN
```

### **Example 4**

In the following example, the command returns all the violations for lint turbo rules in the design with formal status is "*proved*".

```
sg_shell> get_lint_formal_results -filter  
av_property_status==PROVED
```



---

Debug Commands

## See Also

None

## report\_lint\_formal\_results

**Reports properties of violations of lint turbo rules in the current design**

### Syntax

```
report_lint_formal_results  
  [<Lint turbo rule violations>]
```

### Scope

Project

### Return Value

None

### Description

The *report\_lint\_formal\_results* reports the properties of lint turbo rules in the current design.

**NOTE:** *The report\_lint\_formal\_results command fails if an improper collection of objects, which does not contain lint turbo rule violations returned by the get\_lint\_formal\_results command, has been provided as an input to the report\_lint\_formal\_results command.*

### Arguments

<Lint turbo rule violations>

Collection used in the *report\_lint\_formal\_results* command is a set of violations returned by the *get\_lint\_formal\_results* command.

### Examples

#### Example 1

In the following example, the *report\_lint\_formal\_results* command reports all the violations for lint turbo rules in the design.

```
sg_shell> report_lint_formal_results  
[get_lint_formal_results ]
```

## Example 2

In the following example, the command reports all the violations for lint turbo rules in the design unit (module) 'sub'.

```
sg_shell> report_lint_formal_results  
[get_lint_formal_results -modules sub]
```

## SpyGlass Constraints Commands

The following table describes the various Tcl commands that are a part of the SpyGlass Constraints product:

<b>Command</b>	<b>Description</b>
<i>autofix_sdc</i>	Generates SDC file having list of missing/incorrect constraints specified through the constraint rules
<i>get_constrained_muxes</i>	Used to get a list of MUXes where select pins are unconstrained and data pins are constrained
<i>get_sdc</i>	Used to get a list of SDC commands on the basis of filtering criteria, if specified
<i>write_sdc_node</i>	Used to print the SDC constraints for the given SDC nodes
<i>export_sdc</i>	Used to export the SDC from user updated CSV
<i>update_crossing_file</i>	Translates the crossing file into user csv

## autofix\_sdc

Generates SDC file having list of missing/incorrect constraints specified through the constraint rules

### Syntax

```
autofix_sdc <rules> <-f>
```

### Scope

Project

### Return Value

Returns a string containing name of SDC file that has been generated.

### Description

The *autofix\_sdc* command is used to generate SDC file having a list of missing/incorrect constraints specified through the constraint rules.

### Arguments

**<rules>**

Use this argument to specify a comma separated list of rule names.

**<-f>**

(Optional) Use this argument to forcibly generate the SDC file. If an SDC file already exists, then use this argument to overwrite the existing file.

### Examples

```
sg_shell> autofix_sdc
autofix_sdc: error: Please specify some rules
sg_shell> autofix_sdc Inp_Del01a
out/test/test_goal/spyglass_reports/constraints
autofix_sdc.sdc is successfully generated
```

## See Also

None

## get\_constrained\_muxes

Used to get a list of MUXes where select pins are unconstrained and data pins are constrained

### Syntax

```
get_constrained_muxes
```

### Scope

Project

### Return Value

Returns an empty string or a list of MUXes in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_constrained\_muxes* command is used to get a list of MUXes in the design, where select pins are unconstrained and data pins are constrained.

### Arguments

None

### Examples

```
sg_shell> get_constrained_muxes  
top.m1 top.m2
```

### See Also

[filter\\_collection](#)

## get\_sdc

Used to get a list of SDC commands on the basis of filtering criteria, if specified

### Syntax

```
get_sdc
  <constr_name | -of_objects objects>
  [-filter expression]
  [-regexp | -exact]
```

### Scope

Project

### Return Value

Returns an empty string or a collection of SDC commands in case of successful execution. An empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *get\_sdc* command is used to get a list of user-specified SDC commands on the basis of filtering criteria, if specified.

### Arguments

The *get\_sdc* command has the following arguments:

#### <constr\_name>

Use this argument to specify the constraint name to get SDC commands. Otherwise, SDC commands of all constraints are reported on the basis of filtering criteria, if any.

#### <-of\_objects objects>

Use this argument to create a collection of SDC commands defined on the specified objects. The objects that can be specified are pin names, collection of pins, ports, instances, or nets.

**NOTE:** *The constr\_name and -of\_objects arguments are mutually exclusive.*



**[-filter expression]**

Use this argument to filter a group with an expression. The supported criterion in filter for this command is `sdc_type`. The following examples illustrate the usage of this argument:

```
-filter sdc_type==create_clock
-filter sdc_type==set_input_delay
```

**[-exact]**

Use this argument to consider wildcard characters as plain characters for constraint names.

**Examples****Example 1: Without -filter**

```
sg_shell> get_sdc
_sggrp6
```

**Example 2: With -filter**

```
sg_shell> get_sdc -of_objects <objects> -filter
sdc_type==create_clock
_sggrp7
```

**Example 3: get\_sdc usage with write\_sdcnode**

```
sg_shell> set sdcs [get_sdc -of_objects <objects> -filter
sdc_type==create_clock]
_sggrp9
sg_shell> foreach_in_collection node $sdcs { write_sdc_node
$node }
create_clock -name clk1 -period 10 -waveform { 0 5 }
[get_ports clk1]
create_clock -name clk2 -period 12 -waveform { 0 6 }
[get_ports clk2]
```

**See Also**

[write\\_sdc\\_node](#), [sdc\\_type](#)

## write\_sdc\_node

Used to print the SDC constraints for the given SDC nodes

### Syntax

```
write_sdc_node <sdc_collection>
```

### Scope

Project

### Return Value

Returns an empty string or a string containing printed SDC constraints corresponding to the SDC nodes. In case of unsuccessful execution, an error is returned that can be trapped by using the *catch* command.

### Description

The *write\_sdc\_node* command prints the SDC constraint supplied by the user as an SDC node or a collection of SDC nodes.

### Arguments

The *write\_sdc\_node* command has the following argument:

#### <sdc\_collection>

Use this argument to specify a collection of SDC nodes. It can be obtained by using the [get\\_sdc](#) command.

### Examples

```
sg_shell> set clocks [get_sdc create_clock]
sg_shell> write_sdc_node $clocks
create_clock -name clk1 -period 10 -waveform { 0 5 }
[get_ports clk1]
create_clock -name clk2 -period 12 -waveform { 0 6 }
[get_ports clk2]
```

### See Also

[get\\_sdc](#)

## export\_sdc

Generates the SDC file from the user updated CSV

### Syntax

```
export_sdc <out_dir> <-f>
```

### Scope

Project

### Return Value

Returns a string containing the names of the CSV files that have been exported to the SDC along with the file path.

### Description

The *export\_sdc* command generates the actual SDC files from the updated CSV files that were generated after running the `SDC_GenerateIncr` rule. The command reports an error if no CSV files are present or the `SDC_GenerateIncr` rule was not executed before running this command.

### Arguments

The *export\_sdc* command has the following argument:

#### <out\_dir>

Use this argument to specify the path where the SDC file should be generated.

**NOTE:** *By default, the SDC file is generated in the `spyglass_reports/constraints/exportSDCGen` directory. To specify a different directory, make sure you have created the directory in which you want the SDC files to be generated. In addition, ensure you have write permissions to the directory.*

#### <-f>

(Optional) Use this argument to forcibly generate the SDC files into the user-specified directory, using `<out_dir>` argument. If an SDC file is already present in the user-specified directory, then this option can be used

to forcibly generate or overwrite the existing file.

## Examples

```
sg_shell> export_sdc
```

```
Block1.csv is successfully translated to ./test/test_goal
```

```
spyglass_reports/constraints/exportSDCgen/Block1.sdc
```

```
top.csv is successfully translated to ./test/test_goal
```

```
spyglass_reports/constraints/exportSDCgen/top.sdc
```

```
sg_shell> export_sdc sdc_files
```

```
Block1.csv is successfully translated to ./sdc_files
```

```
Block1.sdc
```

```
top.csv is successfully translated to ./sdc_files/top.sdc
```

## See Also

[\*update\\_crossing\\_file\*](#)

## update\_crossing\_file

Translates the crossing file into user csv

### Syntax

```
update_crossing_file
```

### Scope

Project

### Return Value

Returns a string showing the number of crossing pairs that have been translated to CSV along with the file path.

### Description

The *update\_crossing\_file* command is used to translate the crossing file to user CSV file.

### Examples

```
sg_shell> update_crossing_file  
1 pair(s) as classified in crossing file ./test/test_goal/  
spyglass_reports/constraints/top-crossings.csv have been  
updated successfully
```

### See Also

[export\\_sdc](#)

## SpyGlass CDC Commands

The following table describes the various Tcl commands that are a part of the SpyGlass CDC product. You can also refer to the [Appendix E: CDC Application Commands](#) section.

Command	Description
<i>get_cdc</i>	Creates a list of clock domain crossings in the current design that match certain criteria
<i>get_cdc_coherency</i>	Returns the collection of Ac_conv issue based in field values
<i>get_cdc_glitch</i>	Creates collection of clock domain crossings in current design that may have glitches and match certain criteria
<i>get_cdc_sources</i>	Returns source of a crossing given by a destination name or an object returned from get_cdc collection
<i>get_conv_sync_signals</i>	Returns a collection of crossings for a given convergence object that is an element of collection returned from get_cdc_coherency
<i>get_glitch_sources</i>	Returns a collection of sources for a given glitch prone crossing or a destination
<i>get_multi_flop_sync_info</i>	Returns a collection of synchronizer flip-flops for a given crossing
<i>get_paths</i>	Reports the complete paths between the specified start and end points
<i>get_reset_sync</i>	Returns status of flip-flops with the reset synchronization issues in the current design that match certain criteria
<i>get_reset_sync_names</i>	Return the reset synchronizer information in the current design that match certain criteria
<i>report_cdc</i>	Reports clock domain crossing details
<i>report_cdc_coherency</i>	Displays the collection of coherency/convergence issues reported by get_cdc_coherency
<i>report_cdc_glitch</i>	Reports clock domain crossing with glitches
<i>report_paths</i>	Reports elements in a defined path in current design
<i>report_reset_sync</i>	Reports reset synchronization issues related information
<i>report_reset_sync_names</i>	Reports reset synchronization related information

## get\_cdc

**Creates a collection of clock domain crossings in the current design that match certain criteria**

### Syntax

```
get_cdc
    [-from <from_pattern>]
    [-to <to_pattern>]
    [-from_clocks <f_clock>]
    [ -to_clocks <t_clock>]
    [-from_domains <f_domain> ]
    [-to_domains <t_domain>]
    [-from_objects <f_object>]
    [-to_objects <t_object>]
    [-of_objects <obj>]
    [-regexp]
    [-filter <expr>]
    [-dump_crossing_path]
    [-crossing_path_limit <value>]
    [-disable_flop_chain]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of clock domain crossing objects in case of successful execution. An empty string is returned if nothing matches the filtering criterion. In case of unsuccessful execution, an error is returned which can be trapped using the **catch** command.

### Description

Creates a collection of clock domain crossings in the current design or instance that matches a certain criteria. The command returns a collection, if any crossing matches the various input specifications and also passes the filtering criteria, if specified. If no objects match the criteria, an empty string is returned.

The `get_cdc` command fails when:

- Flattened design view does not exist.
- No data is available from `Ac_sync_group` rules.

## Arguments

The `get_cdc` command has the following arguments:

**< -from >**

Use this argument to specify net names, pin names, or port names of source object in crossing.

**< -to >**

Use this argument to specify net names, pin names, or port names of destination object in crossing.

**[ -from\_clocks ]**

Use this argument to specify name of the source clock in the `sgdc/sdc` file. Clock name can be port name, pin name, net name, or clock tag name.

**[ -to\_clocks ]**

Use this argument to specify name of the destination clock specified in `sgdc/sdc` file. Clock name can be port name, pin name, net name, or clock tag name.

**[ -from\_domains ]**

Use this argument to specify name of the source domain specified in `sgdc` or `sdc` file.

**[ -to\_domains ]**

Use this argument to specify the name of the destination in the `sgdc/sdc` file.

**[ -from\_objects ]**

Use this argument to specify collection of source clock objects or collection of domain objects or collection of net objects.



## Debug Commands

**[ -to\_objects ]**

Use this argument to specify collection of destination clock objects or collection of domain objects or collection of net objects.

**[ -of\_objects ]**

Collection of clock domain crossings.

**[-filter <filter\_expression>]**

Use this argument to specify filter expression over named attributes of crossings such as 'is\_synchronized', 'is\_data', 'num\_sources', 'num\_source\_domains', 'sync\_method', 'failure\_reason', 'src\_type', 'des\_type'.

Following table describes the filter attributes available for this argument:

<b>Filter Attribute</b>	<b>Description</b>
is_synchronized	(Boolean) True if the crossing is synchronized, false otherwise
is_data	(Boolean) True if the crossing is a data crossing
num_sources	(String) The number of sources in a crossing
num_source_domains	(String) The number of source domains in a crossing
sync_method	(String) The synchronization method as reported by Ac_sync01 and Ac_sync02 rules. For details, refer to the Reasons for Synchronized Crossings section of the SpyGlass CDC Rules Reference Guide. In the reason string, replace the space with '_' to use the string as a filter value. For example, specify 'Conventional Multi-Flop Synchronization Scheme' as 'Conventional_Multi-Flop_Synchronization_Scheme'

<b>Filter Attribute</b>	<b>Description</b>
failure_reason	(String) Reason the crossing is unsynchronized as reported by Ac_unsync01 and Ac_unsync02 rules. For details, refer to the Reasons for Unsynchronized Crossings section of the SpyGlass CDC Rules Reference Guide. In the reason string, replace the space with '_' to use the string as filter value. For example specify 'Qualifier not found' as 'Qualifier_not_found'
src_type	(String) The type of the source as reported by Ac_sync_group rules. It can have values - flop, library-cell, latch, primary-input, black-box.
dest_type	(String) The type of the destination as reported by Ac_sync_group rules. It can have values - flop, library-cell, latch, primary-output, black-box.
cdc_rule_name	(String) Name of the rule which generate the violation
dest_name	(String) The output net name of the destination of a crossing
dest_clocks	(String) Name of clocks which drive the destination. In case of multiple clocks, all clock names are shown.
dest_domain	(String) The clock domain of destination. If the domain name is not specified, the clock domain name is the same as the clock name. If destination is driven by multiple clocks, all the clock domains are shown.
dest_internal_domain_id	(Integer) A unique tag number generated for the destination clock domain.
dest_clock_tag	(String) the tag defined in sgdc file for destination clock. This attribute returns a value if a tag has been defined for at least one clock. Else returns and empty string. Clocks for which tags have not been defined will receive internally generated tags. In case of multiple clocks, all tags are shown.
dest_file_line	(String) Design file name and line number where the destination is used. File name and line number is separated by a colon.

## Debug Commands

<b>Filter Attribute</b>	<b>Description</b>
overall_failure_reason	(String) Overall reason for unsynchronized crossing
overall_synch_scheme	(String) Name of the synchronization method used to synchronize a crossing
dest_parent_inst_name	(String) Name of the parent instance of the destination
crossing_module_name	(String) Name of the crossing module
potential_qualifier_name	(String) Name of the potential qualifier
multi_flop_synchronizer_names	(String) Names of flip flops which are used to synchronize the crossing
multi_flop_synchronizer_stages	(Integer) Number of flip-flops used to synchronize the crossing
dest_module_name	(String) Module name of the destination of the crossing
dest_objects	(Collection) Destination net objects of the crossing
multi_flop_synchronizer_objects	(Collection) Module name of the destination of the crossing

**[-regexp]**

If '-regexp' switch is given then patterns are seen as real regular expressions rather than simple wildcard patterns. This switch is not needed for wildcard pattern matching. Wildcard pattern matching is on by default. Regular expression can only be specified for design objects names in fields '-from', '-to', '-from\_clock', '-to\_clock'.

**[-dump\_crossing\_path]**

When the `-dump_crossing_path` is specified, the crossing paths are computed for the specified crossings and reported by the subsequent `report_cdc` commands.

**[-crossing\_path\_limit <value>]**

This argument accepts an integer value to limit the number of crossings

that will show the crossing paths. This argument accepts values from 0 to any number. By default it shows all crossing. If the value is 0, no crossings are shown.

### **[-disable\_flop\_chain]**

Disables populating chain flops (shift registers) beyond the synchronized point.

## **Examples**

```
sg_shell> get_cdc -from top.q_sync1 //returns collection of crossings having source as 'top.q_sync1'  
_sggrp4
```

```
sg_shell> get_cdc -to_clock top.clk2 //returns collection of crossings having destination clock as 'top.clk2'  
_sggrp5
```

```
sg_shell> get_cdc -filter is_synchronized==false //returns collection of all unsynchronized crossings  
_sggrp6
```

```
sg_shell> get_cdc -to_object [get_clocks tag4] //return collection of crossing having destination clock tag as 'tag4'  
_sggrp10
```

```
sg_shell> get_cdc -from q_sync1.* -regexp //return collection of crossings having source matching regular expression 'q_sync1.*'  
_sggrp1
```

```
sg_shell> get_cdc -filter sync_method==Conventional_multi-flop_for_metastability_technique //return collection of crossings having synchronization method as specified  
_sggrp10
```

```
sg_shell> get_cdc -from top.FF1 -to top.FF2 -dump_crossing_path
```

---

Debug Commands

## See Also

[report\\_cdc](#), [get\\_clocks](#), [get\\_domains](#)

## get\_cdc\_coherency

Returns the collection of Ac\_conv issue based in field values

### Syntax

```
get_cdc_coherency
    [-from <from_pattern>]
    [-to <to_pattern>]
    [-from_clock <f_clock>]
    [ -to_clock <t_clock>]
    [-from_domain <f_domain> ]
    [-to_domain <t_domain>]
    [-from_object <f_object>]
    [-to_object <t_object>]
    [-of_object <obj>]
    [-regexp]
    [-filter <expr> ]
    [-num_convergences <num_conv_paths> ]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of coherency issues in case of successful execution. The empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution an error is returned.

### Description

This command is used to return the collection of Ac\_conv issue based in some field values. The collection returned can also be filtered on the basis of some attributes.

**NOTE:** *Each Ac\_conv violation/convergence issue has multiple crossings involved. Match with a crossing's attribute is sufficient for it to be returned as part of the output collection.*

### Arguments

The get\_cdc\_coherency command has the following arguments:

## Debug Commands

**-from**

Use this argument to get the list of net name, pin name, and port name of crossings source.

**-to**

Use this argument to get the list of net name, pin name, and port name of crossings destination.

**-from\_clock**

Use this argument to get the list of source clocks specified in the sgdc/sdc file. Source clock is the clock of crossing's source.

**-to\_clock**

Use this argument get the list of destination clocks specified in the sgdc/sdc file. Destination clock means the clock of crossing's destination.

**-from\_domain**

Use this argument to get the list of source domains specified in the sgdc/sdc file. Source domain means the domain of crossing's source.

**-to\_domain**

Use this argument to get the list of destination domains specified in the sgdc/sdc file. Destination domain means the domain of crossing's destination.

**-from\_object**

Use this argument to get the list of source clock object, domain objects, net objects, pin objects that are returned from other tcl commands, for example, `get_nets`, `get_pins`, and `get_clocks`.

**-to\_object**

Use this argument to get the list of destination clock object, domain objects, net objects, pin objects that are returned from other tcl commands, for example, `get_nets`, `get_pins`, and `get_clocks`.

**-of\_object**

Use this argument to get a collection of objects returned by the tcl command `get_cdc` or `get_cdc_coherency`.

**-regexp**

Use this argument to indicate that all the arguments are regular expressions.

**-filter**

Following table describes the filter attributes available for this argument:

Filter Attribute	Description
<code>is_graycoded</code>	(String) Yes, no, unknown or disabled to return convergences with functional check as PASSED, FAILED, Partially-proved or DISABLED respectively
<code>is_comb_conv</code>	(Boolean) True if the conv is in <code>Ac_conv02</code>
<code>is_seq_conv</code>	(Boolean) True if the conv is in <code>Ac_conv01</code>
<code>is_nonconv_bus</code>	(Boolean) True if the conv is in <code>Ac_conv04</code>
<code>is_user_defined</code>	(Boolean) True if the conv is due to user-defined gray-coding constraint reported in <code>Ac_conv05</code>
<code>num_sources</code>	(Integer) The number of converging signals
<code>num_source_domains</code>	(Integer) The number of source domains in a crossing (will be more than 1 for <code>Ac_conv03</code> )
<code>cdc_rule_name</code>	(String) Name of rule which generates the violation
<code>conv_gate</code>	(String) Output net name of gate on which signals converge. Returns N.A. for <code>Ac_conv04</code> and <code>Ac_conv05</code> .
<code>sync_count</code>	(String) Number of signals converge
<code>status</code>	(String) Result of the gray encoding check. Returns N.A. for <code>Ac_conv01</code> , <code>Ac_conv02</code> and <code>Ac_conv03</code> .

**-num\_convergences**

Use this argument to specify the number of convergences to be reported



for the same set of synchronizers. The default value is 1. Specify -1 to report all the computed parallel convergences. The value specified for the `num_convergences` parameter should be less than or equal to the value specified to the `compute_num_convergences` parameter.

Note that parallel convergences are not checked for gray encoding and therefore the status is shown as Not-Analyzed.

**NOTE:** *To use this argument, run the `Ac_conv01`, `Ac_conv02`, or `Ac_conv03` rule with the `compute_num_convergences <num>` parameter. The number specified with this parameter is the maximum number of convergences you can query with the TCL command.*

## Examples

Consider the following commands:

```
sg_shell> get_cdc_coherency //return collection of all
coherency issues
_sggrp3
```

```
sg_shell> get_cdc_coherency -from q12 //returns collection of
coherency issues where crossing source matches 'q12'
_sggrp4
```

```
sg_shell> get_cdc_coherency -from q12_typo //error case
get_cdc_coherency: error: Please specify valid net or pin
with '-from' field
```

```
sg_shell> get_cdc_coherency -filter is_comb_conv==false //
return crossing which are of type other than Ac_conv02
_sggrp5
```

```
sg_shell> get_cdc_coherency -filter is_seq_conv==true //
return coherency issue of type Ac_conv01
_sggrp6
```

```
sg_shell> get_cdc_coherency -filter is_user_defined==true //  
return coherency issue of type Ac_conv05  
_sggrp7
```

```
sg_shell> get_cdc_coherency -num_convergences=3 //calculates  
three parallel convergences
```

```
sg_shell> get_cdc_coherency -num_convergences='-1' //  
calculates all possible parallel convergences
```

## See Also

[report\\_cdc](#), [report\\_cdc\\_coherency](#)

## get\_cdc\_glitch

**Creates collection of clock domain crossings in current design that may have glitches and match certain criteria**

### Syntax

```
get_cdc_glitch
  [-to <to_pattern>]
  [ -to_clocks <t_clock>]
  [-to_domains <t_domain>]
  [-to_objects <t_object>]
  [-of_objects <obj>]
  [-regexp]
  [-filter <expr> ]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of clock domain crossing objects that may have glitches, in case of successful execution. The empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution an error is returned that can be trapped using the **catch** command.

### Description

This command creates a collection of clock domain crossings in the current design/instance that may have glitches and match certain criteria. The command returns a collection, if any crossing with glitches matches the various input specifications and also passes the filtering criteria, if specified. If no objects match the criteria, the empty string is returned.

The *get\_cdc\_glitch* command fails when:

- Flattened design view does not exist.
- No data is available from the *Ac\_glitch03* rule.

## Arguments

The `get_cdc_glitch` command has the following arguments:

### `<-to>`

Use this argument to specify net names, pin names, or port names of destination object in crossing.

### `[-to_clocks]`

Use this argument to specify the name of the destination clock in the `sgdc/sdc` file. Clock name can be port name, pin name, net name, or clock tag name.

### `[-to_domains]`

Use this argument to specify the domain of the destination in the `sgdc/sdc` file.

### `[-to_objects]`

Use this argument to specify collection of destination clock objects using `get_clocks` or collection of domain objects using `get_domains` or collection of net objects using `get_nets`.

### `[-of_objects]`

Collection of clock domain crossings returned by the `get_cdc_glitch` or `get_cdc` command.

### `[-regexp]`

If `'-regexp'` switch is given then patterns are seen as real regular expressions rather than simple wildcard patterns. This switch is not needed for wildcard pattern matching. Wildcard pattern matching is on by default. Regular expression can only be specified for design objects names in fields `'-from'`, `'-to'`, `'-from_clocks'`, and `'-to_clocks'`.

### `[-filter <filter_expression>]`

Use this argument to specify filter expression over named attributes of crossings such as `'multi_source_glitch_check'`, `'has_reconvergent_sources'`, `'has_multi_domain_sources'`,

'has\_destination\_domain', 'num\_sources', and 'num\_source\_domains.'

Following table describes the filter attributes available for this argument:

Filter Attribute	Description
multi_source_glitch_check	(String) Status of grey-encoding check performed on the same domain sources. It can have "pass", "fail" or "unknown" values.
has_reconvergent_sources	(Boolean) True if the crossing has a reconvergent source
has_multi_domain_sources	(Boolean) True if the crossing has multiple domain sources
has_destination_domain	(Boolean) True if the crossing has a destination domain signal in fanin
num_sources	(String) The number of sources in a crossing
num_source_domains	(String) The number of source domains in a crossing
cdc_rule_name	(String) Name of rule which generates the violation
dest_name	(String) Output net name of the destination
dest_clocks	(String) Names of clocks that drive the destination. In case of multiple clocks, all clock names are shown.
dest_domain	(String) Clock domain of destination. If the domain name is not specified, the clock domain name is the same as the clock name. If destination is driven by multiple clocks, all the clock domains are shown.
dest_internal_domain_id	(String) Unique tag number generated for the destination clock domain
dest_clock_tag	(String) Tag name specified in the SGDC file for clock net connected to destination. This attribute returns a value if a tag has been defined for at least one clock. Else, returns an empty string. Clocks for which tags have not been defined will receive internally generated tags. In case of multiple clocks, all tags are shown.
status	(String) Result of gray encoding check
failure_reason	(String) Reason for the status

## Examples

Consider the following commands:

```
sg_shell> get_cdc_glitch -to_clocks top.clk2 //returns  
collection of glitchy crossings having destination clock as  
'top.clk2'  
_sggrp5
```

```
sg_shell> get_cdc_glitch -filter  
has_reconvergent_sources==false //returns collection of  
those glitchy crossings which have reconvergent sources  
_sggrp6
```

```
sg_shell> get_cdc_glitch -to_objects [get_clocks tag4] //  
returns collection of crossing having destination clock tag  
as 'tag4'  
_sggrp10
```

```
sg_shell> sg_shell> get_cdc_glitch -to q_syncl.* -regexp //  
return collection of crossings having destination matching  
regular expression 'q_syncl.*'  
_sggrp1
```

## See Also

[report\\_cdc\\_glitch](#), [get\\_clocks](#), [get\\_domains](#), [get\\_cdc](#)

## get\_cdc\_sources

Returns source of a crossing given by a destination name or an object returned from `get_cdc` collection

### Syntax

```
get_cdc_sources
    [<destination_name>]
    [-of_objects <cdc-object>]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of clock domain crossing's source objects in case of successful execution.

### Description

This command gives source list of a clock domain crossing in the current design/instance that matches certain criteria.

The `get_cdc_sources` command fails when:

- Flattened design view does not exist.
- No data is available from the `Ac_sysnc_group` rules.

### Arguments

The `get_cdc_sources` command has the following arguments:

#### <-destination\_name>

Use this argument to specify net names, pin names or port names of a clock domain crossings object.

#### <-of\_objects>

Use this argument to specify the clock domain crossing object list of filter attributes.

**List of filter attributes:**

Filter Attribute	Description
source_name	(String) Source name
source_clocks	(String) Source clocks name
source_domain	(String) Source domain
source_type	(String) The type of the source as reported by the Ac_sync_group rules. It can have values - flop, library-cell, latch, primary-input, and black-box.
source_file_line	(String) RTL file and line of the source definition
source_parent_inst_name	(String) Parent instance name
source_internal_domain_id	(Integer) Internal domain id of the source
source_clock_tag	(String) User-defined tag name of the clock
source_failure_reason	(String) Reason the crossing is unsynchronized as reported by the Ac_unsync01 and Ac_unsync02 rules. Refer section Reasons for <i>Unsynchronized Crossings</i> in the <i>SpyGlass CDC Rules Reference Guide</i> . In the reason string, space needs to be replaced with '_' to use the string as filter value. For example: 'Qualifier not found' need to be specified as 'Qualifier_not_found'.
source_synch_scheme	(String) Sync scheme name
source_qualifier_name	(String) Name of the qualifier
source_qualifier_depth	(String) Depth of the qualifier

**Examples**

Consider the following commands:

```
sg_shell> set cdc_vars [eval [concat get_cdc]];
          foreach_in_collection var $cdc_vars {
            set src_vars [get_cdc_sources -of_objects
                          $var];
          }
```

**See Also**

[report\\_cdc](#), [get\\_clocks](#), [get\\_domains](#)



## get\_conv\_sync\_signals

Returns a collection of crossings for a given convergence object that is an element of collection returned from `get_cdc_coherency`

### Syntax

```
get_conv_sync_signals  
    [<converegnce>]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of clock domain crossing objects in case of successful execution.

### Description

The `get_conv_sync_signals` command returns a collection of crossings for a given convergence object which is an element of collection returned from the [get\\_cdc\\_coherency](#) command.

The `get_conv_sync_signals` command fails when:

- Flattened design view does not exist.
- No data is available from the *Ac\_conv* rules.

### Arguments

The `get_conv_sync_signals` command has the following arguments:

#### <converegnce>

Use this argument to specify an element of collection returned from the `get_cdc_coherency` comamnd.

**List of filter attributes:**

<b>Filter Attribute</b>	<b>Description</b>
dest_name	(string) Output net name of destination
source_names	(string) Output net names of sources
seq_depth	(Integer) Number of sequential elements from destination to the convergent object
diverging_nets	(String) Output net names of sources which diverge and then converge
dest_clocks	(String) Names of clocks which drive the destination
dest_clock_tag	(String) Tag name specified in the SGDC file for clock net connected to destination. This attribute returns a value if a tag value has been defined for at least one clock. Else, returns an empty string. Clocks for which tags have not been defined will receive internally generated tags. In case of multiple clocks, all tags are shown.
source_clocks	(String) Names of clocks which drive the source
source_clock_tag	(String) Tag name specified in the SGDC file for clock net connected to source. This attribute returns a value if a tag value has been defined for at least one clock. Else, returns an empty string. Clocks for which tags have not been defined will receive internally generated tags. In case of multiple clocks, all tags are shown.
dest_file_line	(String) Design file name and line number where the destination is used. File name and line number are separated by a colon.

**Examples**

Consider the following commands:

```
sg_shell> set cdc_vars [eval [concat get_cdc_coherency]];
           foreach_in_collection var $cdc_vars {
               set src_vars [get_conv_sync_signals $var];
           }
```

**See Also**

[get\\_cdc\\_coherency](#), [report\\_cdc\\_coherency](#), [get\\_cdc](#)

## get\_glitch\_sources

Returns a collection of sources for a given glitch prone crossing or a destination

### Syntax

```
get_glitch_sources  
    [<destination_name>]  
    [-of_objects <cdc-object>]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of source objects in case of successful execution.

### Description

The *get\_glitch\_sources* command returns a collection of sources for a given glitch prone crossing or a destination. The crossing object should be an element of collection returned from the [get\\_cdc\\_glitch](#) command.

The *get\_glitch\_sources* command fails when:

- Flattened design view does not exist.
- No data is available from the *Ac\_glitch03* rule.

### Arguments

The *get\_glitch\_sources* command has the following arguments:

#### <destination\_name>

Use this argument to specify net names, pin names or port names of a destination of a glitch prone crossing.

#### <-of\_objects>

Use this argument to specify collection of elements returned from the *get\_cdc\_glitch* command.

**List of filter attributes:**

Filter Attribute	Description
source_name	(String) Output net name of source
source_type	(String) Source type as flop, latch, library-cell, port, or black box
source_clocks	(String) Names of clocks which drive the source. In case of multiple clocks, all clock names are shown.
source_clock_tag	(String) Tag name specified in the SGDC file for clock net connected to source. This attribute returns a value if a tag value has been defined for at least one clock. Else, returns an empty string. Clocks for which tags have not been defined will receive internally generated tags. In case of multiple clocks, all tags are shown.
source_domain	(String) Clock domain of source. If the domain name is not specified, the clock domain name is the same as the clock name. If source is driven by multiple clocks, all the clock domains are shown.
source_internal_domain_id	(Integer) Unique tag number generated for the source clock domain
is_reconv	(Boolean) True if source reaches its destination through multiple paths
is_async	(Boolean) True if source is not synchronized with its the destination

**Examples**

Consider the following commands:

```
sg_shell> set cdc_vars [eval [concat get_cdc_glitch]];
             foreach_in_collection var $cdc_vars {
                 set src_vars [get_glitch_sources $var];
             }
```

**See Also**

[get\\_cdc\\_glitch](#), [report\\_cdc\\_glitch](#), [get\\_cdc](#)

## get\_multi\_flop\_sync\_info

Returns a collection of synchronizer flip-flops for a given crossing

### Syntax

```
get_multi_flop_sync_info [<cdc-object>]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of synchronizer flops in case of successful execution.

### Description

The *get\_multi\_flop\_sync\_info* command returns a collection of synchronizer flip flops for a given crossing, which is synchronized by multiple flip-flop synchronization scheme. The collection is considered empty if the crossing is unsynchronized or synchronized by other method.

The *get\_multi\_flop\_sync\_info* command fails when:

- Flattened design view does not exist.
- No data is available from the *Ac\_sync* rule group.

### Arguments

The *get\_multi\_flop\_sync\_info* command has the following arguments:

#### <cdc-object>

Use this argument to specify the element of collection returned by the *get\_cdc* command.

#### List of filter attributes:

Filter Attribute	Description
sync_name	(String) Output net name of the synchronizer flip-flop
sync_module_name	(String) Module name of the synchronizer flip-flop

## Examples

Consider the following commands:

```
sg_shell> set cdc_vars [eval [concat get_cdc]];
             foreach_in_collection var $cdc_vars {
               set flops [get_multi_flop_sync_info $var];
               foreach_in_collection sync $flops {
                 set syncname [get_attribute $sync
                 sync_name];
               }
             }
```

## See Also

[report\\_cdc](#), [get\\_cdc\\_sources](#), [get\\_cdc](#)

## get\_reset\_sync

Returns status of flip-flops with the reset synchronization issues in the current design that match certain criteria

### Syntax

```
get_reset_sync [<reset>] [-reset_objects <rst_obj>] [-to_objects <to_objects>] [-of_objects <obj>] [-to <to_flop_net>] [-resets <resets>] [-regexp] [-filter <expr>]
```

### Scope

Goal

### Return Value

Returns an empty string or a collection of reset flip-flops in case of successful execution. The empty string is returned if nothing matched the filtering criterion. In case of unsuccessful execution, an error is returned that can be identified using the `catch` command.

### Description

The `get_reset_sync` command creates a collection of reset violation objects in the current design/instance that match certain criteria. The command returns a collection of resets related violations reported by the `Ar_sync01`, `Ar_unsync01`, `Ar_asyncdeassert01`, and `Ar_syncdeassert01` rules and also passes the filtering criteria (if specified). If no objects match the criteria, the empty string is returned.

The `get_reset_sync` command fails, if:

- Flattened design view does not exist.
- No data is available from the `Ar_sync01`, `Ar_unsync01`, `Ar_asyncdeassert01`, `Ar_syncdeassert01` rules.

### Arguments

The `get_reset_sync` command has the following arguments:

**<reset>**

Collection of resets returned by the `get_resets` command.



## Debug Commands

**-reset\_objects <rst\_obj>**

Collection of resets returned by the *get\_resets* command.

**-to\_objects <object>**

Collection of destination net objects returned by the *get\_nets* or *get\_pins* command.

**-of\_objects <object>**

Collection of reset sync objects returned by the *get\_reset\_sync* command.

**-to <to\_flop\_net>**

Name of destination flop output net or pin name.

**-resets <resets>**

Name of flop output net or pin name of a reset.

**-filter <filter\_expression>**

Filter expression can be specified over named attributes of crossings such as *is\_synchronized*, *is\_sync\_deasserted*, *missing\_synchronizer*, *invalid\_reset\_syn-synchronizer*, *different\_domain\_synchronizer*, *multi-flop\_reset\_synchronizer*, *user-defined\_reset\_synchronizer*, *reset\_constrained\_through\_abstract\_port\_constraint*, *reset\_constrained\_through\_input\_constraint*, and *generated\_reset*.

The following table describes the filter attributes available for this argument:

Filter Attribute	Description
<i>dest_clocks</i>	(String) Returns the clocks applied on the reset flip-flop
<i>dest_resets</i>	(String) Returns the resets applied on the reset flip-flop
<i>dest_domain</i>	(String) Returns the domain name of the clock on the reset flip-flop
<i>multi_flop_synchronizer_names</i>	(String) Returns names of driven nets of the reset synchronizer flip-flop chain
<i>dest_file_line</i>	(String) Returns the destination file

## **-regexp**

If this argument is specified, the patterns are considered as real regular expressions rather than simple wildcard patterns. This argument is not needed for wildcard pattern matching. Wildcard pattern matching is on by default. Regular expression can only be specified for design object names in the `-to` and `-resets` arguments.

## **Examples**

```
sg_shell> get_reset_sync //returns all possible reset flop
information related to flaged violations
```

```
    _sggrp1
```

```
sg_shell> report_reset_sync [get_reset_sync -to
{top111.top0_.inst_22.q}] //returns reset flop information
related to violations flaged on provided destination
```

```
    //Retrieving related attributes:
```

```
    set sync_flops [get_reset_sync];
```

```
    foreach_in_collection flop $sync_flops {
        set dest_resets [get_attribute $flop dest_resets];
        set dest_clocks [get_attribute $flop dest_clocks];
        set dest_domain [get_attribute $flop dest_domain];
        set dest_file_line [get_attribute $flop
dest_file_line];

        puts [format "dest_resets:%s dest_clocks:%s
dest_domain:%s dest_file_line:%s" $dest_resets $dest_clocks
$dest_domain $dest_file_line];
    }
```

## **See Also**

[\*get\\_reset\\_sync\\_names\*](#), [\*report\\_reset\\_sync\\_names\*](#), [\*report\\_reset\\_sync\*](#)

## get\_reset\_sync\_names

**Return the reset synchronizer information in the current design that match certain criteria**

### Syntax

```
get_reset_sync_names [<reset>] [-of_objects <obj>] [-resets  
<resets>][-regexp] [-filter <expr>]
```

### Scope

Goal

### Return Value

Returns the available reset synchronizer information in case of successful execution. In case of unsuccessful execution, an error is returned that can be identified using the `catch` command.

### Description

The *get\_reset\_sync\_names* command creates a collection of reset synchronizer information objects in the current design/instance that match certain criteria. The command returns a collection of reset synchronizers related to violations reported by the `Ar_sync01`, `Ar_unsync01`, `Ar_asynceassert01`, `Ar_syncdeassert01` rules and also pass the filtering criteria (if specified). If no objects match the criteria, an empty string is returned.

The *get\_reset\_sync\_names* command fails, if:

- Flattened design view does not exist.
- No data is available from the `Ar_sync01`, `Ar_unsync01`, `Ar_asynceassert01`, and `Ar_syncdeassert01` rules.

### Arguments

The *get\_reset\_sync\_names* command has the following arguments:

**<reset>**

Specifies collection of resets returned by the `get_reset` command.

**-of\_objects <object>**

Collection of reset synchronizers returned from the [get\\_reset\\_sync](#) command.

**-resets <resets>**

Specifies name of flip-flop output net name of a reset.

**-filter <filter\_expression>**

Filter expression can be specified over named attributes of crossings such as `is_synchronized`, `is_sync_deasserted`, `missing_synchronizer`, `invalid_reset_synchronizer`, `different_domain_synchronizer`, `multi-flop_reset_synchronizer`, `user-defined_reset_synchronizer`, `reset_con strained_through_abstract_port_constraint`, `reset_con strained_through_input_constraint`, and `generated_reset`.

The following table describes the filter attributes available for this argument:

Filter Attribute	Description
reset	(String) Returns the reset for the synchronizer
clock	(String) Returns the clock applied on the reset flip-flop
sync_name	(String) Returns the synchronizer flip-flop names of that synchronizer
module	(String) Returns names of reset synchronizer module
sync_type	(String) Returns the synchronization type
sync_count	(String) Returns the synchronizer flip-flop count

**-regexp**

If this argument is specified, then the patterns are seen as real regular expressions rather than simple wildcard patterns. This switch is not needed for wildcard pattern matching. Wildcard pattern matching is on by default. Regular expression can only be specified for design objects names in the `-to` and `-resets` arguments.

## Examples

```
sg_shell> get_reset_sync_names
    _sggrp1

sg_shell> set sync_nms [get_reset_sync_names -of_objects
[get_reset_sync]];

    //Retrieving related attributes:

foreach_in_collection nm $sync_nms {
    set reset [get_attribute $nm reset];
    set clock [get_attribute $nm clock];
    set sync_name [get_attribute $nm sync_name];
    set module [get_attribute $nm module];
    set sync_type [get_attribute $nm sync_type];
    set sync_count [get_attribute $nm sync_count];

    puts [format "reset:%s clock:%s sync_name:%s module:%s
type:%s sync_count:%s" $reset $clock $sync_name $module
$sync_type $sync_count ];
}
```

## See Also

[get\\_reset\\_sync](#), [report\\_reset\\_sync\\_names](#), [report\\_reset\\_sync](#)

## get\_paths

**Reports the complete paths between the specified start and end points**

### Syntax

```
get_paths [-from <from_pattern>] [-to <to_pattern>] [-  
from_objects <f_object>] [-to_objects <t_object>] [-  
of_objects <obj>] [-regex] [-type  
clock|async_reset|sync_reset|data] [-path_limit  
<limit>]
```

### Scope

Project

### Return Value

Returns a collection of complete paths between the specified start and/or end points.

### Description

The *get\_paths* command returns the complete path between the specified start and end points in current design matching specified criteria.

It is mandatory to run the *propagate\_clocks* and/or *propagate\_resets* command before *get\_paths* when using with *-type clock/async\_reset/sync\_reset*, so that the flip-flops/latches driven by clocks/resets are reported.

If the start or end point is a clock or reset, the path is computed between the flip-flops data path driven by the specified clocks/resets.

The following are the stop points while finding a path:

- Black-box without *assume\_path/abstract\_port path\_type combo|buf|inv*
- Blocked path
- Flops, latches, clock gating cells
- Library sequential elements without combinational arc

The *get\_paths* command fails if:

- The design query view type selected is not flat
- The flattened view of design is not present

## Arguments

The *get\_paths* command has the following arguments:

[ **<from>** ]

The argument returns the list of net name, pin name, port name of a start point.

[ **<to>** ]

The argument returns the list of net name, pin name, port name of an end point.

[ **<from\_objects>** ]

The argument returns the clock/reset objects returned from *get\_clocks/**get\_resets* commands, net objects, pin objects that are returned from other Tcl commands, (for example, *get\_nets* or *get\_pins*). This is for start points.

[ **<to\_objects>** ]

The argument returns the clock/reset objects returned from *get\_clocks/**get\_resets* commands, net objects, pin objects that are returned from other Tcl commands (for example, *get\_nets* or *get\_pins*). This is for end points.

[ **<of\_objects>** ]

The argument specifies collection of flat cells, flat nets and flat pins. Collection of resets driving specified flat objects are returned.

[ **<regexp>** ]

The collection indicates that all the arguments are regular expressions.

[ **<type>** ]

By default, it is set to data. It can have clock, *async\_reset*, *sync\_reset* if user wants to see data paths for specific clock, asynchronous or

synchronous resets.

[ <path\_limit> ]

This argument limits the number of paths to be shown. By default, it shows paths for 1000 start/end points. Specify 0 to show all the paths.

## Examples

```

      _____
|FF1|      |--\      |--\      |--\      |FF2|
|    |-----|N1 |-----|N2 |-----|N3 |-----|    |---
|>__|      |--/      |--/      |--/      |>__|

```

```
sg_shell> get_paths -from top.FF1 -to top.FF2
```

```
sg_shell> get_paths -from {top.FF1} -to {top.FF2}
```

```
sg_shell> get_paths -from {top.FF11 top.FF12} -to {top.FF21
top.FF22}
```

```
sg_shell> get_paths -of_objects [get_paths -from {top.FF11
top.FF12} -to {top.FF21 top.FF22}
```

## See Also

[report\\_paths](#)



## report\_cdc

### Reports clock domain crossing details

#### Syntax

```
report_cdc [<crossings>] [-dump_csv <file-name>]
```

#### Scope

Goal

#### Return Value

None

#### Description

The report\_cdc command is used to report crossing details as given in the Ac\_sync\_group\_detail.

The report\_cdc command fails when:

- flattened design view does not exist.
- No data is available from Ac\_sync\_group rules.
- An improper collection is provided as an input to the report\_cdc command. A collection is called improper when it does not consist of crossings, that is, a collection returned by a command other than the get\_cdc command.

#### Arguments

The report\_cdc command has the following arguments:

##### <crossings>

Use this argument to specify a collection of crossings returned by the get\_cdc command. This command reports crossing details in a tabular format. If this option is not specified, all the crossing in current\_design will be reported.

##### [-dump\_csv <file-name>]

Use this argument to generate the report in CSV file specified as <file-

name>. It generates the file in the `spyglass_reports/clock-reset/tcl_dump` directory. The report is generated with the columns *S.No.*, *Rule Name*, *Dest Name*, *Dest Clock(s)*, *Source Name*, *Source Clock(s)*, *Failure Reason*, *Synch. Scheme*, *Dest File:Line*, and *Source File:Line*.

## Examples

```
sg_shell> report_cdc //reports all the crossing in
current_design
```

Following is the output of the above command:

```
=====
Synchronized Scalar Signal Crossings (Ac_sync01)
=====
+++++
S.No.  Dest.Name  Dest.Clock Names      Source Name      Source Clock Names  Sync. Scheme
=====
1      top.q_sync2  top.clk3,top.clk4,   top.q_sync1      top.clk1,top.clk2   Conventional
                top.clk5                                     Multi-flop for
                                                Metastability
                                                Technique
+++++
=====
Synchronized Vector Signal Crossings (Ac_sync02)
=====
+++++
S.No.  Dest. Name  Dest.Clock Names      Source Name      Source Clock Names  Sync. Scheme
=====
1      top.q_sync2  top.clk3,top.clk4,   top.q_sync1      top.clk1,top.clk2   Conventional
      _bus[7:0]  ,top.clk5             _bus[7:0]
                                                Multi-flop for
                                                Metastability
                                                Technique
+++++
sg_shell> report_cdc [get_cdc -to_object [get_clocks tag4]]
//report all the crossing returned by get_cdc command
sg_shell> report_cdc [get_clocks] //negative scenario
```

where improper collection is provided as input

**See Also**

[\*get\\_cdc\*](#)

## report\_cdc\_coherency

**Displays the collection of coherency/convergence issues reported by get\_cdc\_coherency**

### Syntax

```
report_cdc_coherency [<conv_issues>]
```

### Scope

Goal

### Return Value

None

### Description

This command is used to display the coherency/convergence issues present in design, in tabular format.

### Arguments

The `report_cdc_coherency` command has the following arguments:

#### **conv\_issues**

Use this argument to pass the list of convergence issues returned by the `get_cdc_coherency` command. If no collection is passed to this command, all the coherency issues present in design are reported.

### Examples

Consider the following commands:

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter
is_user_defined==true]
```

```
=====
=====
(Ac_conv05)
=====
```

## Debug Commands

```

=====

+++++
+++++
S.No.   Signals                               Status
=====
=====
1       gray.reg2[1:0],gray.reg2_in[1:0] PASSED
+++++
+++++

sg_shell> report_cdc_coherency [get_cdc_coherency -from
current_state]

=====
=====
(Ac_conv04)
=====
=====

+++++
+++++
S.No.   Count  Signals                               Status
=====
=====
1       3      gray.reg1[0:2] PASSED
+++++
+++++

sg_shell> report_cdc_coherency

=====

```

```

=====
(Ac_conv01)
=====
=====

```

```

+++++
+++++

```

S.No.	Count	Signals	Converging Gate
-------	-------	---------	-----------------

```

=====
=====

```

1	2	test2.sync2.r[0] test2.sync1.r[0]	test2.o1
---	---	--------------------------------------	----------

```

+++++
+++++

```

```

=====
=====

```

(Ac\_conv03)

```

=====
=====

```

```

+++++
+++++

```

S.No.	Count	Signals	Converging Gate
-------	-------	---------	-----------------

```

=====
=====

```

1	3	test2.sync3.r[0] test2.sync2.r[0] test2.sync1.r[0]	test2.o1
---	---	--	----------

```

+++++

```

---

**Debug Commands**

+++++

**See Also**

[\*report\\_cdc\*](#), [\*get\\_cdc\\_coherency\*](#)

## report\_cdc\_glitch

Reports clock domain crossing with glitches

### Syntax

```
report_cdc_glitch [<crossings>]
```

### Scope

Goal

### Return Value

None

### Description

The *report\_cdc\_glitch* command is used to report crossing that may have glitches as reported by the *Ac\_glitch03* rule.

This command fails when:

- flattened design view does not exist.
- No data is available from the *Ac\_glitch03* rule.
- An improper collection is provided as an input to *the report\_cdc\_glitch* command. A collection is called improper when it does not consist of crossings, that is, a collection returned by a command other than the *get\_cdc\_glitch* command.

### Arguments

The report\_cdc command has the following arguments:

#### <crossings>

Use this argument to specify a collection of crossings with glitches returned by the *get\_cdc\_glitch* command. This command reports details of the crossing with glitches in a tabular format. If this option is not specified, all the crossings with glitches in *current\_design* are reported.

### Examples

```
sg_shell> report_cdc_glitch //reports all the crossings with
```



```
glitches in current_design
  1. Destination Name: top.q1
     Destination Internal Domain ID: 5
     Destination Clock(s): Domain : Internal Domain ID: Tag Name
     top.hierInst1.c1: d1: 1: T1
     top.hierInst2.c2: d2: 2: <NA>
     Gray Encoding Check: DISABLED
     Reason: Source reconverges
     Total Sources: 1
     Total Source Domains: 1
```

```
sg_shell> report_cdc_glitch [get_cdc_glitch -to_objects
[get_clocks tag4]] //report all the glitchy crossin
gs returned by get_cdc_glitch command
```

```
sg_shell> report_cdc_glitch [get_clocks] //negative scenario
where improper collection is provided as input
report_cdc_glitch: error: collection type mismatch. Refer
'report_cdc_glitch -help' or 'man report_cdc_glitch'
```

## See Also

[\*get\\_cdc\\_glitch\*](#)

## report\_paths

Reports elements in a defined path in current design

### Syntax

```
report_paths [<paths>]
```

### Scope

Project

### Return Value

None

### Description

The *report\_paths* command is used to report results returned by the *get\_paths* command in the current design.

It is mandatory to run the *get\_paths* command before running *report\_paths*, so that paths are reported.

The *report\_paths* command fails, if:

- Design query view type selected is not flat.
- Flattened design view is not present.
- An improper collection (a collection that is not of paths as returned by the *get\_paths* command) of objects has been provided as input to the *report\_paths* command.

### Arguments

The *report\_paths* command has the following arguments:

[ <paths> ]

The argument reports the collection returned by the *get\_paths* command in current design.

### Examples

```
sg_shell> report_paths
```

Start Point: Top.FF1

End Point: Top.FF2

Path Type: Data

Path:

a) Top.N1

Top.N2

Top.N3

b) Top.N11

Top.N21

Top.N31

## See Also

[\*get\\_paths\*](#)

## report\_reset\_sync

Reports reset synchronization issues related information

### Syntax

```
report_reset_sync [<reset_syncs>]
```

### Scope

Goal

### Return Value

None

### Description

The *report\_reset\_sync* command is used to display the collection of reset synchronizer issues returned by the *get\_reset\_sync* command.

The *report\_reset\_sync* command fails, if:

- Flattened design view does not exist.
- No data is available from the Ar\_sync01, Ar\_unsync01, Ar\_asyncdeassert01, and Ar\_syncdeassert01 rules.
- An improper collection (a collection that is not of reset\_flop\_node, that is a collection returned by a command other than get\_reset\_sync) has been provided as input to the report\_reset\_sync command.

### Arguments

The *report\_reset\_sync* command has the following arguments:

**<reset\_syncs>**

Collection of reset synchronization issues returned by the *get\_reset\_sync* command. If this option is not specified, all the sync issues in *current\_design* are reported.

### Examples

```
sg_shell> report_reset_sync //reports all the crossing  
in current_design
```

```

*****
Report : flops of reset synchronizer
*****

=====

Dest Name : test.t
Dest Type : flop
Dest Reset(s) : test.rst (clear)
Dest Clock(s) : test.clk1
Sync Method : User-defined reset synchronizer
(Name:sync_cell)
Failure Reason : N.A.
De-assertion Status : Synchronously de-asserted
relative to clock signal 'test.clk1'
Reset Synchronizer Names : N.A.

=====

*****
Report : flops of reset synchronizer
*****

=====

Dest Name : test.t
Dest Type : flop
Dest Reset(s) : test.rst (clear)
Dest Clock(s) : test.clk1
Sync Method : User-defined reset synchronizer
(Name:sync_cell)

```

```
Failure Reason : N.A.  
De-assertion Status : Synchronously de-asserted  
relative to clock signal 'test.clk1'  
Reset Synchronizer Names : N.A.  
=====
```

## See Also

[get\\_reset\\_sync](#), [get\\_reset\\_sync\\_names](#), [report\\_reset\\_sync\\_names](#)

## report\_reset\_sync\_names

Reports reset synchronization related information

### Syntax

```
report_reset_sync_names [<reset_syncs>]
```

### Scope

Goal

### Return Value

None

### Description

The `report_reset_sync_names` command is used to report crossing details as given in the `Ac_sync_group_detail` report.

The `report_reset_sync_names` command fails, if:

- Flattened design view does not exist.
- No data is available from the `Ar_sync01`, `Ar_unsync01`, `Ar_asyncdeassert01`, and `Ar_syncdeassert01` rules.
- An improper collection (a collection that is not of `reset_sync_node`, that is the collection returned by a command other than `get_reset_sync_names`) has been provided as input to the `report_reset_sync_names` command.

### Arguments

The `report_reset_sync_names` command has the following arguments:

#### <crossings>

Collection of crossings returned by the `get_reset_sync_names` command. If this option is not specified, all the reset synchronizations in `current_design` will be reported.

### Examples

```
sg_shell> report_reset_sync_names //reports all reset
```

synchronizers in the design

```
*****  
Report : reset synchronizer information  
*****  
  
=====
```

Reset : test.rst  
Synchronizer Flop(s) : N.A.  
Synchronizer Flop Count : N.A.  
Synchronizer Clock(s) : test.clk1  
Type : User-defined reset synchronizer  
=====

## See Also

[get\\_reset\\_sync](#), [get\\_reset\\_sync\\_names](#), [report\\_reset\\_sync](#)



## SpyGlass DFT Commands

The following table describes the Tcl commands that are a part of the SpyGlass DFT product:

<b>Command</b>	<b>Description</b>
<i>dft_generate_coverage</i>	Generates coverage information for selected modules and instances.
<i>dft_generate_fault_report</i>	Generates pin wise fault information for selected modules and instances.
<i>dft_generate_scan_report</i>	Generates instance-wise scannability information of flip-flops or latches in the selected modules and scannability information for selected flip-flops or latch instances in the current design.
<i>dft_generate_latch_status_report</i>	Generates instance-wise transparency information of latches in selected modules and transparency information for selected latch instances in current design.
<i>cv_is_cmt_present</i>	Check if the passed constraint_message_tag is present on the specified flat-object.
<i>dsm_assert_illegal_path</i>	Defines the illegal connectivity check for a path.
<i>dsm_assert_illegal_value</i>	Defines a check that a logic value should not be present on a design node.
<i>cv_define_user_macro</i>	Define a new user macro.
<i>cv_delete_user_macro</i>	Delete a user macro.
<i>cv_reset_user_macros</i>	Delete all user macros.
<i>cv_get_list_of_user_macros</i>	Get list of user macros.
<i>cv_add_element_to_user_macro</i>	Add flat-object to user macro's collection.
<i>cv_get_cell_list_of_user_macro</i>	Get cell list of user macro.
<i>cv_get_pin_list_of_user_macro</i>	Get pin list of user macro.
<i>cv_get_port_list_of_user_macro</i>	Get port list of user macro.
<i>cv_is_element_present_in_user_macro</i>	Check if flat-object is present in the user macro collection.
<i>cv_remove_element_from_user_macro</i>	Remove flat-object from user macro's collection.

## dft\_generate\_coverage

**Generates coverage information for selected modules and instances**

### Syntax

```
dft_generate_coverage
    [-top <top_name>]
    [-report_file <report_file>]
    [-fault_model <fault_model>]
    [-module <list_of_modules>]
    [-instance <list_of_instances>]
```

### Scope

Goal

### Return Value

The *dft\_generate\_coverage* command returns info, warning and error messages:

- The info message is shown for the values of different fields used as shown in the following example:  
info: Using 'stuck\_at' (default) for fault\_model  
info: top field not defined
- warning message is shown if something expected is not found as shown in the following example:  
warning: No module with name 'moid' found in design 'my\_design'
- error message is shown if command can not proceed further as shown in the following example:  
error: Unable to open 'my\_design\_sa\_file' for writing  
error: No top design 'my\_design' found

### Description

This command is used to generate selective coverage information for specific set of modules, instances or ports.

This command uses already completed coverage analysis. Therefore, it is necessary to run the *Info\_coverage*, *Info\_transitionCoverage*, and *Info\_random\_resistance* rules, which perform stuck-at, transition and random resistance coverage analysis, respectively.

## Arguments

The `dft_generate_coverage` command has the following arguments:

### **-top <top-name>**

This argument takes the name of the top design unit, coverage information for specified modules and instance will be reported for this top only. If there is only one design unit then this argument is optional.

### **-report\_file <report-file>**

This is an optional argument, its default value is stdout. It takes the full path name of the report file.

**NOTE:** *Using this option is recommended.*

### **fault\_model <fault-model>**

This is an optional argument. Its default value is `stuck_at`. It is used to specify the fault model for which information needs to be reported. Supported values are `stuck_at`, `transition`, and `rrf`.

- **stuck\_at** is used for stuck-at fault model. The *Info\_coverage* rule of SpyGlass DFT generates the data for this fault model.
- **transition** is used for transition fault model. The *Info\_transitionCoverage* rule of SpyGlass DFT generates the data for this fault model. The `dsm_launch_method` parameter needs to be set as `loc/los` to generate the required data.
- **rrf** is used for random resistance fault model. The *Info\_random\_resistance rule* of SpyGlass DFT generates the data for this fault model.

### **-module <list-of-modules>**

This is an optional argument. It is used to specify all the modules for which the coverage information needs to be reported. This argument takes a list.

**-instance <list-of-instances>**

This is an optional argument. It is used to specify all the instances (full path of the instantiation) for which the coverage information needs to be reported. This argument takes a list.

**Examples**

```
sg_shell > dft_generate_coverage \  
-top my_design -report_file path_to_target_file/  
stuck_at_DT_for_mod.rpt \  
-fault_model stuck_at \  
-module "modeA modB
```

```
sg_shell > dft_generate_coverage
```

**See Also**

N/A

## dft\_generate\_fault\_report

**Generates pin wise fault information for selected modules and instances**

### Syntax

```
dft_generate_fault_report
    [-top <top-name>]
    [-report_file <report-file>]
    [-fault_model <fault-model>]
    [-fault_type <fault-type>]
    [-module <list-of-modules>]
    [-instance <list-of-instances>]
    [-port]
```

### Scope

Goal

### Return Value

Returns returns info, warning and error messages.

### Description

This command is used to generate selective fault information for specific set of modules, instances or ports. It uses already completed fault analysis, so it is necessary to run the *Info\_coverage* and *Info\_transitionCoverage* rules, which perform stuck at and transition fault analysis respectively.

### Arguments

The `dft_generate_fault_report` command has the following arguments:

**-top <top-name>**

This argument takes the name of the top design unit, fault information for

specified modules and instance will be reported for this top only. If there is only one design unit then this argument is optional.

**-report\_file <report-file>**

This is an optional argument, its default value is stdout. It takes the full path name of the report file.

**NOTE:** *Using this option is recommended*

**fault\_model <fault-model>**

This is an optional argument. It's default value is `stuck_at`. It is used to specify the fault model for which information needs to be reported. Supported values are `stuck_at`, `transition_loc` and `transition_los`.

- `stuck_at` is used for stuck-at fault model. The *Info\_coverage* rule of SpyGlass DFT generates the data for this fault model.
- `transition_loc` is used for transition fault model with launch on capture. The *Info\_transitionCoverage* rule of SpyGlass DFTDSM generates the data for this fault model. The `dsm_launch_method` parameter needs to be set as `loc`.
- `transition_los` is used for transition fault model with launch on capture. The *Info\_transitionCoverage* rule of SpyGlass DFTDSM generates the data for this fault model. The `dsm_launch_method` parameter needs to be set as `los`.

**-fault\_type <fault-type>**

This is an optional argument, its default value is `all`. It specifies the categories of the fault that needs to be reported. The supported options are `all`, `detected`, `undetected`, `synthesis_redundant`, `false_path`, `unused`, `untestable`, `tied`, `blocked`, `logical_redundant`, and `all_except_detected`.

**-module <list-of-modules>**

This is an optional argument. It is used to specify all the modules for which the fault information needs to be reported. This argument takes a list.

**-instance <list-of-instances>**

This is an optional argument. It is used to specify all the instances (full

path of the instantiation) for which the fault information needs to be reported. This argument takes a list.

### **-port**

This is an optional argument. If specified then the fault information for all ports is reported.

**NOTE:** *If module, instance, and port are not defined then fault information for complete design unit is reported.*

## **Examples**

```
sg_shell > dft_generate_fault_report \  
                -top my_design -report_file \  
path_to_target_file/stuck_at_detected_for_mod.rpt \  
                -fault_model stuck_at -fault_type detected \  
                -module "modeA modB"
```

```
sg_shell > dft_generate_fault_report
```

## **See Also**

[dft\\_generate\\_scan\\_report](#), [dft\\_generate\\_latch\\_status\\_report](#)

## dft\_generate\_scan\_report

**Generates instance-wise scannability information of flip-flops or latches in the selected modules and scannability information for selected flip-flops or latch instances in the current design**

### Syntax

```
dft_generate_scan_report
    [-top <top-name>]
    [-type <scan-type>]
    [-module <list-of-modules>]
    [-instance <list-of-instances>]
```

### Scope

Goal

### Return Value

Returns info, warning and error messages.

### Description

This command is used to generate a report of scannability information of flip-flops or latches in a specific set of modules and scannability information for selected flip-flops or latch instances in current design. This report should be generated after one of the DFT goals has been run.

### Arguments

The `dft_generate_scan_report` command has the following arguments:

#### **-top <top-name>**

This argument takes the name of the top design unit, scan information for specified modules and instance will be reported for this top only. If there is only one design unit then this argument is optional.

#### **-type <scan-type>**

This is an optional argument. Its default value is 'all'. It specifies the scannability types that need to be reported. The supported options are



all, scan\_forced, scan\_inferred, no\_scan\_forced, no\_scan\_inferred, unscannable\_clock, unscannable\_reset, and unscannable\_clock\_reset.

**-module <list-of-modules>**

This is an optional argument. It is used to specify all the modules for which the scan information needs to be reported. This argument takes a list.

**-instance <list-of-instances>**

This is an optional argument. It is used to specify all the instances (full path of the instantiation) for which the scan information needs to be reported. This argument takes a list.

## Examples

```
sg_shell > dft_generate_scan_report \  
             -top my_design \  
             -type scan_forced \  
             -module "modeA modB"  
sg_shell > dft_generate_scan_report
```

## See Also

[dft\\_generate\\_fault\\_report](#), [dft\\_generate\\_latch\\_status\\_report](#)

## dft\_generate\_latch\_status\_report

**Generates instance-wise transparency information of latches in selected modules and transparency information for selected latch instances in current design**

### Syntax

```
dft_generate_latch_status_report
    [-top <top-name>]
    [-mode <simulation-mode>]
    [-type <transparency-type>]
    [-module <list-of-modules>]
    [-instance <list-of-instances>]
```

### Scope

Goal

### Return Value

Returns info, warning and error messages.

### Description

This command is used to generate a report of transparency data of latches in a specific set of modules or transparency information for selected latch instances in the current design. This report should be generated after one of the DFT/DFT\_DSM goals has been run.

### Arguments

The `dft_generate_latch_status_report` command has the following arguments:

#### **-top <top-name>**

This argument takes the name of the top design unit, transparency information for specified modules and instance will be reported for this top only. If there is only one design unit then this argument is optional.

#### **-mode <simulation-mode>**

This is an optional argument that specifies the simulation mode which the

transparency information should generate. The supported options are `shift`, `capture`, and `atspeed`. The default mode is `capture` mode.

**-type <transparency-type>**

This is an optional argument, its default value is 'all'. It specifies the transparency types that needs to be reported. The supported options are `all`, `transparent_forced`, `transparent_clock_off`, `transparent_control`, `shadow`, `lockup_source`, and `lockup_destination`.

**-module <list-of-modules>**

This is an optional argument. It is used to specify all the modules for which the transparency information needs to be reported. This argument takes a list.

**-instances <list-of-instances>**

This is an optional argument. It is used to specify all the instances (full path of the instantiation) for which the transparency information needs to be reported. This argument takes a list.

## Examples

```
sg_shell > dft_generate_latch_status_report \  
           -top my_design -mode capture \  
           -type transparent_control \  
           -module "modeA modB"
```

```
sg_shell > dft_generate_latch_status_report \  
           -mode atspeed
```

## See Also

[dft\\_generate\\_fault\\_report](#), [dft\\_generate\\_scan\\_report](#)

## cv\_is\_cmt\_present

Check if the given `constraint_message_tag` is present on the specified flat-object.

### Syntax

```
cv_is_cmt_present
  -flat_object <flat_object>
  -cmt_name <constraint_message_tag_name>
```

### Scope

Goal

### Return Value

- Returns 1 if the specified `constraint_message_tag`, along with modifier, is found and returns 0 if it is not found.
- Error messages are shown if something unexpected is found and command cannot proceed further.

### Description

The `cv_is_cmt_present` command is used to query whether a given `flat_object` has passed or failed a specific connectivity check captured via `constraint_message_tag`.

### Arguments

The `cv_is_cmt_present` command has the following arguments:

**-flat\_object <flat\_object>**

This argument takes a `flat_object` and not its name.

**-cmt\_name <constraint\_message\_tag\_name>**

This is the `constraint_message_tag` name along with `:PASS` or `:FAIL` modifier.

## Examples

```
sg_shell > cv_is_cmt_present \  
-flat_object [get_ports clkout] \  
-cmt_name OUTPUT_CLK_CHECK:PASS
```

## See Also

None

## dsm\_assert\_illegal\_path

Defines the illegal connectivity check for a path

### Syntax

```
dsm_assert_illegal_path
  -from <from_list> -to <to_list>
  [-path_type <type>]
  [-use_shift]
  [-use_capture]
  [-use_capture_at_speed]
  [-waveform]
```

### Scope

dsm

### Return Value

The `dsm_assert_illegal_path` command returns connectivity status between specified pairs of nodes.

### Description

This command defines an illegal-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, when `dsm_assert_illegal_path` command is issued.

### Arguments

The `dsm_assert_illegal_path` command has the 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 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

## Debug Commands

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.

**-use\_shift | -use\_capture | -use\_capture\_at\_speed**

These are optional arguments. For any of these modifiers, the `dsm_assert_illegal_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.

**-waveform**

This is an optional argument. This option generates a waveform for better debugging in GUI mode.

**Examples**

```
sg_shell > dsm_assert_illegal_path -from top.a[2:0] top.b  
-to top.c[3:0]
```

**See Also**

None

## dsm\_assert\_illegal\_value

Defines a check that a logic value should not be present on a design node

### Syntax

```
dsm_assert_illegal_value
  -name <node_list>
  -value <value_list>
  [-match_n_bits <size>]
  [-use_shift]
  [-use_capture]
  [-use_capture_at_speed]
  [-waveform]
```

### Scope

dsm

### Return Value

The `dsm_assert_illegal_value` command returns the following warning messages when node has the illegal value:

```
dsm_assert_illegal_value returns "Node '<node_name>' has
illegal value <current_sim_val u
```

### Description

This command is used to check whether specific value on user specified design node is not achieved under specific simulation condition. The simulation condition is the state of the circuit, when the `dsm_assert_illegal_value` command is issued.

### Arguments

The `dsm_assert_illegal_value` command has the following arguments:



## Debug Commands

**<node\_list>**

This argument can assume one of the following values: Top-module port name, Internal net name, or 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.

**<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_illegal_value` command simulates testmode of that particular mode. If more than one of the -tag, -use\_shift, -use\_capture, or -use\_capture\_at\_speed arguments is specified, an error condition occurs. Only one of these modifiers should be specified with the `dsm_assert_illegal_value` command.

**-waveform**

This is an optional argument. This option generates a waveform for better debugging in GUI mode.

**Examples**

```
sg_shell > dsm_assert_illegal_value -name top.U_CGC.TE -  
value 1 -use_capture
```

**See Also**

None

## cv\_define\_user\_macro

Define a new user marco.

### Syntax

```
cv_define_user_macro
    -macro <user_macro_name>
```

### Scope

Goal

### Return Value

- Returns 1 if macro creation was successful.
- Error messages are shown if something unexpected is found.

### Description

The `cv_define_user_macro` command is used to define a new user macro which can be used for subsequent connectivity checks.

### Arguments

The `cv_define_user_macro` command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes new user macro name.

### Examples

```
sg_shell > cv_define_user_macro \  
    -macro user_macro_1
```

### See Also

None

## cv\_delete\_user\_macro

Delete a user macro.

### Syntax

```
cv_delete_user_macro
    -macro <user_macro_name>
```

### Scope

Goal

### Return Value

- Returns 1 if macro deletion was successful.
- Error messages are shown if something unexpected is found.

### Description

The `cv_delete_user_macro` command is used to delete a user macro.

### Arguments

The `cv_delete_user_macro` command has the following arguments:

**-macro** <user\_macro\_name>

This argument takes user macro name to be deleted.

### Examples

```
sg_shell > cv_delete_user_macro \  
    -macro user_macro_1
```

### See Also

None

## cv\_reset\_user\_macros

Delete all user macros.

### Syntax

```
cv_reset_user_macros
```

### Scope

Goal

### Return Value

- Returns 1 on successful deletion.
- Error messages are shown if something unexpected is found.

### Description

The `cv_reset_user_macros` command is used to delete all user macros.

### Examples

```
sg_shell > cv_reset_user_macros
```

### See Also

None

## **cv\_get\_list\_of\_user\_macros**

Get list of user macros.

### **Syntax**

```
cv_get_list_of_user_macros
```

### **Scope**

Goal

### **Return Value**

- Returns list of user macros.
- Error messages are shown if something unexpected is found.

### **Description**

The `cv_get_list_of_user_macros` command is used to get list of user macros.

### **Examples**

```
sg_shell > cv_get_list_of_user_macros
```

### **See Also**

None

## cv\_add\_element\_to\_user\_macro

**Add flat-object to user macro's collection.**

### Syntax

```
cv_add_element_to_user_macro
    -macro <user_macro_name>
    -flat_object <flat_object>
```

### Scope

Goal

### Return Value

- Returns 1 if flat\_object was successfully added to the user macro.
- Error messages are shown if something unexpected is found.

### Description

The `cv_add_element_to_user_macro` command is used to add a flat\_object to the user macro's collection.

### Arguments

The `cv_add_element_to_user_macro` command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes user macro name.

**-flat\_object <flat\_object>**

This argument takes a flat\_object and not its name

### Examples

```
cv_add_element_to_user_macro \  
    -macro    user_macro_1 \  
    -flat_object [get_pins test.mod_i1.and_1.A0]
```

---

Debug Commands

## See Also

None

## cv\_get\_cell\_list\_of\_user\_macro

Get cell list of user macro.

### Syntax

```
cv_get_cell_list_of_user_macro  
-macro <user_macro_name>
```

### Scope

Goal

### Return Value

- Returns collection of cells.
- Error messages are shown if something un-expected is found.

### Description

The `cv_get_cell_list_of_user_macro` is used to get list of cells present in the collection of the passed user macro.

### Arguments

The `cv_get_cell_list_of_user_macro` command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes user macro name.

### Examples

```
cv_get_cell_list_of_user_macro \  
-macro user_macro_1
```

### See Also

None



## cv\_get\_pin\_list\_of\_user\_macro

Get pin list of user macro.

### Syntax

```
cv_get_pin_list_of_user_macro  
-macro <user_macro_name>
```

### Scope

Goal

### Return Value

- Returns collection of pins.
- Error messages are shown if something un-expected is found.

### Description

The `cv_get_pin_list_of_user_macro` is used to get list of pins present in the collection of the passed user macro

### Arguments

The `cv_get_pin_list_of_user_macro` command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes user macro name.

### Examples

```
cv_get_pin_list_of_user_macro \  
-macro user_macro_1
```

### See Also

None

## cv\_get\_port\_list\_of\_user\_macro

Get port list of user macro.

### Syntax

```
cv_get_port_list_of_user_macro  
-macro <user_macro_name>
```

### Scope

Goal

### Return Value

- Returns collection of ports.
- Error messages are shown if something un-expected is found.

### Description

The `cv_get_port_list_of_user_macro` is used to get list of ports present in the collection of the passed user macro.

### Arguments

The `cv_get_port_list_of_user_macro` command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes user macro name.

### Examples

```
cv_get_port_list_of_user_macro \  
-macro user_macro_1
```

### See Also

None

## cv\_is\_element\_present\_in\_user\_macro

Check if flat-object is present in the user macro's collection.

### Syntax

```
cv_is_element_present_in_user_macro
    -macro <user_macro_name>
    -flat_object <flat_object>
```

### Scope

Goal

### Return Value

- Returns 1 if flat\_object is present in passed macro's collection and 0 otherwise.
- Error messages are shown if something unexpected is found.

### Description

The `cv_is_element_present_in_user_macro` is used to check if `flat_object` is present in user macro's collection.

### Arguments

The `cv_is_element_present_in_user_macro` command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes user macro name.

**-flat\_object <flat\_object>**

This argument takes a `flat_object` and not its name.

### Examples

```
cv_is_element_present_in_user_macro \  
    -macro    user_macro_1 \  
    -flat_object [get_pins test.mod_i1.and_1.A0]
```

## See Also

None

## cv\_remove\_element\_from\_user\_macro

Remove flat-object from user macro's collection.

### Syntax

```
cv_remove_element_from_user_macro  
-macro <user_macro_name>  
-flat_object <flat_object>
```

### Scope

Goal

### Return Value

- Returns 1 if flat\_object was successfully removed from macro's collection and 0 otherwise.
- Error messages are shown if something unexpected is found.

### Description

The cv\_remove\_element\_from\_user\_macro is used to remove a flat\_object from the user macro's collection.

### Arguments

The cv\_remove\_element\_from\_user\_macro command has the following arguments:

**-macro <user\_macro\_name>**

This argument takes user macro name.

**-flat\_object <flat\_object>**

This argument takes a flat\_object and not its name.

### Examples

```
cv_remove_element_from_user_macro \  
-macro user_macro_1 \  
-flat_object [get_pins test.mod_i1.and_1.A0]
```

## See Also

None

## SpyGlass Power Verify Commands

The following table describes the various Tcl commands that are a part of the SpyGlass Power Verify product:

<b>Command</b>	<b>Description</b>
<i>check_pwr_intent_crossing</i>	Displays the crossing type between the two power intent nodes
<i>get_pwr_intent</i>	Gives a power intent node on which user can query information
<i>get_retention_info</i>	Gives a power retention node containing information related to retention strategy applied on an instance
<i>get_isolation_info</i>	Gives a power isolation node containing information related to isolation strategy applied on an instance
<i>get_power_switch_info</i>	Gives a power PSW node containing information related to create_power_switch strategy applied on an instance
<i>get_level_shifter_info</i>	Gives a power level shifter node containing information related to level shifter strategy applied on an instance
<i>get_supply_info</i>	Gives a power supply node containing information related to power supply corresponding to a design net
<i>report_retention_info</i>	Displays the information of the power retention node
<i>report_isolation_info</i>	Displays the information of the power isolation node
<i>report_pwr_intent</i>	Displays the information of the power intent node that is given as input
<i>report_power_switch_info</i>	Displays the information of the power PSW node
<i>report_level_shifter_info</i>	Displays the information of the power level shifter node
<i>report_supply_info</i>	Displays the information of the power supply node

## get\_pwr\_intent

**Gives a power intent node for a design element on which user can query information**

### Syntax

```
get_pwr_intent [get_cells | get_pins | get_ports]
```

### Scope

Goal

### Return Value

Returns a collection of power intent nodes.

### Description

This `get_pwr_intent` command gives a power intent node on which user can query information, such as domain name or power supply, etc.

This command works on ports and leaf level cells (instances and pins) and returns a collection of power intent nodes.

### Arguments

None

### Examples

```
sg_shell> get_pwr_intent [ get_cells ]  
sg_shell> get_pwr_intent [ get_cells * ]  
sg_shell> get_pwr_intent [ get_cells top.inst1 ]
```

### See Also

[report\\_pwr\\_intent](#), [check\\_pwr\\_intent\\_crossing](#), [power\\_domain](#), [power\\_supply](#), [ground\\_supply](#), [voltage\\_range\\_min](#), [voltage\\_range\\_max](#)



## report\_pwr\_intent

**Displays the information of the power intent node**

### Syntax

```
report_pwr_intent <pwr_intent_node>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

The `report_pwr_intent` command displays the information of the power intent node that is given as an input. The following information is displayed, in columns, for an element that is being probed:

- Domain name: It shows the name of the domain in which the element is present.
- Voltage range: It shows the voltage range of the domain. The voltage range displayed is in the following format:  
`voltage_range_min: voltage_range_max`  
Voltage values for `voltage_range_min` and `voltage_range_max` can be equal depending on the power intent. The `off` and `0` volt values are not shown for a domain.
- Power supply: It shows the power supply connected to the element.
- Ground supply: It shows the ground supply connected to the element.

**NOTE:** *Power supply and ground supply are not shown in the SGDC format.*

### Arguments

The `report_pwr_intent` command has the following arguments:

#### `pwr_intent_node`

Use this argument to report the details power intent.

## Examples

```
sg_shell> report_pwr_intent [ get_pwr_intent [ get_cells ] ]
```

```
sg_shell> report_pwr_intent [ get_pwr_intent [get_cells  
top.inst1 ]
```

The following is the output of the above command:

```
Domain           :      top  
Voltage Range    :      0.6:0.6  
Power Supply     :      VTOP  
Ground Supply    :      VSS
```

```
sg_shell> report_pwr_intent [ get_pwr_intent [get_cells  
top.inst2 ]
```

The following is the output of the above command:

```
Domain           :      PD1  
Voltage Range    :      0.6:0.8  
Power Supply     :      VDD1  
Ground Supply    :      VSS
```

```
sg_shell> report_pwr_intent [ get_pwr_intent [get_cells  
top.inst2 ]
```

The following is the output of the above command in the SGDC format:

```
Domain           :      PD1  
Voltage Range    :      0.6:0.8
```

## See Also

[get\\_pwr\\_intent](#), [check\\_pwr\\_intent\\_crossing](#), [power\\_domain](#), [power\\_supply](#),  
[ground\\_supply](#), [voltage\\_range\\_min](#), [voltage\\_range\\_max](#)

## check\_pwr\_intent\_crossing

Displays the crossing type between the two power intent nodes

### Syntax

```
check_pwr_intent_crossing -from <pwr_intent_node1> -to  
<pwr_intent_node2> -type <isolation | level_shift>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

This command takes as an input a pair of power intent node and type of the check (isolation or level\_shift) based on the state defined in power intent and returns a string.

### Arguments

The `check_pwr_intent_crossing` command has the following arguments:

#### **-from**

Use this argument to specify the source power intent node.

#### **-to**

Use this argument to specify destination power intent node.

#### **-type**

Use this argument to specify the relationship type (isolation or level\_shift).

### Examples

```
sg_shell> check_pwr_intent_crossing -from [ get_pwr_intent [ get_cells top.inst1 ] ] -to [ get_pwr_intent [ get_cells
```

```
top.inst2 ] ] -type isolation
sg_shell> check_pwr_intent_crossing -from [ get_pwr_intent [
get_cells top.inst1 ] ] -to [ get_pwr_intent [ get_cells
top.inst2 ] ] -type level_shift
```

## See Also

[get\\_pwr\\_intent](#), [report\\_pwr\\_intent](#), [power\\_domain](#), [power\\_supply](#),  
[ground\\_supply](#), [voltage\\_range\\_min](#), [voltage\\_range\\_max](#)

## get\_retention\_info

**Gives a power retention node containing information related to retention strategy applied on an instance**

### Syntax

```
get_retention_info [get_cells]
```

### Scope

Goal

### Return Value

Returns a collection of power retention nodes.

### Description

This `get_retention_info` command gives a power retention node on which user can query the following information:

- Strategy name
- Domain name
- Retention power supply
- Retention ground supply
- Save signal
- Restore signal

This command works on instances on which retention strategy is applied.

### Arguments

None

### Examples

```
sg_shell> get_retention_info [ get_cells ]  
sg_shell> get_retention_info [ get_cells * ]  
sg_shell> get_retention_info [ get_cells top.inst1.buf1 ]
```

**See Also**

*name, power\_domain, retention\_ground\_supply, retention\_power\_supply, save\_signal, restore\_signal, report\_retention\_info*

## get\_isolation\_info

**Gives a power isolation node containing information related to isolation strategy applied on an instance**

### Syntax

```
get_isolation_info [get_cells]
```

### Scope

Goal

### Return Value

Returns a collection of power isolation nodes.

### Description

This `get_isolation_info` command gives a power isolation node on which user can query the following information:

- Strategy name
- Domain name
- Source supply
- Sink supply
- Isolation power supply
- Isolation ground supply
- Clamp value
- Location
- Isolation signal
- Isolation sense

This command works on instances on which isolation strategy is applied.

### Arguments

None

## Examples

```
sg_shell> get_isolation_info [ get_cells ]  
sg_shell> get_isolation_info [ get_cells * ]  
sg_shell> get_isolation_info [ get_cells top.inst1.iso ]
```

## See Also

*name, power\_domain, isolation\_power\_net, isolation\_sense, isolation\_signal, location, clamp\_value, source, sink, report\_isolation\_info*



## get\_power\_switch\_info

Gives a power PSW node containing information related to create\_power\_switch strategy applied on an instance

### Syntax

```
get_power_switch_info [get_cells]
```

### Scope

Goal

### Return Value

Returns a collection of power PSW nodes.

### Description

This `get_power_switch_info` command gives a power PSW node on which user can query the following information:

- Strategy name
- Domain name
- Input supply
- Output supply
- Control signal

This command works on instances on which power switch strategy is applied.

### Arguments

None

### Examples

```
sg_shell> get_power_switch_info [ get_cells ]
```

```
sg_shell> get_power_switch_info [ get_cells * ]
```

```
sg_shell> get_power_switch_info [ get_cells top.inst1.psw ]
```

**See Also**

*name, power\_domain, input\_supply\_port, output\_supply\_port, control\_port, report\_power\_switch\_info*

## get\_level\_shifter\_info

**Gives a power level shifter node containing information related to level shifter strategy applied on an instance**

### Syntax

```
get_level_shifter_info [get_cells]
```

### Scope

Goal

### Return Value

Returns a collection of power level shifter nodes.

### Description

This `get_level_shifter_info` command gives a power level shifter node on which user can query the following information:

- Strategy name
- Domain name
- Source supply
- Sink supply
- Input supply
- Output supply
- Location
- Type

This command works on instances on which power level shifter strategy is applied.

### Arguments

None

### Examples

```
sg_shell> get_level_shifter_info [ get_cells ]
```

```
sg_shell> get_level_shifter_info [ get_cells * ]  
sg_shell> get_level_shifter_info [ get_cells top.inst1.ls ]
```

## See Also

*name, power\_domain, source, sink, input\_supply\_port, output\_supply\_port, location, rule, report\_level\_shifter\_info*

## get\_supply\_info

Gives a power supply node containing information related to power supply corresponding to a design net

### Syntax

```
get_supply_info [get_nets]
```

### Scope

Goal

### Return Value

Returns a collection of power supply nodes.

### Description

This `get_supply_info` command gives a power supply node on which user can query the following information:

- Supply name
- Type
- Voltage range min
- Voltage range max

This command works on design nets on that corresponds to a power supply in UPF.

### Arguments

None

### Examples

```
sg_shell> get_supply_info [ get_nets ]  
sg_shell> get_supply_info [ get_nets * ]  
sg_shell> get_supply_info [ get_nets top.vdd ]
```

### See Also

[supply\\_name](#), [rule](#), [voltage\\_range\\_min](#), [voltage\\_range\\_max](#), [report\\_supply\\_info](#)

## report\_retention\_info

**Displays the information of the power retention node**

### Syntax

```
report_retention_info <pwr_retention_node>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

The `report_retention_info` command displays the information of the power retention node that is given as an input. The following information is displayed, in columns, for an element that is being probed:

- Strategy name: Name of the strategy
- Domain name: Name of the domain as defined in power intent
- Retention power supply: Name of the retention power supply net
- Retention ground supply: Name of the retention ground supply net
- Save signal: Full Name of the retention save signal along with the value
- Restore signal: Full Name of the retention restore signal along with the value

### Arguments

The `report_retention_info` command has the following arguments:

#### **pwr\_retention\_node**

Use this argument to report the details of the retention node.

### Examples

```
sg_shell> report_retention_info [ get_retention_info [ get_cells ] ]
```

**See Also**

*name, power\_domain, retention\_ground\_supply, retention\_power\_supply, save\_signal, restore\_signal, get\_retention\_info*

## report\_isolation\_info

**Displays the information of the power isolation node**

### Syntax

```
report_isolation_info <pwr_isolation_node>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

The `report_isolation_info` command displays the information of the power isolation node that is given as an input. The following information is displayed, in columns, for an element that is being probed:

- Strategy name: Name of the strategy
- Domain name: Name of the domain as defined in power intent
- Source supply: Name of the source supply set as defined in power intent
- Sink supply: Name of the sink supply set as defined in power intent
- Isolation power supply: Name of the isolation power supply as defined in power intent
- Isolation ground supply: Name of the isolation ground supply as defined in power intent
- Clamp value: <0 | 1 | any | Z | latch> as defined in power intent
- Location: Location as defined in power intent
- Isolation signal: List of isolation signals as defined in power intent
- Isolation sense: <low | high>

### Arguments

The `report_isolation_info` command has the following arguments:



**pwr\_isolation\_node**

Use this argument to report the details of the isolation node.

**Examples**

```
sg_shell> report_isolation_info [ get_isolation_info [
get_cells ] ]
```

**See Also**

*name, power\_domain, isolation\_power\_net, isolation\_sense, isolation\_signal, location, clamp\_value, source, sink, get\_isolation\_info, report\_power\_switch\_info*

## report\_power\_switch\_info

Displays the information of the power PSW node

### Syntax

```
report_power_switch_info <pwr_psw_node>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

The `report_power_switch_info` command displays the information of the power PSW node that is given as an input. The following information is displayed, in columns, for an element that is being probed:

- Strategy name: Name of the strategy
- Domain name: Name of the domain as defined in power intent
- Input supply: Name of the input supply net
- Output supply: Name of the output supply\_net supply net
- Control signal: Full Name of the power switch control signal

### Arguments

The `report_power_switch_info` command has the following arguments:

#### `pwr_psw_node`

Use this argument to report the details of the power PSW node.

### Examples

```
sg_shell> report_power_switch_info [ get_power_switch_info [ get_cells ] ]
```

**See Also**

*name, power\_domain, input\_supply\_port, output\_supply\_port, control\_port, get\_power\_switch\_info*

## report\_level\_shifter\_info

**Displays the information of the power level shifter node**

### Syntax

```
report_level_shifter_info <pwr_level_shift_node>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

The `report_level_shifter_info` command displays the information of the power level shifter node that is given as an input. The following information is displayed, in columns, for an element that is being probed:

- Strategy name: Name of the strategy
- Domain name: Name of the domain as defined in power intent
- Source supply: Name of the source supply set
- Sink supply: Name of the sink supply set
- Input supply: Name of the input supply set
- Output supply: Name of the output supply set
- Location: Location as defined in power intent
- Type: High\_To\_Low or Low\_To\_High or Both

### Arguments

The `report_level_shifter_info` command has the following arguments:

#### **pwr\_level\_shift\_node**

Use this argument to report the details of the power level shifter node.

## Examples

```
sg_shell> report_level_shifter_info [ get_level_shifter_info  
[ get_cells ] ]
```

## See Also

*name, power\_domain, source, sink, input\_supply\_port, output\_supply\_port,  
location, rule, get\_level\_shifter\_info*

## report\_supply\_info

Displays the information of the power supply node

### Syntax

```
report_supply_info <pwr_supply_node>
```

### Scope

Goal

### Return Value

Displays a string.

### Description

The `report_supply_info` command displays the information of the power level shifter node that is given as an input. The following information is displayed, in columns, for an element that is being probed:

- Supply name: Name of the power supply
- Type: Type of the supply power/ground/pwell/nwell
- Voltage range min: The maximum voltage value of the range of the supply net
- Voltage range max: The maximum voltage value of the range of the supply net

### Arguments

The `report_supply_info` command has the following arguments:

#### `pwr_supply_node`

Use this argument to report the details of the power supply node.

### Examples

```
sg_shell> report_supply_info [ get_supply_info [ get_nets]]
```

**See Also**

*[supply\\_name](#), [rule](#), [voltage\\_range\\_min](#), [voltage\\_range\\_max](#), [get\\_supply\\_info](#)*

## SpyGlass Power Estimate and Reduce Commands

The following table describes the various Tcl commands that are a part of the SpyGlass Power Estimate and SpyGlass Power Reduce products:

<b>Command</b>	<b>Description</b>
<i>report_power_stats_for _cell</i>	Generates power attributes report for the given flat cells
<i>report_power_stats_for _reg</i>	Generates a report for given flat cells of register type with required attributes



## report\_power\_stats\_for\_cell

Generates power attributes report for the given flat cells

### Syntax

```
report_power_stats_for_cell [-cell_size_for_power]
[-internal_power] [-leakage_power] [-switching_power]
[-power_type] [-max_capacitance] [-area]
[-leakage_power_model] [-all] <cell names>
```

### Scope

Goal

### Return Value

Displays a report.

### Description

The *report\_power\_stats\_for\_cell* command is used to generate a report for the given attributes.

### Arguments

The *report\_power\_stats\_for\_cell* command has the following arguments:

#### **-cell\_size\_for\_power**

Gives the relative size of a flat cell as used for *set\_cell\_allocation*.

#### **-internal\_power**

Gives the total internal power consumed by the given flat cell.

#### **-leakage\_power**

Gives the total leakage power consumed by the given flat cell.

#### **-switching\_power**

Gives the total switching power consumed by switching on all the fan-out nets.

**-power\_type**

Gives the power contribution type of the given flat cell.

**-area**

Gives the area of corresponding library cell as defined in the technology library.

**-max\_capacitance**

Indicates the maximum capacitive load that this flat cell can drive.

**-all**

Displays all the attributes on given cells.

**<cellnames>**

It is the list of flat cell names used for report generation.

**Examples**

```
sg_shell> report_power_stats_for_cell -internal_power -  
max_capacitance [get_cells
```

**out put:**

```
-----  
cellName    max_capacitance  internal_power  
-----  
Top.F2.F1   20.0             2.23279994315817e-6  
Top.F1.F1   20.0             3.4325228170928312e-6  
Top.F4.F1   20.0             4.284173428459326e-6  
Top.F5.F1   20.0             1.155974587163655e-6  
Top.PP1.rtlc_I2  20.0           2.0269306233444695e-8  
Top.PP.N1   20.0             3.0516972060468106e-8
```

```
sg_shell> report_power_stats_for_cell -all Top.NAN11
```

**out put:**

```
-----  
cellName cell_size_for_power internal_power leakage_power  
switching_power power_type area max_capacitance  
leakage_power_model  
-----
```

```
Top.NAN11 not_defined 3.045476262286684e-7  
6.072121139233388e-10 6.84218150581728e-7 combinational  
3.444000005722046 20.0 state_dependent
```

**See Also**

[get\\_attribute](#), [get\\_cells](#)

## report\_power\_stats\_for\_reg

Generates a report for given flat cells of register type with required attributes

### Syntax

```
report_power_stats_for_reg [-cell_size_for_power]
[-internal_power] [-leakage_power] [-switching_power]
[-max_capacitance] [-area] [-leakage_power_model]
[gating_efficiency][is_clock_gated][clock_net]
[clock_net_frequency][root_clock_for_power]
[root_clock_frequency][output_frequency]
[output_capacitance][-all] <cell names>
```

### Scope

Goal

### Return Value

Displays a report.

### Description

The *report\_power\_stats\_for\_reg* command is used to generate a report for the given attributes on given registers.

### Arguments

The *report\_power\_stats\_for\_reg* command has the following arguments:

#### **-cell\_size\_for\_power**

Gives the relative size of a flat cell used in `set_cell_allocation`.

#### **-internal\_power**

Gives the total internal power consumed by the given flat cell.

#### **-leakage\_power**

Gives the total leakage power consumed by the given flat cell.

## Debug Commands

**-switching\_power**

Gives the total switching power consumed by switching on all the fan-out nets.

**-area**

Gives the area of corresponding library cell as defined in the technology library.

**-max\_capacitance**

Indicates the maximum capacitive load that this flat cell can drive.

**-root\_clock\_for\_power**

Gives the root clock name for the given register flat cell.

**-root\_clock\_frequency**

Gives the frequency of the root clock for a register flat cell.

**-clock\_net**

Gives the immediate net connected to clock pin of a register flat cell.

**-clock\_net\_frequency**

Gives the frequency of the immediate net connected to clock pin of a register flat cell.

**-output\_frequency**

Gives the frequency of immediate net connected to the output of a register flat cell.

**-output\_capacitance**

Gives the total load (wire and pin capacitances) on the output of register flat cell.

**-gating\_efficiency**

Gives the efficiency of gating for a particular register flat cell. It is indicative of how good the gating is on a particular register.

It is equivalent to:

$$(1 - (\text{clock\_net\_frequency}/\text{root\_clock\_frequency})) * 100$$

### **-is\_clock\_gated**

Shows whether a register is gated or not.

### **-all**

Show all the attributes on given cells.

### **<reg\_names>**

It is the list of register names used for report generation.

## **Examples**

```
sg_shell> report_power_stats_for_reg -internal_power -
max_capacitance [get_cells]
```

**out put:**

```
-----
cellName    max_capacitance  internal_power
-----
Top.F2.F1   20.0             2.23279994315817e-6
Top.F1.F1   20.0             3.4325228170928312e-6
Top.F4.F1   20.0             4.284173428459326e-6
Top.F5.F1   20.0             1.155974587163655e-6
Top.PP1.rtlc_I2  20.0           2.0269306233444695e-8
Top.PP.N1   20.0             3.0516972060468106e-8
```

```
sg_shell> report_power_stats_for_reg -internal_power
Top.F5.F1
```

**out put:**

```
-----
cellName    internal_power
-----
Top.F5.F1   1.155974587163655e-6
```

---

Debug Commands

## See Also

[get\\_attribute](#), [get\\_cells](#)

## Built-in Attributes

Built-in application attributes are classified on the basis of the following object classes:

- *lib*
- *lib\_cell*
- *lib\_pin*
- *lib\_timing\_arcs*
- *cdc\_conv\_signal\_node*
- *cdc\_conv\_node*
- *cdc\_glitch\_node*
- *cdc\_glitch\_source\_node*
- *cdc\_node*
- *cdc\_source\_node*
- *design*
- *du\_cell*
- *du\_pin*
- *du\_port*
- *du\_net*
- *flat\_inst*
- *flat\_cell*
- *flat\_pin*
- *flat\_port*
- *flat\_net*
- *adc\_node*
- *sdc\_node*
- *clock*
- *clock\_domain*
- *message*
- *rule*



---

**Debug Commands**

- *reset*
- *reset\_flop\_node*
- *reset\_sync\_node*
- *paths\_node* (no attributes)

Refer to [List of Built-in Attributes](#) for the complete list of built-in application attributes defined in SpyGlass.

## lib

The following table describes the various application attributes that are a part of the *lib* object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given library. For example, for the <code>mylib_20c</code> library, <i>full_name</i> is <code>mylib_20c</code>
base_name	string	Basic name of a given library. For a <i>lib</i> object, <i>full_name</i> and <i>base_name</i> are same
file_name	string	File path from which the library is read
default_cell_leakage_power	float	Default cell leakage power as defined in the library
default_fanout_load	float	Default fan-out load as defined in the library
default_inout_pin_cap	float	Default capacitance of an inout pin as defined in the library
default_input_pin_cap	float	Default capacitance of an input pin as defined in the library
default_leakage_power_density	float	Default leakage power density as defined in the library
default_max_capacitance	float	Default maximum capacitance as defined in the library
default_max_fanout	float	Default maximum fan-out as defined in the library
default_max_transition	float	Default maximum transition as defined in the library
default_output_pin_cap	float	Default capacitance of an output pin as defined in the library
default_wire_load_area	float	Default wire load area as defined in the library
default_wire_load_capacitance	float	Default wire load capacitance as defined in the library

## Debug Commands

<b>Attribute Name</b>	<b>Type</b>	<b>Description</b>
default_wire_load_resistance	float	Default wire load resistance as defined in the library
nom_process	float	Process of the design as defined in the library
nom_temperature	float	Temperature of the design as defined in the library
nom_voltage	float	Voltage of the design as defined in the library
time_unit	string	Time unit used in the library
voltage_unit	string	Voltage unit used in the library
current_unit	string	Current unit used in the library
capacitive_load_unit	string	Capacitive load unit used in the library
pulling_resistance_unit	string	Pulling resistance unit used in the library
leakage_power_unit	string	Leakage power unit used in the library
default_connection_class	string	Default value of the connection class in the library
default_operating_conditions	string	Default value of the operating conditions in the library
default_power_rail	string	Default power rail in the library
default_threshold_voltage_group	string	Default threshold voltage group in the library
default_wire_load	string	Default value of the wire load in the library
default_wire_load_mode	string	Default value of the wire load mode in the library
default_wire_load_selection	string	Default value of the wire load selection in the library
define_cell_area	string	Definition of library cell area inside the library
delay_model	string	Delay model used in this library
technology	string	Technology used in the library

Attribute Name	Type	Description
object_class	string	Class of a given object. In this case, it is always <code>lib</code>
sglib_name	string	Name of the sglib that contains the object

## lib\_cell

The following table describes the various application attributes that are a part of the `lib_cell` object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given library cell. For example, for the <code>AN2</code> cell of the <code>mylib_20c</code> library, <code>full_name</code> is <code>mylib_20c.AN2</code>
base_name	string	Basic name of a given library cell. For example, for the <code>AN2</code> cell of the <code>mylib_20c</code> library, <code>base_name</code> is <code>AN2</code>
file_name	string	File path from which the library cell is read
line_num	int	Line number of the file where the library cell definition is started
is_flop	boolean	Value is <code>true</code> if the library cell is a flip-flop, otherwise the value is <code>false</code>
is_latch	boolean	Value is <code>true</code> if the library cell is a latch, otherwise the value is <code>false</code>
is_sequential	boolean	Value is <code>true</code> if the library cell is a sequential cell
is_combinational	boolean	Value is <code>true</code> if the library cell is not a sequential cell
is_level_shifter	boolean	Value is <code>true</code> if the library cell is a level shifter cell

## Debug Commands

Attribute Name	Type	Description
is_clock_gating_cell	boolean	Value is <code>true</code> if the library cell is a clock gating cell
is_macro_cell	boolean	Value is <code>true</code> if the lib cell is a macro cell
is_memory_cell	boolean	Value is <code>true</code> if the library cell is a memory cell
is_pad_cell	boolean	Value is <code>true</code> if the library cell is used as an I/O pad cell
is_three_state	boolean	Value is <code>true</code> if the library cell is a three-state device
is_mux	boolean	Value is <code>true</code> if the library cell is a multiplexer
is_isolation_cell	boolean	Value is <code>true</code> if the library cell is an isolation cell
dont_touch	boolean	The <code>true</code> value indicates that this library cell is not optimized during compilation
dont_use	boolean	The <code>true</code> value indicates that this library cell is not used in the design
number_of_pins	int	Number of pins of the library cell
area	float	Area of the library cell
object_class	string	Class of a given object. In this case, it is always <code>lib_cell</code>
sglib_name	string	Name of the sglib that contains the object
always_on	boolean	Value is <code>true</code> if the library cell is always on
input_voltage_range	string	Returns the value of input voltage range of the library cell
level_shifter_type	string	Indicates the level shifter type of the library cell
output_voltage_range	string	Returns the value of <code>output_voltage_range</code> of the library cell
retention_cell	string	Indicates the type of retention register
switch_cell_type	string	Indicates the switch cell type of the library cell

## lib\_pin

The following table describes the various application attributes that are a part of the *lib\_pin* object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given library pin. For example, for the A pin of the AN2 cell of the mylib_20c library, <i>full_name</i> is mylib_20c.AN2.A
base_name	string	Basic name of a given library pin. For example, for the A pin of the AN2 cell of the mylib_20c library, <i>base_name</i> is A
direction	string	Direction of the library pin
fanout_load	float	Fan-out load of an input library pin
max_capacitance	float	Indicates the maximum capacitive load that this output or inout library pin can drive
min_capacitance	float	Indicates the minimum capacitive load that this output or inout library pin can drive.
capacitance	float	Capacitance of the library pin
max_fanout	float	Maximum fan-out load that an output pin can drive
min_fanout	float	Minimum fan-out load that an output pin can drive
max_transition	float	Maximum acceptable transition time of a library pin
min_transition	float	Minimum acceptable transition time of a library pin
function	string	Function of the library pin relative to input pins
connection_class	string	Indicates which pins to connect to the pins of other cells
driver_type	string	Indicates the driver information (pullup or pulldown) of a library pin

## Debug Commands

Attribute Name	Type	Description
x_function	string	Indicates the condition relative to the input pins for which the output is at the X state
memory_read	boolean	The <code>true</code> value indicates that it is a memory read pin of a memory cell
memory_write	boolean	The <code>true</code> value indicates that it is a memory write pin of a memory cell
clock_gate_clock_pin	boolean	Indicates whether the library pin is a clock pin of a clock gating cell
clock_gate_enable_pin	boolean	Indicates whether the library pin is the enable pin of a clock gating cell
clock_gate_obs_pin	boolean	Indicates whether the output pin of a clock gating cell is connected to an observability signal
clock_gate_out_pin	boolean	Indicates whether the library pin is an output pin of a clock gating cell
clock_gate_test_pin	boolean	Indicates whether the input pin of a clock gating cell is connected to a <i>test_mode</i> signal
is_preset_pin	boolean	Indicates whether the pin is a preset pin of a library cell
is_clear_pin	boolean	Indicates whether the pin is a clear pin of a library cell
is_data_pin	boolean	Indicates whether the pin is a data pin of a library cell
is_clock_pin	boolean	Indicates whether the pin is a clock pin of a library cell
is_mux_select_pin	boolean	Indicates whether the pin is a select pin of a multiplexer
is_pad	boolean	Indicates whether the pin is a pad pin of a pad cell
is_enable_pin	boolean	Indicates whether the pin is an enable pin of a latch
is_load_pin	boolean	Indicates whether the pin is a load pin of a library cell

Attribute Name	Type	Description
is_async_pin	boolean	Indicates whether the pin is an asynchronous input pin of a library cell
is_three_state_enable_pin	boolean	The <code>true</code> value indicates it is an enable pin of a three-state device
is_three_state_output_pin	boolean	The <code>true</code> value indicates it is an output pin of a three-state device
is_vector	boolean	Indicates whether the pin is a vector (bus) pin of a library cell
bus_width	int	Bus width of the bus containing the library pin
lsb	int	LSB of the bus containing the library pin
msb	int	MSB of the bus containing the library pin
index	int	Indicates the index of the bus pin
object_class	string	Class of a given object. In this case, it is always <i>lib_pin</i>
sglib_name	string	Name of the sglib that contains the object
related_power_pin	collection	Related power pin corresponding to the library pin
related_ground_pin	collection	Related ground pin corresponding to the library pin
pg_type	string	Returns the <code>pg_type</code> of the <code>pg_pin</code> as defined in the library
is_power_pin	boolean	Indicates whether the library pin is a power pin of a cell
is_ground_pin	boolean	Indicates whether the library pin is a ground pin of a cell
is_pg_pin	boolean	Indicates whether the library pin is a power or ground pin of a cell
always_on	boolean	Value is <code>true</code> if the library pin is always on
input_voltage_range	string	Returns the value of input voltage range of the library pin



## Debug Commands

Attribute Name	Type	Description
isolation_cell_data_pin	boolean	Returns the value of isolation_cell_data_pin as defined on the library pin
isolation_cell_enable_pin	boolean	Returns the value of isolation_cell_enable_pin as defined on the library pin
level_shifter_data_pin	boolean	Returns the value of level_shifter_data_pin as defined on the library pin
level_shifter_enable_pin	boolean	Returns the value of level_shifter_enable_pin as defined on the library pin
output_voltage_range	string	Returns the value of output_voltage_range of the library pin
pg_function	string	Returns the pg_function of the pg_pin as defined in the library
power_down_function	string	Returns the value of power_down_function of the library pin
retention_pin	string	Returns the value of retention_pin as defined on the library pin
std_cell_main_rail	boolean	Returns the value of std_cell_main_rail as defined on the library pin
switch_function	string	Returns the value of switch_function as defined on the library pin
switch_pin	boolean	Indicates whether the library pin is a switch_pin or not
voltage_value	float	Returns the voltage value of the pg_pin
is_isolated	boolean	Returns the value of is_isolated as defined on the library pin

## lib\_timing\_arcs

The following table describes the various application attributes that are a part of the *lib\_timing\_arcs* object class.

Attribute Name	Type	Description
from_lib_pin	collection	The library pin from which library timing arc originates
to_lib_pin	collection	The library pin at which library timing arc ends
timing_type	string	Timing type of the library timing arc
timing_sense	string	Timing sense of the library timing arc
sdf_cond	string	Shows the Standard Delay Format condition related to the library timing arc
object_class	string	Class of a given object. In this case, it is always <i>lib_timing_arcs</i>
sglib_name	string	Name of the sglib that contains the object

## cdc\_conv\_signal\_node

The following table describes the various application attributes that are a part of the *cdc\_conv\_signal\_node* object class.

Attribute Name	Type	Description
dest_clock_tag	char	Returns the destination clock's tag
dest_clocks	char	Returns the destination clock's information
dest_file_line	char	Returns the file line information of the destination
dest_name	char	Returns the name of destination
diverging_nets	char	Gets the divergent point of a convergent signal
source_clock_tag	char	Returns the source clock's tag name
source_clocks	char	Returns the source clocks
source_names	char	Returns the source name
seq_depth	int	Gets the count of sequential cells from a destination to convergent point

## cdc\_conv\_node

The following table describes the various application attributes that are a

part of the *cdc\_conv\_node* object class.

Attribute Name	Type	Description
is_comb_conv	boolean	Checks the presence of Ac_conv02 rule violation
is_nonconv_bus	boolean	Checks the presence of Ac_conv04 rule violation
is_seq_conv	boolean	Checks the presence of Ac_conv01 rule violation
is_user_defined	boolean	Checks the presence of Ac_conv05 rule violation
cdc_rule_name	boolean	Returns the rule name which the convergence reported
conv_gate	boolean	Returns converging gate
is_graycoded	boolean	Checks if the converging signals are grey encoded
status	boolean	Returns the status
sync_count	boolean	Returns the number of synchronizers
num_source_domains	int	Checks the number of source domains in a crossing
num_sources	int	Checks the number of sources in a crossing

## cdc\_glitch\_node

The following table describes the various application attributes that are a part of the *cdc\_glitch\_node* object class.

Attribute Name	Type	Description
has_destination_domain	boolean	Returns those glitch crossings which have destination domain signal
has_multi_domain_sources	boolean	Returns those glitch crossings which have multi-domain sources
has_reconvergent_sources	boolean	Returns those glitch crossings which have reconvergent sources
cdc_rule_name	char	Returns the rule name
dest_clock_tag	char	Returns the destination clock's tag name

Attribute Name	Type	Description
dest_clocks	char	Returns the destination clocks
dest_domain	char	Returns the destination domain
dest_name	char	Returns the destination name
failure_reason	char	Checks the crossing synchronization failure reason
multi_source_glitch_check	char	Return those glitch crossings which have gray-encoding check status as PASSED, FAILED or unknown
status	char	Return the status
dest_internal_domain_id	int	Returns the destination domain id
num_source_domains	int	Checks the number of source domains in a crossing
num_sources	int	Checks the number of sources in a crossing

## cdc\_glitch\_source\_node

The following table describes the various application attributes that are a part of the *cdc\_glitch\_source\_node* object class.

Attribute Name	Type	Description
is_async	boolean	Checks whether crossing is asynchronous
is_reconv	boolean	Checks whether crossing is asynchronous
source_clock_tag	char	Returns the source clock's tag name
source_clocks	char	Returns the source clocks
source_domain	char	Returns the source domain
source_name	char	Returns the source name
source_type	char	Checks the source type of a single source as reported by Ac_sync_group rules. It can have values: flip-flop, library-cell, latch, primary input, black-box
source_internal_domain_id	int	Returns the source domain id

## cdc\_node

The following table describes the various application attributes that are a part of the *cdc\_node* object class.

Attribute Name	Type	Description
is_data	boolean	Checks if crossing is a data crossing
is_synchronized	boolean	Checks if crossing is synchronized or not
cdc_rule_name	char	Returns the rule name
crossing_module_name	char	Checks the crossing module name
dest_clock_tag	char	Returns the destination clock's tag name
dest_clocks	char	Returns the destination clocks
dest_domain	char	Returns the destination domain
dest_file_line	char	Returns the file line information of the destination
dest_module_name	char	Gets the module name of the destination of the crossing
dest_name	char	Returns the destination name
dest_parent_instance	char	Checks the crossing destination parent instance
dest_type	char	Checks the destination type as reported by Ac_sync_group rules. It can have values: [flop,library-cell,latch,primary-output,black-box]
failure_reason	char	Checks the crossing synchronization failure reason
multi_flop_synchronizer_names	char	Checks the multi flip-flop synchronizer information of a crossing
overall_failure_reason	char	Checks the crossing synchronization failure reason
overall_synch_scheme	char	Returns the overall synchronization scheme
potential_qualifier_name	char	Checks the qualifier of a crossing

Attribute Name	Type	Description
src_type	char	Checks the source type as reported by Ac_sync_group rules. It can have values: flip-flop, library-cell, latch, primary input, black-box
sync_method	char	Check the crossing synchronization method
dest_internal_domain_id	int	Returns the destination domain id
multi_flop_synchronizer_stages	int	Checks the number of flip-flip flops used to synchronize the crossing
num_source_domains	int	Checks the number of source domains in a crossing
num_sources	int	Checks the number of sources in a crossing
dest_objects	collection	Destination net objects of the crossing
multi_flop_synchronizer_objects	collection	Module name of the destination of the crossing

## cdc\_source\_node

The following table describes the various application attributes that are a part of the *cdc\_source\_node* object class.

Attribute Name	Type	Description
source_clock_tag	char	Returns the source clock tag name
source_clocks	char	Returns the source clocks
source_domain	char	Returns the source domain
source_failure_reason	char	Checks the crossing source failure reason
source_file_line	char	Returns the file line information of the source
source_name	char	Returns the source name
source_parent_instance_name	char	Checks the crossing source parent instance
source_qualifier_name	char	Checks the crossing source qualifier name
source_sync_scheme	char	Checks the crossing source sync scheme

Attribute Name	Type	Description
source_type	char	Checks the source type of a single source as reported by Ac_sync_group rules. It can have values: flip-flop, library-cell, latch,primary input, black-box
source_internal_domain_id	int	Returns the source domain id
source_qualifier_depth	int	Checks the crossing source qualifier depth

## design

The following table describes the various application attributes that are a part of the *design* object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given design
base_name	string	Basic name of a given design
file_name	string	File path from which the design is read
line_num	int	Line number of the file where the design definition starts
is_flop	boolean	Value is <code>true</code> if the design is a flip-flop, otherwise the value is <code>false</code>
is_latch	boolean	Value is <code>true</code> if the design is a latch, otherwise the value is <code>false</code>
is_sequential	boolean	Value is <code>true</code> if the design is a sequential cell
is_combinational	boolean	Value is <code>true</code> if the design is not a sequential cell
is_level_shifter	boolean	Value is <code>true</code> if the design is a level shifter module
is_clock_gating_cell	boolean	Value is <code>true</code> if the design is a clock gating cell

Attribute Name	Type	Description
is_memory_cell	boolean	Value is <code>true</code> if the design is a memory cell
is_pad_cell	boolean	Value is <code>true</code> if the design is used as an I/O pad cell
is_three_state	boolean	Value is <code>true</code> if the design is a three-state device
is_mux	boolean	Value is <code>true</code> if the design is a multiplexer
number_of_pins	int	Number of ports of a design
rtl_name	string	RTL name of the design
language	string	Language in which the design is written
is_primitive	boolean	The <code>true</code> value indicates that the design is a primitive gate type
is_macro	boolean	The <code>true</code> value indicates that the design is a hard macro
is_user_module	boolean	The <code>true</code> value indicates that the definition of this design is supplied by the user
is_lib	boolean	The <code>true</code> value indicates that the design is a library cell
is_blackbox	boolean	The <code>true</code> value indicates that the design is a black box
is_hierarchical	boolean	The <code>true</code> value indicates that the design is hierarchical
is_leaf	boolean	The <code>true</code> value indicates that the design is leaf level cell
object_class	string	Class of a given object. In this case, it is always <i>design</i>
is_stop	boolean	The <code>true</code> value indicates that the design has been stopped with the <i>stop</i> project file command
is_celldefine	boolean	The <code>true</code> value indicates that the design is defined as <code>celldefine</code>
is_empty	boolean	The <code>true</code> value indicates that the design has no logical definition inside its interface definition



Attribute Name	Type	Description
is_top	boolean	The <code>true</code> value indicates that the design is inferred or set as a top module
level_shifter_type	string	Indicates the level shifter type of the design
switch_cell_type	string	Indicates the switch cell type of the design

## du\_cell

The following table describes the various application attributes that are a part of the *du\_cell* object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given du cell. For example, for the <code>inst_IV</code> cell, <i>full_name</i> is <code>inst_IV</code>
base_name	string	Basic name of a given du cell. For <i>du_cell</i> , <i>full_name</i> and <i>base_name</i> are same
master_name	string	Master module name of a given du cell
file_name	string	File path from which the du cell is read
line_num	int	Line number of the file where the du cell is defined
number_of_pins	int	Number of pins of a du cell
is_hierarchical	boolean	The <code>true</code> value means the design is not a leaf level cell
is_leaf	boolean	The <code>true</code> value means the design is a leaf level cell
is_user_module	boolean	The <code>true</code> value indicates that the definition of this du cell is supplied by the user
object_class	string	Class of a given object. In this case, it is always <i>du_cell</i>
is_blackbox	boolean	The <code>true</code> value indicates that the du cell is a black box

Attribute Name	Type	Description
is_stop	boolean	The <code>true</code> value indicates that the du cell has been stopped with the <code>stop</code> project file command
is_celldefine	boolean	The <code>true</code> value indicates that the du cell is defined as <code>celldefine</code>
is_empty	boolean	The <code>true</code> value indicates that the du cell has no logical definition inside its interface definition
is_top	boolean	The <code>true</code> value indicates that the du cell is inferred or set as a top module
level_shifter_type	string	Indicates the level shifter type of the du cell
retention_cell	string	Indicates the type of retention register
switch_cell_type	string	Indicates the switch cell type of the du cell

The following table describes the application attributes that are a part of the `du_cell` object class and that give meaningful results only for the leaf-level cells.

Attribute Name	Type	Description
is_flop	boolean	Value is <code>true</code> if the du cell is a flip-flop, otherwise the value is <code>false</code>
is_latch	boolean	Value is <code>true</code> if the du cell is a latch, otherwise the value is <code>false</code>
is_sequential	boolean	Value is <code>true</code> if the du cell is a sequential cell
is_combinational	boolean	Value is <code>true</code> if the du cell is a pure combinational cell
is_level_shifter	boolean	Value is <code>true</code> if the du cell is used in the design module as a level shifter cell
is_clock_gating_cell	boolean	Value is <code>true</code> if the du cell is a clock gating cell
is_macro_cell	boolean	Value is <code>true</code> if the du cell is a macro cell

## Debug Commands

Attribute Name	Type	Description
is_memory_cell	boolean	Value is <code>true</code> if the du cell is a library memory cell instance
is_pad_cell	boolean	Value is <code>true</code> if the du cell is used as an I/O pad cell
is_three_state	boolean	Value is <code>true</code> if the du cell is a three-state device
is_mux	boolean	Value is <code>true</code> if the du cell is a multiplexer
is_isolation_cell	boolean	Value is <code>true</code> if the du cell is an isolation cell
dont_touch	boolean	The <code>true</code> value indicates that the du cell is an instance of a library cell with the <i>dont_touch</i> attribute
dont_use	boolean	The <code>true</code> value indicates that the du cell is an instance of a library cell with the <i>dont_use</i> attribute
area	float	Area of the du cell
is_primitive	boolean	The <code>true</code> value indicates that the du cell is a primitive gate type
is_macro	boolean	The <code>true</code> value indicates that the du cell is a design module macro
is_lib	boolean	The <code>true</code> value indicates that the du cell is an instance of a library cell

## du\_pin

The following table describes the various application attributes that are a part of the *du\_pin* object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given du pin. For example, for the A pin of the <code>inst1</code> cell, <i>full_name</i> is <code>inst1.A</code>
base_name	string	Basic name of a given du pin. For example, for the A pin of the <code>inst1</code> cell, <i>base_name</i> is <code>A</code>
file_name	string	File path from which the du pin is read
line_num	int	Line number of the file where the du pin is defined
direction	string	Direction (input/output/inout) of a du pin
is_vector	boolean	Indicates whether the pin is a vector (bus) pin of a du cell
bus_width	int	Bus width of the bus containing this du pin
lsb	int	LSB of the bus containing this du pin
msb	int	MSB of the bus containing this du pin
object_class	string	Class of a given object. In this case, it is always <i>du_pin</i>
related_power_pin	collection	Related power pin corresponding to the du pin
related_ground_pin	collection	Related ground pin corresponding to the du pin
is_power_pin	boolean	Indicates whether the du pin is a power pin of a cell
is_ground_pin	boolean	Indicates whether the du pin is a ground pin of a cell
is_pg_pin	boolean	Indicates whether the du pin is a power or ground pin of a cell
isolation_cell_data_pin	boolean	Returns the value of <code>isolation_cell_data_pin</code> as defined on the du pin
isolation_cell_enable_pin	boolean	Returns the value of <code>isolation_cell_enable_pin</code> as defined on the du pin

## Debug Commands

Attribute Name	Type	Description
level_shifter_data_pin	boolean	Returns the value of level_shifter_data_pin as defined on the du pin
level_shifter_enable_pin	boolean	Returns the value of level_shifter_enable_pin as defined on the du pin
power_down_function	string	Returns the value of power_down_function of the du pin
switch_function	string	Returns the value of switch_function as defined on the du pin
switch_pin	boolean	Indicates whether the du pin is a switch_pin or not
is_isolated	boolean	Returns the value of is_isolated as defined on the du pin

The following table describes the application attributes that are a part of the *du\_pin* object class and that give meaningful results only for the leaf-level pins.

Attribute Name	Type	Description
memory_read	boolean	The <code>true</code> value indicates a memory read pin of a memory cell
memory_write	boolean	The <code>true</code> value indicates a memory write pin of a memory cell
clock_gate_clock_pin	boolean	Indicates whether the du pin is a clock pin of a clock gating cell
clock_gate_enable_pin	boolean	Indicates whether the du pin is the enable pin of a clock gating cell
clock_gate_obs_pin	boolean	Indicates whether the output pin of a clock gating cell is connected to an observability signal
clock_gate_out_pin	boolean	Indicates whether the du pin is the output pin of a clock gating cell
clock_gate_test_pin	boolean	Indicates whether the input pin of a clock gating cell is connected to a <i>test_mode</i> signal

Attribute Name	Type	Description
is_preset_pin	boolean	Indicates whether the pin is a preset pin of a du cell
is_clear_pin	boolean	Indicates whether the pin is a clear pin of a du cell
is_data_pin	boolean	Indicates whether the pin is a data pin of a du cell
is_clock_pin	boolean	Indicates whether the pin is a clock pin of a du cell
is_mux_select_pin	boolean	Indicates whether the pin is a select pin of a multiplexer
is_pad	boolean	Indicates whether the pin is a pad pin of a pad cell
is_enable_pin	boolean	Indicates whether the pin is an enable pin of a latch
is_load_pin	boolean	Indicates whether the pin is a load pin of a du cell
is_async_pin	boolean	Indicates whether the pin is an asynchronous input pin of a du cell
is_three_state_enable_pin	boolean	The <code>true</code> value indicates that it is an enable pin of a three-state device
is_three_state_output_pin	boolean	The <code>true</code> value indicates that it is an output pin of a three-state device

## du\_port

The following table describes the various application attributes that are a part of the *du\_port* object class.

## Debug Commands

Attribute Name	Type	Description
full_name	string	Bit-blasted name of a given port of a design or module. For example, for the $\bar{A}$ bus port that contains two bits, <i>full_name</i> is { $\bar{A}[0]$ , $\bar{A}[1]$ }
base_name	string	Basic name of a given port of a design or module. For example, for the $\bar{A}$ bus port that contains two bits, <i>base_name</i> is $\bar{A}$
file_name	string	File path from which the du port is read
line_num	int	Line number of the file where the du port is defined
direction	string	Direction of the du port
is_vector	boolean	Indicates whether the port is a vector (bus) pin of a design or module
bus_width	int	Bus width of the bus containing this du port
lsb	int	LSB of the bus containing this du port
msb	int	MSB of the bus containing this du port
object_class	string	Class of a given object. In this case, it is always <i>du_port</i>

## du\_net

The following table describes the various application attributes that are a part of the *du\_net* object class.

Attribute Name	Type	Description
full_name	string	Bit-blasted name of a given net of a design or module. For example, for the $\bar{A}$ net that contains two bits, <i>full_name</i> is { $\bar{A}[0]$ , $\bar{A}[1]$ }
base_name	string	Basic name of a given net of a design or a module. For example, for the $\bar{A}$ net that contains two bits, <i>base_name</i> is $\bar{A}$

Attribute Name	Type	Description
file_name	string	File path from which the du net is read
line_num	int	Line number of the file where the du net is defined
direction	string	Direction of the du net
is_vector	boolean	Indicates whether the net is a vector net
bus_width	int	Number of bits of the vector net
lsb	int	LSB of the vector net
msb	int	MSB of the vector net
object_class	string	Class of a given object. In this case, it is always <i>du_net</i>
is_generated	boolean	The <code>true</code> value indicates that the given du net has been generated internally
is_record	boolean	The <code>true</code> value indicates that the given du net is a record net
is_multidim	boolean	The <code>true</code> value indicates that the given du net is a multidimensional net

## flat\_inst

The following table describes the various application attributes that are a part of the *flat\_inst* object class.

Attribute Name	Type	Description
sync_module_name	char	Gets the module name of a synchronizer flip-flop.
sync_name	char	Gets the output net name of a synchronizer flip-flop.



## flat\_cell

The following table describes the various application attributes that are a part of the *flat\_cell* object class.

Attribute Name	Type	Description
full_name	string	Full qualified name of a given flat cell. For example, after flattening of a design, <i>full_name</i> of a cell may be <code>top.inst_mid1.inst_b1.rtlc_I1</code>
base_name	string	Basic name of a given flat cell. For example, after flattening of a design, <i>base_name</i> of the same cell is <code>rtlc_I1</code>
master_name	string	Master module name of a given flat cell
file_name	string	File path from which the flat cell is read
line_num	int	Line number of the file where the flat cell instance is declared
is_flop	boolean	Value is <code>true</code> if the flat cell is a flip-flop, otherwise the value is <code>false</code>
is_latch	boolean	Value is <code>true</code> if the flat cell is a latch, otherwise the value is <code>false</code>
is_sequential	boolean	Value is <code>true</code> if the flat cell is a sequential cell
is_combinational	boolean	Value is <code>true</code> if the flat cell is a pure combinational cell
is_level_shifter	boolean	Value is <code>true</code> if the flat cell is used in the design as a level shifter cell
is_clock_gating_cell	boolean	Value is <code>true</code> if the flat cell is a clock gating cell
is_macro_cell	boolean	Value is <code>true</code> if the flat cell is a macro cell
is_memory_cell	boolean	Value is <code>true</code> if the flat cell is a library memory cell instance

Attribute Name	Type	Description
is_pad_cell	boolean	Value is <code>true</code> if the flat cell is used as an I/O pad cell
is_three_state	boolean	Value is <code>true</code> if the flat cell is a three-state device
is_mux	boolean	Value is <code>true</code> if the flat cell is a multiplexer
is_isolation_cell	boolean	Value is <code>true</code> if the flat cell is an isolation cell
dont_touch	boolean	The <code>true</code> value indicates the flat cell instance of a library cell with the <code>dont_touch</code> attribute
dont_use	boolean	The <code>true</code> value indicates the flat cell instance of a library cell with the <code>dont_use</code> attribute
number_of_pins	int	Number of pins of the flat cell
area	float	Area of the flat cell
is_lib	boolean	The <code>true</code> value indicates the flat cell is an instantiation of a library cell
is_blackbox	boolean	The <code>true</code> value indicates the flat cell is a black box
is_hierarchical	boolean	The <code>false</code> value indicates the flat cell is not a hierarchical cell. This is always <code>false</code>
is_leaf	boolean	The <code>true</code> value indicates the flat cell is a leaf level flat cell. This is always <code>true</code>
path_name	string	Full path name of a given flat cell
object_class	string	Class of a given object. In this case, it is always <code>flat_cell</code>
is_stop	boolean	The <code>true</code> value indicates that the flat cell has been stopped with the <code>stop</code> project file command
is_celldefine	boolean	The <code>true</code> value indicates that the flat cell is defined as <code>celldefine</code>

## Debug Commands

Attribute Name	Type	Description
is_empty	boolean	The <code>true</code> value indicates that the flat cell has no logical definition inside its interface definition
is_top	boolean	The <code>true</code> value indicates that the flat cell is inferred or set as a top module
clocks	collection	When this attribute is retrieved on a <i>flat_cell</i> , it returns a collection of clocks through which the cell is driven
domain	collection	When this attribute is retrieved on a <i>flat_cell</i> , it returns a collection of domains of clocks through which the cell is driven
level_shifter_type	string	Indicates the level shifter type of the flat cell
retention_cell	string	Indicates the type of retention register
switch_cell_type	string	Indicates the switch cell type of the flat cell
standard_sdc_name	string	Provides the un-changed standard SDC names of a given flat cell and are available only after SDC files/commands have been read. For example, after flattening of a design, <code>standard_sdc_name</code> of a cell may be <code>{ IF1.FOR1[0].genblk1.genblk1[0].IF2.inst/fifo_out_reg}</code> .

Attribute Name	Type	Description
non_standard_sdc_name	string	<p>Provides the non-standard SDC names of a given net of a design or module without taking into account effect of <code>define_name_rules/change_names</code> SDC commands, but difference in name from <code>standard_sdc</code> is only visible if <code>allow_non_standard_sdc</code> is specified which works with the following options:</p> <ul style="list-style-type: none"> <li>• <code>set_option use_if_generate_prefix {true/false}</code></li> <li>• <code>set_option use_for_generate_prefix {true/false}</code></li> <li>• <code>set_option unlabeled_generate_prefix {true/false}</code></li> </ul> <p>For example, after flattening of a design, <code>non_standard_sdc_name</code> of a cell may be { IF1.FOR1[0].IF2.inst.[0]/fifo_out_reg }.</p>
modified_sdc_name	string	<p>Provides the SpyGlass names after the combined effect of attribute <code>non_standard_sdc_name</code> and <code>define_name_rules/change_names</code> SDC commands, and are available only after SDC files/commands have been read</p> <p>For example, <code>change_name</code> rules { "inst", "I" } is specified in sdc file, after flattening of a design, <code>modified_sdc_name</code> of a cell may be { IF1.FOR1[0].IF2.I.[0]/fifo_out_reg }.</p>

## flat\_pin

The following table describes the various application attributes that are a part of the *flat\_pin* object class.

## Debug Commands

Attribute Name	Type	Description
full_name	string	Full qualified name of a given flat pin. For example, for the A pin of the <code>inst1</code> cell, <i>full_name</i> is <code>inst1.A</code>
base_name	string	Basic name of a given flat pin. For example, for the A pin of the <code>inst1</code> cell, <i>base_name</i> is A
file_name	string	File path from which the flat pin is read
line_num	int	Line number of the file where the flat pin is defined
direction	string	Direction of the flat pin
memory_read	boolean	The <code>true</code> value indicates that the flat pin is a memory read pin of a memory cell
memory_write	boolean	The <code>true</code> value indicates that the flat pin is a memory write pin of a memory cell
clock_gate_clock_pin	boolean	Indicates whether the flat pin is a clock pin of a clock gating cell
clock_gate_enable_pin	boolean	Indicates whether the flat pin is an enable pin of a clock gating cell
clock_gate_obs_pin	boolean	Indicates whether this output pin of a clock gating cell is connected to an observability signal
clock_gate_out_pin	boolean	Indicates whether the flat pin is the output pin of a clock gating cell
clock_gate_test_pin	boolean	Indicates whether the input pin of a clock gating cell is connected to a <i>test_mode</i> signal
is_preset_pin	boolean	Indicates whether the pin is a preset pin of a flat cell
is_clear_pin	boolean	Indicates whether the pin is a clear pin of a flat cell
is_data_pin	boolean	Indicates whether the pin is a data pin of a flat cell
is_clock_pin	boolean	Indicates whether the pin is a clock pin of a flat cell

Attribute Name	Type	Description
is_mux_select_pin	boolean	Indicates whether the pin is a select pin of a multiplexer
is_pad	boolean	Indicates whether the pin is a pad pin of a pad cell
is_enable_pin	boolean	Indicates whether the pin is an enable pin of a latch
is_load_pin	boolean	Indicates whether the pin is a load pin of a flat cell
is_async_pin	boolean	Indicates whether the pin is an asynchronous input pin of a flat cell
is_three_state_enable_pin	boolean	The <code>true</code> value indicates the pin is an enable pin of a three-state device
is_three_state_output_pin	boolean	The <code>true</code> value indicates the pin is an output pin of a three-state device
is_vector	boolean	Indicates whether the pin is a vector (bus) pin of a flat cell
bus_width	int	Bus width of the bus containing the flat pin
lsb	int	LSB of the bus containing the flat pin
msb	int	MSB of the bus containing the flat pin
index	int	Index of the vector pin
object_class	string	Class of a given object. In this case, it is always <i>flat_pin</i>
related_power_pin	collection	Related power pin corresponding to the flat pin
related_ground_pin	collection	Related ground pin corresponding to the flat pin
is_power_pin	boolean	Indicates whether the flat pin is a power pin of a cell
is_ground_pin	boolean	Indicates whether the flat pin is a ground pin of a cell
is_pg_pin	boolean	Indicates whether the flat pin is a power or ground pin of a cell
clocks	collection	When this attribute is retrieved on a <i>flat_pin</i> , it returns a collection of clocks through which the pin is driven

## Debug Commands

Attribute Name	Type	Description
domain	collection	When this attribute is retrieved on a <i>flat_pin</i> , it returns a collection of domains of clocks through which the pin is driven
isolation_cell_data_pin	boolean	Returns the value of isolation_cell_data_pin as defined on the flat pin
isolation_cell_enable_pin	boolean	Returns the value of isolation_cell_enable_pin as defined on the flat pin.
level_shifter_data_pin	boolean	Returns the value of level_shifter_data_pin as defined on the flat pin
level_shifter_enable_pin	boolean	Returns the value of level_shifter_enable_pin as defined on the flat pin
power_down_function	string	Returns the value of power_down_function of the flat pin
switch_function	string	Returns the value of switch_function as defined on the flat pin
switch_pin	boolean	Indicates whether the flat pin is a switch_pin or not
is_isolated	boolean	Returns the value of is_isolated as defined on the flat pin
standard_sdc_name	string	Provides the un-changed standard SDC names of a given flat pin and are available only after SDC files/commands have been read. For example, for the Q pin, standard_sdc_name may be { IF1.FOR1[0].genblk1.genblk1[0].IF2.inst/fifo_out_reg/Q}.

Attribute Name	Type	Description
non_standard_sdc_name	string	<p>Provides the non-standard SDC names of a given net of a design or module without taking into account effect of <code>define_name_rules/change_names</code> SDC commands, but difference in name from <code>standard_sdc</code> is only visible if <code>allow_non_standard_sdc</code> is specified, which works with the following options:</p> <ul style="list-style-type: none"> <li>• <code>set_option use_if_generate_prefix {true/false}</code></li> <li>• <code>set_option use_for_generate_prefix {true/false}</code></li> <li>• <code>set_option unlabeled_generate_prefix {true/false}</code></li> </ul> <p>For example, for the A pin of the cell, <code>non_standard_sdc_name</code> is <code>{ IF1.FOR1[0].IF2.inst.[0]/fifo_out_reg/Q }</code>.</p>
modified_sdc_name	string	<p>Provides the SpyGlass names after the combined effect of attribute <code>non_standard_sdc_name</code> and <code>define_name_rules/change_names</code> SDC commands, and are available only after SDC files/commands have been read</p> <p>For example, <code>change_name rules { "inst", "I" }</code> is specified in sdc file, for the Q pin of the cell, <code>modified_sdc_name</code> may be <code>{ IF1.FOR1[0].IF2.I.[0]/fifo_out_reg/Q }</code>.</p>

## flat\_port

The following table describes the various application attributes that are a part of the *flat\_port* object class.



## Debug Commands

Attribute Name	Type	Description
full_name	string	Bit-blasted name of a given port of a design or module. For example, for A bus port that contains two bits, <i>full_name</i> is { A [ 0 ] , A [ 1 ] }
base_name	string	Basic name of a given port of a design or module. For example, for A bus port that contains two bits, <i>base_name</i> is A
file_name	string	File path from which the flat port is read
line_num	int	Line number of the file where the flat port is defined
direction	string	Direction of the flat port
is_vector	boolean	Indicates whether the port is a vector (bus) pin of a design or module
bus_width	int	Bus width of the bus containing this flat port
lsb	int	LSB of the bus containing this flat port
msb	int	MSB of the bus containing this flat port
index	int	Indicates the index of the vector port
object_class	string	Class of a given object. In this case, it is always <i>flat_port</i>
standard_sdc_name	string	Provides the un-changed standard SDC names of a given port of a design or module and are available only after SDC files/commands have been read. For example, for the A bus port that contains two bits, <i>standard_sdc_name</i> may be { A[0], A[1] }.

Attribute Name	Type	Description
non_standard_sdc_name	string	<p>Provides the non-standard SDC names of a given port of a design or module without taking into account effect of define_name_rules/change_names SDC commands, but difference in name from standard_sdc are only visible if allow_non_standard_sdc is specified, which works with the following options:</p> <ul style="list-style-type: none"> <li>• set_option use_if_generate_prefix {true/false}</li> <li>• set_option use_for_generate_prefix {true/false}</li> <li>• set_option unlabeled_generate_prefix {true/false}</li> </ul> <p>For example, for the A bus port that contains two bits, non_standard_sdc_name is {A[0], A[1]}.</p>
modified_sdc_name	string	<p>Provides the SpyGlass names after the combined effect of attribute non_standard_sdc_name and define_name_rules/change_names SDC commands, and are available only after SDC files/commands have been read</p> <p>For example, for the A bus port that contain two bits and change_name rules {"A", "a"} is specified in sdc file, modified_sdc_name may be { a[0],a[1]}.</p>

## flat\_net

The following table describes the various application attributes that are a part of the *flat\_net* object class.

Attribute Name	Type	Description
full_name	string	<p>Bit-blasted name of a given net of a design or module. For example, for A net that contains two bits, <i>full_name</i> is {A[0], A[1]}</p>
base_name	string	<p>Basic name of a given net of a design/module. For example, for A net that contains two bits, <i>base_name</i> is A</p>

## Debug Commands

Attribute Name	Type	Description
file_name	string	File path from which the flat net is read
line_num	int	Line number of the file where the flat net is defined
direction	string	Direction of the flat net
path_name	string	Full path name of a given flat net
is_vector	boolean	Indicates whether the net is a vector net
bus_width	int	Number of bits of the vector net
lsb	int	LSB of the vector net
msb	int	MSB of the vector net
index	int	Indicates the index of the vector net
net_type	string	Indicates if the net is any kind of supply or tristate net type
object_class	string	Class of a given object. In this case, it is always <i>flat_net</i>
is_generated	boolean	The <code>true</code> value indicates that the given flat net has been generated internally
is_record	boolean	The <code>true</code> value indicates that the given flat net is a record net
is_multidim	boolean	The <code>true</code> value indicates that the given flat net is a multidimensional net
standard_sdc_name	string	Provides the un-changed standard SDC names of a given net of a design or module and are available only after SDC files/commands have been read. For example, for the A net , <code>standard_sdc_name</code> may be {IF1.FOR1[0].genblk1.genblk1[1].IF2.inst/A}.

Attribute Name	Type	Description
non_standard_sdc_name	string	<p>Provide the non-standard SDC names of a given net of a design or module without taking into account effect of <code>define_name_rules/change_names</code> SDC commands, but difference in name from <code>standard_sdc</code> is only visible if <code>allow_non_standard_sdc</code> is specified, which works with the following options:</p> <ul style="list-style-type: none"> <li>• <code>set_option use_if_generate_prefix {true/false}</code></li> <li>• <code>set_option use_for_generate_prefix {true/false}</code></li> <li>• <code>iset_option unlabeled_generate_prefix {true/false}</code></li> </ul> <p>For example, for the A net, <code>non_standard_sdc_name</code> may be <code>{IF1.FOR1[0].IF2.inst..[1]/A}</code>.</p>
modified_sdc_name	string	<p>Provides the SpyGlass names after the combined effect of attribute <code>non_standard_sdc_name</code> and <code>define_name_rules/change_names</code> SDC commands, and are available only after SDC files/commands have been read.</p> <p>For example, <code>change_name</code> rules <code>{“inst”, “I”}</code> is specified in sdc file, for the A net <code>modified_sdc_name</code> may be <code>{IF1.FOR1[0].IF2.I..[1]/A}</code>.</p>

## adc\_node

The following table describes the various application attributes that are a part of the `adc_node` object class.

Attribute Name	Type	Description
file_name	string	File path from which the Atrenta Design Constraint, or ADC, node is read
current_design	string	Name of the current design
object_class	string	Class of a given object. In this case, it is always <code>adc_node</code>

## sdc\_node

The following table describes the various application attributes that are a part of the *sdc\_node* object class.

Attribute Name	Type	Description
object_class	string	Class of a given object. In this case, it is always <i>sdc_node</i>

## clock

The following table describes the various application attributes that are a part of the *clock* object class.

Attribute Name	Type	Description
clk_name	string	Clock tag name corresponding to a given clock
clk_net	collection	Clock net, which is a flat net, corresponding to a given clock
domain	collection	Domain corresponding to a given clock. This collection can be used as an input to commands, such as <a href="#">get_clocks</a> , <a href="#">report_domains</a> , and <a href="#">get_registers</a>
edgelist	string	Edgelist of a given clock
period	float	Period of a given clock
file_name	string	File name from which a clock is read
clock_type	string	Type of a given clock

## clock\_domain

The following table describes the various application attributes that are a part of the *clock\_domain* object class.

Attribute Name	Type	Description
clocks	collection	Collection of clocks of a specified domain. This collection can be used as an input to commands, such as <i>get_domains</i> , <i>report_clocks</i> , and <i>get_registers</i>
domain_name	string	Domain name corresponding to a specified clock domain

## message

The following table describes the various application attributes that are a part of the *message* object class.

Attribute Name	Type	Description
file_name	string	Returns the file path from which the object is read
has_arglabels	boolean	Returns if a message has argument labels defined or not
line_number	int	Returns the line number of the message
severity_label	string	Returns the severity label of the message
severity_class	string	Returns the severity class of the message
weight	int	Returns the weight of the message
msg	string	Returns the complete message
static_message	string	Returns the static part of the message
secondary_messages	message collection	Return the list of secondary messages of a given message, if present.
is_waived	boolean	Returns the status of the message, if it was waived or not
message_id	string	Returns the message id of the message
object_class	string	Returns the class of a given object
du_name	string	Returns the name of the design unit if present

## rule

The following table describes the various application attributes that are a

part of the *rule* object class.

Attribute Name	Type	Description
rule_name	string	Returns the registered name of the rule
alias_name	string	Returns the alias of the rule
policy_name	string	Returns the name of parent policy
is_enabled	boolean	Returns true or false, whether the rule is enabled for the current run
rule_type	int	Returns the view type of the rule
violation_count	int	Gives the number of violations reported by the rule
rule_language	string	Returns the language type as Verilog, VHDL, or Mixed
is_mandatory	boolean	Returns true if the rule is mandatory otherwise false
return_status	int	Returns the status of the execution function of rule
object_class	string	Returns the class of a given object

## reset

The following table describes the various application attributes that are a part of the *reset* object class.

Attribute Name	Type	Description
reset_name	char	Returns the name of the reset

## reset\_flop\_node

The following table describes the various application attributes that are a part of the *reset\_flop\_node* object class.

Attribute Name	Type	Description
dest_clocks	char	Returns the destination clocks
dest_domain	char	Returns the destination domain

Attribute Name	Type	Description
dest_file_line	char	Returns the file line information of the destination
dest_resets	char	Returns the resets applied on the flip-flop
multi_flop_synchronizer_names	char	Checks the multi flop synchronizer information of a crossing

## reset\_sync\_node

The following table describes the various application attributes that are a part of the *reset\_sync\_node* object class.

Attribute Name	Type	Description
clock	char	Returns the clock for provided synchronizer
module	char	Returns the reset synchronizer module
reset	char	Returns the reset for provided synchronizer
sync_count	char	Returns the number of synchronizers
sync_name	char	Gets the output net name of a synchronizer flip-flop
sync_type	char	Returns the synchronization type



## Product Attributes

Product attributes can be further categorized in the following groups:

- *Base Attributes*
- *CDC Attributes*
- *Constraints Attributes*
- *DFT Attributes*
- *Power Attributes*
- *Power Verify Attributes*

Refer to [List of Product Attributes](#) for the complete list of product attributes defined in SpyGlass.

Product attributes are saved/restored so that these are accessible when a user comes back to a previously run goal. The [save\\_goal](#) and [restore\\_goal](#) commands are there to save/restore the design query data for the currently selected goal. However, there are following product attributes for which save/restore is not supported:

Attribute Name	Object	Type	Product
<a href="#">dest_type</a>	cdc_node	string	SpyGlass CDC
<a href="#">failure_reason</a>	cdc_node	string	SpyGlass CDC
<a href="#">is_comb_conv</a>	cdc_conv_node	boolean	SpyGlass CDC
<a href="#">is_data</a>	cdc_node	boolean	SpyGlass CDC
<a href="#">is_graycoded</a>	cdc_conv_node	boolean	SpyGlass CDC
<a href="#">is_nonconv_bus</a>	cdc_conv_node	boolean	SpyGlass CDC
<a href="#">is_seq_conv</a>	cdc_conv_node	boolean	SpyGlass CDC
<a href="#">is_synchronized</a>	cdc_node	boolean	SpyGlass CDC
<a href="#">is_user_defined</a>	cdc_conv_node	boolean	SpyGlass CDC
<a href="#">num_source_domains</a>	cdc_node	int	SpyGlass CDC
<a href="#">num_source_domains</a>	cdc_conv_node	int	SpyGlass CDC
<a href="#">num_sources</a>	cdc_node	int	SpyGlass CDC
<a href="#">num_sources</a>	cdc_conv_node	int	SpyGlass CDC
<a href="#">src_type</a>	cdc_node	string	SpyGlass CDC

<i>sync_method</i>	cdc_node	string	SpyGlass CDC
<i>ground_supply</i>	pwr_intent_node	string	SpyGlass Power Verify
<i>power_domain</i>	pwr_intent_node	string	SpyGlass Power Verify
<i>power_supply</i>	pwr_intent_node	string	SpyGlass Power Verify
<i>voltage_range_max</i>	pwr_intent_node	float	SpyGlass Power Verify
<i>voltage_range_min</i>	pwr_intent_node	float	SpyGlass Power Verify
<i>sdc_type</i>	sdc_node	string	SpyGlass Constraints

## Base Attributes

The following table describes the various Tcl application attributes that are a part of the SpyGlass products.

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>is_async_sync_reset</i>	flat_net	boolean	Returns the reset net used as both synchronously and asynchronously
<i>is_clock_used_as_no_nclock</i>	flat_net	boolean	Returns the flip-flop clock signal net which is used as non clock signal in a design
<i>is_clock_used_with_both_edges</i>	flat_net	boolean	Returns the clock signal driving on both edges
<i>is_constant_pin</i>	flat_pin	boolean	Returns the pin of an instance at which a constant value reaches
<i>is_disabled_cell</i>	flat_cell	boolean	Returns the disabled gate
<i>is_internally_generated_reset</i>	flat_net	boolean	Returns the internally generated reset
<i>is_latch_clock_driven_on_both_edges</i>	flat_net	boolean	Returns the clock net trigger latches
<i>is_multiple_driver</i>	flat_net	boolean	Returns the flattened net that has multiple drivers

## Debug Commands

---

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>is_reset_used_as_no_nreset</i>	flat_net	boolean	Returns the asynchronous reset or preset net which is used as non-asynchronous reset or preset signal
<i>is_reset_used_with_both_polarity</i>	flat_net	boolean	Returns the reset or set net which is used as both positive and negative polarity in same design unit
<i>is_unregistered_port</i>	du_port	boolean	Returns the ports, which are not registered, of the module

---

## is\_async\_sync\_reset

Returns the reset net used as both synchronously and asynchronously

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true if reset is used as both synchronously and asynchronously to reset latch.

### Description

The `is_async_sync_reset` attribute returns the reset nets that are used to reset latch, both synchronously and asynchronously, in design. To get the list of these reset net run the `LatchReset` rule of the `latch` policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
"is_async_sync_reset == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_clock\_used\_as\_nonclock

Returns the flip-flop clock signal net which is used as non clock signal in a design

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true if clock signal is used as non clock signal.

### Description

The `is_clock_used_as_nonclock` attribute returns the flip-flop clock signal that is used as a non-clock signal in a design. To get the list of clock net that are used as non-clock in a design, you can run the *STARC-1.4.3.4* rule of *starc* policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
" is_clock_used_as_nonclock == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_clock\_used\_with\_both\_edges

Returns the clock signal driving on both edges

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true if clock signal is driven on both the edges.

### Description

The *is\_clock\_used\_with\_both\_edges* attribute returns the clock signal that is driven on both the edges. To get the list of these clocks in a design, you can run the *W391* rule of the *lint* policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
"is_clock_used_with_both_edges == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_constant\_pin

Returns the pin of an instance at which a constant value reaches

### Scope

Goal

### Object

This attribute is defined on the `flat_pin` object type.

### Return Value

Returns `true` if a constant value reaches at the flattened pin

### Description

The *is\_constant\_pin* attribute returns the pin of an instance at which a constant value reaches. To get a list of constant pins of instances, run any of the following rules from the SpyGlass ERC product:

- **FlopClockConstant:** This rule reports flip-flop instances for which a clock pin is tied to a constant value. Use this rule to get a constant clock pin of a flip-flop.
- **FlopDataConstant:** This rule reports flip-flop instances for which the data pin is tied to a constant value. Use this rule to get a constant data pin of a flip-flop.
- **FlopSRConst:** This rule reports flip-flop instances for which the set or reset pins are permanently enabled. Use this rule to get a set or reset pin that is permanently enabled.
- **FlopEConst:** This rule reports flip-flop instances for which the enable pin is permanently disabled or enabled. Use this rule to get an enable pin that is permanently disabled or enabled.
- **MuxSelConst:** This rule reports MUX gate instances for which the select pin is tied to a constant value. Use this rule to get the MUX select pin that is constant.
- **LatchDataConstant:** *This rule reports latch instances for which the data pin is tied to a constant value. Use this rule to get latch data pin driven by a constant value.*

- **LatchEnableConstant:** This rule reports latch instances for which the enable pin is tied to a constant value. Use this rule to get latch enable pin driven by a constant value.

The *FlopClockConstant*, *FlopEConst*, and *FlopSRConst* rules are a part of the following GuideWare methodologies:

- `./New_RTL/rtl_handoff/lint/structure`
- `./New_RTL/initial_rtl/lint/structure`
- `./New_RTL/detailed_rtl/lint/structure`
- `./New_RTL/ip_handoff/lint/structure`

Please select the appropriate methodology as per the requirement or create a custom methodology to run these rules.

## Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set flop_iter [get_cells * -filter "is_flop ==
true"]
sg_shell> set flop_iter_with_const_clk [get_pins -of_objects
$flop_iter -filter "is_clock_pin == true && is_constant_pin
== true"]
```

**NOTE:** You can iterate on the *flop\_iter\_with\_const\_clk* list, depending on the requirement.

## See Also

[get\\_attribute](#), [filter\\_collection](#)



## is\_disabled\_cell

Returns the disabled gate

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns `true` if flattened gate is disabled

### Description

The *is\_disabled\_cell* attribute returns the flattened instance of AND or NAND gate in which at least one pin is tied low, or NOR or OR gate in which at least one pin is tied high.

Run the *DisabledAnd* or *DisabledOr* rule of the SpyGlass ERC product, or create a custom goal to run these rules.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set disabled_gate_list [get_cells -filter {
is_disabled_cell == true } ]
sg_shell> set disabled_gate_list [get_cells -filter {
is_disabled_cell== true } ]
```

**NOTE:** You can iterate on the “disabled\_gate\_list” list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_internally\_generated\_reset

Returns the internally generated reset

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true on a net if it is used as a internally generated reset in the design.

### Description

The *is\_internally\_generated\_reset* attribute returns the internally generated reset in design. To get the list of these internally generated reset nets, run the *IntReset* rule of *openmore* policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
"is_internally_generated_reset == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_latch\_clock\_driven\_on\_both\_edges

Returns the clock net trigger latches

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true if different level of user defined clocks trigger the latches in design

### Description

The *is\_latch\_clock\_driven\_on\_both\_edges* attribute returns those clock nets whose different levels trigger the latches in the design. To get the list of these clock nets, you can run the *ClockEdges* rule of *latch* policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
" is_latch_clock_driven_on_both_edges == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_multiple\_driver

Returns the flattened net that has multiple drivers

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns `true` if flattened net has multiple drivers

### Description

The *is\_multiple\_driver* attribute returns the nets that have multiple drivers. To get a list of multiple-driven nets in a design, run the *W415* rule of the SpyGlass lint product. This rule is a part of the following GuideWare Methodologies:

- `./IP_netlist/ip_risk/lint/ip_netlist`
- `./IP_netlist/ip_audit/lint/ip_netlist`
- `./IP_RTL/ip_risk/lint/ip_rtl`
- `./IP_RTL/ip_audit/lint/ip_rtl`
- `./New_RTL/rtl_handoff/lint/simulation`
- `./New_RTL/initial_rtl/lint/simulation`
- `./New_RTL/detailed_rtl/lint/simulation`
- `./New_RTL/ip_handoff/lint/simulation`
- `./SoC/soc_postlayout/lint/soc_netlist`
- `./soc_postlayout/lint/soc_netlist`
- `./soc_postsynth/lint/soc_netlist`
- `./soc_prelim/lint/soc_netlist`
- `./soc_rtl/lint/soc_rtl`

Please select the appropriate methodology as per the requirement or

create a custom methodology to run the rule.

## Examples

```
sg_shell> set_pref dq_design_view_type flat  
sg_shell> set net_iter [get_nets * -filter  
"is_multiple_driver == true"]
```

**NOTE:** *You can iterate on the "net\_iter" list, depending on the requirement.*

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_reset\_used\_as\_nonreset

Returns the asynchronous reset or preset net which is used as non asynchronous reset or preset signal

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true if asynchronous reset or preset net is used as a non-asynchronous reset or preset signal.

### Description

The `is_reset_used_as_nonreset` attribute returns the asynchronous reset or preset net that is used as a non-asynchronous reset or preset signal.

To get the list of reset net that are used as non reset in a design, you can run the *STARC-1.3.1.3* rule of the *starc* policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
"is_reset_used_as_nonreset == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_reset\_used\_with\_both\_polarity

Returns the reset or set net which is used as both positive and negative polarity in same design unit

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns true if reset or set is used as both positive and negative polarity in a design.

### Description

The *is\_reset\_used\_with\_both\_polarity* attribute returns those reset or set signals that are used as both positive and negative polarity in the same design unit. To get the list of both polarity reset in a design, you can run the *W392* rule of the *lint* policy.

Select the above rule or create a custom methodology.

### Examples

```
sg_shell> set_pref dq_design_view_type flat
sg_shell> set net_iter [get_nets * -filter
"is_reset_used_with_both_polarity == true"]
```

**NOTE:** You can iterate on the "net\_iter" list, depending on the requirement.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_unregistered\_port

Returns the ports, which are not registered, of the module

### Scope

Goal

### Object

This attribute is defined on the `du_port` object type.

### Return Value

Returns `true` if a DU port is unregistered

### Description

The *is\_unregistered\_port* attribute returns the port of the module that is not registered. To set the attribute, run the *RegInputOutput-ML* rule of the SpyGlass moreLint product. The attribute is set on the DU object of type port.

The *RegInputOutput-ML* rule is a part of the following GuideWare Methodologies:

- `./New_RTL/rtl_handoff/audit/datasheet_io_audit`
- `./New_RTL/initial_rtl/audit/datasheet_io_audit`
- `./New_RTL/detailed_rtl/audit/datasheet_io_audit`
- `./New_RTL/ip_handoff/audit/datasheet_io_audit`
- `./IP_RTL/ip_audit/audit/datasheet_io_audit`
- `./IP_netlist/ip_audit/audit/datasheet_io_audit`
- `./SoC/soc_postlayout/audit/datasheet_io_audit`
- `./SoC/soc_postsynth/audit/datasheet_io_audit`
- `./SoC/soc_prelim/audit/datasheet_io_audit`
- `./SoC/soc_rtl/audit/datasheet_io_audit`

Please select the appropriate methodology as per the requirement or create a custom methodology to run the rule.



## Examples

```
sg_shell> set_pref dq_design_view_type du
sg_shell> set port_iter [get_ports * -filter
"is_unregistered_port == true"]
```

**NOTE:** *You can iterate on the "port\_iter" list, depending on the requirement.*

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## CDC Attributes

The following table describes the various Tcl application attributes that are a part of the SpyGlass CDC product.

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>dest_type</i>	cdc_node	string	Checks the type of destination instance present in a crossing
<i>failure_reason</i>	cdc_node	string	Reports the reason of unsynchronization of a CDC crossing
<i>is_comb_conv</i>	cdc_conv_node	boolean	Checks whether the converging signals are reported by the Ac_conv02 rule
<i>is_data</i>	cdc_node	boolean	Checks whether a CDC crossing is a data crossing
<i>is_graycoded</i>	cdc_conv_node	string	Checks whether the converging signals are gray encoded
<i>is_nonconv_bus</i>	cdc_conv_node	boolean	Checks whether the converging signals are reported by the Ac_conv04 rule
<i>is_seq_conv</i>	cdc_conv_node	boolean	Checks whether the converging signals are reported by the Ac_conv01 rule
<i>is_synchronized</i>	cdc_node	boolean	Checks whether a CDC crossing is synchronized
<i>is_user_defined</i>	cdc_conv_node	boolean	Checks whether the converging signals are reported by the Ac_conv05 rule
<i>num_sources</i>	cdc_node, cdc_conv_node	int	Checks number of sources present in a crossing
<i>num_source_domains</i>	cdc_node, cdc_conv_node	int	Checks number of source domains present in a crossing

## Debug Commands

---

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>src_type</i>	cdc_node	string	Checks the type of source instance present in a crossing
<i>sync_method</i>	cdc_node	string	Reports the reason of synchronization of a CDC crossing

---

## dest\_type

**Checks the type of destination instance present in a crossing**

### Syntax

```
get_attribute [get_cdc] dest_type
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` object type.

### Return Value

Returns a string of flip-flop, library-cell, latch, primary output, and black box.

### Description

The *dest\_type* attribute tells the type of destination instance in a crossing. Query on this attribute should be made only when the *Ac\_sync\_group* rules (*Ac\_sync01*, *Ac\_sync02*, *Ac\_undef01*, and *Ac\_undef02*) have been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter dest_type==latch]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## failure\_reason

Reports the reason of unsynchronization of a CDC crossing

### Syntax

```
get_attribute [get_cdc] failure_reason
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` object type.

### Return Value

Returns a string value stating the reason of unsynchronization.

### Description

The *failure\_reason* attribute tells the reason of unsynchronization of a CDC crossing.

Query on this attribute should be made only when the *Ac\_unsync01* or *Ac\_unsync02* rule has been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter  
failure_reason==Qualifier_not_found]
```

**NOTE:** *In the reason string, replace the space as "\_" to use the string as a filter value.*

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_comb\_conv

Checks whether the converging signals are reported by the *Ac\_conv02* rule

### Syntax

```
get_attribute [get_cdc_coherency] is_comb_conv
```

### Scope

Goal

### Object

This attribute is defined on the *cdc\_conv\_node* object type.

### Return Value

Returns a boolean value.

### Description

The *is\_comb\_conv* attribute checks whether the converging signals have been violated by the *Ac\_conv02* rule.

### Examples

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
is_comb_conv==true]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_data

**Checks whether a CDC crossing is a data crossing**

### Syntax

```
get_attribute [get_cdc] is_data
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` object type.

### Return Value

Returns a boolean value.

### Description

The *is\_data* attribute checks whether a CDC crossing is a data crossing. Query on this attribute should be made only when the *Ac\_sync\_group* rules have been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter is_data == true]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_graycoded

**Checks whether the converging signals are gray encoded**

### Syntax

```
get_attribute [get_cdc_coherency] is_graycoded
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_conv_node` object type.

### Return Value

Returns a string.

### Description

The *is\_graycoded* attribute checks whether the functional check on the converging signals has been passed.

This attribute has the following values:

- **yes**: When this value is set, the *is\_graycoded* attribute returns only those convergence violations that have been functionally checked for gray coding and their status is PASSED.
- **no**: When this value is set, the *is\_graycoded* attribute returns only those convergence violations that have been functionally checked for gray coding and their status is FAILED.
- **unknown**: When this value is set, the *is\_graycoded* attribute returns only those convergence violations that have been functionally checked for gray coding and their status is PA.
- **disabled**: When this value is set, the *is\_graycoded* attribute reports violation for the convergence rules that do not perform functional gray coding check.

Query on this attribute should be made only when the *Ac\_conv* rules have been run.



## Examples

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
is_graycoded==yes]
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_nonconv\_bus

Checks whether the converging signals are reported by the *Ac\_conv04* rule

### Syntax

```
get_attribute [get_cdc_coherency] is_nonconv_bus
```

### Scope

Goal

### Object

This attribute is defined on the *cdc\_conv\_node* object type.

### Return Value

Returns a boolean value.

### Description

The *is\_nonconv\_bus* attribute checks whether the converging signals have been violated by the *Ac\_conv04* rule.

### Examples

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
is_nonconv_bus==true]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_seq\_conv

Checks whether the converging signals are reported by the `Ac_conv01` rule

### Syntax

```
get_attribute [get_cdc_coherency] is_seq_conv
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_conv_node` object type.

### Return Value

Returns a boolean value.

### Description

The `is_seq_conv` attribute checks whether the converging signals has been violated by the `Ac_conv01` rule.

### Examples

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
is_seq_conv==true]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_synchronized

Checks whether a CDC crossing is synchronized

### Syntax

```
get_attribute [get_cdc] is_synchronized
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` object type.

### Return Value

Returns a boolean value.

### Description

The *is\_synchronized* attribute checks whether a CDC crossing is synchronized.

Query on this attribute should be made only when the *Ac\_sync\_group* rules have been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter  
is_synchronized==false]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_user\_defined

Checks whether the converging signals are reported by the *Ac\_conv05* rule

### Syntax

```
get_attribute [get_cdc_coherency] is_user_defined
```

### Scope

Goal

### Object

This attribute is defined on the *cdc\_conv\_node* object type.

### Return Value

Returns a boolean value.

### Description

The *is\_user\_defined* attribute checks the presence of signal convergence due to the user-defined *gray\_signals* constraint. All of these converging signals are reported by the *Ac\_conv05* rule.

### Examples

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
is_user_defined==true]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## num\_sources

**Checks number of sources present in a crossing**

### Syntax

```
get_attributes [<get_cdc/get_cdc_coherency>] num_sources
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` and `cdc_conv_node` object types.

### Return Value

Returns an integer value.

### Description

The *num\_sources* attribute tells the count of sources in a crossing.

Query on this attribute should be made only when convergence rules (`Ac_conv01`, `Ac_conv02`, `Ac_conv03`, `Ac_conv04`, `Ac_conv05`) or *Ac\_sync\_group* rules (`Ac_sync01`, `Ac_sync02`, `Ac_undef01`, `Ac_undef02`) have been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter num_sources>2]
```

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
num_sources>3]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## num\_source\_domains

Checks number of source domains present in a crossing

### Syntax

```
get_attributes [<get_cdc/get_cdc_coherency>]  
num_source_domains
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` and `cdc_conv_node` object types.

### Return Value

Returns an integer value.

### Description

The *num\_source\_domains* attribute tells the count of source domains in a crossing.

Query on this attribute should be made only when convergence rules (`Ac_conv01`, `c_conv02`, `Ac_conv03`, `Ac_conv04`, `Ac_conv05`) or *Ac\_sync\_group* rules (`Ac_sync01`, `Ac_sync02`, `Ac_unsync01`, `Ac_unsync02`) have been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter num_source_domains>2]
```

```
sg_shell> report_cdc_coherency [get_cdc_coherency -filter  
num_source_domains>3]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## src\_type

**Checks the type of source instance present in a crossing**

### Syntax

```
get_attribute [get_cdc] src_type
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` object type.

### Return Value

Returns a string of flip-flop, library-cell, latch, primary input, and black box.

### Description

The *src\_type* attribute tells the type of source instance in a crossing.

Query on this attribute should be made only when the *Ac\_sync\_group* rules (*Ac\_sync01*, *Ac\_sync02*, *Ac\_unsync01*, *Ac\_unsync02*) have been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter src_type==flop]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)



## sync\_method

**Reports the reason of synchronization of a CDC crossing**

### Syntax

```
get_attribute [get_cdc] sync_method
```

### Scope

Goal

### Object

This attribute is defined on the `cdc_node` object type.

### Return Value

Returns a string value stating the reason of synchronization.

### Description

The *num\_source\_domains* attribute reports the reason of synchronization of a CDC crossing.

Query on this attribute should be made only when the *Ac\_sync01* or *Ac\_sync02* rule has been run.

### Examples

```
sg_shell> report_cdc [get_cdc -filter  
sync_method=~synchronizing_cell*]
```

**NOTE:** *In the reason string, replace the space as "\_" to use the string as a filter value.*

```
sg_shell> report_cdc [get_cdc -filter  
sync_method==Conventional_multi-  
flop_for_metastability_technique]
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## Constraints Attributes

The following table describes the various Tcl application attributes that are a part of the SpyGlass Constraints product.

Attribute Name	Object	Type	Description
<i>sdc_type</i>	sdc_node	string	Returns the SDC constraint type of the SDC node
<i>timing_state</i>	flat_cell flat_pin flat_port flat_net	string	Returns the constrained status of the queried design object

## sdc\_type

Returns the SDC constraint type of the SDC node

### Scope

Goal

### Object

This attribute is defined on the `sdc_node` object type.

### Return Value

Returns a string value if the query is successful and nothing if the query is unsuccessful

### Description

The `sdc_type` attribute returns the SDC constraint type of the SDC node.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_sdc -of_objects top.U0.A ]  
sdc_type
```

The output of this command is as follows:

```
create_clock
```

#### Example 2

Consider the following command:

```
sg_shell> get_sdc -filter { sdc_type == create_clock }
```

This command does not generate any output.

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## timing\_state

Returns the constrained status of the queried design object

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_cell
- flat\_pin
- flat\_port
- flat\_net

### Return Value

Returns a non empty string

### Description

The *timing\_state* attribute returns the constraint status of the queried design object. If the object is constrained with an SDC constraint, it returns *constrained*; otherwise, it returns *unconstrained*.

Consider the following points while using this attribute:

- For data pins of MUXes, this attribute returns *constrained*, if any clock is reaching the pin.
- For select pins of MUXes, this attribute returns *constrained*, if the pin is set to a constant value.

### Examples

```
sg_shell> get_attribute [get_ports top.clk] timing_state  
constrained
```

```
sg_shell> get_ports -filter { timing_state == constrained }  
{top.clk}
```

---

Debug Commands

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## DFT Attributes

The following table describes the various Tcl application attributes that are a part of the SpyGlass DFT product.

Attribute Name	Object	Type	Description
<i>atspeed_sim_value</i>	flat_pin flat_port flat_net	string	Gets atspeed_capture mode simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design
<i>capture_sim_value</i>	flat_pin flat_port flat_net	string	Gets capture mode simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design
<i>get_atspeed_clock_n_phase</i>	flat_pin flat_port flat_net	string	Gets the atspeed mode source clock, source clock phase and phase at a user specified design node in current design
<i>get_capture_clock_n_phase</i>	flat_pin flat_port flat_net	string	Gets the capture mode source clock, source clock phase and phase at a user specified design node in the current design
<i>get_dft_functional_clock_n_phase</i>	flat_pin flat_port flat_net	string	Gets the functional source clock, source clock phase and phase at a user specified design node in current design
<i>get_latch_atspeed_status</i>	flat_inst	string	Gets the atspeed transparency status for user specified latch in current design
<i>get_latch_capture_status</i>	flat_inst	string	Gets the capture transparency status for user specified latch in current design
<i>get_latch_shift_status</i>	flat_inst	string	Gets the shift transparency status for user specified latch in current design
<i>get_scan_status</i>	flat_inst	string	Gets the scannability status for user specified flip-flop or latch in current design

## Debug Commands

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>get_shift_clock_n_phase</i>	flat_inst	string	Gets the shift mode source clock, source clock phase and phase at a user specified design node in current design
<i>is_scannable</i>	flat_cell	boolean	Checks whether a flip-flop is scannable
<i>obs_probability</i>	flat_port flat_pin flat_net	float	Gets the probability that the flat port, pin or net is observable when random test pattern is applied
<i>one_cnt_probability</i>	flat_port flat_pin flat_net	float	Gets the probability that the flat port, pin or net is at 1 when random test pattern is applied
<i>pg_sim_value</i>	flat_pin flat_port flat_net	string	Gets power_ground simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design
<i>rand_fault_cov_estimate</i>	design	float	Gets the fault coverage estimate of top module for the dft_pattern_count random test patterns
<i>rand_test_cov_estimate</i>	design	float	Gets the test coverage estimate of top module for dft_pattern_count random test patterns
<i>sa0_det_probability</i>	flat_port flat_pin flat_net	float	Gets the probability that stuck at 0 fault on flat port, pin or net is detected after dft_pattern_count random test patterns are applied
<i>sa1_det_probability</i>	flat_port flat_pin flat_net	float	Gets the probability that stuck at 1 fault on flat port, pin or net is detected after dft_pattern_count random test patterns are applied
<i>sa0_fault_detectability</i>	flat_port flat_pin	string	Gets the stuck_at zero fault detectability

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>sa1_fault_detectability</i>	flat_port flat_pin	string	Gets the stuck_at one fault detectability
<i>shift_sim_value</i>	flat_pin flat_port flat_net	string	Gets shift mode simulation value (1   0   X   Z) for user-specified design node (flat_net   flat_pin   flat_port) in current design
<i>static_controllability</i>	flat_pin flat_port flat_net	string	Gets static controllability (3-bit-string (y/n): zero-control_one-control_zee-control: nnn   nny   nyn   nyy   ynn   yny   yyn   yyy) for user-specified design node (flat_net   flat_pin   flat_port) in current design
<i>static_observability</i>	flat_pin flat_port	string	Gets static observability (y (yes_observable)   n (not_observable)) for user-specified design node (flat_pin   flat_port) in current design
<i>t01_fault_detectability_los</i>	flat_pin flat_port	string	Gets zero to one transition fault detectability with launch on shift
<i>t10_fault_detectability_los</i>	flat_pin flat_port	string	Gets one to zero transition fault detectability with launch on shift
<i>t01_fault_detectability_loc</i>	flat_pin flat_port	string	Gets zero to one transition fault detectability with launch on capture
<i>t10_fault_detectability_loc</i>	flat_pin flat_port	string	Gets one to zero transition fault detectability with launch on capture
<i>zero_cnt_probability</i>	flat_port flat_pin flat_net	float	Gets the probability that the flat port, pin, or net is at 0 when random test pattern is applied



## atspeed\_sim\_value

Gets *atspeed\_capture* mode simulation value (1 | 0 | X | Z) for user-specified design node (flat\_net | flat\_pin | flat\_port) in current design

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
atspeed_sim_value
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

string: 1 | 0 | X | Z

### Description

The *atspeed\_sim\_value* attribute displays the *atspeed\_capture* mode simulation value (1 | 0 | X | Z) for a user-specified design node (flat\_net | flat\_pin | flat\_port) in the current design.

Query on this attribute should be made only when one of the SpyGlass DFT DSM goals has been run.

### Examples

```
sg_shell> current_goal dft_dsm_clocks  
sg_shell> run_goal  
. .  
<output of run_goal>
```

```
.  
.br/>sg_shell> get_attribute [get_nets top.U0.A]  
atspeed_sim_value  
<1 | 0 | X | Z>
```

## See Also

[capture\\_sim\\_value](#), [pg\\_sim\\_value](#), [capture\\_sim\\_value](#)

## capture\_sim\_value

Gets capture mode simulation value (1 | 0 | X | Z) for user-specified design node (flat\_net | flat\_pin | flat\_port) in current design

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>
capture_sim_value
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

string: 1 | 0 | X | Z

### Description

The *capture\_sim\_value* attribute displays the capture mode simulation value (1 | 0 | X | Z) for a user-specified design node (flat\_net | flat\_pin | flat\_port) in the current design.

Query on this attribute should be made only when one of the SpyGlass DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready
sg_shell> run_goal
.
.
<output of run_goal>
```

```
.  
.   
sg_shell> get_attribute [get_nets top.U0.A]  
capture_sim_value  
<1 | 0 | X | Z>
```

## See Also

[pg\\_sim\\_value](#), [pg\\_sim\\_value](#), [capture\\_sim\\_value](#)

## get\_atspeed\_clock\_n\_phase

**Gets the atspeed mode source clock, source clock phase and phase at a user specified design node in current design**

### Syntax

```
get_attribute < object: flat_port | flat_pin | flat_net >
get_atspeed_clock_n_phase
```

### Scope

Goal

### Object

This attribute is defined on the flat\_port | flat\_pin | flat\_net object type.

### Return Value

String

### Description

The *get\_atspeed\_clock\_n\_phase* attribute gets the atspeed mode source clock, source clock phase and phase at a user specified design node in the current design. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> run_goal
.
.
<output of run_goal>
.
.
sg_shell> get_attribute [get_nets top.tclk1]
get_atspeed_clock_n_phase < <clock_name>, <-1 | 0 | 1>, <-1 |
0 | 1 > >
```

**See Also**

*[get\\_shift\\_clock\\_n\\_phase](#), [get\\_capture\\_clock\\_n\\_phase](#),  
[get\\_dft\\_functional\\_clock\\_n\\_phase](#)*

## get\_capture\_clock\_n\_phase

**Gets the capture mode source clock, source clock phase and phase at a user specified design node in the current design**

### Syntax

```
get_attribute < object: flat_port | flat_pin | flat_net >  
get_capture_clock_n_phase
```

### Scope

Goal

### Object

This attribute is defined on the flat\_port | flat\_pin | flat\_net object type.

### Return Value

String

### Description

The *get\_capture\_clock\_n\_phase* attribute gets the capture mode source clock, source clock phase and phase at a user specified design node in the current design. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready  
sg_shell> run_goal  
.  
.  
<output of run_goal>  
.  
.  
sg_shell> get_attribute [get_nets top.tclk1]
```

```
get_capture_clock_n_phase < <clock_name>, <-1 | 0 | 1>, <-1 |  
0 | 1 > >
```

## See Also

[get\\_shift\\_clock\\_n\\_phase](#), [get\\_atspeed\\_clock\\_n\\_phase](#),  
[get\\_dft\\_functional\\_clock\\_n\\_phase](#)



## get\_dft\_functional\_clock\_n\_phase

**Gets the functional source clock, source clock phase and phase at a user specified design node in current design**

### Syntax

```
get_attribute < object: flat_port | flat_pin | flat_net >  
get_dft_functional_clock_n_phase
```

### Scope

Goal

### Object

This attribute is defined on the flat\_port | flat\_pin | flat\_net object type.

### Return Value

String

### Description

The *get\_dft\_functional\_clock\_n\_phase* attribute gets the functional mode source clock, source clock phase and phase at a user specified design node. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready  
sg_shell> run_goal  
.  
.  
  <output of run_goal>  
.  
.  
sg_shell> get_attribute [get_nets top.tclk1]
```

```
get_dft_functional_clock_n_phase < <clock_name>, <-1 | 0 | 1>, <-1 | 0 | 1 > >
```

## See Also

[get\\_shift\\_clock\\_n\\_phase](#), [get\\_at-speed\\_clock\\_n\\_phase](#),  
[get\\_capture\\_clock\\_n\\_phase](#)

## get\_latch\_atspeed\_status

Gets the *atspeed* transparency status for user specified latch in current design

### Syntax

```
get_attribute <object: flat_inst> get_latch_atspeed_status
```

### Scope

Goal

### Object

This attribute is defined on the `flat_inst` object type.

### Return Value

String

### Description

The *get\_latch\_atspeed\_status* attribute gets the *atspeed* transparency status for the user specified latch. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> current_goal get_latch_atspeed_status
sg_shell> run_goal
.
.
<output of run_goal>
.
.
sg_shell> get_attribute [get_cells top.ld_1]
get_latch_atspeed_status < transparent_forced |
transparent_clock_off | transparent_control | shadow |
lockup_source | lockup_destination >
```

## See Also

[\*get\\_scan\\_status\*](#), [\*get\\_latch\\_shift\\_status\*](#), [\*get\\_latch\\_capture\\_status\*](#)

## get\_latch\_capture\_status

**Gets the capture transparency status for user specified latch in current design**

### Syntax

```
get_attribute <object: flat_inst> get_latch_capture_status
```

### Scope

Goal

### Object

This attribute is defined on the `flat_inst` object type.

### Return Value

String

### Description

The *get\_latch\_capture\_status* attribute gets the capture transparency status for the user specified latch. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> current_goal get_latch_capture_status
```

```
sg_shell> run_goal
```

```
.
```

```
.
```

```
<output of run_goal>
```

```
.
```

```
.
```

```
sg_shell> get_attribute [get_cells top.ld_1]
get_latch_capture_status < transparent_forced |
transparent_clock_off | transparent_control | shadow |
lockup_source | lockup_destination >
```

**See Also**

[\*get\\_scan\\_status\*](#), [\*get\\_latch\\_shift\\_status\*](#), [\*get\\_latch\\_at-speed\\_status\*](#)

## get\_latch\_shift\_status

**Gets the shift transparency status for user specified latch in current design**

### Syntax

```
get_attribute <object: flat_inst> get_latch_shift_status
```

### Scope

Goal

### Object

This attribute is defined on the `flat_inst` object type.

### Return Value

String

### Description

The *get\_latch\_shift\_status* attribute gets the shift transparency status for user specified latch in current design. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> current_goal get_latch_shift_status
sg_shell> run_goal
.
.
<output of run_goal>
.
.

sg_shell> get_attribute [get_cells top.ld_1]
get_latch_shift_status < transparent_forced |
transparent_clock_off | transparent_control | shadow |
lockup_source | lockup_destination >
```

**See Also**

[\*get\\_scan\\_status\*](#), [\*get\\_latch\\_capture\\_status\*](#), [\*get\\_latch\\_atspeed\\_status\*](#),



## get\_scan\_status

**Gets the scannability status for user specified flip-flop or latch in current design**

### Syntax

```
get_attribute <object: flat_inst> get_scan_status
```

### Scope

Goal

### Object

This attribute is defined on the `flat_inst` object type.

### Return Value

String

### Description

The *get\_scan\_status* attribute gets scannability status for user specified flip\_flop or latch in the current design. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> current_goal get_scan_status
           sg_shell> run_goal
           .
           .
           <output of run_goal>
           .
           .
sg_shell> get_attribute [get_cells top.ff_1]
           get_scan_status < scan_forced | scan_inferred |
           no_scan_forced | no_scan_inferred | unscannable_clock |
           unscannable_reset | unscannable_clock_reset >
```

**See Also**

[\*get\\_latch\\_shift\\_status\*](#), [\*get\\_latch\\_capture\\_status\*](#), [\*get\\_latch\\_capture\\_status\*](#),  
[\*get\\_latch\\_atspeed\\_status\*](#)

## get\_shift\_clock\_n\_phase

gets the shift mode source clock, source clock phase and phase at a user specified design node in current design

### Syntax

```
get_attribute < object: flat_port | flat_pin | flat_net >  
get_shift_clock_n_phase
```

### Scope

Goal

### Object

This attribute is defined on the flat\_port | flat\_pin | flat\_net object type.

### Return Value

String

### Description

The *get\_shift\_clock\_n\_phase* attribute gets the shift mode source clock, source clock phase and phase at a user specified design node in the current design. Query on this attribute should be made only when one of the DFT goals has been run.

### Examples

```
sg_shell> get_attribute [get_nets top.tclk1]  
get_shift_clock_n_phase < <clock_name>, <-1 | 0 | 1>, <-1 | 0  
| 1 > >
```

### See Also

[get\\_dft\\_functional\\_clock\\_n\\_phase](#), [get\\_atspeed\\_clock\\_n\\_phase](#),  
[get\\_capture\\_clock\\_n\\_phase](#)

## is\_scannable

Checks whether a flip-flop or a latch is scannable

### Syntax

```
get_attribute <object: flat_cell> is_scannable
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

bool: true | false

### Description

The *is\_scannable* attribute checks whether a flip-flop or a latch is scannable.

Query on this attribute should be made only when one of the SpyGlass DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready
sg_shell> run_goal
.
.
<output of run_goal>
.
.
sg_shell> get_attribute [get_cell FD1] is_scannable
<true | false>
```

### See Also

None

## obs\_probability

**Gets the probability that the flat port, pin or net is observable when random test pattern is applied**

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
obs_probability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

float

### Description

The *obs\_probability* attribute is used to get the probability that the flat port, pin or net is observable when random test pattern is applied.

The *obs\_probability* attribute returns *observe\_1* probability for a single pattern. This is independent of the pattern count specified using the *dft\_pattern\_count* parameter.

Query on this attribute should be made only when *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class flat_pin "test.inst1.pin"  
obs_probability
```

**See Also**

*[zero\\_cnt\\_probability](#), [sa0\\_det\\_probability](#), [sa1\\_det\\_probability](#), [one\\_cnt\\_probability](#)*

## one\_cnt\_probability

**Gets the probability that the flat port, pin or net is at 1 when random test pattern is applied**

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
one_cnt_probability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

float

### Description

The *one\_cnt\_probability* attribute is used to get probability that the flat port, pin or net is at 1 when random test pattern is applied.

The *one\_cnt\_probability* attribute returns control\_1 probability for a single pattern. This is independent of the pattern count specified using the *dft\_pattern\_count* parameter.

Query on this attribute should be made only when *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class flat_pin "test.inst1.pin"  
one_cnt_probability
```

**See Also**

*[zero\\_cnt\\_probability](#), [sa0\\_det\\_probability](#), [sa1\\_det\\_probability](#), [obs\\_probability](#)*



## pg\_sim\_value

Gets *power\_ground* simulation value (1 | 0 | X | Z) for user-specified design node (flat\_net | flat\_pin | flat\_port) in current design

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
pg_sim_value
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

string: 1 | 0 | X | Z

### Description

The *pg\_sim\_value* attribute displays the *power\_ground* simulation value (1 | 0 | X | Z) for a user-specified design node (flat\_net | flat\_pin | flat\_port) in the current design.

Query on this attribute should be made only when one of the SpyGlass DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready  
sg_shell> run_goal  
.  
.  
<output of run_goal>
```

```
.  
.   
sg_shell> get_attribute [get_nets top.U0.A] pg_sim_value  
<1 | 0 | X | Z>
```

## See Also

[pg\\_sim\\_value](#), [capture\\_sim\\_value](#), [capture\\_sim\\_value](#)

## rand\_fault\_cov\_estimate

Gets the fault coverage estimate of top module for the `dft_pattern_count` random test patterns

### Syntax

```
get_attribute <object: design> rand_fault_cov_estimate
```

### Scope

Goal

### Object

This attribute is defined on the `design` object type

### Return Value

float

### Description

The *rand\_fault\_cov\_estimate* attribute is used to get fault coverage estimate of top module for the *dft\_pattern\_count* random test patterns.

Query on this attribute should be made only when the *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class design "top_design"  
rand_fault_cov_estimate
```

### See Also

[rand\\_test\\_cov\\_estimate](#)

## rand\_test\_cov\_estimate

**Gets the test coverage estimate of top module for *dft\_pattern\_count* random test patterns**

### Syntax

```
get_attribute <object: design> rand_test_cov_estimate
```

### Scope

Goal

### Object

This attribute is defined on the `design` object type

### Return Value

float

### Description

The *rand\_test\_cov\_estimate* attribute is used to get the test coverage estimate of top module for *dft\_pattern\_count* random test patterns.

Query on this attribute should be made only when the *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class design "top_design"  
rand_test_cov_estimate
```

### See Also

[rand\\_fault\\_cov\\_estimate](#)

## sa0\_det\_probability

**Gets the probability that stuck at 0 fault on flat port, pin or net is detected after *dft\_pattern\_count* random test patterns are applied**

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
sa0_det_probability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

float

### Description

The *sa0\_det\_probability* attribute is used to get the probability that stuck at 0 fault on flat port, pin or net is detected after *dft\_pattern\_count* random test patterns are applied.

Query on this attribute should be made only when the *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class flat_pin "test.inst1.pin"  
sa0_det_probability
```

### See Also

[zero\\_cnt\\_probability](#), [sa1\\_det\\_probability](#), [one\\_cnt\\_probability](#), [obs\\_probability](#)

## sa1\_det\_probability

**Gets the probability that stuck at 1 fault on flat port, pin or net is detected after `dft_pattern_count` random test patterns are applied**

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
sa1_det_probability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

float

### Description

The *sa1\_det\_probability* attribute is used to get the probability that stuck at 1 fault on flat port, pin or net is detected after *dft\_pattern\_count* random test patterns are applied.

Query on this attribute should be made only when the *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class flat_pin "test.inst1.pin"  
sa1_det_probability
```

### See Also

[zero\\_cnt\\_probability](#), [sa0\\_det\\_probability](#), [one\\_cnt\\_probability](#), [obs\\_probability](#)

## sa0\_fault\_detectability

Gets the stuck\_at zero fault detectability

### Syntax

```
get_attribute <object: flat_pin | flat_port>  
sa0_fault_detectability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string

### Description

The *sa1\_fault\_detectability* attribute is used to get the stuck\_at zero (s/O) fault detectability.

Query on this attribute should be made only when the one of the goals with the *Info\_coverage* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> current_goal dft/dft_scan_ready
```

```
sg_shell> run_goal
```

```
    <output of run_goal>
```

```
sg_shell> get_attribute -class flat_pin "top.myInst.in1"  
sa0_fault_detectability
```

### See Also

[sa1\\_fault\\_detectability](#)

## sa1\_fault\_detectability

Gets the stuck\_at one fault detectability

### Syntax

```
get_attribute <object: flat_pin | flat_port>  
sa1_fault_detectability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string

### Description

The *sa1\_fault\_detectability* attribute is used to get the stuck\_at one (s/1) fault detectability.

Query on this attribute should be made only when the one of the goals with the *Info\_coverage* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> current_goal dft/dft_scan_ready  
sg_shell> run_goal  
          <output of run_goal>  
sg_shell> get_attribute -class flat_pin "top.myInst.in1"  
sa1_fault_detectability
```

### See Also

[sa0\\_fault\\_detectability](#)



## shift\_sim\_value

Gets shift mode simulation value (1 | 0 | X | Z) for user-specified design node (flat\_net | flat\_pin | flat\_port) in current design

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>
shift_sim_value
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

string: 1 | 0 | X | Z

### Description

The *shift\_sim\_value* attribute displays the shift mode simulation value (1 | 0 | X | Z) for a user-specified design node (flat\_net | flat\_pin | flat\_port) in the current design.

Query on this attribute should be made only when one of the SpyGlass DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready
sg_shell> run_goal
.
.
<output of run_goal>
.
```

```
sg_shell> get_attribute [get_nets top.U0.A] shift_sim_value  
<1 | 0 | X | Z>
```

## See Also

[pg\\_sim\\_value](#), [capture\\_sim\\_value](#), [pg\\_sim\\_value](#)

## static\_controllability

Gets static controllability (3-bit-string (y/n): zero-control\_one-control\_zee-control: nnn | nny | nyn | nyy | ynn | yny | yyn | yyy) for user-specified design node (flat\_net | flat\_pin | flat\_port) in current design

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>
static_controllability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_net
- flat\_pin
- flat\_port

### Return Value

string: 3-bit-string (y/n): zero-control\_one-control\_zee-control: nnn | nny | nyn | nyy | ynn | yny | yyn | yyy

### Description

The *static\_controllability* attribute displays static controllability (3-bit-string (y/n): zero-control\_one-control\_zee-control: nnn | nny | nyn | nyy | ynn | yny | yyn | yyy) for a user-specified design node (flat\_net | flat\_pin | flat\_port) in the current design.

Query on this attribute should be made only when one of the SpyGlass DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready
sg_shell> run_goal
.
```

```
.  
<output of run_goal>  
.br/>.br/>sg_shell> get_attribute [get_nets top.U0.A]  
static_controllability <nnn | nny | nyn | nyy | ynn | yny |  
yyn | yyy>
```

## See Also

[\*static\\_observability\*](#)

## static\_observability

Gets static observability (y (yes\_observable) | n (not\_observable)) for user-specified design node (flat\_pin | flat\_port) in current design

### Syntax

```
get_attribute <object: flat_pin | flat_port>
static_observability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string: y (yes\_observable) | n (not\_observable)

### Description

The *static\_observability* attribute displays static observability (y (yes\_observable) | n (not\_observable)) for a user-specified design node (flat\_net | flat\_pin | flat\_port) in the current design.

Query on this attribute should be made only when one of the SpyGlass DFT goals has been run.

### Examples

```
sg_shell> current_goal dft_scan_ready
sg_shell> run_goal
.
.
<output of run_goal>
.
.
```

```
sg_shell> get_attribute [get_pins top.U0.A]  
static_observability  
<y | n>
```

## See Also

[\*static\\_controllability\*](#)

## t01\_fault\_detectability\_los

**Gets zero to one transition fault detectability with launch on shift**

### Syntax

```
get_attribute <object: flat_pin | flat_port>  
t01_fault_detectability_los
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string

### Description

The *t01\_fault\_detectability\_los* attribute is used to get the zero to one transition (slow to rise - t/01) fault detectability with launch on shift.

Query on this attribute should be made only when the one of the goals with the *Info\_transitionCoverage* rule of the SpyGlass DFT DSM has been run with the *dsm\_launch\_method* parameter as los.

### Examples

```
sg_shell> current_goal dft_dsm_best_practice  
sg_shell> run_goal  
          <output of run_goal>  
sg_shell> get_attribute -class flat_pin "top.myInst.in1"  
t01_fault_detectability_los
```

### See Also

[t10\\_fault\\_detectability\\_los](#)

## t10\_fault\_detectability\_los

**Gets one to zero transition fault detectability with launch on shift**

### Syntax

```
get_attribute <object: flat_pin | flat_port>  
t10_fault_detectability_los
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string

### Description

The *t10\_fault\_detectability\_los* attribute is used to get the one to zero transition (slow to fall - t/10) fault detectability with launch on shift.

Query on this attribute should be made only when the one of the goals with the *Info\_transitionCoverage* rule of the SpyGlass DFT DSM has been run with the *dsm\_launch\_method* parameter as los.

### Examples

```
sg_shell> current_goal dft_dsm_best_practice  
sg_shell> run_goal  
          <output of run_goal>  
sg_shell> get_attribute -class flat_pin "top.myInst.in1"  
t10_fault_detectability_los
```

### See Also

[t01\\_fault\\_detectability\\_los](#)



## t01\_fault\_detectability\_loc

**Gets zero to one transition fault detectability with launch on capture**

### Syntax

```
get_attribute <object: flat_pin | flat_port>  
t01_fault_detectability_loc
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string

### Description

The *t01\_fault\_detectability\_loc* attribute is used to get the zero to one transition (slow to rise - t/01) fault detectability with launch on capture.

Query on this attribute should be made only when the one of the goals with the *Info\_transitionCoverage* rule of the SpyGlass DFT DSM has been run with the *dsm\_launch\_method* parameter as *loc*.

### Examples

```
sg_shell> current_goal dft_dsm_best_practice  
sg_shell> run_goal  
          <output of run_goal>  
sg_shell> get_attribute -class flat_pin "top.myInst.in1"  
t01_fault_detectability_loc
```

## See Also

[\*t10\\_fault\\_detectability\\_loc\*](#)

## t10\_fault\_detectability\_loc

**Gets one to zero transition fault detectability with launch on capture**

### Syntax

```
get_attribute <object: flat_pin | flat_port>
t10_fault_detectability_loc
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_pin
- flat\_port

### Return Value

string

### Description

The *t10\_fault\_detectability\_loc* attribute is used to get the one to zero transition (slow to fall - t/10) fault detectability with launch on capture : fault category.

Query on this attribute should be made only when the one of the goals with the *Info\_transitionCoverage* rule of the SpyGlass DFT DSM has been run with the *dsm\_launch\_method* parameter as *loc*.

### Examples

```
sg_shell> current_goal dft_dsm_best_practice
sg_shell> run_goal
           <output of run_goal>
sg_shell> get_attribute -class flat_pin "top.myInst.in1"
t10_fault_detectability_loc
```

## See Also

[\*t01\\_fault\\_detectability\\_loc\*](#)

## zero\_cnt\_probability

**Gets the probability that the flat port, pin, or net is at 0 when random test pattern is applied**

### Syntax

```
get_attribute <object: flat_net | flat_pin | flat_port>  
zero_cnt_probability
```

### Scope

Goal

### Object

This attribute is defined on the following object types:

- flat\_port
- flat\_pin
- flat\_port

### Return Value

float

### Description

The *zero\_cnt\_probability* attribute is used to get the probability that the flat port, pin, or net is at 0 when random test pattern is applied.

The *zero\_cnt\_probability* attribute returns control\_0 probability for a single pattern. This is independent of the pattern count specified using the *dft\_pattern\_count* parameter.

Query on this attribute should be made only when the *Info\_random\_resistance* rule of the SpyGlass DFT DSM has been run.

### Examples

```
sg_shell> get_attribute -class flat_pin "test.inst1.pin"  
zero_cnt_probability
```

**See Also**

[\*sa0\\_det\\_probability\*](#), [\*sa1\\_det\\_probability\*](#), [\*one\\_cnt\\_probability\*](#), [\*obs\\_probability\*](#)

## Power Attributes

The following table describes the various Tcl application attributes that are a part of the SpyGlass Power family.

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>activity</i>	flat_net	float	Returns the activity of a flat net
<i>blackbox_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the black box cells of a hierarchy
<i>blackbox_leakage_power</i>	flat_cell	float	Returns the total leakage power consumed by all the black box cells of a hierarchy
<i>blackbox_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the black box cells of a hierarchy
<i>capacitance_source</i>	flat_net	string	Returns the source of the capacitance of a flat net
<i>cell_size_for_power</i>	flat_cell	float	Returns the relative size of a flat cell as used for set_cell_allocation
<i>clock_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the clock cells of a hierarchy
<i>clock_leakage_power</i>	flat_net	float	Returns the total leakage power consumed by all the clock cells of a hierarchy
<i>clock_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the clock cells of a hierarchy
<i>combinational_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the combinational cells of a hierarchy
<i>combinational_leakage_power</i>	flat_net	float	Returns the total leakage power consumed by all the combinational cells of a hierarchy

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>combinational_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the combinational cells of a hierarchy
<i>fanout_capacitance</i>	flat_net	float	Returns the total pin capacitance of a given net
<i>internal_power</i>	flat_cell	float	Returns total internal power consumed by the given flat cell
<i>io_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the io cells of a hierarchy
<i>io_leakage_power</i>	flat_net	float	Returns the total leakage power consumed by all the io cells of a hierarchy
<i>io_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the io cells of a hierarchy
<i>is_activity_annotated</i>	flat_net	boolean	Returns a Boolean value, as an annotation status of a given net
<i>is_clock_gated</i>	flat_cell	boolean	Returns the gating status for the given flat cell
<i>is_internal_power_defined</i>	lib_cell	boolean	Returns a boolean value, based on whether internal power tables is specified for a library cell or not
<i>is_instantiated</i>	flat_cell	boolean	Returns a boolean value as instantiation status for the given flat cell
<i>leakage_power</i>	flat_cell	float	Returns total leakage power consumed by the given flat cell
<i>leakage_power_model</i>	lib_cell	string	Returns leakage power model of a library cell
<i>megacell_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the megacell cells of a hierarchy



## Debug Commands

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>megacell_leakage_power</i>	flat_cell	float	Returns the total leakage power consumed by all the megacell cells of a hierarchy
<i>megacell_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the megacell cells of a hierarchy
<i>memory_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the memory cells of a hierarchy
<i>memory_leakage_power</i>	flat_cell	float	Returns the total leakage power consumed by all the memory cells of a hierarchy
<i>memory_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the memory cells of a hierarchy
<i>net_frequency</i>	flat_net	float	Returns the frequency of the flat net
<i>other_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all those cells of a hierarchy that do not fall into any standard cell category
<i>other_leakage_power</i>	flat_cell	float	Returns the total leakage power consumed by all those cells of a hierarchy that do not fall into any standard cell category
<i>other_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all those cells of a hierarchy that do not fall into any standard cell category
<i>power_type</i>	flat_cell	string	Returns the type of a flat cell categorized for reporting the power consumption
<i>net_capacitance</i>	flat_net	float	Returns the wire capacitance of a flat net

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>probability</i>	flat_net	float	Returns the probability of a flat net
<i>root_clock_for_power</i>	flat_cell	float	Returns the root clock name for the given register flat cell
<i>sequential_internal_power</i>	flat_cell	float	Returns the total internal power consumed by all the sequential cells of a hierarchy
<i>sequential_leakage_power</i>	flat_cell	float	Returns the total leakage power consumed by all the sequential cells of a hierarchy
<i>sequential_switching_power</i>	flat_cell	float	Returns the total switching power consumed by all the sequential cells of a hierarchy
<i>switching_power</i>	flat_cell	float	Returns total switching power consumed by the given flat cell
<i>virtual_buffer_info</i>	flat_net	string	Returns virtual buffer information for the given flat net
<i>virtual_internal_power</i>	flat_net	float	Returns total internal power consumed by all virtual buffers on a given flat net
<i>virtual_leakage_power</i>	flat_net	float	Returns total leakage power consumed by all virtual buffers on the given flat net
<i>virtual_switching_power</i>	flat_net	float	Returns total switching power consumed all virtual buffers on the given flat net
<i>vt_classification</i>	lib_cell	string	Returns threshold voltage group of a library cell

## activity

Returns the activity of a flat net

### Syntax

```
get_attribute [get_nets <net_name>] activity
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *activity* attribute measures the relative toggles of a net with respect to the fastest clock in the design. The fastest clock toggles two times in a cycle. Therefore, the activity of the fastest clock is 2. If a signal toggles at half the frequency of that of the fastest clock, the activity of that signal will be 1. Therefore, the activity value will range from 0 to 2.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_nets top.U0.A ] activity
```

The output of this command is as follows:

```
0.5
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute [get_nets] -class flat_net top.w1  
activity
```

The output of this command is as follows:

0.6

### Example 3

The following command gives the activity information for all the nets in the design:

```
sg_shell> get_attribute [get_nets -filter  
{defined(activity)}] activity
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## blackbox\_internal\_power

Returns the total internal power consumed by all the black box cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
blackbox_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *blackbox\_internal\_power* attribute returns the total internal power consumed by the black box cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
blackbox_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## blackbox\_leakage\_power

Returns the total leakage power consumed by all the black box cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
blackbox_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *blackbox\_leakage\_power* attribute returns the total leakage power consumed by the black box cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
blackbox_leakage_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)



## blackbox\_switching\_power

Returns the total switching power consumed by all the black box cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
blackbox_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *blackbox\_switching\_power* attribute returns the total switching power consumed by the black box cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
blackbox_switching_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## capacitance\_source

Returns the source of the capacitance of a flat net

### Syntax

```
get_attribute [get_nets <net_name>] capacitance_source
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a string value if the query is successful

### Description

The *capacitance\_source* attribute gives the source for the given flat net. The capacitance of a net can be calculated as follows:

- If you specify the capacitance by using the `wire_load` and `wire_load_table` structures in the technology library, the return status will be `WIRE_LOAD`.
- If you set the capacitance by using an SPEF file, the return status will be `SPEF`.
- If you specify the capacitance by using the `set_load` command in SDC, the return status will be `SDC`.
- If you estimate the capacitance in SpyGlass by using the physical information in the LEF file, the return status will be `LEF`.

## Examples

### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_nets top.U0.A ]  
capacitance_source
```

The output of this command is as follows:

```
WIRE_LOAD
```

### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_net top.w1  
capacitance_source
```

The output of this command is as follows:

```
LEF
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## cell\_size\_for\_power

Returns the relative size of a flat cell as used for `set_cell_allocation`

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value on successful query.

### Description

The `cell_size_for_power` attribute returns the relative size of the flat cell used for `set_cell_allocation`. It's value is greater than 0.0.

### Examples

Consider the following command:

```
sg_shell> get_attribute -class top.U0 cell_size_for_power
```

The output of this command is as follows:

```
1.0
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## clock\_internal\_power

Returns the total internal power consumed by all the clock cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
clock_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *clock\_internal\_power* attribute returns the total internal power consumed by the clock cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
clock_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## clock\_leakage\_power

Returns the total leakage power consumed by all the clock cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
clock_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the flat\_cell object type.

### Return Value

Returns a float value if the query is successful

### Description

The clock\_leakage\_power attribute returns the total leakage power consumed by the clock cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the enable\_hier\_flattening option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
clock_leakage_power
```

The output of this command is as follows:

```
0.0365
```



You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## clock\_switching\_power

Returns the total switching power consumed by all the clock cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
clock_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *clock\_switching\_power* attribute returns the total switching power consumed by the clock cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
clock_switching_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## combinational\_internal\_power

Returns the total internal power consumed by all the combinational cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
combinational_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *combinational\_internal\_power* attribute returns the total internal power consumed by the combinational cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
combinational_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## combinational\_leakage\_power

Returns the total leakage power consumed by all the combinational cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
combinational_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *combinational\_leakage\_power* attribute returns the total leakage power consumed by the combinational cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
combinational_leakage_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## combinational\_switching\_power

Returns the total switching power consumed by all the combinational cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
combinational_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *combinational\_switching\_power* attribute returns the total switching power consumed by the combinational cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
combinational_switching_power
```

The output of this command is as follows:

```
0.0365
```



You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## fanout\_capacitance

Returns the total pin capacitance of a given net

### Syntax

```
get_attribute [get_nets <net_name>] fanout_capacitance
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value if the query is successful and nothing if the query is unsuccessful

### Description

The *fanout\_capacitance* attribute returns the total pin capacitance of the given net. It is the arithmetic sum of the capacitance of the pins that are connected to the given flat net.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_nets top.U0.A ]  
fanout_capacitance
```

The output of this command is as follows:

```
0.5
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_net top.w1  
fanout_capacitance
```

The output of this command is as follows:

```
.32
```

### Example 3

The following command gives the fan-out capacitance of all the nets in the design where the *fanout\_capacitance* attribute is defined:

```
sg_shell> get_attribute [get_nets -filter  
{defined(fanout_capacitance)} ] fanout_capacitance
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## internal\_power

Returns total internal power consumed by the given flat cell or hierarchical cell

### Syntax

```
get_attribute -class flat_cell <cell_name> internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value on successful query

### Description

The *internal\_power* attribute returns total internal power consumed by the given instance during the execution of the goal.

You can get the internal power of a hierarchical cell by setting the `enable_hier_flattening` option to `yes`, before the execution of the goal.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg internal_power
```

The output of this command is as follows:

```
0.0000423
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.mem_reg internal_power
```

The output of this command is as follows:

```
0.0002345
```

### Example 3

For getting `internal_power` of a hierarchical cell, set the following option, before executing the goal (this can also be specified in a `.spq` file):

```
sg_shell> set_option enable_hier_flattening yes
```

Run the goal:

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat` by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## io\_internal\_power

Returns the total internal power consumed by all the io cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
io_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *io\_internal\_power* attribute returns the total internal power consumed by the io cells of the given hierarchical instance during the goal execution. This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
io_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following

option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## io\_leakage\_power

Returns the total leakage power consumed by all the io cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name> io_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *io\_leakage\_power* attribute returns the total leakage power consumed by the io cells of the given hierarchical instance during the goal execution. This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
io_leakage_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:



```
sg_shell> set_pref dq_design_view_type flat
```

**See Also**

[get\\_attribute](#), [filter\\_collection](#)

## io\_switching\_power

Returns the total switching power consumed by all the io cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
io_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *io\_switching\_power* attribute returns the total switching power consumed by the io cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
io_switching_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[\*get\\_attribute\*](#), [\*filter\\_collection\*](#)

## is\_activity\_annotated

Returns a boolean value, as an annotation status of a given net

### Syntax

```
get_attribute [get_nets <net_name>] is_activity_annotated
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns `true`, if the given net is annotated, and `false`, if the given net is not annotated

### Description

The *is\_activity\_annotated* attribute returns the activity annotation status. The attribute determines whether the activity on a net is annotated or not. If the net is captured in the specified simulation file (VCD/FSDB/SAIF) or the SGDC file by using the *activity* or *clock* command, the return status will be `true`. If the activity is not specified or is propagated by using the internal activity engine, the return status will be `false`.

Activity annotation will not be defined, if the given net is a supply net, synthesis-generated net, hanging net, or undriven net.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_nets top.U0.A ]  
is_activity_annotated
```

The output of this command is as follows:

```
true
```

**Example 2**

Consider the following command:

```
sg_shell> get_attribute -class flat_net top.w1  
is_activity_annotated
```

The output of this command is as follows:

```
false
```

**Example 3**

The following command gives the activity annotation for all nets in the design:

```
sg_shell> get_attribute [get_nets -filter  
{defined(is_activity_annotated)}] is_activity_annotated
```

**See Also**

[get\\_attribute](#), [filter\\_collection](#)

## is\_clock\_gated

Returns the gating status for the given flat cell

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a boolean value on successful query.

### Description

The *is\_clock\_gated* attribute returns a boolean value to indicate if given flat cell is clock gated or not. This attribute is not defined for flat cells other than registers.

### Examples

Consider the following commands:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg is_clock_gated
```

The output of this command is as follows

```
true
```

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.mem_reg is_clock_gated
```

The output of this command is as follows:

```
false
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_internal\_power\_defined

Returns a boolean value, based on whether internal power tables are specified for a library cell or not

### Syntax

```
get_attribute -class lib_cell <cell_name>  
is_internal_power_defined
```

### Scope

Goal

### Object

This attribute is defined on the `lib_cell` object type.

### Return Value

Returns a boolean value.

### Description

The *is\_internal\_power\_defined* attribute returns true if internal power table is specified for a library cell, otherwise it returns false.

### Examples

Consider the following command:

```
sg_shell> get_attribute -class lib_cell my_lib.my_cell  
is_internal_power_defined
```

The output of this command is as follows:

```
true
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## is\_instantiated

Returns a boolean value as instantiation status for the given flat cell

### Syntax

```
get_attribute -class flat_cell <cell_name> is_instantiated
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

- Returns true, if the given cell is instantiated in the design (RTL or netlist)
- Returns false, if the given cell is not instantiated and is inferred during synthesis

### Description

The *is\_instantiated* attribute returns true if the flat cell is instantiated in the design. Here, instantiated means that the cell exists in the RTL or netlist given to the tool. A cell is not instantiated if it is inferred by synthesis.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg is_instantiated
```

The output of this command is as follows:

```
false
```



**Example 2**

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.mem_reg is_instantiated
```

The output of this command is as follows:

```
false
```

**See Also**

[get\\_attribute](#), [filter\\_collection](#)

## leakage\_power

Returns total leakage power consumed by the given flat cell or hierarchical cell

### Syntax

```
get_attribute -class flat_cell <cell_name> leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value on successful query

### Description

The *leakage\_power* attribute returns total leakage power consumed by the given instance during the execution of the goal.

You can get the leakage power of a hierarchical cell by setting the `enable_hier_flattening` option to `yes`, before the execution of the goal.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg leakage_power
```

The output of this command is as follows:

```
0.0000423
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.mem_reg leakage_power
```

The output of this command is as follows:

```
0.0002345
```

### Example 3

For getting `leakage_power` of a hierarchical cell, set the following option, before executing the goal (this can also be specified in a `.spq` file):

```
sg_shell> set_option enable_hier_flattening yes
```

Run the goal:

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
leakage_power
```

The output of this command is as follows:

```
0.036523
```

You can switch to `flat_view` from `hier_flat` by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## leakage\_power\_model

Returns leakage power model of a library cell

### Syntax

```
get_attribute -class lib_cell <cell_name>  
leakage_power_model
```

### Scope

Goal

### Object

This attribute is defined on the `lib_cell` object type.

### Return Value

Returns a string value on successful query. Return values can be `state_dependent`, `cell_default`, and `not_defined`.

### Description

The *leakage\_power\_model* attribute returns leakage power model for the given library cell. Leakage power model of a library cell is given as follows:

- If state based leakage power is defined for the given cell, then return status will be `state_dependent`.
- If state based leakage power is not defined for given cell and cell default leakage power is defined, then the return status will be `cell_default`.
- If cell default leakage power is not defined for given cell, and lib default leakage power is defined, then return status will be `lib_default`.
- If library default leakage power is also not defined then return status will be `not_defined`.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class lib_cell my_lib.my_cell_1  
leakage_power_model
```

The output of this command is as follows:

```
state_dependent
```

### **Example 2**

Consider the following command:

```
sg_shell> get_attribute -class lib_cell my_lib.my_cell2  
leakage_power_model
```

The output of this command is as follows:

```
lib_default
```

## **See Also**

[get\\_attribute](#), [filter\\_collection](#)

## megacell\_internal\_power

Returns the total internal power consumed by all the megacell cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>
megacell_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *megacell\_internal\_power* attribute returns the total internal power consumed by the megacell cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
sg_shell> get_attribute -class flat_cell top.U_MEM1
megacell_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## megacell\_leakage\_power

Returns the total leakage power consumed by all the megacell cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
megacell_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *megacell\_leakage\_power* attribute returns the total leakage power consumed by the megacell cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
megacell_leakage_power
```

The output of this command is as follows:

```
0.0365
```



You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## megacell\_switching\_power

Returns the total switching power consumed by all the megacell cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>
megacell_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *megacell\_switching\_power* attribute returns the total switching power consumed by the megacell cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
sg_shell> get_attribute -class flat_cell top.U_MEM1
megacell_switching_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## memory\_internal\_power

Returns the total internal power consumed by all the memory cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
memory_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *memory\_internal\_power* attribute returns the total internal power consumed by the memory cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
memory_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## memory\_leakage\_power

Returns the total leakage power consumed by all the memory cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
memory_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *memory\_leakage\_power* attribute returns the total leakage power consumed by the memory cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
memory_leakage_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## memory\_switching\_power

Returns the total switching power consumed by all the memory cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
memory_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *memory\_switching\_power* attribute returns the total switching power consumed by the memory cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
memory_switching_power
```

The output of this command is as follows:

```
0.0365
```



You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## net\_frequency

Returns the frequency of the flat net

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value on successful query.

### Description

The *net\_frequency* attribute returns the frequency of a flat net. The frequency is the measurement of the number of complete AC cycles that occur in one second for a net on an average. The frequency is measured in Hertz (Hz).

### Examples

Consider the following commands:

```
sg_shell> get_attribute [get_nets top.U0.A ] net_frequency
```

The output of this command is as follows

```
0.0
```

```
sg_shell> get_attribute [get_nets -filter  
{defined(net_frequency)} ] net_frequency
```

The above command gives frequency information of all nets

```
sg_shell> get_attribute -class flat_net top.w1  
net_frequency
```

The output of this command is as follows:

```
400.00
```

**See Also**

[get\\_attribute](#), [filter\\_collection](#)

## other\_internal\_power

Returns the total internal power consumed by all those cells of a hierarchy that do not fall into any standard cell category

### Syntax

```
get_attribute -class flat_cell <cell_name>  
other_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *other\_internal\_power* attribute returns the total internal power consumed during the goal execution by those cells of the given hierarchical instance that do not fall into any standard cell category.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
other_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## other\_leakage\_power

Returns the total leakage power consumed by all those cells of a hierarchy that do not fall into any standard cell category

### Syntax

```
get_attribute -class flat_cell <cell_name>  
other_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *other\_leakage\_power* attribute returns the total leakage power consumed during the goal execution by those cells of the given hierarchical instance that do not fall into any standard cell category.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
other_leakage_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## other\_switching\_power

Returns the total switching power consumed by all those cells of a hierarchy that do not fall into any standard cell category

### Syntax

```
get_attribute -class flat_cell <cell_name>  
other_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *other\_switching\_power* attribute returns the total switching power consumed during the goal execution by those cells of the given hierarchical instance that do not fall into any standard cell category.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
other_switching_power
```

The output of this command is as follows:

```
0.0365
```



You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## power\_type

Returns the category of power to which this flat cell is contributing

### Syntax

```
get_attribute -class flat_cell <cell_name> power_type
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a string value on successful query. Return value can be `combinational`, `sequential`, `clock`, `black_box`, `memory`, `iopad`, and others.

### Description

The *power\_type* attribute returns the power contribution type of the given flat cell during power estimation. That is, while categorizing the power consumption on component basis, each cell power contribution is distributed into one category.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg power_type
```

The output of this command is as follows:

```
sequential
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.mem_reg power_type
```

The output of this command is as follows:

```
combinational
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## net\_capacitance

Returns the wire capacitance of a flat net

### Syntax

```
get_attribute [get_nets <net_name>] net_capacitance
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *net\_capacitance* attribute represents a float value. It specifies the total capacitive load of the specified net, excluding the capacitance of the connected pins. The capacitance will be expressed in the form of picofarads, or pF.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_nets top.U0.A ] net_capacitance
```

The output of this command is as follows:

```
0.5
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_net top.w1  
net_capacitance
```

The output of this command is as follows:

```
0.32
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## probability

Returns the probability of a flat net

### Syntax

```
get_attribute [get_nets <net_name>] probability
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value if the query is successful and nothing if the query is unsuccessful

### Description

The *probability* attribute returns the probability of a flat net. Probability is the percentage of the time in which the signal is high on the given net. For example, if a net is high for 70% of the time, its probability will be 0.7.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute [get_nets top.U0.A ] probability
```

The output of this command is as follows:

```
0.5
```

#### Example 2

Consider the following command:

```
sg_shell> get_attribute [get_nets] -class flat_net top.w1  
probability
```

The output of this command is as follows:

0.6

### Example 3

The following command gives the activity information for all nets in the design:

```
sg_shell> get_attribute [get_nets -filter  
{defined(activity)} ] probability
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)

## root\_clock\_for\_power

Returns the root clock name for the given register flat cell

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *root\_clock\_for\_power* returns the root clock net name for the registers. For non-register flat cells, this attribute will be undefined.

In addition, this command also returns root clock for a latch if the `pe_ignore_latch_in_inferred_clock` parameter is set to no.

**NOTE:** *The Tcl attribute for root clock and gating status is available under the `adv_power_rednso` license.*

### Examples

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg root_clock_for_power
```

The output of this command is as follows:

```
Top.clk1
```

### See Also

[get\\_attribute](#), [filter\\_collection](#)



## sequential\_internal\_power

Returns the total internal power consumed by all the sequential cells of a hierarchy

### Syntax

```
get_attribute [get_nets <net_name>]  
sequential_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *sequential\_internal\_power* attribute returns the total internal power consumed by the sequential cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
sequential_internal_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## sequential\_leakage\_power

Returns the total leakage power consumed by all the sequential cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
sequential_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *sequential\_leakage\_power* attribute returns the total leakage power consumed by the sequential cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
sequential_leakage_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## sequential\_switching\_power

Returns the total switching power consumed by all the sequential cells of a hierarchy

### Syntax

```
get_attribute -class flat_cell <cell_name>  
sequential_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value if the query is successful

### Description

The *sequential\_switching\_power* attribute returns the total switching power consumed by the sequential cells of the given hierarchical instance during the goal execution.

This information is available only for hierarchical instances and not leaf cell. You need to specify the `enable_hier_flattening` option before running the goal.

### Examples

```
sg_shell> set_option enable_hier_flattening yes
```

```
sg_shell> set_pref dq_design_view_type hier_flat  
sg_shell> get_attribute -class flat_cell top.U_MEM1  
sequential_switching_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat`, by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## switching\_power

Returns total switching power consumed by the given flat cell or hierarchical cell

### Syntax

```
get_attribute -class flat_cell <cell_name> switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_cell` object type.

### Return Value

Returns a float value on successful query

### Description

The *switching\_power* attribute returns total switching power consumed by the given instance during the execution of the goal. This switching power is consumed by charging and discharging of the load connected to the output pins of this flat cell.

You can get the switching power of a hierarchical cell by setting the `enable_hier_flattening` option to `yes`, before the execution of the goal.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.fifomem_reg switching_power
```

The output of this command is as follows:

```
0.0000423
```

### Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_cell  
top.U_MEM1.U_FF1.mem_reg switching_power
```

The output of this command is as follows:

```
0.0002345
```

### Example 3

For getting `switching_power` of a hierarchical cell, set the following option, before executing the goal (this can also be specified in a `.spq` file):

```
sg_shell> set_option enable_hier_flattening yes
```

Run the goal:

```
sg_shell> set_pref dq_design_view_type hier_flat
```

```
sg_shell> get_attribute -class flat_cell top.U_MEM1  
switching_power
```

The output of this command is as follows:

```
0.0365
```

You can switch to `flat_view` from `hier_flat` by setting the following option:

```
sg_shell> set_pref dq_design_view_type flat
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)



## virtual\_buffer\_info

Returns virtual buffer information for the given flat net

### Syntax

```
get_attribute -class flat_net <net_name> virtual_buffer_info
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a string with a list of pairs on successful query.

### Description

The *virtual\_buffer\_info* attribute returns all virtual buffers (SpyGlass assumed buffers) and their count on the given flat net during the execution of the goal.

Return value consist of pairs. Each pair represents the buffer name and their count.

Generally, SpyGlass estimates some buffers during power estimation for clock nets and high fanout nets, based on maximum pin capacitance and load on the net.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_net Top.mDec_net_0_2770  
virtual_buffer_info
```

The output of this command is as follows:

```
{BUF_B1 2 }
```

**Example 2**

Consider the following command:

```
sg_shell> get_attribute -class flat_net Top.r1t1c_N0  
virtual_buffer_info
```

The output of this command is as follows:

```
{BUF_B3 1 }
```

**See Also**

[get\\_attribute](#), [filter\\_collection](#)

## virtual\_internal\_power

Returns total internal power consumed by all virtual buffers on a given flat net

### Syntax

```
get_attribute -class flat_net <net_name>  
virtual_internal_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value on successful query.

### Description

The *virtual\_internal\_power* attribute returns total internal power consumed by all virtual buffers on a given flat net during the execution of the goal.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_net  
top.U_MEM1.U_FF1.fifomem_reg.o2 virtual_internal_power
```

The output of this command is as follows:

```
0.0000423
```

## Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_net  
top.U_MEM1.U_FF1.mem_reg.o1 virtual_internal_power
```

The output of this command is as follows:

```
0.0002345
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## virtual\_leakage\_power

Returns total leakage power consumed by all virtual buffers on the given flat net

### Syntax

```
get_attribute -class flat_net <net_name>  
virtual_leakage_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value on successful query.

### Description

The *virtual\_leakage\_power* attribute returns total leakage power consumed by all virtual buffers on the given flat net during the execution of the goal.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_net  
top.U_MEM1.U_FF1.fifomem_reg.I1 virtual_leakage_power
```

The output of this command is as follows:

```
0.0000423
```

## Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_net  
top.U_MEM1.U_FF1.mem_reg.o2 virtual_leakage_power
```

The output of this command is as follows:

```
0.0002345
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## virtual\_switching\_power

Returns total switching power consumed by all virtual buffers on the given flat net

### Syntax

```
get_attribute -class flat_net <net_name>  
virtual_switching_power
```

### Scope

Goal

### Object

This attribute is defined on the `flat_net` object type.

### Return Value

Returns a float value on successful query.

### Description

The *virtual\_switching\_power* attribute returns total switching power consumed by all virtual buffers on the given flat net during the execution of the goal.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class flat_net  
top.U_MEM1.U_FF1.fifomem_reg.il virtual_switching_power
```

The output of this command is as follows:

```
0.0000423
```

## Example 2

Consider the following command:

```
sg_shell> get_attribute -class flat_net  
top.U_MEM1.U_FF1.mem_reg.i2 virtual_switching_power
```

The output of this command is as follows:

```
0.0002345
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)



## vt\_classification

Returns threshold voltage group of a library cell

### Syntax

```
get_attribute -class lib_cell <cell_name> vt_classification
```

### Scope

Goal

### Object

This attribute is defined on the `lib_cell` object type.

### Return Value

Returns a string value on successful query.

### Description

The *vt\_classification* attribute gives the threshold voltage group of given library cell during the power estimation. This can be based on the *threshold\_voltage\_group* attribute in the library or on the *vt\_mix\_percentage* SGDC command. The user can create own groups using the *vt\_mix\_percentage* command, even if the *threshold\_voltage\_group* attribute is not specified in the *.lib* file. This attribute returns a resultant voltage group assigned to each cell using *.lib* information and *vt\_mix\_percentage* information.

### Examples

#### Example 1

Consider the following command:

```
sg_shell> get_attribute -class lib_cell  
SR50_N_25_1.2_CORE_UHD.db.BF006Q vt_classification
```

The output of this command is as follows:

```
-cell:*BF*
```

## Example 2

Consider the following command:

```
sg_shell> get_attribute -class lib_cell typical2.A021X2  
vt_classification
```

The output of this command is as follows:

```
-lib:typical2
```

## See Also

[get\\_attribute](#), [filter\\_collection](#)

## Power Verify Attributes

The following table describes the various Tcl application attributes that are a part of the SpyGlass Power Verify solution.

Attribute Name	Object	Type	Description
<i>clamp_value</i>	pwr_isolation_node	string	Displays the clamp value of the isolation strategy
<i>control_port</i>	pwr_psw_node	string	Displays the control signal of power switch
<i>ground_supply</i>	pwr_intent_node	string	Displays the ground supply name
<i>input_supply_port</i>	pwr_psw_node	string	Displays the input supply of power switch
<i>input_supply_set</i>	pwr_level_shift_node	string	Displays the input supply set of level shifter
<i>isolation_ground_net</i>	pwr_isolation_node	string	Displays the isolation ground net of the isolation strategy
<i>isolation_power_net</i>	pwr_isolation_node	string	Displays the isolation power supply of the isolation strategy
<i>isolation_sense</i>	pwr_isolation_node	string	Displays the isolation sense of the isolation strategy
<i>isolation_signal</i>	pwr_isolation_node	string	Displays the isolation signal of the isolation strategy
<i>location</i>	pwr_isolation_node pwr_level_shift_node	string	Displays the location of the applied strategy
<i>name</i>	pwr_isolation_node pwr_retention_node pwr_psw_node pwr_level_shift_node	string	Displays the name of the strategy

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Description</b>
<i>output_supply_port</i>	pwr_psw_node	string	Displays the input supply of power switch
<i>output_supply_set</i>	pwr_level_shift_node	string	Displays the output supply set of level shifter
<i>power_domain</i>	pwr_intent_node	string	Displays the power domain name
<i>power_supply</i>	pwr_intent_node	string	Displays the power supply name
<i>restore_signal</i>	pwr_retention_node	string	Displays the restore signal of the retention strategy
<i>retention_ground_supply</i>	pwr_retention_node	string	Displays the retention ground supply of the retention strategy
<i>retention_power_supply</i>	pwr_retention_node	string	Displays the retention power supply of the retention strategy
<i>save_signal</i>	pwr_retention_node	string	Displays the save signal of the retention strategy
<i>sink</i>	pwr_isolation_node	string	Displays the sink supply of an applied strategy
<i>source</i>	pwr_isolation_node pwr_level_shift_node	string	Displays the source supply of an applied strategy
<i>supply_name</i>	pwr_supply_node	string	Displays the supply name corresponding to a design net
<i>rule</i>	pwr_level_shift_node	string	Displays the rule of level shifter strategy
<i>type</i>	pwr_supply_node	string	Displays the type of supply net
<i>voltage_range_min</i>	pwr_intent_node	float	Returns the minimum value to voltage range
<i>voltage_range_max</i>	pwr_intent_node	float	Returns the maximum value to voltage range

## clamp\_value

**Displays the clamp value of the isolation strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ] ]  
clamp_value
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *clamp\_value* attribute gives the clamp value of the isolation strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells  
top.iso ] ] clamp_value
```

The output of this command is as follows:

```
1
```

### See Also

[get\\_attribute](#)

## control\_port

**Displays the control signal of power switch**

### Syntax

```
get_attribute [ get_power_switch_info [ get_cells <cell-  
name> ] ] control_port
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_psw_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *control\_port* attribute gives the control signal of the power switch corresponding to cell name.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_power_switch_info [ get_cells  
top.sw1.psw ] ] control_port
```

The output of this command is as follows:

```
top.sw1.CTRL
```

### See Also

[get\\_attribute](#)

## input\_supply\_port

Displays the input supply of power switch

### Syntax

```
get_attribute [ get_power_switch_info [ get_cells <cell-  
name> ] ] input_supply_port
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_psw_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *input\_supply\_port* attribute gives the input supply of the power switch corresponding to cell name.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_power_switch_info [ get_cells  
top.sw1.psw ] ] input_supply_port
```

The output of this command is as follows:

```
VDD1
```

### See Also

[get\\_attribute](#)

## input\_supply\_set

Displays the input supply set of level shifter

### Syntax

```
get_attribute [ get_level_shifter _info [ get_cells  
<cellname> ] ] input_supply_set
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_level_shift_node` object.

### Return Value

Returns a string value on successful query.

### Description

The `input_supply_set` attribute gives the input supply set of the power switch corresponding to cell name.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_level_shifter _info [ get_cells  
top.sw1.psw ] ] input_supply_set
```

The output of this command is as follows:

```
ss1
```

### See Also

[get\\_attribute](#)



## isolation\_ground\_net

Displays the isolation ground net of the isolation strategy

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]  
] isolation_ground_net
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` object.

### Return Value

Returns a string value on successful query.

### Description

The `isolation_ground_net` attribute gives the isolation ground supply of the isolation strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> sg_shell> get_attribute [ get_isolation_info [ get_cells top.iso ] ] isolation_ground_net
```

The output of this command is as follows:

```
VSS
```

### See Also

[get\\_attribute](#)

## isolation\_power\_net

**Displays the isolation power supply of the isolation strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]  
] isolation_power_net
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *isolation\_power\_net* attribute gives the isolation power supply of the isolation strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells  
top.iso ] ] isolation_power_net
```

The output of this command is as follows:

```
VDD
```

### See Also

[get\\_attribute](#)

## isolation\_sense

**Displays the isolation sense of the isolation strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]  
] isolation_sense
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *isolation\_sense* attribute gives the isolation sense of the isolation strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells  
top.iso ] ] isolation_sense
```

The output of this command is as follows:

```
high
```

### See Also

[get\\_attribute](#)

## isolation\_signal

**Displays the isolation signal of the isolation strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]  
] isolation_signal
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *isolation\_signal* attribute gives the isolation signal of the isolation strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells  
top.iso ] ] isolation_signal
```

The output of this command is as follows:

```
top.iso
```

### See Also

[get\\_attribute](#)

## location

**Displays the location of the applied strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]  
] location  
  
get_attribute [ get_level_shifter_info [ get_cells <cell-  
name> ] ] location
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` and `pwr_level_shift_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *location* attribute gives the location of the strategy applied (isolation or level shifter) on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells  
top.iso ] ] location
```

```
sg_shell> get_attribute [ get_level_shifter_info [ get_cells  
top.lsl ] ] location
```

The output of this command is as follows:

```
parent
```

## See Also

[\*get\\_attribute\*](#)

## name

**Displays the name of the strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ] ]  
name  
get_attribute [ get_retention_info [ get_cells <cell-name> ] ]  
name  
get_attribute [ get_power_switch_info [ get_cells <cell-  
name> ] ]  
name  
get_attribute [ get_level_shifter_info [ get_cells <cell-  
name> ] ]
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node`, `pwr_retention_node`, `pwr_psw_node`, and `pwr_level_shift_node` objects.

### Return Value

Returns a string value on successful query.

### Description

The *name* attribute gives the name of the strategy applied (isolation, retention, power switch, or level shifter) on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells  
top.iso ] ] name
```

The output of this command is as follows:

```
iso1
```

```
sg_shell> get_attribute [ get_retention_info [ get_cells  
top.ret ] ] name
```

The output of this command is as follows:

```
ret1
```

```
sg_shell> get_attribute [ get_power_switch_info [ get_cells  
top.psw ] ] name
```

The output of this command is as follows:

```
psw1
```

```
sg_shell> get_attribute [ get_level_shifter_info [ get_cells  
top.ls ] ] name
```

The output of this command is as follows:

```
ls1
```

## See Also

[\*get\\_attribute\*](#)



## output\_supply\_port

Displays the input supply of power switch

### Syntax

```
get_attribute [ get_power_switch_info [ get_cells <cell-  
name> ] ] output_supply_port
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_psw_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *output\_supply\_port* attribute gives the output supply of the power switch corresponding to cell name.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_power_switch_info [ get_cells  
top.sw1.psw ] ] output_supply_port
```

The output of this command is as follows:

```
VDDg
```

### See Also

[get\\_attribute](#)

## output\_supply\_set

**Displays the input supply set of level shifter**

### Syntax

```
get_attribute [ get_level_shifter _info [ get_cells  
<cellname> ] ] output_supply_set
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_level_shift_node` object.

### Return Value

Returns a string value on successful query.

### Description

The `output_supply_set` attribute gives the output supply set of the power switch corresponding to cell name.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_level_shifter _info [ get_cells  
top.sw1.psw ] ] output_supply_set
```

The output of this command is as follows:

```
ss2
```

### See Also

[get\\_attribute](#)

## power\_domain

Displays the power domain name

### Syntax

```
get_attribute [ get_pwr_intent [ get_cells < cell_name > |  
get_pins < pin_name > | get_ports < port_name > ] ]  
power_domain
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_intent_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *power\_domain* attribute gives the name of the power domain.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_pwr_intent [ get_cells  
top.inst1 ] ] power_domain
```

The output of this command is as follows:

```
top
```

### See Also

[get\\_attribute](#)

## power\_supply

**Displays the power supply name**

### Syntax

```
get_attribute [ get_pwr_intent [ get_cells < cell_name > |  
get_pins < pin_name > | get_ports < port_name > ] ]  
power_supply
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_intent_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *power\_supply* attribute gives the name of the power supply.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_pwr_intent [ get_cells  
top.inst1 ] ] power_supply
```

The output of this command is as follows:

```
VTOP
```

### See Also

[get\\_attribute](#)

## ground\_supply

Displays the ground supply name

### Syntax

```
get_attribute [ get_pwr_intent [ get_cells < cell_name > |  
get_pins < pin_name > | get_ports < port_name > ] ]  
ground_supply
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_intent_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *ground\_supply* attribute gives the name of the ground supply.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_pwr_intent [ get_cells  
top.inst1 ] ] ground_supply
```

The output of this command is as follows:

```
VSS
```

### See Also

[get\\_attribute](#)

## restore\_signal

**Displays the restore signal of the retention strategy**

### Syntax

```
get_attribute [ get_retention_info [ get_cells <cell-name> ]  
] restore_signal
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_retention_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *restore\_signal* attribute gives the restore signal of the retention strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_retention_info [ get_cells  
top.inst1.buf ] ] restore_signal
```

The output of this command is as follows:

```
top.inst1.res
```

### See Also

[get\\_attribute](#)

## retention\_ground\_supply

**Displays the retention ground supply of the retention strategy**

### Syntax

```
get_attribute [ get_retention_info [ get_cells <cell-name> ]  
] retention_ground_supply
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_retention_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *retention\_ground\_supply* attribute gives the retention ground supply of the retention strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_retention_info [ get_cells  
top.inst1.buf ] ] retention_ground_supply
```

The output of this command is as follows:

```
VSS
```

### See Also

[get\\_attribute](#)

## retention\_power\_supply

**Displays the retention power supply of the retention strategy**

### Syntax

```
get_attribute [ get_retention_info [ get_cells <cell-name> ]  
] retention_power_supply
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_retention_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *retention\_power\_supply* attribute gives the retention power supply of the retention strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_retention_info [ get_cells  
top.inst1.buf ] ] retention_power_supply
```

The output of this command is as follows:

```
VDD
```

### See Also

[get\\_attribute](#)



## save\_signal

**Displays the save signal of the retention strategy**

### Syntax

```
get_attribute [ get_retention_info [ get_cells <cell-name> ]  
] save_signal
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_retention_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *save\_signal* attribute gives the save signal of the retention strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_retention_info [ get_cells  
top.inst1.buf ] ] save_signal
```

The output of this command is as follows:

```
top.inst1.save
```

### See Also

[get\\_attribute](#)

## sink

**Displays the sink supply of an applied strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]
] sink

get_attribute [ get_level_shifter_info [ get_cells <cell-
name> ] ] sink
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` and `pwr_level_shift_node` objects.

### Return Value

Returns a string value on successful query.

### Description

The *sink* attribute gives the sink supply of the strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells
top.iso ] ] sink
```

```
sg_shell> get_attribute [ get_level_shifter_info [ get_cells
top.lsl ] ] sink
```

The output of this command is as follows:

```
SS1
```

### See Also

[get\\_attribute](#)

## source

**Displays the source supply of an applied strategy**

### Syntax

```
get_attribute [ get_isolation_info [ get_cells <cell-name> ]
] source

get_attribute [ get_level_shifter_info [ get_cells <cell-
name> ] ] source
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_isolation_node` and `pwr_level_shift_node` objects.

### Return Value

Returns a string value on successful query.

### Description

The *source* attribute gives the source supply of the strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_isolation_info [ get_cells
top.iso ] ] source
```

```
sg_shell> get_attribute [ get_level_shifter_info [ get_cells
top.lsl ] ] source
```

The output of this command is as follows:

```
SS2
```

## See Also

[\*get\\_attribute\*](#)

## supply\_name

**Displays the supply name corresponding to a design net**

### Syntax

```
get_attribute [ get_supply_info [get_nets <net-name> ] ]  
supply_name
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_supply_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *supply\_name* attribute gives the supply name corresponding to a design net.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_supply_info [get_nets top.VDD ]  
] supply_name
```

The output of this command is as follows:

```
VDD
```

### See Also

[get\\_attribute](#)

## rule

**Displays the rule of level shifter strategy**

### Syntax

```
get_attribute [ get_level_shifter_info [ get_cells <cell-  
name> ] ] rule
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_level_shift_node` objects.

### Return Value

Returns a string value on successful query.

### Description

The *rule* attribute gives the type of the level shifter strategy applied on the cell.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_level_shifter_info [ get_cells  
top.lsl ] ] rule
```

The output of this command is as follows:

```
Low_To_High
```

### See Also

[get\\_attribute](#)

## type

**Displays the type of supply net**

### Syntax

```
get_attribute [ get_supply_info [get_nets <net-name> ] ] type
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_supply_node` object.

### Return Value

Returns a string value on successful query.

### Description

The *rule* attribute gives the type of the supply net.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_supply_info [get_nets top.VDD ]  
] rule
```

The output of this command is as follows:

```
power
```

### See Also

[get\\_attribute](#)

## voltage\_range\_min

Returns the minimum value to voltage range

### Syntax

```
get_attribute [ get_pwr_intent [ get_cells < cell_name > |  
get_pins < pin_name > | get_ports < port_name > ] ]  
voltage_range_min
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_intent_node` object.

### Return Value

Returns a float value on successful query.

### Description

The `voltage_range_min` attribute gives the minimum value to voltage range.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_pwr_intent [ get_cells  
top.inst1 ] ] voltage_range_min
```

The output of this command is as follows:

```
0.6000000238418579
```

### See Also

[get\\_attribute](#)



## voltage\_range\_max

Returns the maximum value to voltage range

### Syntax

```
get_attribute [ get_pwr_intent [ get_cells < cell_name > |  
get_pins < pin_name > | get_ports < port_name > ] ]  
voltage_range_max
```

### Scope

Goal

### Object

This attribute is defined on the `pwr_intent_node` object.

### Return Value

Returns a float value on successful query.

### Description

The `voltage_range_max` attribute gives the maximum value to voltage range.

### Examples

Consider the following command:

```
sg_shell> get_attribute [ get_pwr_intent [ get_cells  
top.inst1 ] ] voltage_range_max
```

The output of this command is as follows:

```
0.6000000238418579
```

### See Also

[get\\_attribute](#)

## Miscellaneous Commands

Miscellaneous commands include commands related to shell features. The following table describes the various miscellaneous commands:

<b>Command</b>	<b>Description</b>
<i>alias</i>	Creates an alias for a group of word(s)
<i>benchmark</i>	Monitors run time and memory usage between two designated check points in sg_shell
<i>capture</i>	Captures output (stdout or stderr) of script to a file
<i>gui_set_preference</i>	Specifies SpyGlass preferences using Tcl Shell
<i>gui_add_menu</i>	Creates a new toolbar menu item and returns the Id
<i>show_error</i>	Displays the last error that occurred during a particular command invocation along with its trace
<i>source</i>	Evaluates a file or resource as a Tcl script
<i>unalias</i>	Removes an alias set for a group of word(s)

## alias

**Creates an alias for a group of word(s)**

### Syntax

```
alias <alias-name>  
    [<expansion>]
```

### Scope

Any

### Return Value

String, *<alias-name>*, if a new alias is being created. Otherwise, returns nothing.

### Description

The *alias* command creates an alias for a group of words, the first of which should be an existing command. This command is like the usual UNIX shell's *alias* command.

The *alias* command can be used in the following ways:

- To display the list of existing aliases  
Specify the *alias* command without any argument to display a list of existing aliases. This command returns nothing.
- To display the value of a particular alias  
Specify the command, *alias <alias-name>*, to display the value of the specified alias (*<alias-name>*). This command returns nothing.
- To create a new alias  
Specify the command, *alias <alias-name> [<expansion>]*, to create a new alias of the specified name (*<alias-name>*), which expands to the words as specified in the expansion (*<expansion>*).  
This command returns the string, *<alias-name>*.

### Examples

```
sg_shell> alias np { new_project start.prj -projectwdir ./
```

```
start }
new_project start.prj -projectwdir ./start
sg_shell> np

sg_shell> alias ho { help -option }
help -option
sg_shell> ho

sg_shell> alias
[ho] -> help -option
[np] -> new_project start.prj -projectwdir ./start
[quit] -> exit

sg_shell> alias np
[np] -> new_project start.prj -projectwdir ./start
```

## See Also

[\*unalias\*](#)

## benchmark

**Monitors run-time and memory usage between two designated check points in `sg_shell`**

### Syntax

```
benchmark
  [-start <check-point-name>] | [-show <data-object>]
  [-h]
```

### Scope

Any

### Return Value

- Returns a data object when the `-start` argument is specified
- Returns nothing when the `-show` argument is specified

### Description

The *benchmark* command monitors runtime and memory usage at various stages. The benchmark data shown is calculated with respect to some previous check points as marked by the `benchmark -start` command. Therefore, `sg_shell` first determines a check point by using the `benchmark -start point-1` command, and stores the result in some variable. Then, at a later time, whenever you wish to see the details such as memory usage and run time, with respect to the check-point "point-1", you can specify the command, `benchmark -show $data`.

The invocation `benchmark -start point-1` returns a list containing some data. The contents of this list are as follows:

```
{ malloc'ed-memory heap-memory user-time system-time
  cpu-time elapsed-time check-point-name }
```

The data in the above list denotes the absolute figures at the time of fetching this list. This list should be stored in some variable, say `check_point_start_data`. This variable should be used to see the incremental statistics at any later stage by using the command, `benchmark -show $check_point_start_data`.

## Arguments

This command has the following arguments:

**-h**

(Optional) Prints memory figures in human readable units, such as KB and MB.

**-show <data-object>**

(Optional) Displays the information for the specified data, <data-object>.

**-start <check-point-name>**

(Optional) Marks a start check point for benchmarking.

## Examples

```
sg_shell> set lnk_dsg_data [benchmark -start
link_design_start]
mmem 11008308 hmem 88879104 usr_tm 0.11 sys_tm 0.11 cpu_tm
0.22 elapsed_tm 1252041322.090611 name link_design_start
```

```
sg_shell> link_design
```

```
.
.
.
```

```
sg_shell> benchmark -show $lnk_dsg_data
Benchmark (memory/time related) information
```

```
-----
check-point: link_design_start
malloc'ed memory = 35426708 bytes ( 24418400 bytes )
heap size        = 118157312 bytes ( 29278208 bytes )
user time        = 2.480 sec ( 2.370 sec)
system time      = 1.060 sec ( 0.950 sec)
cpu time         = 3.540 sec ( 3.320 sec)
```

Note: figures in the parentheses show incremental data wrt the nearest check-point named 'link\_design\_start'

(Use '-h' for human readable units)

```
sg_shell> benchmark -show $lnk_dsg_data -h  
Benchmark (memory/time related) information
```

```
-----  
check-point: link_design_start  
malloc'ed memory = 33.79 MB ( 23.29 MB )  
heap size       = 112.68 MB ( 27.92 MB )  
user time       = 2.480 sec ( 2.370 sec)  
system time     = 1.060 sec ( 0.950 sec)  
cpu time        = 3.540 sec ( 3.320 sec)
```

Note: figures in the parentheses show incremental data wrt the nearest check-point named 'link\_design\_start'

## capture

**Captures output (stdout or stderr) of script to a file**

### Syntax

```
capture
  [-stdout] [-stderr] [-append]
  <output-file>
  <command-string>
```

### Scope

Any

### Return Value

None

### Description

The *capture* command redirects the output (stdout or stderr) of the specified command, *<command-string>*, to the specified output file, *<output-file>*. Here, *<command-string>* must be provided in curly brackets `{ }`.

If there is any error generated by a command, that error is also captured in the specified file depending on whether the error is flagged on stdout or stderr.

For example, consider that an error has been reported during the execution of the `write_report <rpt_name>` command as part of the following command:

```
sg_shell> capture my.rpt {write_report <rpt_name>}
```

In this case, the error is captured in the `my.rpt` file itself.

The *capture* command is useful to store the output of commands generating voluminous output, such as the [link\\_design](#), [run\\_goal](#), and [write\\_report](#) commands.

### Arguments

This command has the following arguments:



**-stdout**

(Optional) Redirects stdout to the specified output file.

**-stderr**

(Optional) Redirects stderr to the specified output file.

**-append**

(Optional) Appends the output to the specified output file. If the specified output file does not exist, `sg_shell` creates that output file.

**<output-file>**

Specifies the output file in which the output needs to be stored.

**<command-string>**

Specifies one or more command strings.

## Examples

```
sg_shell> capture -stdout output.log {link_design}
```

In the above example, only `stdout` (default) part of the output from `link_design` is captured and written into `output.log`. If `output.log` exists, the content of the file is overwritten. If `output.log` does not exist, it will be created. Nothing is returned.

```
sg_shell> capture -stderr output.log {link_design}
```

In the above example, only `stderr` part of the output from `link_design` is captured and written into `output.log`. If `output.log` exists, the content of the file is overwritten. If `output.log` does not exist, it will be created. Nothing is returned.

```
sg_shell> capture -stdout -stderr output.log {link_design}
```

In the above example, both `stdout` and `stderr` parts of the output from `link_design` is captured and written into `output.log`. If `output.log` exists, the content of the file is overwritten. If `output.log` does not exist, it will be created. Nothing is returned.

```
sg_shell> capture -append -stdout output.log {link_design}
```

In the above example, only `stdout` part of the output from `link_design` is

captured and written into `output.log`. If `output.log` exists, the content of the file is appended. If `output.log` does not exist, it will be created. Nothing is returned.

## gui\_set\_preference

Specifies SpyGlass preferences using Tcl Shell

### Syntax

```
gui_set_preference <preference-key> <preference-value>
```

### Scope

Any

### Return Value

None

### Description

The *gui\_set\_preference* command is used to set the SpyGlass preferences using Tcl Shell User Interface.

Sanity checks are used to check the compatibility of the provided value with the allowed set of values for the given preference. See the [Setting SpyGlass Preferences Using Tcl Shell Interface](#) section for more details about using this command.

### Arguments

This command has the following arguments:

#### <preference-key>

Specifies a string that is used to uniquely identify a particular preference.

#### <preference-value>

Specifies the value that should be set to the given preference.

### Examples

```
sg_shell> gui_set_preference AutoReloadProject true
```

In the above example, `AutoReloadProject` is a boolean type preference and can have any of the `true`, `false`, `yes`, `no`, `1`, and `0`

values.

## gui\_add\_menu

Creates a new toolbar menu item and returns the Id

### Syntax

```
gui_add_menu
  [-label label_name]
  [-icon icon_name]
  [-callback callback_function]
  [-in menu_id]
  [-args list_of_callback_arguments]
  [-window window_name]
  [-context context]
```

### Scope

Any

### Return Value

None

### Description

The *gui\_add\_menu* command is used to create a new toolbar menu item and return the Id.

### Arguments

This command has the following arguments:

**-args** <list\_of\_callback\_arguments>

Specify the arguments for the callback function.

**-callback** <callback\_function>

Specifies the callback function associates with the menu item.

**-context** <rmb|context>

Used to add menu item to (RMB) context menu

**-icon <icon\_name>**

Specifies an icon to the menu item.

**-in <menu\_id>**

Specifies the parent menu id.

**-label <label\_name>**

Specifies a label to the menu item.

**-window <window\_name(s)>**

Used to specify windows to add menu item.

## Examples

NA

## show\_error

Displays the last error that occurred during a particular command invocation along with its trace

### Syntax

```
show_error
```

### Scope

Any

### Return Value

None

### Description

The *show\_error* command displays the last error that occurred during a particular command execution along with its trace.

### Examples

```
sg_shell> this_is_an_erroneous_command
invalid command name "this_is_an_erroneous_command"
sg_shell> show_error
Detailed Error Trace
-----

invalid command name "this_is_an_erroneous_command"
  while executing
  "this_is_an_erroneous_command"
```

## source

Evaluates a file or resource as a Tcl script

### Syntax

```
source [-encoding <encoding_name>]
       [-continue_on_error]
       <file_name>
```

### Scope

Any

### Return Value

Return value of the last command executed in the script *<file\_name>*

### Description

The *source* command takes the contents of the specified file or resource and passes it to the Tcl interpreter as a text script. The return value of this command is the return value of the last command executed in the script. If an error occurs in evaluating the contents of the script, the *source* command returns that error. If a *return* command is invoked from within the script, the remainder of the file is skipped and the *source* command returns normally with the result from the *return* command.

The end-of-file character for files is "\32" (^Z) for all platforms. The *source* command reads files up to this character. This restriction does not exist for the *read* or *gets* commands, allowing for files containing code and data segments (scripted documents). If you require a "^Z" in code for string comparison, you can use "\032" or "\u001a", which will be safely substituted by the Tcl interpreter into "^Z".

### Arguments

This command has the following arguments:

#### **-continue\_on\_error**

Use this option to continue with the execution of the rest of the script even after encountering errors.



**NOTE:** *The `-continue_on_error` option, similar to the `-tcl_file_continue_on_error` option for startup files, should be used only when you require all the errors, such as ADC syntax errors, to be reported in one go. Refer to [Continue-on-error mechanism for the Tcl startup file](#) section for more details on continue-on-error mechanism.*

### **-encoding**

Use this option to specify encoding of the data stored in the script file `<file_name>`. When the `-encoding` option is omitted, the system encoding is assumed.

### **file\_name**

Use this option to specify the Tcl script file to be read.

## **Examples**

Consider a Tcl command file, `cmds.tcl`, as follows:

```
cmds.tcl
new_project -f test.prj
read_file test.v
## error in the below line
no_such_command -opt1 -opt2
puts "Compiling design ..."
compile_design
puts "Compilation finished"
run_design_query_proc [get_cells]
puts "Design query proc executed"
puts "Finished"
=====
```

If you now source the above file in `sg_shell`, compilation would stop at line number 4, and the `source` command exits and reports an error at line number 4.

If you want to skip line number 4 and continue even after encountering errors in the startup file, use the `-continue_on_error` option as follows:

```
sg_shell> source cmds.tcl -continue_on_error
```

## unalias

**Removes an alias set for a group of word(s)**

### Syntax

```
unalias <alias-name>
```

### Scope

Any

### Return Value

None

### Description

The *unalias* command is used to unalias a previously set alias for a group of words. This command is similar to the UNIX shell's *unalias* command.

### Examples

```
sg_shell> alias np { new_project start.prj -projectwdir ./  
start }  
new_project start.prj -projectwdir ./start  
sg_shell> unalias np
```

### See Also

[alias](#)

---

# Appendix A: Deprecated Command Names and Their Corresponding New Commands

---

The following table lists the old commands and their corresponding new commands in Console:

<b>Old Command</b>	<b>New Command</b>
set_option current_methodology	current_methodology
set_option stop_module	set_option stop
set_option define_macro	set_option define <macro>
set_option include_file_search_paths	set_option incdir <path>
set_option verilog_library_extensions	set_option libext <exts>
set_option verilog_library_files	set_option v <files>
set_option verilog_library_directories	set_option y <dirs>
set_option verilog_standard	set_option enableSV <yes   no> set_option disablev2k <yes   no>
set_option allow_duplicate_modules	set_option allow_module_override <yes   no>
set_option vhdl_standard	set_option 87 <yes   no>
set_option work_logical_name	set_option work <arg>

<b>Old Command</b>	<b>New Command</b>
set_option vhdl_sort_method	set_option sort <yes   no>
set_option hdl_parameter	set_option param <parameter>
set_option analyze_designware	set_option dw <yes   no>
set_option black_boxes	set_option inferblackbox <yes   no> set_option inferblackbox_rtl <yes   no> set_option nobb <yes   no>
set_option message_printing_limit	set_option lvpr <value>
set_option define_synthesis_pragma	set_option pragma <list-of-pragmas>
set_option enable_clock_gating	set_option sgsyn_clock_gating
set_option clock_gating_threshold	set_option sgsyn_clock_gating_threshold
set_option memory_size_limit	set_option mthresh <value>
set_option handle_large_memory	set_option handlememory <yes   no>
set_option enable_save_restore	set_option enable_save_restore
set_option precomp_lib_check	set_option hdlldu <yes   no>
set_option cache_directory	set_option cachedir <dir-name>
add_file	read_file
create_report	set_option report
create_report -file	set_option reportfile
create_report -maxsize	set_option report_max_size
create_report -style	set_option report_style
define_precomp_lib	set_option libhdlfiles <lib-name>
define_lib_map	set_option lib <lib-name>
define_goal_setup	current_goal

---

# Appendix B: Application Attributes

---

The following is a list of the applicaiton attributes:

- [List of Built-in Attributes](#)
- [List of Product Attributes](#)

## List of Built-in Attributes

The following is the list of built-in attributes defined in SpyGlass:

Attribute Name	Object	Type
<i>area</i>	lib_cell	float
<i>area</i>	du_cell	float
<i>area</i>	flat_cell	float
<i>always_on</i>	lib_cell	boolean
<i>always_on</i>	lib_pin	boolean
<i>base_name</i>	lib	string
<i>base_name</i>	lib_cell	string
<i>base_name</i>	lib_pin	string
<i>base_name</i>	design	string
<i>base_name</i>	du_cell	string
<i>base_name</i>	du_pin	string
<i>base_name</i>	du_port	string
<i>base_name</i>	du_net	string
<i>base_name</i>	flat_cell	string
<i>base_name</i>	flat_pin	string
<i>base_name</i>	flat_port	string
<i>base_name</i>	flat_net	string
<i>bus_width</i>	lib_pin	int
<i>bus_width</i>	du_pin	int
<i>bus_width</i>	du_port	int
<i>bus_width</i>	du_net	int
<i>bus_width</i>	flat_pin	int
<i>bus_width</i>	flat_port	int
<i>bus_width</i>	flat_net	int
<i>capacitance</i>	lib_pin	float
<i>capacitive_load_unit</i>	lib	string
<i>clk_name</i>	clock	string
<i>clk_net</i>	clock	collection

## List of Built-in Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>clock_gate_clock_pin</i>	lib_pin	boolean
<i>clock_gate_clock_pin</i>	du_pin	boolean
<i>clock_gate_clock_pin</i>	flat_pin	boolean
<i>clock_gate_enable_pin</i>	lib_pin	boolean
<i>clock_gate_enable_pin</i>	du_pin	boolean
<i>clock_gate_enable_pin</i>	flat_pin	boolean
<i>clock_gate_obs_pin</i>	lib_pin	boolean
<i>clock_gate_obs_pin</i>	du_pin	boolean
<i>clock_gate_obs_pin</i>	flat_pin	boolean
<i>clock_gate_out_pin</i>	lib_pin	boolean
<i>clock_gate_out_pin</i>	du_pin	boolean
<i>clock_gate_out_pin</i>	flat_pin	boolean
<i>clock_gate_test_pin</i>	lib_pin	boolean
<i>clock_gate_test_pin</i>	du_pin	boolean
<i>clock_gate_test_pin</i>	flat_pin	boolean
<i>clock_type</i>	clock	string
<i>clocks</i>	flat_cell	collection
<i>clocks</i>	flat_pin	collection
<i>clocks</i>	clock_domain	collection
<i>connection_class</i>	lib_pin	string
<i>current_design</i>	adc_node	string
<i>current_unit</i>	lib	string
<i>default_cell_leakage_power</i>	lib	float
<i>default_connection_class</i>	lib	string
<i>default_fanout_load</i>	lib	float
<i>default_inout_pin_cap</i>	lib	float
<i>default_input_pin_cap</i>	lib	float
<i>default_leakage_power_density</i>	lib	float
<i>default_max_capacitance</i>	lib	float
<i>default_max_fanout</i>	lib	float
<i>default_max_transition</i>	lib	float
<i>default_operating_conditions</i>	lib	string

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>default_output_pin_cap</i>	lib	float
<i>default_power_rail</i>	lib	string
<i>default_threshold_voltage_group</i>	lib	string
<i>default_wire_load</i>	lib	string
<i>default_wire_load_area</i>	lib	float
<i>default_wire_load_capacitance</i>	lib	float
<i>default_wire_load_mode</i>	lib	string
<i>default_wire_load_resistance</i>	lib	float
<i>default_wire_load_selection</i>	lib	string
<i>define_cell_area</i>	lib	string
<i>delay_model</i>	lib	string
<i>direction</i>	lib_pin	string
<i>direction</i>	du_pin	string
<i>direction</i>	du_port	string
<i>direction</i>	du_net	string
<i>direction</i>	flat_pin	string
<i>direction</i>	flat_port	string
<i>direction</i>	flat_net	string
<i>domain</i>	flat_cell	collection
<i>domain</i>	flat_pin	collection
<i>domain</i>	clock	collection
<i>domain_name</i>	clock_domain	string
<i>dont_touch</i>	lib_cell	boolean
<i>dont_touch</i>	du_cell	boolean
<i>dont_touch</i>	flat_cell	boolean
<i>dont_use</i>	lib_cell	boolean
<i>dont_use</i>	du_cell	boolean
<i>dont_use</i>	flat_cell	boolean
<i>driver_type</i>	lib_pin	string
<i>edgelist</i>	clock	string
<i>fanout_load</i>	lib_pin	float
<i>file_name</i>	lib	string



## List of Built-in Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>file_name</i>	lib_cell	string
<i>file_name</i>	design	string
<i>file_name</i>	du_cell	string
<i>file_name</i>	du_pin	string
<i>file_name</i>	du_port	string
<i>file_name</i>	du_net	string
<i>file_name</i>	flat_cell	string
<i>file_name</i>	flat_pin	string
<i>file_name</i>	flat_port	string
<i>file_name</i>	flat_net	string
<i>file_name</i>	adc_node	string
<i>file_name</i>	clock	string
<i>file_name</i>	message	string
<i>from_lib_pin</i>	lib_timing_arcs	collection
<i>full_name</i>	lib	string
<i>full_name</i>	lib_cell	string
<i>full_name</i>	lib_pin	string
<i>full_name</i>	design	string
<i>full_name</i>	du_cell	string
<i>full_name</i>	du_pin	string
<i>full_name</i>	du_port	string
<i>full_name</i>	du_net	string
<i>full_name</i>	flat_cell	string
<i>full_name</i>	flat_pin	string
<i>full_name</i>	flat_port	string
<i>full_name</i>	flat_net	string
<i>function</i>	lib_pin	string
<i>has_arglabels</i>	message	boolean
<i>index</i>	lib_pin	int
<i>index</i>	flat_pin	int
<i>index</i>	flat_port	int
<i>index</i>	flat_net	int

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>input_voltage_range</i>	lib_cell	string
<i>input_voltage_range</i>	lib_pin	string
<i>is_async_pin</i>	lib_pin	boolean
<i>is_async_pin</i>	du_pin	boolean
<i>is_async_pin</i>	flat_pin	boolean
<i>is_blackbox</i>	design	boolean
<i>is_blackbox</i>	du_cell	boolean
<i>is_blackbox</i>	flat_cell	boolean
<i>is_celldefine</i>	design	boolean
<i>is_celldefine</i>	du_cell	boolean
<i>is_celldefine</i>	flat_cell	boolean
<i>is_clear_pin</i>	lib_pin	boolean
<i>is_clear_pin</i>	du_pin	boolean
<i>is_clear_pin</i>	flat_pin	boolean
<i>is_clock_gating_cell</i>	lib_cell	boolean
<i>is_clock_gating_cell</i>	design	boolean
<i>is_clock_gating_cell</i>	du_cell	boolean
<i>is_clock_gating_cell</i>	flat_cell	boolean
<i>is_clock_pin</i>	lib_pin	boolean
<i>is_clock_pin</i>	du_pin	boolean
<i>is_clock_pin</i>	flat_pin	boolean
<i>is_combinational</i>	lib_cell	boolean
<i>is_combinational</i>	design	boolean
<i>is_combinational</i>	du_cell	boolean
<i>is_combinational</i>	flat_cell	boolean
<i>is_data_pin</i>	lib_pin	boolean
<i>is_data_pin</i>	du_pin	boolean
<i>is_data_pin</i>	flat_pin	boolean
<i>is_empty</i>	design	boolean
<i>is_empty</i>	du_cell	boolean
<i>is_empty</i>	flat_cell	boolean
<i>is_enable_pin</i>	lib_pin	boolean

## List of Built-in Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>is_enable_pin</i>	du_pin	boolean
<i>is_enable_pin</i>	flat_pin	boolean
<i>is_flop</i>	lib_cell	boolean
<i>is_flop</i>	design	boolean
<i>is_flop</i>	du_cell	boolean
<i>is_flop</i>	flat_cell	boolean
<i>is_generated</i>	du_net	boolean
<i>is_generated</i>	flat_net	boolean
<i>is_ground_pin</i>	lib_pin	boolean
<i>is_ground_pin</i>	du_pin	boolean
<i>is_ground_pin</i>	flat_pin	boolean
<i>is_hierarchical</i>	design	boolean
<i>is_hierarchical</i>	du_cell	boolean
<i>is_hierarchical</i>	flat_cell	boolean
<i>is_isolated</i>	lib_pin	boolean
<i>is_isolated</i>	du_pin	boolean
<i>is_isolated</i>	flat_pin	boolean
<i>is_isolation_cell</i>	lib_cell	boolean
<i>is_isolation_cell</i>	du_cell	boolean
<i>is_isolation_cell</i>	flat_cell	boolean
<i>is_latch</i>	lib_cell	boolean
<i>is_latch</i>	design	boolean
<i>is_latch</i>	du_cell	boolean
<i>is_latch</i>	flat_cell	boolean
<i>is_leaf</i>	design	boolean
<i>is_leaf</i>	du_cell	boolean
<i>is_leaf</i>	flat_cell	boolean
<i>is_level_shifter</i>	lib_cell	boolean
<i>is_level_shifter</i>	design	boolean
<i>is_level_shifter</i>	du_cell	boolean
<i>is_level_shifter</i>	flat_cell	boolean
<i>is_lib</i>	design	boolean

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>is_lib</i>	du_cell	boolean
<i>is_lib</i>	flat_cell	boolean
<i>is_load_pin</i>	lib_pin	boolean
<i>is_load_pin</i>	du_pin	boolean
<i>is_load_pin</i>	flat_pin	boolean
<i>is_macro</i>	design	boolean
<i>is_macro</i>	du_cell	boolean
<i>is_macro_cell</i>	lib_cell	boolean
<i>is_macro_cell</i>	du_cell	boolean
<i>is_macro_cell</i>	flat_cell	boolean
<i>is_memory_cell</i>	lib_cell	boolean
<i>is_memory_cell</i>	design	boolean
<i>is_memory_cell</i>	du_cell	boolean
<i>is_memory_cell</i>	flat_cell	boolean
<i>is_multidim</i>	du_net	boolean
<i>is_multidim</i>	flat_net	boolean
<i>is_multiple_driver</i>	flat_net	boolean
<i>is_mux</i>	lib_cell	boolean
<i>is_mux</i>	design	boolean
<i>is_mux</i>	du_cell	boolean
<i>is_mux</i>	flat_cell	boolean
<i>is_mux_select_pin</i>	lib_pin	boolean
<i>is_mux_select_pin</i>	du_pin	boolean
<i>is_mux_select_pin</i>	flat_pin	boolean
<i>is_pad</i>	lib_pin	boolean
<i>is_pad</i>	du_pin	boolean
<i>is_pad</i>	flat_pin	boolean
<i>is_pad_cell</i>	lib_cell	boolean
<i>is_pad_cell</i>	design	boolean
<i>is_pad_cell</i>	du_cell	boolean
<i>is_pad_cell</i>	flat_cell	boolean
<i>is_pg_pin</i>	lib_pin	boolean

## List of Built-in Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>is_pg_pin</i>	du_pin	boolean
<i>is_pg_pin</i>	flat_pin	boolean
<i>is_power_pin</i>	lib_pin	boolean
<i>is_power_pin</i>	du_pin	boolean
<i>is_power_pin</i>	flat_pin	boolean
<i>is_preset_pin</i>	lib_pin	boolean
<i>is_preset_pin</i>	du_pin	boolean
<i>is_preset_pin</i>	flat_pin	boolean
<i>is_primitive</i>	design	boolean
<i>is_primitive</i>	du_cell	boolean
<i>is_record</i>	du_net	boolean
<i>is_record</i>	flat_net	boolean
<i>is_sequential</i>	lib_cell	boolean
<i>is_sequential</i>	design	boolean
<i>is_sequential</i>	du_cell	boolean
<i>is_sequential</i>	flat_cell	boolean
<i>is_stop</i>	design	boolean
<i>is_stop</i>	du_cell	boolean
<i>is_stop</i>	flat_cell	boolean
<i>is_three_state</i>	lib_cell	boolean
<i>is_three_state</i>	design	boolean
<i>is_three_state</i>	du_cell	boolean
<i>is_three_state</i>	flat_cell	boolean
<i>is_three_state_enable_pin</i>	lib_pin	boolean
<i>is_three_state_enable_pin</i>	du_pin	boolean
<i>is_three_state_enable_pin</i>	flat_pin	boolean
<i>is_three_state_output_pin</i>	lib_pin	boolean
<i>is_three_state_output_pin</i>	du_pin	boolean
<i>is_three_state_output_pin</i>	flat_pin	boolean
<i>is_top</i>	design	boolean
<i>is_top</i>	du_cell	boolean
<i>is_top</i>	flat_cell	boolean

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>is_user_module</i>	design	boolean
<i>is_user_module</i>	du_cell	boolean
<i>is_waived</i>	message	boolean
<i>is_vector</i>	lib_pin	boolean
<i>is_vector</i>	du_pin	boolean
<i>is_vector</i>	du_port	boolean
<i>is_vector</i>	du_net	boolean
<i>is_vector</i>	flat_pin	boolean
<i>is_vector</i>	flat_port	boolean
<i>is_vector</i>	flat_net	boolean
<i>isolation_cell_data_pin</i>	lib_pin	boolean
<i>isolation_cell_data_pin</i>	du_pin	boolean
<i>isolation_cell_data_pin</i>	flat_pin	boolean
<i>isolation_cell_enable_pin</i>	lib_pin	boolean
<i>isolation_cell_enable_pin</i>	du_pin	boolean
<i>isolation_cell_enable_pin</i>	flat_pin	boolean
<i>language</i>	design	string
<i>leakage_power_unit</i>	lib	string
<i>level_shifter_data_pin</i>	lib_pin	boolean
<i>level_shifter_data_pin</i>	du_pin	boolean
<i>level_shifter_data_pin</i>	flat_pin	boolean
<i>level_shifter_enable_pin</i>	lib_pin	boolean
<i>level_shifter_enable_pin</i>	du_pin	boolean
<i>level_shifter_enable_pin</i>	flat_pin	boolean
<i>level_shifter_type</i>	lib_cell	string
<i>level_shifter_type</i>	design	string
<i>level_shifter_type</i>	du_cell	string
<i>level_shifter_type</i>	flat_cell	string
<i>line_num</i>	lib_cell	int
<i>line_num</i>	design	int
<i>line_num</i>	du_cell	int
<i>line_num</i>	du_pin	int

## List of Built-in Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>line_num</i>	du_port	int
<i>line_num</i>	du_net	int
<i>line_num</i>	flat_cell	int
<i>line_num</i>	flat_pin	int
<i>line_num</i>	flat_port	int
<i>line_num</i>	flat_net	int
<i>line_number</i>	message	int
<i>lsb</i>	lib_pin	int
<i>lsb</i>	du_pin	int
<i>lsb</i>	du_port	int
<i>lsb</i>	du_net	int
<i>lsb</i>	flat_pin	int
<i>lsb</i>	flat_port	int
<i>lsb</i>	flat_net	int
<i>master_name</i>	du_cell	string
<i>master_name</i>	flat_cell	string
<i>max_capacitance</i>	lib_pin	float
<i>max_fanout</i>	lib_pin	float
<i>max_transition</i>	lib_pin	float
<i>memory_read</i>	lib_pin	boolean
<i>memory_read</i>	du_pin	boolean
<i>memory_read</i>	flat_pin	boolean
<i>memory_write</i>	lib_pin	boolean
<i>memory_write</i>	du_pin	boolean
<i>memory_write</i>	flat_pin	boolean
<i>message_id</i>	message	string
<i>min_capacitance</i>	lib_pin	float
<i>min_fanout</i>	lib_pin	float
<i>min_transition</i>	lib_pin	float
<i>modified_sdc_name</i>	flat_cell	string
<i>modified_sdc_name</i>	flat_pin	string
<i>modified_sdc_name</i>	flat_port	string

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>modified_sdc_name</i>	flat_net	string
<i>msb</i>	lib_pin	int
<i>msb</i>	du_pin	int
<i>msb</i>	du_port	int
<i>msb</i>	du_net	int
<i>msb</i>	flat_pin	int
<i>msb</i>	flat_port	int
<i>msb</i>	flat_net	int
<i>msg</i>	message	string
<i>net_type</i>	flat_net	string
<i>nom_process</i>	lib	float
<i>nom_temperature</i>	lib	float
<i>nom_voltage</i>	lib	float
<i>non_standard_sdc_name</i>	flat_cell	string
<i>non_standard_sdc_name</i>	flat_pin	string
<i>non_standard_sdc_name</i>	flat_port	string
<i>non_standard_sdc_name</i>	flat_net	string
<i>number_of_pins</i>	lib_cell	int
<i>number_of_pins'</i>	design	int
<i>number_of_pins</i>	du_cell	int
<i>number_of_pins</i>	flat_cell	int
<i>object_class</i>	lib	string
<i>object_class</i>	lib_cell	string
<i>object_class</i>	lib_pin	string
<i>object_class</i>	lib_timing_arcs	string
<i>object_class</i>	design	string
<i>object_class</i>	du_cell	string
<i>object_class</i>	du_pin	string
<i>object_class</i>	du_port	string
<i>object_class</i>	du_net	string
<i>object_class</i>	flat_cell	string
<i>object_class</i>	flat_pin	string



## List of Built-in Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>object_class</i>	flat_port	string
<i>object_class</i>	flat_net	string
<i>object_class</i>	adc_node	string
<i>object_class</i>	sdc_node	string
<i>object_class</i>	message	string
<i>output_voltage_range</i>	lib_cell	string
<i>output_voltage_range</i>	lib_pin	string
<i>path_name</i>	flat_cell	string
<i>path_name</i>	flat_net	string
<i>period</i>	clock	float
<i>pg_function</i>	lib_pin	string
<i>pg_type</i>	lib_pin	string
<i>power_down_function</i>	lib_pin	string
<i>power_down_function</i>	du_pin	string
<i>power_down_function</i>	flat_pin	string
<i>probability</i>	flat_net	float
<i>pulling_resistance_unit</i>	lib	string
<i>related_ground_pin</i>	lib_pin	collection
<i>related_ground_pin</i>	du_pin	collection
<i>related_ground_pin</i>	flat_pin	collection
<i>related_power_pin</i>	lib_pin	collection
<i>related_power_pin</i>	du_pin	collection
<i>related_power_pin</i>	flat_pin	collection
<i>retention_cell</i>	lib_cell	string
<i>retention_cell</i>	du_cell	string
<i>retention_cell</i>	flat_cell	string
<i>retention_pin</i>	lib_pin	string
<i>rtl_name</i>	design	string
<i>severity_label</i>	message	string
<i>severity_class</i>	message	string
<i>sdf_cond</i>	lib_timing_arcs	string
<i>sglib_name</i>	lib	string

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>
<i>sglib_name</i>	lib_cell	string
<i>sglib_name</i>	lib_pin	string
<i>sglib_name</i>	lib_timing_arcs	string
<i>std_cell_main_rail</i>	lib_pin	boolean
<i>standard_sdc_name</i>	flat_cell	string
<i>standard_sdc_name</i>	flat_pin	string
<i>standard_sdc_name</i>	flat_port	string
<i>standard_sdc_name</i>	flat_net	string
<i>static_message</i>	message	string
<i>switch_cell_type</i>	lib_cell	string
<i>switch_cell_type</i>	design	string
<i>switch_cell_type</i>	du_cell	string
<i>switch_cell_type</i>	flat_cell	string
<i>switch_function</i>	lib_pin	string
<i>switch_function</i>	du_pin	string
<i>switch_function</i>	flat_pin	string
<i>switch_pin</i>	lib_pin	boolean
<i>switch_pin</i>	du_pin	boolean
<i>switch_pin</i>	flat_pin	boolean
<i>technology</i>	lib	string
<i>time_unit</i>	lib	string
<i>timing_sense</i>	lib_timing_arcs	string
<i>timing_type</i>	lib_timing_arcs	string
<i>to_lib_pin</i>	lib_timing_arcs	collection
<i>weight</i>	message	int
<i>voltage_unit</i>	lib	string
<i>voltage_value</i>	lib_pin	float
<i>x_function</i>	lib_pin	string

## List of Product Attributes

The following is the list of product attributes defined in SpyGlass:

Attribute Name	Object	Type	Product
<i>activity</i>	flat_net	float	SpyGlass Power family
<i>blackbox_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>blackbox_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>blackbox_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>atspeed_sim_value</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>capacitance_source</i>	flat_net	string	SpyGlass Power family
<i>capture_sim_value</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>cell_size_for_power</i>	flat_cell	float	SpyGlass Power family
<i>clock_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>clock_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>clock_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>clamp_value</i>	pwr_isolation_node	string	SpyGlass Power Verify
<i>control_port</i>	pwr_psw_node	string	SpyGlass Power Verify
<i>combinational_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>combinational_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>combinational_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>dest_type</i>	cdc_node	string	SpyGlass CDC
<i>failure_reason</i>	cdc_node	string	SpyGlass CDC
<i>fanout_capacitance</i>	flat_net	float	SpyGlass Power family

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Product</b>
<i>get_atspeed_clock_n_phase</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>get_capture_clock_n_phase</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>get_dft_functional_clock_n_phase</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>get_latch_atspeed_statuses</i>	flat_inst	string	SpyGlass DFT
<i>get_latch_capture_statuses</i>	flat_inst	string	SpyGlass DFT
<i>get_latch_shift_status</i>	flat_inst	string	SpyGlass DFT
<i>get_scan_status</i>	flat_inst	string	SpyGlass DFT
<i>get_shift_clock_n_phase</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>ground_supply</i>	pwr_intent_node	string	SpyGlass Power Verify
<i>input_supply_port</i>	pwr_psw_node	string	SpyGlass Power Verify
<i>input_supply_set</i>	pwr_level_shift_node	string	SpyGlass Power Verify
<i>internal_power</i>	flat_cell	float	SpyGlass Power family
<i>io_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>io_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>io_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>is_activity_annotated</i>	flat_net	boolean	SpyGlass Power family
<i>is_async_sync_reset</i>	flat_net	boolean	SpyGlass latch
<i>is_clock_gated</i>	flat_cell	boolean	SpyGlass Power family
<i>is_clock_used_as_nonclock</i>	flat_net	boolean	SpyGlass starc

## List of Product Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Product</b>
<i>is_clock_used_with_both_edges</i>	flat_net	boolean	SpyGlass lint
<i>is_constant_pin</i>	flat_pin	boolean	SpyGlass ERC
<i>is_comb_conv</i>	cdc_conv_node	boolean	SpyGlass CDC
<i>is_data</i>	cdc_node	boolean	SpyGlass CDC
<i>is_disabled_cell</i>	flat_cell	boolean	SpyGlass ERC
<i>is_graycoded</i>	cdc_conv_node	string	SpyGlass CDC
<i>is_internal_power_defined</i>	lib_cell	boolean	SpyGlass Power family
<i>is_instantiated</i>	flat_cell	boolean	SpyGlass Power family
<i>is_multiple_driver</i>	flat_net	boolean	SpyGlass lint
<i>is_nonconv_bus</i>	cdc_conv_node	boolean	SpyGlass CDC
<i>is_latch_clock_driven_on_both_edges</i>	flat_net	boolean	SpyGlass latch
<i>is_internally_generated_reset</i>	flat_net	boolean	SpyGlass openmore
<i>is_reset_used_as_nonreset</i>	flat_net	boolean	SpyGlass starc
<i>is_reset_used_with_both_polarity</i>	flat_net	boolean	SpyGlass lint
<i>is_scannable</i>	flat_cell	boolean	SpyGlass DFT
<i>is_seq_conv</i>	cdc_conv_node	boolean	SpyGlass CDC
<i>is_synchronized</i>	cdc_node	boolean	SpyGlass CDC
<i>is_unregistered_port</i>	du_port	boolean	SpyGlass moreLint
<i>is_user_defined</i>	cdc_conv_node	boolean	SpyGlass CDC
<i>isolation_ground_net</i>	pwr_isolation_node	string	SpyGlass Power Verify
<i>isolation_power_net</i>	pwr_isolation_node	string	SpyGlass Power Verify

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Product</b>
<i>isolation_sense</i>	pwr_isolation_node	string	SpyGlass Power Verify
<i>isolation_signal</i>	pwr_isolation_node	string	SpyGlass Power Verify
<i>leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>leakage_power_model</i>	lib_cell	string	SpyGlass Power family
<i>location</i>	pwr_isolation_node pwr_level_shift_node	string	SpyGlass Power Verify
<i>name</i>	pwr_isolation_node pwr_retention_node pwr_psw_node pwr_level_shift_node	string	SpyGlass Power Verify
<i>megacell_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>megacell_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>megacell_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>memory_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>memory_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>memory_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>net_capacitance</i>	flat_net	float	SpyGlass Power family
<i>net_frequency</i>	flat_net	float	SpyGlass Power family
<i>num_source_domains</i>	cdc_node cdc_conversion_node	int	SpyGlass CDC
<i>num_sources</i>	cdc_node cdc_conversion_node	int	SpyGlass CDC

## List of Product Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Product</b>
<i>obs_probability</i>	flat_port flat_pin flat_net	float	SpyGlass DFT
<i>one_cnt_probability</i>	flat_port flat_pin flat_net	float	SpyGlass DFT
<i>other_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>other_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>other_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>output_supply_port</i>	pwr_psw_n ode	string	SpyGlass Power Verify
<i>output_supply_set</i>	pwr_level_ shift_node	string	SpyGlass Power Verify
<i>pg_sim_value</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>power_domain</i>	pwr_intent_ _node	string	SpyGlass Power Verify
<i>power_supply</i>	pwr_intent_ _node	string	SpyGlass Power Verify
<i>power_type</i>	flat_cell	string	SpyGlass Power family
<i>probability</i>	flat_net	float	SpyGlass Power family
<i>rand_fault_cov_estimate</i>	design	float	SpyGlass DFT
<i>rand_test_cov_estimate</i>	design	float	SpyGlass DFT
<i>restore_signal</i>	pwr_retenti on_node	string	SpyGlass Power Verify
<i>retention_ground_supply</i>	pwr_retenti on_node	string	SpyGlass Power Verify
<i>retention_power_supply</i>	pwr_retenti on_node	string	SpyGlass Power Verify
<i>root_clock_for_power</i>	flat_cell	float	SpyGlass Power family
<i>rule</i>	pwr_level_ shift_node	string	SpyGlass Power Verify

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Product</b>
<i>sa0_det_probability</i>	flat_port flat_pin flat_net	float	SpyGlass DFT
<i>sa1_det_probability</i>	flat_port flat_pin flat_net	float	SpyGlass DFT
<i>sa0_fault_detectability</i>	flat_port flat_pin	string	SpyGlass DFT
<i>sa1_fault_detectability</i>	flat_port flat_pin	string	SpyGlass DFT
<i>save_signal</i>	pwr_retenti on_node	string	SpyGlass Power Verify
<i>sink</i>	pwr_isolati on_node	string	SpyGlass Power Verify
<i>sequential_internal_power</i>	flat_cell	float	SpyGlass Power family
<i>sequential_leakage_power</i>	flat_cell	float	SpyGlass Power family
<i>sequential_switching_power</i>	flat_cell	float	SpyGlass Power family
<i>sdc_type</i>	sdc_node	string	SpyGlass Constraints
<i>shift_sim_value</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>source</i>	pwr_isolati on_node pwr_level_ shift_node	string	SpyGlass Power Verify
<i>src_type</i>	cdc_node	string	SpyGlass CDC
<i>static_controllability</i>	flat_pin flat_port flat_net	string	SpyGlass DFT
<i>static_observability</i>	flat_pin flat_port	string	SpyGlass DFT
<i>supply_name</i>	pwr_supply _node	string	SpyGlass Power Verify



## List of Product Attributes

<b>Attribute Name</b>	<b>Object</b>	<b>Type</b>	<b>Product</b>
<i>switching_power</i>	flat_cell	float	SpyGlass Power family
<i>sync_method</i>	cdc_node	string	SpyGlass CDC
<i>timing_state</i>	flat_cell flat_pin flat_port flat_net	string	SpyGlass Constraints
<i>t01_fault_detectability_ios</i>	flat_port flat_pin	string	SpyGlass DFT
<i>t10_fault_detectability_ios</i>	flat_port flat_pin	string	SpyGlass DFT
<i>t01_fault_detectability_oc</i>	flat_port flat_pin	string	SpyGlass DFT
<i>t10_fault_detectability_oc</i>	flat_port flat_pin	string	SpyGlass DFT
<i>type</i>	pwr_supply_node	string	SpyGlass Power Verify
<i>virtual_buffer_info</i>	flat_net	string	SpyGlass Power family
<i>virtual_internal_power</i>	flat_net	float	SpyGlass Power family
<i>virtual_leakage_power</i>	flat_net	float	SpyGlass Power family
<i>virtual_switching_power</i>	flat_net	float	SpyGlass Power family
<i>voltage_range_max</i>	pwr_intent_node	float	SpyGlass Power Verify
<i>voltage_range_min</i>	pwr_intent_node	float	SpyGlass Power Verify
<i>vt_classification</i>	lib_cell	string	SpyGlass Power family
<i>zero_cnt_probability</i>	flat_port flat_pin flat_net	float	SpyGlass DFT



---

# Appendix C: SpyGlass Report Names

---

You can specify a report name to the `<report-names>` argument of the following commands:

- `set_option report <report-name>`  
Refer to the *SpyGlass Console Reference Guide* for details.
- [write\\_report <report-name>](#)
- [define\\_report <report-name>](#)

This Appendix provides links to the sections containing valid report names of each product that can be used with the `<report-names>` argument.

Use the following links to refer to the valid reports names for required product:

- Generic Reports
  - [General Reports](#)
  - [Custom Reports](#)
  - [Default Reports](#)
- Product-specific Reports
  - [SpyGlass area Reports](#)
  - [SpyGlass audits Reports](#)

- SpyGlass lint Reports*
- SpyGlass morelint Reports*
- SpyGlass OpenMore Reports*
- SpyGlass STARC Reports*
- SpyGlass STARC02 Reports*
- SpyGlass STARC05 Reports*
- SpyGlass CDC Reports*
- SpyGlass Constraints Reports*
- SpyGlass DFT Reports*
- SpyGlass DFT DSM Reports*
- SpyGlass Power Family Reports*
- SpyGlass Power Verify Reports*
- SpyGlass TXV Reports*

## General Reports

The following general reports are generated by SpyGlass.

Report Name	Description
count	This report lists the number of times SpyGlass found each type of message.
inline	This report displays the contents of the RTL source file annotated with violation messages.
sign_off	This report lists summary and detailed information about the SpyGlass analysis run.
moresimple_filesort	This report is similar to the moresimple report except that the rule messages are sorted in the following order: <ul style="list-style-type: none"> <li>• File name for a given file</li> <li>• Severity</li> <li>• Weight</li> <li>• Rule</li> <li>• Line number</li> </ul>
moresimple_rulesort	This report is similar to the moresimple report except that the rule messages are sorted by the rule name first and for a given rule, by their file and line
moresimple_sevclass	This report is similar to the moresimple report with additional information displaying the severity class.
score	This report provides a built-in scoring system for code checks.
ignosimple	This report truncates long names and messages to fit the contents in the reports layout.
summary	This report displays a summary list of message counts by each particular rule type along with the severity class and rule short help.

## Custom Reports

The following custom reports are generated by SpyGlass.

<b>Report Name</b>	<b>Description</b>
count_sevsort	This report displays the severity label name, the rule name, and the rule message count for each rule.
moresimple_csv	The moresimple_csv report has the same details as the moresimple report but in a comma-separated format without the header/footer lines. The moresimple_csv Report.
score_detail	This report is the enhanced version of the score_report.

## Default Reports

The following default reports are generated by SpyGlass.

Report Name	Description
ignore_summary	This report lists design units/files that are ignored in the current run.
moresimple	The moresimple report is similar to the simple report. However, it does not truncate long names and messages.
no_msg_reporting_rules	This report displays a list of rules that did not report any violation or waived during SpyGlass run.
stop_summary	This report lists design units and files that are skipped from SpyGlass checking in the current run.
stop_summary	This report lists design units and files that are skipped from SpyGlass checking in the current run.
elab_summary	This report shows the following design information: <ul style="list-style-type: none"><li>• RTL design unit names</li><li>• Elaborated names</li><li>• Parameter values</li></ul>
waiver	This report generates information about waived messages during SpyGlass run.

## SpyGlass area Reports

The following reports are generated by the SpyGlass area product:

Report Name	Description
diff	This report compares the results of two SpyGlass Analysis runs (one before modifications and the second after modifications) to generate area-related differences in the two runs.
GateCountReport	This report contains module-wise gate count information. In addition, this report contains the information, such as area factor, prototype NAND gate name, prototype NAND gate area, gate count, and instance count of the complete design.



## SpyGlass audits Reports

The following reports are generated by the SpyGlass audits product:

<b>Report Name</b>	<b>Description</b>
Audit-RTL	This report is generated when the <code>rtl_audit</code> goal is run and provides information about the RTL of the design.
Audit-Structure	This report is generated when the <code>structure_audit</code> goal is run and provides information about the structural data of the design.

## SpyGlass lint Reports

The following reports are generated by the SpyGlass lint product:

<b>Report Name</b>	<b>Description</b>
SignalUsageReport	This report contains the details of the violating bits of multi-dimensional arrays, memory, and vector signals.
W448_Report	This report contains the details of the reset nets used synchronously and asynchronously.

## SpyGlass morelint Reports

The following report is generated by the SpyGlass morelint product:

<b>Report Name</b>	<b>Description</b>
ReportPortInfo	The report displays all top-level modules/entities port first followed by the black box instances.

## SpyGlass OpenMore Reports

The following report is generated by the SpyGlass OpenMore product:

<b>Report Name</b>	<b>Description</b>
CombLoopReport	This report contain the details of all the combinational loops detected by the CombLoop rule.

## SpyGlass STARC Reports

The following reports are generated by the SpyGlass STARC product:

<b>Report Name</b>	<b>Description</b>
STARC-1213	This report contains the details of all the combinational loops detected by the STARC-1.2.1.3 rule.
STARC_1316	This report shows all the synchronous and asynchronous usage of the reset net.

## SpyGlass STARC02 Reports

The following reports are generated by the SpyGlass STARC02 product:

<b>Report Name</b>	<b>Description</b>
STARC_1316	This report shows all the synchronous and asynchronous usage of the reset net.
STARC02_2414	This report contains the details of all the combinational loops detected by the <i>STARC02-2.4.1.4</i> rule.

## SpyGlass STARC05 Reports

The following report is generated by the SpyGlass STARC05 product:

<b>Report Name</b>	<b>Description</b>
STARC_1316	This report shows all the synchronous and asynchronous usage of the reset net.

## SpyGlass CDC Reports

The following reports are generated by the SpyGlass CDC product:

Report Name	Description
Clock-Reset-Summary	Provides information on clocks, resets, unconstrained clock nets, and clock-domain crossings in a design.
Clock-Reset-Detail	Provides information on the following: <ul style="list-style-type: none"> <li>• Synchronized and unsynchronized clock domain crossings</li> <li>• Crossings filtered by using the <code>cdc_false_path</code> constraint</li> <li>• Flip-flops that have their data pin tied to a constant</li> <li>• Synchronization techniques</li> </ul>
CKTree	Shows clock hierarchy in a tree-like format (also called clock tree).
CKCondensedTree	Shows a condensed clock tree in which unlike the CKTree report, this report shows the number of leaves instead of actually listing the leaves.
RSTree	Shows set/reset trees in a design.
PortClockMatrix	Provides information on constraints coverage of ports.
SynchInfo	Provides information on destinations and synchronizers involved in different synchronization schemes.
CrossingInfo	Provides information on source and destination flip-flops for all synchronized and unsynchronized crossings.
CKPathInfo	Provides information on clock cells on clock paths.
CKSGDCInfo	Provides information on user-specified constraints.
CKSync01	Provides information on unsynchronized crossings in a design
CDC-report	Provides summary on design, design setup, and verification results.
adv_cdc	Provides information that helps you to analyze the cause of a bug and helps you to gather functional analysis statistics.
adv_reg	Provides information on clocks, resets, and registers in a design.



## SpyGlass CDC Reports

<b>Report Name</b>	<b>Description</b>
NoClockCell-Summary	Provides information on the objects specified by the noclockcell_start, noclockcell_stop_instance, noclockcell_stop_module, and noclockcell_stop_signal constraints.
DeltaDelay-Concise	Shows a list of delta delay values for each clock in a design and number of flip-flops and latches for each delta delay value.
DeltaDelay-Detailed	Shows a list of delta delays for each clock, net names of flip-flops for each delay value, and net names of latches for each delay value.
DeltaDelay02-Detailed	Shows a list of flip-flops that can cause simulation problems due to delta delay issues.
DeltaDelay-Summary	Provides information on objects specified by the deltacheck_start, deltacheck_stop_instance, deltacheck_stop_module, deltacheck_stop_signal/ constraints.
Ac_sync_group_detail	Shows details of violations reported by the Ac_sync02, Ac_sync01, Ac_unsync02, and Ac_unsync01 rules.
Ac_sync_qualifier	Shows all the control-crossing synchronizers with their qualifiers usage status in synchronizing data crossings.
Glitch_detailed	Shows a summary of all the sources that are crossing destinations and contain glitch-related issues.

## SpyGlass Constraints Reports

The following reports are generated by the SpyGlass Constraints product:

Report Name	Description
tc_report_disable_timing	Lists disabled timing points set by set_disable_timing.
tc_unparsed_commands	Lists unparsed and unpopulated commands.
tc_block11_info	Provides a list of registered input/output/inout ports
False_Path01	Provides the unconnected points specified in set_false_pat.
MCP01	Provides the unconnected points specified in -from/-through/-to of set_multicycle_path.
TE_Consis01	Provides the unconnected points specified in -from/-through/-to of set_max_delay.
TE_Consis02	Provides the unconnected points specified in -from/-through/-to of set_min_delay.
Clock-Mapping	Provides information about the matched and ambiguous clocks specified in reference and implement
EQUIV_SDC_NON_EQUIVALENT_DESIGN_REPORT	Provides information about the sequential elements (registers) and ports for which the mapping information is not provided in the mapping file
tc_dont_touch_info	Lists modules with the set_dont_touch constraint
tc_dont_use_info	Lists cells with the set_dont_use or dont_use attribute set
clk_domain_false_path	Determine CDC that have different clocks.
sync_clock_uncertainty	Identify domain crossing synchronous clocks that do not have set_clock_uncertainty.
tc_clk_gen01a_info	Identify sequential cells that have clock pins driven by the reported clock net
tc_clk_gen01b_info	Identify all sequential cells that have clock pins driven by the reported constant or hanging clock net
tc_clock_info	View all generated clocks and source clocks
tc_latency_info	View the sum of source and network latencies of clocks

---

**SpyGlass Constraints Reports**

---

SDC_Methodology31	Identify clock paths comprising the feedback path
mcp_domain_<integer>	Determine the domain relationships between clocks used by set_multicycle_path.

---

## SpyGlass DFT Reports

The following reports are generated by the SpyGlass DFT product:

Report Name	Description
dft_ff_edge	Lists all the flip-flops that are triggered by positive and negative edges of a clock
dft_ff_set_reset_active	Lists all flip-flops and the combination that makes the Async pins of these flip-flops simultaneously active
dft_ff_set_reset_sequential_in_capture	Lists all flip-flops whose set/reset pins are driven by sequential elements/black box in capture mode
dft_ff_set_reset_sequential_in_shift	Lists all flip-flops whose set/reset pins are driven by sequential elements/black box in shift mode
dft_ff_X_source_for_tristate_enable	Lists all the flip-flops that could cause X on Tri- Enable's
dft_latch_enable	Contains the details regarding info latch mapping
dft_mandatory_sgdc	Lists all the missing mandatory constraints needed by a dft rule
dft_no_fault_instances	Lists all blocks for which all faults, internal and boundary pins, are treated as no fault
dft_optional_sgdc	Lists all the missing optional constraints needed by a SpyGlass DFT rule.
dft_summary	Lists a summary of autofix performed by the SpyGlass DFT product
dft_tristate	Lists all the tristate buses in the circuit
dft_initialized_ffs	list all the initialized flip-flops after an initialization sequence is applied
potentially_detected_faults	Lists all the potentially detectable faults
scan_chain	Lists all flip-flops that are or are not a part of some scan chains
scan_wrap	Contains a scanwrap list of all black boxes and their related information

## SpyGlass DFT Reports

---

soc_06_rpt	Lists the hierarchical name of all the instances appended with a pin name
stil_file	Lists all STIL constructs
stuck_at_faults	Lists all stuck-at faults
stuck_at_coverage	Contains the summary and details of stuck at fault numbers
stuck_at_coverage_audit	Contains the coverage audit summary
test_points_selected_2	Lists suggested test points
undetected_faults	Lists undetected faults and their causes

---

## SpyGlass DFT DSM Reports

The following reports are generated by the SpyGlass DFT DSM product:

Report Name	Description
atspeed_03_rpt	Lists all the cross-domain paths between flip-flops that are asynchronous in functional domain and synchronous in the at-speed testmode domain.
atspeed_04_rpt	Lists all the cross-domain paths between flip-flops that are synchronous in functional domain and asynchronous in the at-speed testmode domain.
atspeed_05_rpt	Lists all maximum and minimum number of logic levels for paths that are true 'testmode false' path found in the at-speed testmode
atspeed_07_rpt	Lists all the at-speed test clocks that are gated by the same set of logic.
atspeed_08_rpt	Lists all the at-speed clocks that need to be balanced and that have different gates along the clock paths.
atspeed_clock_synchronization	Lists an ordered list of domain pairs.
dft_dsm_clock_frequency	Lists frequency assignment data of a design, as given in the constraint files.
dft_dsm_clock_gating_cell	Lists information about clock gating cells (CGCs) in a design, name of the master module, type, test-enable pin clock source (shift mode), number of flip-flops connected, and CG rule violations (if any).
dft_dsm_constr-err-file	Lists all the violations of the dftDsmConstraintCheck_01 rule.
dft_dsm_enabled_flipflops	Lists information, such as flip-flop count, domain name, clock name, and flip-flop name on a per clock domain basis.
dft_dsm_ip_report	Lists information on values, such as for mode, for control, for observe, and for clock on per instance basis.

## SpyGlass DFT DSM Reports

<b>Report Name</b>	<b>Description</b>
ff_clock_reconvergence	Lists all the re-convergent combinational paths.
random_pattern_coverage	Lists the coverage estimate associated with a module.
random_pattern_faults	Lists information related to fault detection, controllability, and observability probabilities at ports and terminals.
transition_coverage	Lists the summary and details of transition fault numbers.
transition_coverage_audit	Lists the predicted improvement in the transition coverage number after each major step.
transition_coverage_clockdomain	Lists the number of different types of faults in a clock domain with the fault coverage and test coverage within the domain, and shows a summary below the detailed information.
transition_faults	Lists the pin-based fault data for transition faults.

## SpyGlass Power Family Reports

The following reports are generated by SpyGlass Power Family:

Report Name	Description
pe_activity	Contains activity information.
pe_adv_reduction	Shows clock gating statistics. In addition, contains information about the power saved by new clock gating candidates.
pe_audit	Contains detailed information about the rules in the SpyGlass Power Estimate and SpyGlass Power Reduce solutions.
pe_custom_wireload_debug	Lists the fan-outs and the corresponding number of nets used for generating a wireload table.
pe_cycle_power	Specifies the power consumption for each clock cycle in the design.
pe_debug_info	Contains debug information of nets that are not set from the simulation file.
pe_design_stats	Shows the details of the technology cells used to synthesize the design. For a gate level design, the statistics of the cells in the gate-level design are shown here.
pe_enable_scorecard	Contains the module-wise and instance-wise summaries of existing clock enables and new gating opportunities in the design.
pe_non_gated_flop_scorecard	Contains a summary of existing non-gated flip-flops.
pe_power_savings_non_gated_flop_scorecard	Contains a summary of existing non-gated flip-flops and estimates their power savings.
pe_summary	Describes the various aspects of power consumption of the design.
pe_reduction	Shows clock-gating statistics and information about the power saving.
pe_cell_allocation	Provides the activity details for the complete design reported per gating cell.



## SpyGlass Power Family Reports

<b>Report Name</b>	<b>Description</b>
pe_cell_sizing_info	Contains sizing information of the libraries used in the SpyGlass run. You can gain clarity on the library interpretation of SpyGlass and you can check the size of a library cell. In addition, view the report to verify the calculated cell size.
pe_spef_info	Contains information of multiple and missing capacitance data available from spef files.
pe_tvlg_info	Contains multi-voltage leakage estimates.
pe_wireload	Contains information about wireload used for calculating the capacitance of nets present in the design. It also reports the detailed net capacitance for each net in the design.
rme_summary	Lists AutoFix information.
pe_clock_gating_summary	Contains information on clock-gating count based on the clock domains in a design.
sec_status	Contains the information on status of assertion and type of equivalence of the equivalent points formally checked by SEC for equivalence.

## SpyGlass Power Verify Reports

The following reports are generated by the SpyGlass Power Verify product:

Report Name	Description
lp_assertion_info	This report contains information on the OVL assertions.
lp_autofix_info	This report contains information on auto insertion of level shifters.
lp_constr_info	This report contains information on voltage domain specific design information provided using SGDC/UPF/CPF files.
lp_cons_req	This report lists missing constraints that are required by the rules.
lp_crossing_data	This report contains domain crossing data information as inferred by SpyGlass using the SGDC/CPF/UPF files and library files.
lp_domain_info	This report contains domain related information for the domain crossings.
lp_lib_data	This report contains information from the library files as interpreted by the SpyGlass Power Verify solution.
lp_multivt_perblock	This report contains the instantiation information about VT type library cells in each hierarchy.
lp_multivtreport	This report contains the instantiation information about VT type library cells in the complete design.
lp_power_data	This report contains information about user-specified CPF/UPF commands parsed by SpyGlass.
lp_power_state	This report shows the power state information specified in the power intent SGDC/ CPF/UPF files.
lp_psw_info	This report contains the list of valid power switch strategies defined in the UPF file.
lp_ret_info	This report contains the list of valid retention strategies defined in the UPF file.
lp_retention_cell_list	This report lists the retention instances found in the design.
lp_supply_connection	This report displays the instance path name, followed by the power pin and corresponding supply net connection.
lp_vd_info	This report provides level-shift, isolation and power-switch information for domains in the design.

## SpyGlass Power Verify Reports

<b>Report Name</b>	<b>Description</b>
lp_wild_card	This report contains the matches found for the cell names specified in the SGDC constraints with wildcards (* or ?).
LP-report	This report is generated when you specify the <code>set_option report LP-report</code> command in the project file.
lp_strategy_info	This report contains the list of valid isolation and/or level shifter strategies defined in the UPF file. In addition, this report contains the expanded list of elements to which that strategy is applied.
lp_multi_supply_instance	This report lists the multi-supply instances that do not have an associated <code>connect_supply_net</code> command.
lp_special_pin_connection	The report shows the primary port connection to special pins.

## SpyGlass TXV Reports

The following reports are generated by the SpyGlass TXV product:

Report Name	Description
txv.rpt	Consists of information related to: <ul style="list-style-type: none"><li>• Run, Options, and Design Audit</li><li>• Setup Audit Information</li><li>• Analysis and Verification</li></ul>
txv_compactpath_file	Displays information about the paths before path compaction and the paths-after-path compaction.
txv_path_summary_report.rpt	Contains the path summary of the false path and multi-cycle path constraints analyzed by the SpyGlass TXV solution.
txv_uninitialized_reg_file	Contains the names of the associated uninitialized register names for each functionally verified sequentially failed constraint.

---

# Appendix D: Preference Variables Supported by the `set_pref` Command

---

## Overview

The following preference variables are supported by the `set_pref` command:

- `sh_command_log_file`
- `goal_show_hidden`
- `goal_enforce_prerequisite`
- `dq_design_view_type`
- `collection_display_limit`

## `sh_command_log_file`

Specifies the path name for `sg_shell`'s command log file

### Syntax

```
set_pref sh_command_log_file "./mycommands.log"
```

### Scope

Any

### Return Value

None

### Description

The `sh_command_log_file` preference variable is used to set the path name of the command log file. The default value for this preference variable is `./sg_shell_command.log`. This means that if `sg_shell` is being run in interactive mode, then all the commands being entered on `sg_shell`'s prompt will be logged into a file called `sg_shell_command.log` in the current directory. If this log file already exists, then it is truncated.

To preserve the commands log for each run of `sg_shell`, you should specify a unique name for the commands log file for each interactive run of `sg_shell` by inserting the following command in your `.sg_shell.startup` file (either `$HOME/.sg_shell.startup` or `$CWD/.sg_shell.startup` or through `"-tcl $MYPATH/startup.tcl"`):

```
set_pref sh_command_log_file "./sg_shell_command_[clock seconds].log"
```

You can set the `sh_command_log_file` variable at any stage of `sg_shell`'s interactive run and its effect will be visible immediately after being set. The commands entered before setting this variable (including the `set_pref` command to set the new log file) are logged into the previously set command log file. All the commands entered after setting the new log file will be logged in the new log file.

## Examples

The following example shows how to display the variable/value pairs.

You start the `sg_shell` as:

```
unix_shell> sg_shell
```

```
sg_shell banner ....
```

```
sg_shell> help
```

```
...
```

```
...
```

```
sg_shell> open_project xyz.prj
```

```
...
```

```
sg_shell> run_goal
```

```
...
```

```
sg_shell> set_pref sh_command_log_file "new_log_file.log"
```

```
set_pref: info: command log file is now at `new_log_file.log'
```

```
(Previously, it was at `./sg_shell_command.log')
```

```
sg_shell> close_project
```

```
...
```

```
sg_shell> exit
```

Considering the sequence above, the contents of the default log file, `./sg_shell_command.log`, are as follows:

```
-----  
help  
open_project xyz
```

```
run_goal
set_pref sh_command_log_file "new_log_file.log"
```

-----

The contents of the new log file, new\_log\_file.log, are as follows:

```
-----
close_project
exit
-----
```

If you specify a path name for sh\_command\_log\_file, then ensure that intermediate directories in the path name already exist. If they do not exist, then an error message will appear and this variable specification will be ignored. Refer to the following example:

```
sg_shell> pwd
/u/sam/prj
sg_shell> exec ls ./logs/
other_cmds.log
sg_shell_command.log
sg_shell> set_pref sh_command_log_file "/u/sam/prj/logs/
today/sg_shell_command_[clock seconds].log"
set_pref: error: could not open `/u/sam/prj/logs/today/
sg_shell_command_1255417358.log' for writing
(Please re-check the value of `sh_command_log_file' variable)

sg_shell> get_pref
sh_command_log_file ./sg_shell_command.log
```



`sg_shell>`

## See Also

[set\\_pref](#), [get\\_pref](#), [help](#)

## goal\_show\_hidden

**Enables visibility of hidden goals of methodology**

### Syntax

```
set_pref goal_show_hidden yes
```

### Scope

Any

### Return Value

None

### Description

The *goal\_show\_hidden* preference variable makes the hidden goals of methodology visible. After enabling this preference variable, the hidden goals are shown like normal goals and get visible in the goal summary report. These goals can also be visible in the available goal list by using the [help -goal](#) command.

Default value of this preference variable is false.

### Examples

```
sg_shell> current_goal initial_rtl/audit -alltop
ERROR [158]      'initial_rtl/audit-mixed.spq' : Unable to
locate goal file in following search paths -
$(SPYGLASS_HOME)/GuideWare/New_RTL

Trying to find best-matches for given goal specification ...
Following best-matches found for this goal specification:
initial_rtl/audit/block_profile
ERROR [38]      Failed loading goal as one or more goals
specified could not be found or do(es) not have read
permissions
current_goal: error: loading of goal 'initial_rtl/audit'
failed
sg_shell> set_pref goal_show_hidden true
sg_shell> current_goal initial_rtl/audit -alltop
```

```
ERROR [158]      'initial_rtl/audit-mixed.spq' : Unable to
locate goal file in following search paths -
$(SPYGLASS_HOME)/GuideWare/New_RTL
Trying to find best-matches for given goal specification ...
Following best-matches found for this goal specification:
initial_rtl/audit/block_profile
initial_rtl/audit/rtl_audit          ## hidden goal
initial_rtl/audit/structure_audit   ## hidden goal
initial_rtl/audit/datasheet_io_audit ## hidden goal
```

## See Also

[\*set\\_pref\*](#), [\*get\\_pref\*](#), [\*help\*](#)

## goal\_enforce\_prerequisite

**Enforces prerequisite goal run before goal run**

### Syntax

```
set_pref goal_enforce_prerequisite yes
```

### Scope

Any

### Return Value

None

### Description

The *goal\_enforce\_prerequisite* preference variable enforces the running of prerequisite goals before running some goals. If you try running a goal without first running its prerequisite goals, *sg\_shell* reports an error and lists the prerequisite goals that need to be run before proceeding with this goal run.

Default value of this preference variable is `false`.

### Examples

```
sg_shell> set_pref goal_enforce_prerequisite yes
sg_shell> current_goal detailed_rtl/constraint_generation/
gen_sdc -alltop
...
#assuming 'detailed_rtl/constraint/sdc_quick_check' is pre-
requisite goal of this goal and was not run yet
...
sg_shell> run_goal
run_goal: error: pre-requisite goal for detailed_rtl/
constraint_generation/gen_sdc was not run
please run detailed_rtl/constraint/sdc_quick_check goal
before running this goal
```

Overview

## See Also

[set\\_pref](#), [get\\_pref](#), [help](#)

## dq\_design\_view\_type

Specifies the view on which design query commands are executed

### Syntax

```
set_pref dq_design_view_type du
```

### Scope

Any

### Return Value

None

### Description

The *dq\_design\_view\_type* preference variable specifies the view on which design query commands work. The values that can be specified are `du`, `flat`, or `hier_flat`. The default value of this preference variable is `flat`.

The preference variable impacts the *Netlist Commands* as follows:

Value of the <i>dq_design_view_type</i> preference variable	Impact on the <i>Netlist Commands</i>
<code>du</code>	These commands return a collection of cells, nets, pins, and ports of the current design unit (du). You can dive into hierarchies by using the <i>current_design</i> command.
<code>flat</code>	These commands return a collection of flattened leaf objects. While working with this view, set the <i>current_design</i> command to <code>top</code> .
<code>hier_flat</code>	These commands return a collection of flattened objects, which may be leaf or hierarchical. You can dive into hierarchies by using the <i>current_instance</i> command. The <code>hier_flat</code> view is only effective if you compile your design or run a goal by using the <i>enable_hier_flattening</i> option.

## Examples

```

sg_shell> current_design
{"SISO"}
sg_shell> set_pref dq_design_view_type du
du
sg_shell> get_cells
{"icgc_inst", "comb_cloud_inst_1", "sf_inst_1",
"comb_cloud_inst_2", "sf_inst_2"}
sg_shell> current_design [get_master sf_inst_1]
{"SCAN_FLOP"}
sg_shell> get_cells
{"mux_inst_sf", "flop_inst_sf", "bbox_inst"}
sg_shell> set_pref dq_design_view_type flat
flat
sg_shell> get_cells
get_cells: error: current_design is set as non-top. Please
set current_design to a top module for executing design query
commands on flattened view
sg_shell> current_design SISO
{"SISO"}
sg_shell> get_cells
{"SISO.sf_inst_1.flop_inst_sf",
"SISO.sf_inst_2.flop_inst_sf",
"SISO.icgc_inst.latch_inst_icgc",
"SISO.sf_inst_1.bbox_inst", "SISO.sf_inst_2.bbox_inst",
"SISO.sf_inst_1.mux_inst_sf.leaf_mux_inst",
"SISO.sf_inst_2.mux_inst_sf.leaf_mux_inst",
"SISO.icgc_inst.and_inst_icgc", "SISO.comb_cloud_inst_1",
"SISO.comb_cloud_inst_2"}
sg_shell> set_pref dq_design_view_type hier_flat
hier_flat
sg_shell> get_cells
{"SISO.comb_cloud_inst_2", "SISO.comb_cloud_inst_1",
"SISO.icgc_inst", "SISO.sf_inst_2", "SISO.sf_inst_1"}
sg_shell> current_instance SISO.sf_inst_1
{"SISO.sf_inst_1"}
sg_shell> get_cells

```

```
{ "SISO.sf_inst_1.bbox_inst", "SISO.sf_inst_1.flop_inst_sf",  
  "SISO.sf_inst_1.mux_inst_sf" }
```

## See Also

[set\\_pref](#), [get\\_pref](#), [current\\_design](#), [current\\_instance](#), [help](#)



## collection\_display\_limit

Specifies the maximum number of objects that can be displayed by any command that returns a displayable collection

### Syntax

```
set_pref collection_display_limit 10
```

### Scope

Any

### Return Value

None

### Description

The *collection\_display\_limit* preference variable controls the maximum number of objects that can be displayed by any command that returns a displayable collection. The default value of the preference variable is 100.

When a command, such as *get\_cells*, is used at the command prompt, the result is implicitly queried. If the collection contains displayable objects, their full names are displayed. You can limit the number of objects displayed by setting this preference variable to an appropriate integer. A value of 0 displays the string representation of the collection handle, instead of the object names in the collection.

To determine the current value of this preference variable, you can use the following command:

```
get_pref collection_display_limit
```

### Examples

```
sg_shell> get_pref collection_display_limit  
100
```

```
sg_shell> set_pref collection_display_limit 5  
5
```

```
sg_shell> get_cells  
{ "U0", "U1", "U2", "U3", "U4", ... }
```

```
sg_shell> set_pref collection_display_limit 0  
0  
sg_shell> get_cells  
_sggrp4
```

## See Also

[set\\_pref](#), [get\\_pref](#), [help](#)

---

# Appendix E: CDC Application Commands

---

The following is a list of the CDC application commands:

- [List of CDC Commands](#)

## List of CDC Commands

The following is the list of CDC application commands defined in SpyGlass:

Command Name	Description	Return Value	Syntax
<i>get_clocks</i>	Obtains clocks	Collection of clock objects	<code>get_clocks</code> [<clkName>] [-filter <filter_expression>] [-of_objects <obj>]
<i>get_domains</i>	Obtains domains	Collection of domain objects	<code>get_domains</code> [<dName>] [-of_objects <obj>]
<i>propagate_clocks</i>	Propagates clocks forward	None	<code>propagate_clocks // no argument, works on get_clocks</code>
<i>get_registers</i>	Obtains registers	Collection of cells driven by specified clocks/resets	<code>get_registers</code> <patterns> [-all ] [ -edge_triggered   -level_sensitive   -posedge_triggered   -negedge_triggered ]
<i>report_clocks</i>	Reports clocks	None	<code>report_clocks</code> [<clocks>] [-verbose]
<i>report_domains</i>	Reports domains	None	<code>report_clocks</code> [<clocks>] [-verbose]

## List of CDC Commands

Command Name	Description	Return Value	Syntax
<i>get_cdc</i>	Obtains crossings	Collection of crossing objects	<pre>get_cdc [-from &lt;from_pattern&gt;] [-to &lt;to_pattern&gt;] [-from_clock &lt;f_clock&gt;] [-to_clock &lt;t_clock&gt;] [-from_domain &lt;f_domain&gt; ] [- to_domain &lt;t_domain&gt;] [-from_object &lt;f_object&gt;] [-to_object &lt;t_object&gt;] [-of_object &lt;obj&gt;] [-regexp] [-filter &lt;expr&gt;]</pre>
<i>report_cdc</i>	Reports crossings	None	
<i>get_cdc_glitch</i>	Obtains glitches in CDC paths	Collection of crossing objects that may have glitches	<pre>get_cdc_glitch [-to &lt;to_pattern&gt;] [-to_clocks &lt;t_clock&gt;] [-to_domains &lt;t_domain&gt;] [-to_objects &lt;t_object&gt;] [-of_objects &lt;obj&gt;] [-regexp] [-filter &lt;expr&gt; ]</pre>
<i>report_cdc_glitch</i>	Reports glitches in CDC paths	None	<pre>report_cdc_glitch [&lt;crossings&gt;]</pre>

Command Name	Description	Return Value	Syntax
<i>get_cdc_coherency</i>	Obtains convergence	Collection of <conv-issues> objects	get_cdc_coherency [-from <from_pattern>] [-to <to_pattern>] [-from_clock <f_clock>] [-to_clock <t_clock>] [-from_domain <f_domain> ] [-to_domain <t_domain>] [-from_object <f_object>] [-to_object <t_object>] [-of_object <obj>] [-regexp] [-filter <expr> ]
<i>report_cdc_coherency</i>	Reports convergence	None	report_cdc_coherency [<conv_issues>]
<i>get_cdc_sources</i>	Obtains the list of sources of a crossing	Collection of crossing's source objects	get_cdc_sources [<destination_name>] [-of_objects <cdc-object>]
<i>get_conv_sync_signals</i>	Obtains all synchronizers of a convergence	Collection of crossing objects	get_conv_sync_signals [<convergence>]
<i>get_glitch_sources</i>	Obtains glitch prone source-list	Collection of crossing's source objects	get_glitch_sources [<destination_name>] [-of_objects <cdc-object>]

---

# List of Topics

---

About This Book .....	31
abstract_block_violation .....	967
abstract_file .....	969
abstract_interface_param .....	972
abstract_interface_port .....	974
abstract_port .....	976
activity .....	1000
activity_data .....	1006
ADC Commands .....	233
ADC Setup Commands .....	233
adc_node .....	644
add_fault .....	1010
allow_combo_logic .....	1017
allow_test_point .....	1020
always_on_buffer .....	1020
always_on_cell .....	1022
always_on_path .....	1024
always_on_pin .....	1023
antenna_cell .....	1025
aon_buffered_signals .....	1026
assertion_signal .....	1027
associate_lib .....	1029
assume_path .....	1031
assume_waveform .....	1030
atspeed_clock_frequency .....	1034
Attribute Commands .....	459
balanced_clock .....	1036
Base Attributes .....	650
blackbox_power .....	1038
block .....	1041
blocksize .....	1044
breakpoint .....	1046
Built-in Attributes .....	608
bypass .....	1047
Capturing stdout and stderr .....	64
CDC Attributes .....	666

---

cdc_attribute .....	1048
cdc_check_glitch .....	1053
cdc_conv_node .....	618
cdc_conv_signal_node .....	618
cdc_define_transition .....	1053
cdc_false_path .....	1055
cdc_filter_coherency .....	1068
cdc_filter_path .....	1072
cdc_glitch_node .....	619
cdc_glitch_source_node .....	620
cdc_matrix_attributes .....	1073
cdc_node .....	621
cdc_source_node .....	622
cell_hookup .....	1076
cell_pin_info .....	1077
cell_tie_class .....	1078
clock .....	1080
clock .....	645
clock_buffer .....	1095
clock_domain .....	645
clockgating .....	1124
clock_group .....	1096
clock_path_wrapper_module .....	1100
clock_pin .....	1102
clock_root .....	1103
clock_sense .....	1104
clock_shaper .....	1105
Collection Commands .....	437
Command Logging in sg_shell .....	66
Common SDC Flow .....	73
complex_cell .....	1125
compressor .....	1127
Constraints Attributes .....	682
Contents of This Book .....	32
Control Signal .....	1597
Custom Reports .....	902
Data Signal .....	1599
dbist .....	1128
Debug Commands .....	345
decompressor .....	1129
Default Reports .....	903



---

define_clock_tree .....	1131
define_illegal_input_values .....	1137
define_legal_input_values .....	1138
define_library_group .....	1140
define_macro .....	1141
define_reset_order .....	1146
define_tag .....	1150
delay_buffer .....	1162
deltacheck_ignore_instance .....	1164
deltacheck_ignore_module .....	1165
deltacheck_start .....	1166
deltacheck_stop_instance .....	1168
deltacheck_stop_module .....	1169
deltacheck_stop_signal .....	1171
Design Query Commands .....	351
Design Setup Commands .....	114
design .....	623
design_map_info .....	1172
DFT Attributes .....	686
dftmax_partition .....	1175
dft_report_coverage .....	1181
dft_report_fault_list .....	1178
dft_stitching_exception .....	1179
disable_timing .....	1182
disallow_modification_type .....	1183
disallow_upf_command .....	1185
domain .....	1186
domain_inputs .....	1188
domain_outputs .....	1189
domain_signal .....	1191
dont_touch .....	1193
Dual Design Read Flow .....	75
du_cell .....	625
du_net .....	631
du_pin .....	628
du_port .....	630
enable_seq_propagation .....	1546
Error Handling in Tcl Commands .....	51
Error Scenarios and Messages .....	53
Errors and Messages Flagged in sg_shell .....	50
Example .....	1357

---

Exit Codes Reported by sg_shell .....	58
expect_frequency .....	1195
false_path .....	1198
Features of sg_shell .....	62
fifo .....	1199
flat_cell .....	633
flat_inst .....	632
flat_net .....	642
flat_pin .....	636
flat_port .....	640
For All Products .....	1591
For SpyGlass Auto Verify solution and SpyGlass CDC solution .....	1152
For SpyGlass CDC solution and SpyGlass Auto Verify Solution .....	1272
For SpyGlass CDC solution .....	1406
For SpyGlass CDC Solution .....	1527
For SpyGlass CDC Solution .....	1589
For SpyGlass CDC solution, SpyGlass Constraints solution, and SpyGlass Auto Verify solution .....	1081
For SpyGlass Constraints solution .....	1521
For SpyGlass DFT DSM solution .....	1525
For SpyGlass DFT DSM Solution .....	1651
For SpyGlass DFT solution and SpyGlass DFT DSM solution .....	1154
For SpyGlass DFT solution and SpyGlass DFT DSM solution .....	1273
For SpyGlass DFT solution and SpyGlass DFT DSM solution .....	1623
For SpyGlass DFT solution, SpyGlass DFT DSM solution .....	1087
For SpyGlass Power Estimation and SpyGlass Power Reduction Solutions .....	1666
For SpyGlass Power Family .....	1410
For SpyGlass Power Verify solution .....	1609
For SpyGlass Power Verify solution .....	1653
For SpyGlass Power Verify solution, SpyGlass ERC Product, and SpyGlass Power Estimation and SpyGlass Power Reduction solutions .....	1093
For SpyGlass STARC Product, SpyGlass STARC02 product, SpyGlass STARC05 product, and SpyGlass STARCad-21 product .....	1640
For SpyGlass TXV solution .....	1404
For the SpyGlass Power Estimation and SpyGlass Power Reduction Solutions ...	1520
For the SpyGlass Power Estimation and SpyGlass Power Reduction Solutions ...	1612
force_no_scan .....	1201
force_probability .....	1209
force_scan .....	1498
force_stable_value .....	1501
force_ta .....	1206

force_unstable_value .....	1503
formal_analysis_filter.....	1211
fsm .....	1213
gating_cell .....	1216
gating_cell_enable.....	1222
General Debug Commands .....	345
General Reports .....	901
generated_clock.....	1223
glitch_free_module.....	1228
Goal Setup or Run Commands.....	161
gray_signals .....	1229
Handling of Duplicate Constraint Specifications .....	959
History Support in sg_shell .....	65
ignore_clock_gating.....	1230
ignore_crossing.....	1232
ignore_nets .....	1570
ignore_supply_pin .....	1234
illegal_constraint_message_tag.....	1235
illegal_path .....	1238
illegal_value .....	1248
initialize_for_bist .....	1259
initstate .....	1259
input .....	1262
input_drive_strength .....	1265
input_isocell .....	1267
instance_trace .....	1269
Invoking Tcl Shell From Command-Line .....	39
Invoking Tcl Shell From SpyGlass .....	42
Invoking the Tcl Shell Interface .....	39
ip_block .....	1271
isolation_cell.....	1275
isolation_wrapper.....	1278
keeper .....	1279
latched_port .....	1281
levelshifter .....	1282
lib .....	610
lib_cell.....	612
lib_pin .....	614
Library Commands .....	352
lib_timing_arcs .....	617
List of Built-in Attributes.....	878

---

List of CDC Commands.....	940
List of Preferences.....	77
List of Product Attributes.....	891
lp_ignore_cells_for_erc.....	1284
make_mandatory_upf_commands_options.....	1285
mapped_pin_map.....	1286
mcp_info.....	1288
memory.....	1298
memory_force.....	1299
memory_inst_port.....	1306
memory_port.....	1300
memory_read_pin.....	1307
memory_tristate.....	1309
memory_type.....	1310
memory_write_disable.....	1312
memory_write_pin.....	1313
message.....	646
meta_design_hier.....	1315
meta_inst.....	1316
meta_module.....	1317
meta_monitor_options.....	1319
Miscellaneous Commands.....	858
mode_condition.....	1321
module_bypass.....	1323
module_pin.....	1325
monitor_time.....	1327
multivt_lib.....	1329
Netlist Commands.....	369
network_allowed_cells.....	1331
no_atspeed.....	1333
noclockcell_start.....	1345
noclockcell_stop_instance.....	1346
noclockcell_stop_module.....	1347
noclockcell_stop_signal.....	1349
no_convergence_check.....	1337
no_fault.....	1338
non_pd_inputcells.....	1350
non_safety_related.....	1574
no_test_point.....	1344
num_flops.....	1351
operating_mode_set.....	1356

---

output .....	1358
output_not_used .....	1360
Overriding GUI Preferences.....	81
Overview.....	925
pg_cell.....	1362
pg_pins_naming.....	1364
pin_voltage .....	1365
pll .....	1369
port_time_delay.....	1370
Power Attributes .....	735
Power Verify Attributes .....	827
power_data .....	1373
power_down.....	1374
power_down_sequence .....	1376
power_management_test_control_cell .....	1378
power_management_unit .....	1379
power_rail_mapping .....	1380
power_state .....	1382
power_switch.....	1384
Product Attributes .....	649
Product Commands.....	474
Project File .....	37
Properties of Tcl Command Arguments .....	46
pr_safe_clocks .....	1386
pulldown .....	1388
pullup .....	1390
Purpose .....	1424
qualifier .....	1391
quasi_static .....	1404
quasi_static_style.....	1411
ram_instance.....	1414
ram_switch .....	1415
rdc_false_path .....	1418
reference_toplevel_isolation_signal.....	1423
ref_power_data.....	1421
Renamed Constraints.....	960
repeater.....	1424
repeater_buffer.....	1425
Reporting Commands.....	305
require_constraint_message_tag .....	1430
require_path.....	1433

---

require_pulse.....	1445
require_stable_value .....	1448
require_strict_path .....	1451
require_structure .....	1458
require_value .....	1461
reset -async .....	1474
reset .....	1472
reset .....	647
reset_filter_path .....	1476
reset_flop_node .....	647
reset_pin.....	1481
reset_sense.....	1054
reset_synchronizer .....	1482
reset_sync_node .....	648
retention_cell.....	1484
retention_instance.....	1488
rme_config .....	1489
rule .....	646
safety_related.....	1573
scan_cell .....	1506
scan_chain .....	1507
scan_enable_source .....	1511
scan_ratio .....	1512
scan_type .....	1513
scan_wrap.....	1516
Screen Output Logging in sg_shell .....	67
sdc_data .....	1520
SDC-Equivalent Commands .....	240
sdc_node .....	645
select_wireload_model .....	1527
seq_atpg.....	1531
ser_data .....	1572
Session Commands .....	84
set .....	1532
set_case_analysis.....	1533
set_cell_allocation .....	1547
set_cell_name_pattern.....	1553
set_clock_gating_type .....	1559
set_fully_decoded_bus.....	1562
set_lib_name .....	1577
set_lib_timing_mode .....	1576

---

set_mega_cell.....	1563
set_monitor_cell .....	1578
set_pin .....	1581
set_power_info .....	1564
set_power_scaling.....	1582
set_slew .....	1495
set_supply_node .....	1586
Setting SpyGlass Preferences Using Tcl Shell Interface .....	77
sg_clock_group.....	1587
sgdc .....	1588
sg_multicycle_path.....	1566
shadow_ratio .....	1592
show_power_calc_details .....	1593
Signal Handling in sg_shell .....	69
signal_in_domain .....	1594
signal_type.....	1597
simulation_data .....	1600
special_cell.....	1604
special_module .....	1605
Specifying a Project File as the Startup File.....	56
Specifying a Tcl File as the Startup File.....	54
Specifying Collection Objects in ADC Commands.....	290
Specifying Inputs to the sg_shell .....	44
Specifying SGDC File to SpyGlass .....	955
spef_data .....	1606
SpyGlass area Reports .....	904
SpyGlass audits Reports .....	905
SpyGlass Base Commands .....	475
SpyGlass CDC Commands .....	494
SpyGlass CDC Reports .....	912
SpyGlass Constraints Commands.....	484
SpyGlass Constraints Reports.....	914
SpyGlass Design Constraints.....	963
SpyGlass DFT Commands .....	545
SpyGlass DFT DSM Reports.....	918
SpyGlass DFT Reports.....	916
SpyGlass lint Reports.....	906
SpyGlass Lint Turbo Commands .....	478
SpyGlass morelint Reports .....	907
SpyGlass OpenMore Reports .....	908
SpyGlass Power Estimate and Reduce Commands.....	600

---

SpyGlass Power Family Reports .....	920
SpyGlass Power Verify Commands .....	575
SpyGlass Power Verify Reports .....	922
SpyGlass STARC Reports .....	909
SpyGlass STARC02 Reports .....	910
SpyGlass STARC05 Reports .....	911
SpyGlass TXV Reports .....	924
Startup Files in sg_shell .....	52
stil_data .....	1565
supply .....	1609
switchoff_wrapper_instance .....	1613
sync_cell .....	1615
sync_reset_style .....	1620
syn_set_dont_use .....	1568
Syntax .....	1430
test_mode .....	1623
test_point .....	1641
tie_x .....	1643
tristate_cell .....	1645
Typographical Conventions .....	33
ungroup_cells .....	1646
use_library_group .....	1649
Using Escape Names in sg_shell .....	71
Using Key Combinations for Performing Actions .....	77
Using Named and Positional Arguments .....	45
Using sg_shell Commands .....	45
Using the Help Feature .....	62
Using the Tab Completion Feature .....	63
Utility Commands .....	291
validation_filter_path .....	1525
voltage_domain .....	1651
vt_mix_percentage .....	1670
Waiver Commands .....	322
watchpoint .....	1674
wireload_selection .....	1676
Working with SGDC Files .....	956
Writing Constraints in an SGDC File .....	954



---

# Appendix: SpyGlass Design Constraints

---

SpyGlass<sup>®</sup> Design Constraints (SGDC) are used to:

- Provide additional design information that is not apparent in RTL.
- Restrict SpyGlass analysis to a set of design objects.

## Writing Constraints in an SGDC File

You can write constraints in a text file that can have any extension. However, it is recommended that you use the .sgdc extension to distinguish this file from other files.

## Specifying SGDC File to SpyGlass

Specify SGDC files in any of the following ways:

- By using the `read_file -type sgdc <SGDC-file-name>` command in a project file
- By using the *Add Files(s)* option under the *Add Design Files* tab in Atrenta Console GUI

You can specify multiple SGDC files. In this case, SpyGlass processes these files in the specified order.

## Working with SGDC Files

The following table describes various tasks that you can perform in SGDC files:

Task	Description
Adding comments	Use # or // to add comments.
Defining a scope for constraints	<p>A scope defines a design unit for which the specified constraints are applicable. To define a scope, use the <code>current_design &lt;design-unit&gt;</code> command before writing SGDC commands, where <code>&lt;design-unit&gt;</code> can be any of the following:</p> <ul style="list-style-type: none"> <li>• For Verilog: <code>&lt;module-name&gt;</code></li> <li>• For VHDL:           <ul style="list-style-type: none"> <li><code>&lt;entity-name&gt;</code>,</li> <li><code>&lt;entity-name&gt;.&lt;archname&gt;</code>,</li> <li><code>&lt;configuration-name&gt;</code>,</li> <li><code>&lt;libname&gt;.&lt;configuration-name&gt;</code></li> </ul> </li> </ul> <p>For details, refer to the <i>Defining a Scope for Constraints</i> topic of the <i>Atrenta Console User Guide</i>.</p>
Specifying multiple values to constraint arguments	<p>You can specify multiple values to arguments in different ways. For details, refer to the <i>Specifying Multiple Values for a Constraint Argument</i> topic of the <i>Atrenta Console User Guide</i>.</p>
Handle interdependencies between constraint arguments	<p>When two arguments of the same constraint are interdependent, specify the exact matching number of values with each argument. For details, refer to the <i>Handling Interdependencies between Different Arguments</i> topic of the <i>Atrenta Console User Guide</i>.</p>

Task	Description
Specify signals names in the correct format	<p>Certain constraint arguments accept names of signals, such as clock signals and low power signals.</p> <p>Based on the type of signals, such as scalar or vector signals or the design hierarchy in which signals are present, you must specify signal names in a correct format so that SpyGlass can identify them correctly.</p> <p>For details, refer to the <i>Specifying Signal Names</i> topic of the <i>Atrenta Console User Guide</i>.</p>
Define and use variables	<p>Variables are used to store values that can be used as argument values of constraints.</p> <p>Once you define a variable and assign a value to it, you can use that variable name as the value of a constraint argument. SpyGlass internally expands that variable name to its value for that argument.</p> <p>For details, refer to the <i>Defining and Using Variables</i> topic of the <i>Atrenta Console User Guide</i>.</p>
Include an SGDC file in another SGDC file	<p>Use the <code>include</code> directive to include an SGDC file in another SGDC. The <code>include</code> directive is used in the following format:</p> <pre>include &lt;file-name&gt;</pre> <p>For details, refer to the <i>Including an SGDC File in Another SGDC File</i> topic of the <i>Atrenta Console User Guide</i>.</p>

Task	Description
Implement compilation of SGDC commands based on certain conditions	To use the same SGDC file for different functional and testing analysis modes, compile different commands from the same SGDC file based on different conditions. For details, refer to the <i>Conditionally Specifying SGDC Constraints</i> topic of the <i>Atrenta Console User Guide</i> .
Importing Block-Level SGDC Commands to Chip-Level	While integrating design blocks at chip-level, you can migrate block-level SGDC files to the chip-level for performing chip-level analysis. For details, refer to the <i>Importing Block-Level SGDC Commands to Chip-Level</i> topic of the <i>Atrenta Console User Guide</i> .
Using the :: operator to define a scope for searching instances in design units	For details, refer to the <i>Implementing Scoping in SGDC Commands</i> topic of the <i>Atrenta Console User Guide</i> .

## Handling of Duplicate Constraint Specifications

If you have given multiple specifications for a constraint that can be applied only once on a design object, the following actions occur:

- SpyGlass considers only the last specification of that constraint.
- SpyGlass reports the *SGDCWRN\_115* warning and ignores the rest of the specifications of that constraint.

Consider the following example:

```
current_design top  
  
set_case_analysis -name in -value 0  
set_case_analysis -name in -value 1
```

For the above example, SpyGlass considers only the last `set_case_analysis` constraint specification and ignores the first constraint specification that sets the value of the `in` pin to 0.

## Renamed Constraints

Some SGDC constraints have been renamed to improve readability and consistency with all the other constraints in SpyGlass. The original names, however, are still supported for backward compatibility.

The following table lists the old names and the corresponding new names of constraints:

<b>Old Name</b>	<b>New Name</b>
aonbuffer	<i>always_on_buffer</i>
aonbufferedsignals	<i>aon_buffered_signals</i>
apcell	<i>antenna_cell</i>
balancedClock	<i>balanced_clock</i>
cellhookup	<i>cell_hookup</i>
clockPin	<i>clock_pin</i>
clockshaper	<i>clock_shaper</i>
defineTag	<i>define_tag</i>
expectFrequency	<i>expect_frequency</i>
gatingcell	<i>gating_cell</i>
gatingcell_enable	<i>gating_cell_enable</i>
ignorepdcrossing	<i>ignore_crossing</i>
iniscell	<i>input_isocell</i>
IP_block	<i>ip_block</i>
isocell	<i>isolation_cell</i>
lpsignal	<i>assertion_signal</i>
memory3s	<i>memory_tristate</i>
memoryreadpin	<i>memory_read_pin</i>
memorytype	<i>memory_type</i>
memorywritedisable	<i>memory_write_disable</i>
memorywritepin	<i>memory_write_pin</i>
moduleByPass	<i>module_bypass</i>
modulePin	<i>module_pin</i>
nofault	<i>no_fault</i>



## Renamed Constraints

<b>Old Name</b>	<b>New Name</b>
noScan	<i>force_no_scan</i>
pdsequence	<i>power_down_sequence</i>
pdsignal	<i>domain_signal</i>
pgcell	<i>pg_cell</i>
pgpins_naming	<i>pg_pins_naming</i>
pi_drive_strength	<i>input_drive_strength</i>
pinvoltage	<i>pin_voltage</i>
porttimedelay	<i>port_time_delay</i>
powerdomaininputs	<i>domain_inputs</i>
powerdomainoutputs	<i>domain_outputs</i>
powerswitch	<i>power_switch</i>
pullDown	<i>pulldown</i>
pullUp	<i>pullup</i>
raminstance	<i>ram_instance</i>
ramswitch	<i>ram_switch</i>
requirePath	<i>require_path</i>
requireValue	<i>require_value</i>
resetPin	<i>reset_pin</i>
retain_instance	<i>retention_instance</i>
retencell	<i>retention_cell</i>
scanchain	<i>scan_chain</i>
scanratio	<i>scan_ratio</i>
scantype	<i>scan_type</i>
scanwrap	<i>scan_wrap</i>
sdcschema	<i>sdc_data</i>
selectwireloadmodel	<i>select_wireload_model</i>
seqATPG	<i>seq_atpg</i>
set_black_box_power	<i>blackbox_power</i>
set_cell_pin_info	<i>cell_pin_info</i>
set_cell_tie_class	<i>cell_tie_class</i>
set_design_map_info	<i>design_map_info</i>
set_instance_for_activity_trace	<i>instance_trace</i>

<b>Old Name</b>	<b>New Name</b>
setPin	<i>set_pin</i>
shadowratio	<i>shadow_ratio</i>
smodule	<i>special_module</i>
specialcell	<i>special_cell</i>
testclock_frequency	<i>atspeed_clock_frequency</i>
testclockFrequency	<i>atspeed_clock_frequency</i>
testmode	<i>test_mode</i>
testpoint	<i>test_point</i>
voltagedomain	<i>voltage_domain</i>
wireloadselection	<i>wireload_selection</i>

## SpyGlass Design Constraints

The following table lists the SpyGlass Design Constraints (SGDC) used by various SpyGlass products:

<b>SpyGlass Auto Verify Solution</b>		
<i>clock</i>	<i>reset</i>	<i>define_tag</i>
<i>set_case_analysis</i>	<i>special_module</i>	<i>breakpoint</i>
<i>watchpoint</i>	<i>formal_analysis_filter</i>	
<b>SpyGlass CDC Solution</b>		
<i>assume_path</i>	<i>breakpoint</i>	<i>cdc_false_path</i>
<i>cdc_filter_path</i>	<i>clock</i>	<i>deltacheck_stop_instance</i>
<i>define_tag</i>	<i>deltacheck_ignore_instance</i>	<i>deltacheck_ignore_module</i>
<i>deltacheck_start</i>	<i>deltacheck_stop_module</i>	<i>deltacheck_stop_signal</i>
<i>fifo</i>	<i>input</i>	<i>ip_block</i>
<i>network_allowed_cells</i>	<i>noclockcell_start</i>	<i>noclockcell_stop_instance</i>
<i>noclockcell_stop_module</i>	<i>noclockcell_stop_signal</i>	<i>noclockcell_stop_instance</i>
<i>noclockcell_stop_module</i>	<i>noclockcell_stop_signal</i>	<i>no_convergence_check</i>
<i>num_flops</i>	<i>output</i>	<i>output_not_used</i>
<i>port_time_delay</i>	<i>reset</i>	<i>set_case_analysis</i>
<i>signal_in_domain</i>	<i>watchpoint</i>	<i>quasi_static</i>
<i>define_reset_order</i>	<i>allow_combo_logic</i>	<i>abstract_port</i>
<i>sync_cell</i>	<i>sgdc</i>	<i>quasi_static_style</i>
<i>noclockcell_stop_instance</i>	<i>signal_type</i>	<i>abstract_file</i>
<i>monitor_time</i>	<i>gray_signals</i>	<i>clock_path_wrapper_module</i>
<i>clock_sense</i>	<i>cdc_matrix_attributes</i>	<i>repeater</i>
<i>cdc_attribute</i>	<i>meta_inst</i>	<i>meta_module</i>
<i>reset_filter_path</i>	<i>rdc_false_path</i>	<i>sg_clock_group</i>
<i>cdc_check_glitch</i>	<i>cdc_define_transition</i>	

<i>meta_monitor_options</i>	<i>sdc_data</i>	<i>validation_filter_path</i>
<b>SpyGlass Constraints Solution</b>		
<i>assume_path</i>	<i>clock</i>	<i>domain</i>
<i>mapped_pin_map</i>	<i>sdc_data</i>	<i>block</i>
<i>blocksize</i>	<i>clock_group</i>	<i>abstract_file</i>
<b>SpyGlass DFT Solution</b>		
<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>dft_stitching_exception</i>	<i>force_ta</i>	<i>gating_cell</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>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_file</i>	<i>illegal_path</i>	<i>illegal_constraint_message_tag</i>
<i>require_constraint_message_tag</i>		
<b>SpyGlass DFT DSM Solution</b>		
<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>

## SpyGlass Design Constraints

<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>	<i>scan_chain</i>
<i>scan_enable_source</i>	<i>scan_wrap</i>	<i>test_mode</i>
<i>test_point</i>	<i>voltage_domain</i>	<i>abstract_file</i>
<i>sg_multicycle_path</i>		
<b>SpyGlass ERC Product</b>		
<i>set_case_analysis</i>	<i>clock</i>	<i>reset</i>
<i>set</i>	<i>abstract_file</i>	
<b>SpyGlass latch Product</b>		
<i>assume_path</i>	<i>set_case_analysis</i>	<i>abstract_file</i>
<b>SpyGlass Power Verify Solution</b>		
<i>always_on_cell</i>	<i>always_on_pin</i>	<i>always_on_buffer</i>
<i>always_on_buffer</i>	<i>assume_path</i>	<i>cell_hookup</i>
<i>clock</i>	<i>ignore_crossing</i>	<i>input_isocell</i>
<i>isolation_cell</i>	<i>delay_buffer</i>	<i>assertion_signal</i>
<i>multivt_lib</i>	<i>non_pd_inputcells</i>	<i>power_down_sequence</i>
<i>domain_signal</i>	<i>pg_cell</i>	<i>pg_pins_naming</i>
<i>pin_voltage</i>	<i>power_down</i>	<i>domain_inputs</i>
<i>domain_outputs</i>	<i>power_switch</i>	<i>power_state</i>
<i>ram_instance</i>	<i>ram_switch</i>	<i>retention_instance</i>
<i>retention_cell</i>	<i>set_case_analysis</i>	<i>cell_pin_info</i>
<i>cell_tie_class</i>	<i>special_cell</i>	<i>supply</i>
<i>switchoff_wrapper_instance</i>	<i>voltage_domain</i>	<i>power_data</i>
<i>ignore_supply_pin</i>	<i>antenna_cell</i>	<i>isolation_wrapper</i>
<i>aon_buffered_signals</i>	<i>levelshifter</i>	<i>set_power_info</i>
<i>associate_lib</i>	<i>set_supply_node</i>	<i>disallow_upf_command</i>
<i>make_mandatory_upf_commands_options</i>	<i>reference_toplevel_isolation_signal</i>	
<b>SpyGlass OpenMore Product</b>		

<i>set_case_analysis</i>	<i>abstract_file</i>	
<b>SpyGlass Power Estimation and SpyGlass Power Reduction Solutions</b>		
<i>activity</i>	<i>activity_data</i>	<i>blackbox_power</i>
<i>clock</i>	<i>clock_buffer</i>	<i>define_clock_tree</i>
<i>define_library_group</i>	<i>design_map_info</i>	<i>gating_cell</i>
<i>input_drive_strength</i>	<i>instance_trace</i>	<i>memory</i>
<i>memory_port</i>	<i>mode_condition</i>	<i>operating_mode_set</i>
<i>pg_cell</i>	<i>power_rail_mapping</i>	<i>power_state</i>
<i>pr_safe_clocks</i>	<i>repeater_buffer</i>	<i>rme_config</i>
<i>sdc_data</i>	<i>select_wireload_model</i>	<i>set_case_analysis</i>
<i>set_cell_allocation</i>	<i>set_clock_gating_type</i>	<i>syn_set_dont_use</i>
<i>ignore_nets</i>	<i>set_monitor_cell</i>	<i>set_power_scaling</i>
<i>set_mega_cell</i>	<i>spef_data</i>	<i>supply</i>
<i>ungroup_cells</i>	<i>use_library_group</i>	<i>voltage_domain</i>
<i>memory_inst_port</i>	<i>vt_mix_percentage</i>	<i>wireload_selection</i>
<i>ignore_clock_gating</i>	<i>disallow_modification_type</i>	<i>set_slew</i>
<b>SpyGlass STARC Product</b>		
<i>set_case_analysis</i>	<i>test_mode</i>	<i>abstract_file</i>
<b>SpyGlass STARC02 Product</b>		
<i>test_mode</i>	<i>abstract_file</i>	
<b>SpyGlass STARC05 Product</b>		
<i>test_mode</i>	<i>abstract_file</i>	
<b>SpyGlass STARCad-21 Product</b>		
<i>test_mode</i>	<i>abstract_file</i>	
<b>SpyGlass TXV Solution</b>		
<i>clock</i>	<i>initstate</i>	<i>quasi_static</i>
<i>reset</i>	<i>simulation_data</i>	<i>mcp_info</i>
<i>assume_waveform</i>		

## abstract\_block\_violation

### Purpose

The `abstract_block_violation` constraint specifies information about the violation reported during the generation of abstract view of a design block. This constraint is generated in the SGDC file representing the abstract view.

**NOTE:** *SpyGlass generates this constraint for its internal use.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass CDC solution, SpyGlass Constraints solution, SpyGlass base products

### Syntax

The syntax to specify the `abstract_block_violation` constraint is as follows:

```
abstract_block_violation
  -name <rule-name>
  -sev <severity>
  [ -count <violation-count> | -waived_count
    <waived-violation-count> ]
  [ -is_builtin ]
```

### Arguments

**-name <rule-name>**

Specifies the name of the rule that reports a violation.

**-sev <severity>**

Specifies the rule severity.

**-count <violation-count>**

Specifies the number of violations reported for the rule specified in the `-name <rule-name>` argument.

**-waived\_count <waived-violation-count>**

Specifies the number of violations waived for the rule specified in the *-name <rule-name>* argument.

**-is\_builtin**

Specifies that the rule specified in the *-name <rule-name>* argument is a built-in rule.

**Example**

Consider the following constraint generated during abstract block generation:

```
abstract_block_violation -name WarnAnalyzeBBox -sev Warn  
-count 1 -is_builtin
```

The above constraint specifies that during abstract block generation, one violation of the *WarnAnalyzeBBox* built-in rule is reported, and the severity of the rule is Warning.



## abstract\_file

### Purpose

The `abstract_file` constraint is used to specify the product version and the block SGDC file using which the abstract view of a block was generated.

The advantage of specifying this information is that if an abstract view becomes incompatible in a particular release, you can specify the product version and the original block SGDC file of that abstract view to SpyGlass. This way, SpyGlass uses the specified abstract view during the SoC validation.

In some cases, the block SGDC file specified by this constraint refers to some additional files through any of the following specifications:

- `include <sgdc_file>`
- `sgdc -import <block_name> <block_constraint_file>`
- `sdc_data -type <sdc_file>`
- `power_data -format <cpf|upf> -file <cpf/upf-file>`
- `activity_data -format <vcd|fsdb|saif> -file <file>`

In such cases, package the additional files manually along with the block SGDC file.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass CDC solution, SpyGlass Constraints solution, SpyGlass base products

### Syntax

The syntax to specify the `abstract_file` constraint is as follows:

```
abstract_file
  -version <version-string>
  -scope <dft|const|base|cdc>
  [ -block_file <original-block-sgdc-files> ]
  [ -abstract_searchpath <search_paths for block-sgdc
file(s)> ]
```

## Arguments

### **-version <version-string>**

Specifies the version number of an abstract view. This version number is product-specific.

During SoC validation run, SpyGlass uses this value to perform following abstraction model version checks:

- When the current product version supports higher abstraction version than the supplied abstracted model SGDC, the `SGDC_abstract_file01` rule reports the following warning message:  
SoC results for product <product> as abstracted model for block <block-name> is not up-to-date. Please regenerate these abstract model using latest <product> policy
- If the supplied abstracted SGDC is generated using a product that has a higher abstraction version than the current product version, the `SGDC_abstract_file02` rule reports the following error message:  
Abstracted models provided for product <product> are generated with newer <product> policy, and are not compatible with the current <product> product used. Please re-generate these abstract models with current <policy-name> policy

### **-scope <dft|const|base|cdc>**

Specifies the scope in which an abstract view is generated.

A scope specifies a product.

### **-block\_file <original-block-sgdc-files>**

Specifies a space-separated list of block SGDC files using which the abstract view was generated.

### **-abstract\_searchpath <search\_paths\_for\_block-sgdc\_file(s)>**

Specifies a space-separated list of search paths to locate block SGDC files.

## Examples

Consider the following example:

```
abstract_file -version 1.0 -scope dft  
-block_file blk1_dft.sgdc
```

The above example means that the abstract view being used during SoC validation was generated from the blk1\_dft.sgdc file. This abstract view is of the 1.0 version and was generated by using SpyGlass DFT.

## abstract\_interface\_param

### Purpose

The `abstract_interface_param` constraint specifies the definition of the parameter abstracted.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass CDC solution, SpyGlass Constraints solution, SpyGlass base products

### Syntax

The syntax to specify the `abstract_interface_param` constraint is as follows:

```
abstract_interface_param
  -name <param-name>
  -value <param-value>
```

### Arguments

**-name <param-name>**

Specifies the name of the parameter.

**-value <param-value>**

Specifies the value of the parameter.

### Example

Consider the following RTL specification:

```
module reorder_bits(in,out);
  parameter SIZE = 10;
  parameter FLIPBIT = ((SIZE + 1) >> 2);
  input [SIZE:0] in;
  output reg [SIZE:0] out;
```

```
endmodule
```

```
abstract_interface_parameter -name "SIZE" -value "10"
```

```
abstract_interface_parameter -name "FLIPBIT" -value "((SIZE  
+ 1) >> 2)"
```

## abstract\_interface\_port

### Purpose

The `abstract_interface_port` constraint specifies the definition of the port abstracted.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass CDC solution, SpyGlass Constraints solution, SpyGlass base products

### Syntax

The syntax to specify the `abstract_interface_port` constraint is as follows:

```
abstract_interface_port
  -name <port-name>
  -definition <port-def>
```

### Arguments

**-name <port-name>**

Specifies the name of the port net.

**-definition <port-def>**

Specifies the way it is defined in the RTL.

### Example

Consider the following RTL specification:

```
module reorder_bits(in,out);
  parameter SIZE = 10;
  parameter FLIPBIT = ((SIZE + 1) >> 2);
  input [SIZE:0] in;
  output reg [SIZE:0] out;
```

```
endmodule
```

```
abstract_interface_port -name "in"    -definition "input  
[SIZE:0] in; "
```

```
abstract_interface_port -name "out" -definition "output reg  
[SIZE:0] out; "
```

## abstract\_port

### Purpose

The `abstract_port` constraint is used to define abstracted information for block ports.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass CDC solution, SpyGlass Constraints solution, SpyGlass base products

### Syntax




The syntax to specify the `abstract_port` constraint is as follows:

```
abstract_port
  -module <module-name>
  -ports <port-name-list>
  -clock <clock-port-list >
  [ -reset <reset-name> ]
  [ -combo <yes | no | unknown> ]
  [ -sync <active | inactive>
    -from <src-clk list> -to <dest-clk list>
    [ -seq <yes | no> ]
    [ -sync_names <sync-names> ]
  ]
  [ -related_ports <related-ports> ]
  [ -scope <dft | cdc | const | base> ]
  [ -mode <mode-name> ]
  [ -connected_inst <instance-name> ]
  [ -inst_master <instance-master> ]
  [ -inst_pin <instance-pin> ]
  [ -path_logic <path-logic> ]
  [ -path_polarity <path-polarity> ]
  [ -phase_list <min|max|rise|fall|setup|hold|start|end> ]
  [ -multiplier_value <multiplier-value> ]
  [ -path_constraint <min_delay | max_delay> ]
  [ -ignore ]
  [ -combo_ifn <clock_port> ]
```



```
[ -start ]
[ -end ]
[ -direction <input/output> ]
[ -init <initial_value> ]
```

**NOTE:** *The SpyGlass DFT DSM solution uses only the following options of this constraint:*

-  `-module`
-  `-ports`
-  `-clock`

## Arguments

### **-module <module-name>**

Specifies the name of a module for which this constraint is being specified.

### **-ports <port-name-list>**

Specifies a space-separated list of port names of a module specified in the `-module <module-name>` argument.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-ports` constraint for the output ports and [Example 9](#) for specifying the `-ports` argument for input ports.

### **-clock <clock-port-list>**

Specifies clock input, inout, or output ports of a module by which the ports specified by the `-ports <port-name-list>` argument is driven. You can also specify a clock as a virtual clock.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-clock` constraint for the output ports and [Example 9](#) for specifying the `-clock` argument for input ports.

### **-reset <reset-name>**

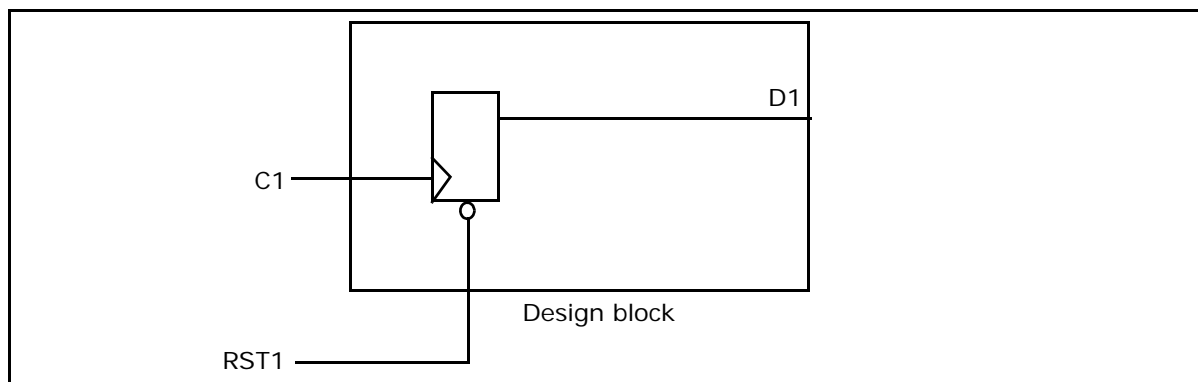
Specifies the reset name assigned to the port of an abstract view.

While creating abstract views for design blocks, if a sequential element with a reset pin reaches a block port, the `Ac_abstract01` and `Ac_blksgdc01` rules dump the `-reset <reset-name>` argument to the `abstract_port`

constraint generated for the abstract view. However, if an input port of the design block reaches a sequential element having a reset pin, the *Clock\_info15* and *Setup\_port01* rule dumps this argument.

**NOTE:** *This functionality works only if the enable\_soc\_rdc parameter is set to yes.*

Consider the following scenario when the `enable_soc_rdc` parameter is set to `yes`:



**FIGURE 1.**

For the above case, the *Ac\_abstract01* rule generates the following *abstract\_port* constraint for the block port D1:

```
abstract_port -ports D1 -clock C1 -reset RST1
```

Based on the generated reset information, the *Ac\_abstract\_validation01*, *Ac\_abstract01*, *Ac\_resetcross01*, and *Ar\_resetcross01* rules perform checks related to invalid reset crossings between the abstract views and top-level modules.

To specify the `-reset` argument of the *abstract\_port* constraint on an input port, RST1, specify the following constraint:

```
abstract_port -ports RST1 -clock C1 -reset RST1
```

**-combo** <yes | no | unknown>

Specifies the presence of a combinational logic in the input cone of a port specified with the *-ports* <port-name-list> argument.

By default, the value of this argument is `unknown`, which means that

related validation checks should not be performed.

You can set this argument to `yes` or `no`.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-combo` constraint for the O4 output port and [Example 9](#) for specifying the `-combo` argument for the IN4 input port.

#### **-sync <active | inactive>**

Specifies whether a port is driven by a control synchronizer or a data synchronizer.

Set this argument to `active` if a port is driven by a control synchronizer that can act as a synchronized enable for other data crossings.

Set this argument to `inactive` if a port is driven by a data synchronizer that cannot act as a synchronized enable for other data crossings.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-sync` constraint for the O2 and O3 output ports. The `-sync` argument is not applicable for input ports.

#### **-from <src-clk-list>**

Specifies a space-separated list of clock ports or virtual clock names that are reaching to the source of a synchronizer. The `-from` argument is used only with the `-sync` argument.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-from` constraint for the O2 and O3 output ports. The `-from` argument is not applicable for input ports.

#### **-to <dest-clk list>**

Specifies a space-separated list of clock ports or virtual clock names that are reaching to the destination of a synchronizer. The `-to` argument is used only with the `-sync` argument.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-to` constraint for the O2 and O3 output ports. The `-to` argument is not applicable for input ports.

**-seq <yes | no>**

Specifies whether sequential elements exist in the input cone of a port between the synchronizer and the port specified by using the *-ports <port-name-list>* argument.

You can set this argument to *yes* or *no*. By default, it is set to *no*.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port -seq` constraint for the O3 output port. The `-seq` argument is not applicable for input ports.

**-sync\_names <sync-names>**

Specifies a space-separated list of hierarchical net or hierarchical pin names of synchronizers. The `-sync_names` argument is used only with the `-sync` argument.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port -sync_names` constraint for the O2 and O3 output ports. The `-sync_names` argument is not applicable for input ports.

**-related\_ports <related-ports>**

Specifies a list of ports that have a sequential path to the ports specified by the *-ports <port-name-list>* argument. The direction of related ports should either be *inout* or *reverse* of the direction of the ports specified by the *-ports <port-name-list>* argument.

For example, if you specify `-ports P[1:0]` and `-related_ports in1[2:0]` for this constraint, it implies that 0,1, and 2 bits of the `in1` port are driving both the `P[0]` and port `P[1]` ports.

**NOTE:** *The `-related_ports` and `-sync <active | inactive>` arguments of this constraint are mutually exclusive.*

See [Example 8](#) for the SpyGlass CDC generated `abstract_port -related_ports` constraint for the O1 and O4 output ports. The `-related_ports` argument is not applicable for input ports.

**-scope <dft | cdc | const | base>**

Specifies the scope in which an abstract view should be generated. A scope specifies a product.

**-mode <mode-name>**

Specifies the mode for which the Interface Logic Model (ILM) modeling information is generated.

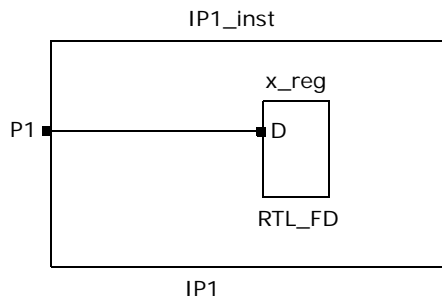
The following table describes the mode for the scopes specified by the *-scope <dft / cdc / const / base>* argument:

Scope	Mode for the Scope
dft	The modes can be shift, capture, and atspeed under which SpyGlass DFT analysis is performed. The abstract_port modeling is generated for these modes.
cdc	The mode is set_case_analysis because SpyGlass CDC analysis is based on this constraint.
base	The mode is set_case_analysis because SpyGlass Base analysis is based on this constraint.
const	The mode is the same as the one specified by the sdc_data constraint. The abstract_port modeling is done for each mode of SDC data.

**-connected\_inst <instance-name>**

Specifies the name of the instance whose pin should be connected with the IP port. The master of this instance is specified by the *-inst\_master <instance-master>* argument.

For example, consider the following figure:



**FIGURE 2.** Specifying the instance name

In the above figure, `x_reg` is the instance name of the master flip-flop

RTL\_FD.

The following is the `abstract_port` constraint specification in this case:

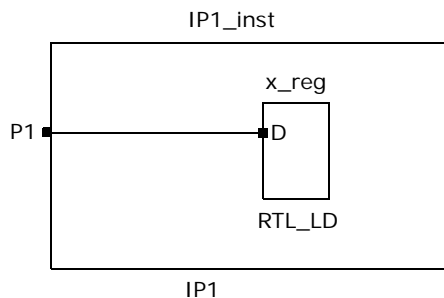
```
current_design IP1
```

```
abstract_port -ports P1 -inst_master RTL_FD -connected_inst
x_reg -inst_pin D
```

### **-inst\_master <instance-master>**

Specifies the master type, such as RTL\_FD and RTL\_LD, of the instance specified by the `-connected_inst <instance-name>` argument.

For example, consider the following figure in which RTL\_LD is the master of the `x_reg` latch:



**FIGURE 3.** Specifying the instance master

In the above figure, specify RTL\_LD in the `-inst_master <instance-master>` argument while connecting the P1 port with the D pin of the `x_reg` instance. The following is the `abstract_port` constraint specification in this case:

```
current_design IP1
```

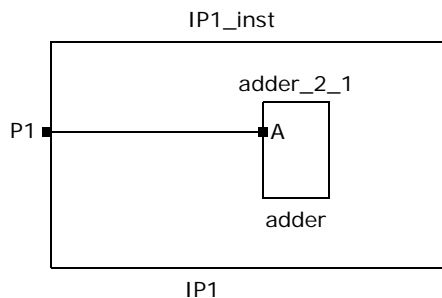
```
abstract_port -ports P1 -inst_master RTL_LD -connected_inst
x_reg -inst_pin D
```

## Specifying a Black Box Master

In the `-inst_master <instance-master>` argument, you can also specify a black box master name corresponding to the black box instance specified by the `-connected_inst <instance-name>` argument. Use the following format while specifying the black box master name:

```
-inst_master "BBOX:<black-box-master-name>"
```

For example, consider the following figure in which `adder` is the master of the `adder_2_1` instance:



**FIGURE 4.** Specifying the black box master

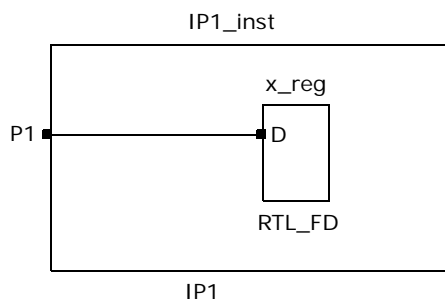
For the above scenario, specify `adder` in the `-inst_master` argument while connecting the `P1` port with the `A` pin of the `adder_2_1` instance, as shown below:

```
current_design IP1
abstract_port -ports P1 -inst_master "BBOX:adder"
-connected_inst adder_2_1 -inst_pin A
```

### `-inst_pin <instance-pin>`

Specifies the pin that should be connected with the port specified by the `-ports <port-name-list>` argument. This pin is present on the instance of the master specified by the `-inst_master <instance-master>` argument.

For example, consider the following figure:



**FIGURE 5.** Specifying the instance pin

In the above figure, D is the instance pin present on the `x_reg` instance whose master is `RTL_FD`. In this case, specify the following constraint to connect the D pin of the `x_reg` instance with the P1 port:

```
current_design IP1
```

```
abstract_port -ports P1 -inst_master RTL_FD -connected_inst
x_reg -inst_pin D
```

### **-path\_logic <path-logic>**

Specifies the logic that exists between the connection from:

- An input port to an output port.
- A port (specified by `-ports <port-name-list>`) to an instance pin (specified by `-inst_pin <instance-pin>`).
- An instance pin (specified by `-inst_pin <instance-pin>`) to a port (specified by `-ports <port-name-list>`).
- A port to a hanging path by using the logic specified by the `-path_logic` argument.

You can specify any of the following values as the path logic:

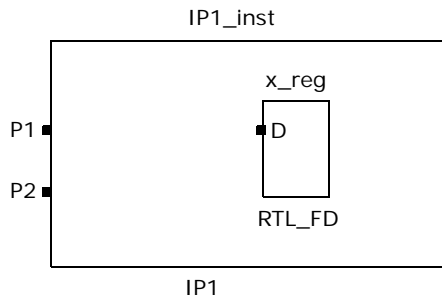
Path Logic	Description
combo	Specifies that the connection is through a combinational logic.



Path Logic	Description
buf	Specifies that the connection is through a buffer.
inv	Specifies that the connection is through an inverter.

See [Example 8](#) for the SpyGlass CDC generated `abstract_port` `-path_logic` constraint for the O4 and O5 output ports and [Example 9](#) for specifying the `-path_logic` argument for the IN5 input port.

For example, consider the following figure:



**FIGURE 6.** Specifying the path logic

In the above figure, consider that you want to perform the following actions:

- Create a connection from P1 to D such that a combinational logic exists between the connection. Specify the following constraints in this case:

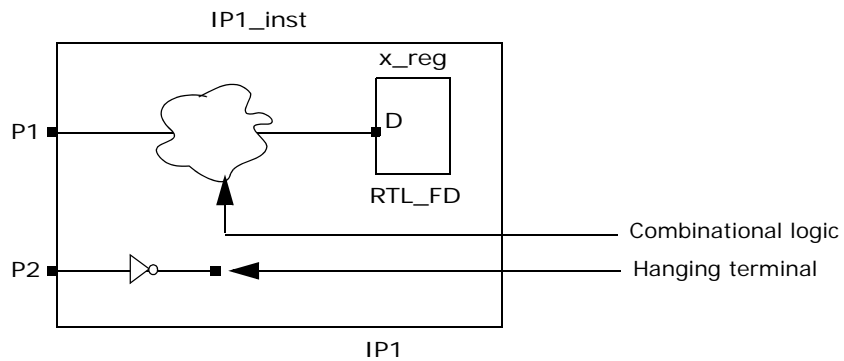
```
current_design IP1
```

```
abstract_port -ports P1 -inst_master RTL_FD -
connected_inst x_reg -inst_pin D -path_logic combo
```

- Create a connection from the P2 port to a hanging terminal by using an inverter. Specify the following constraint in this case:

```
abstract_port -ports P2 -path_logic inv
```

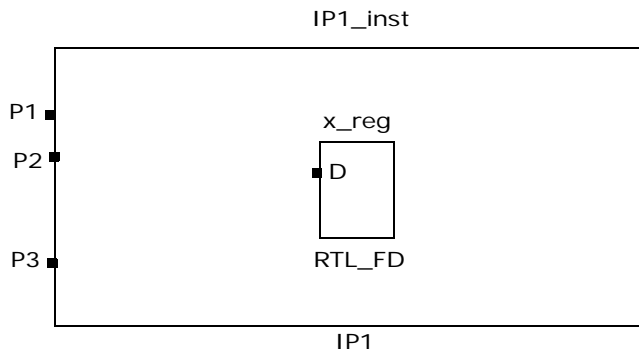
After specifying the above constraints, [Figure 6](#) changes to the following:



**FIGURE 7.** Inserting a combinational logic

## Handling Multiple Connection Paths Belonging to Different Scopes

Consider the following figure:



**FIGURE 8.** Handling Connections of Different Scopes

In the above figure, consider that you want to connect the P1, P2, and P3 ports with the D pin of the x\_reg instance such that:

- The connection from P1 to D belongs to the CDC scope.

To create this connection, specify the following constraint:

```
abstract_port -ports P1 -inst_master RTL_FD
-connected_inst x_reg -inst_pin D -path_logic combo
```

## SpyGlass Design Constraints

```
-scope cdc
```

- The connection from P2 to D belongs to the DFT scope in the shift mode.

To create this connection, specify the following constraint:

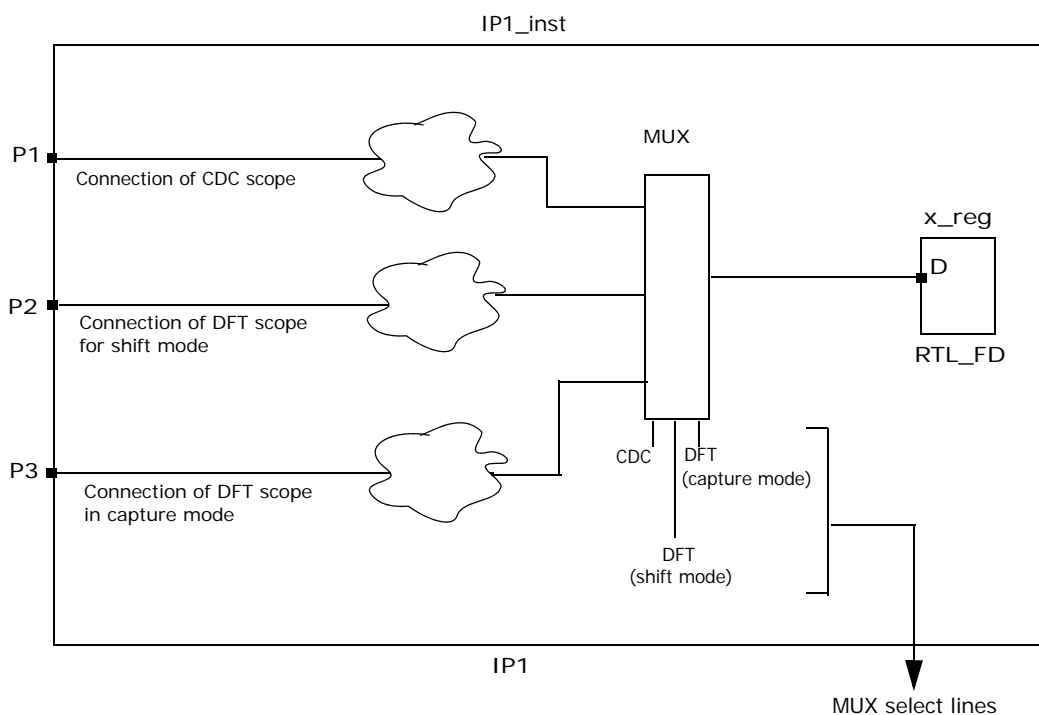
```
abstract_port -ports P2 -inst_master RTL_FD  
-connected_inst x_reg -inst_pin D -path_logic combo  
-scope dft -mode shift
```

- The connection from P3 to D belongs to the DFT scope in the capture mode.

To create this connection, specify the following constraint:

```
abstract_port -ports P3 -inst_master RTL_FD  
-connected_inst x_reg -inst_pin D -path_logic combo  
-scope dft -mode capture
```

After specifying the above constraints, SpyGlass generates connections, as shown in the following figure:



**FIGURE 9.** Handling Connections of Different Scopes

In the above figure, notice the MUX present between the connections.

SpyGlass inserts this MUX so that only one connection of a particular scope (CDC, DFT in the shift mode, or DFT in the capture mode) is active at a time. Therefore:

- When the CDC select line of the MUX is on, the connection between P1 and D is active.
- When the DFT select line of the MUX is on for the shift mode, the connection between P2 and D is active.
- When the DFT select line of the MUX is on for the capture mode, the connection between P3 and D is active.

**-path\_polarity <path-polarity>**

Specifies the polarity of the path containing a combinational logic.

You can specify any of the following values as the path polarity:

Path Polarity	Description
buf	Specifies that the path containing the combinational logic is buffered.
inv	Specifies that the path containing the combinational logic is inverted.

**NOTE:** *Specify this argument only if you specify `-path_logic combo` in the `abstract_port` constraint specification.*

**-phase\_list <min|max|rise|fall|setup|hold|start|end>**

Specifies options associated with SDC commands used by SpyGlass Constraints solution.

For example, when you set the value of this argument to `setup` for multi-cycle paths specified by the `set_multicycle_path` SDC command, SpyGlass abstracts the setup-related information while generating an abstract view.

**-multiplier\_value <multiplier-value>**

Specifies a multiplier value for the `set_multicycle_path` SDC command.

**-path\_constraint <min\_delay | max\_delay>**

Set this argument to `min_delay` or `max_delay` if a path is constrained by using the `set_max_delay` or `set_min_delay` SDC command, respectively.

**-ignore**

Indicates that the specified block port is not considered for SpyGlass analysis.

The `Clock_info15` rule of the SpyGlass CDC solution generates the `abstract_port -ignore` constraint if all the fan-out of an input port

are hanging or blocking. See [Example 4](#).

The `Ac_abstract01` rule of the SpyGlass CDC solution generates the `abstract_port -ignore` constraint if all the fan-in of an output port are hanging or blocking.

**NOTE:** *It is not mandatory to specify the `-clock` argument of the `abstract_port` constraint when you specify the `-ignore` argument of this constraint.*

#### **[`-combo_ifn <clock_port>` ]**

Indicates the clock port associated with the control synchronizer.

See [Example 9](#) for specifying the `-combo_ifn` argument for IN4 input port. The `-combo_ifn` argument is not applicable for output ports.

#### **[`-start>` ]**

Set this argument to validate the domain assumption and other CDC attributes applied on the design object by using the `abstract_port` constraint from its fanin cone.

Note that CDC verification is performed from the design object, which has the `abstract_port` constraint specified, to the output cone of the specified port/pin.

**NOTE:** *This argument is used by the SpyGlass CDC solution only.*

Table 1 below summarizes verification/validation with regard to CDC start points.

**TABLE 1**

	<b>CDC start point</b>
Verification	Primary Input/Blackbox output
Validation	Primary Output/Blackbox input

If the `-start` argument is not specified for an `abstract_port`, SpyGlass CDC considers the design objects specified in Table 2 as the start point.

**TABLE 2**

Object Type	Default	Description
Top-level input port	<code>-start</code>	
Top-level inout port	<code>-start</code> <code>-end</code>	For input side, it will be considered as <code>-start</code> and for output side, it will be treated as <code>-end</code>
Black-box output	<code>-start</code>	
Black-box inout	<code>-start</code> <code>-end</code>	For output side, it will be considered as <code>-start</code> and for input side, it will be considered as <code>-end</code>
Abstracted block output (seen at higher level of hierarchy)	<code>-start</code>	Similar to black box handling
Abstracted block input (seen at the higher level of hierarchy)	<code>-start</code>	Differs from blackbox handling as well as the description in Table 1 to avoid backward compatibility issues for existing SoC flow. SpyGlass CDC validates the abstract port and therefore consider it as CDC start point rather than as CDC end point.
Abstracted block inout (seen at the higher level of hierarchy)	<code>-start</code>	Differs from blackbox handling as well as the description in Table 1 to avoid backward compatibility issues for existing SoC flow. SpyGlass CDC validates abstract port and therefore consider it as CDC start point rather than as CDC end point.

[ `-end`> ]

Set this argument to validate the domain assumption and other CDC attributes applied on the design object by using the `abstract_port` constraint from its fan-out cone.

Note that CDC verification is performed from the design object, which has

the `abstract_port` constraint specified, to the input cone of the specified port/pin.

**NOTE:** *This argument is used by the SpyGlass CDC solution only.*

Table 3 summarizes verification/validation with regard to CDC end points.

**TABLE 3**

	CDC end point
Verification	Primary output/Blackbox input
Validation	Primary input/Blackbox output

If the `-end` argument is not specified for an `abstract_port`, SpyGlass CDC considers the design objects specified in Table 4 as the end point.

#### **[`-direction` ]**

Set this argument to specify direction of an inout port that is constrained by using the `abstract_port` constraint to consider it as an input port or output port. Possible values for this field are `input` and `output`.

#### **[`-comment` ]**

Indicates if the input port is hanging, blocked, or the related clock is unconstrained.

The Setup\_port01 rule of the SpyGlass CDC solution generates the `abstract_port -comment` constraint if the input port is purely hanging or blocked. Possible values of the `-comment` field are `hanging path`, `blocked path`, and `unconstrained clock`. The `-comment` field is generated along with the `-ignore` field of the `abstract_port` constraint.

#### **[`-init` ]**

Indicates the initial state of the delay element to be applied in formal modeling.



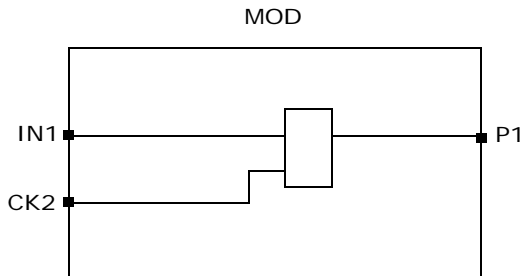
## Examples

### Example 1

Consider the following example:

```
abstract_port -module MOD -ports P1 -clock Ck2
-related_ports IN1 -scope cdc
```

The above example implies that the P1 output port of the MOD module is driven by a flip-flop clocked by the Ck2 clock, and there is no synchronizer in an input cone of the port. The flip-flop is driven by the IN1 input port. This is shown in the following figure:



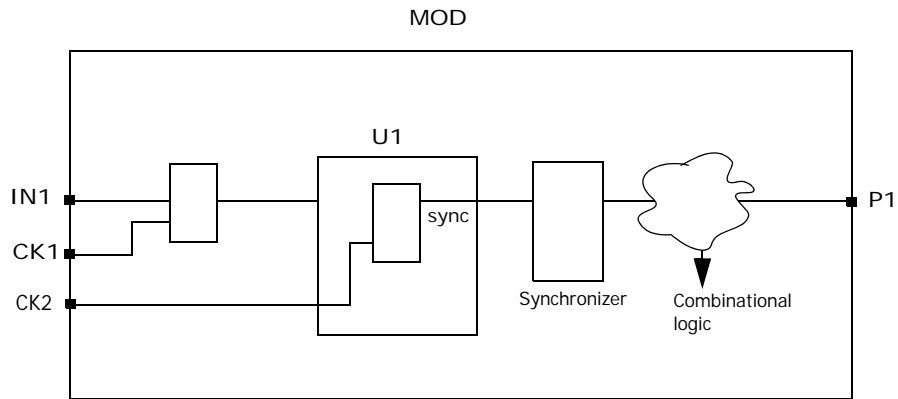
**FIGURE 10.** Example 1

### Example 2

Consider the following example:

```
abstract_port -module MOD -port P1 -clock Ck2
-combo yes -sync active -from Ck1 -to Ck2
-sync_names MOD.U1.sync -scope cdc
```

The above example implies that the P1 port of the MOD module is an output of a control synchronizer (from the source clock Ck1 to the destination clock Ck2) and there is a combinational logic present between a synchronizer and the port. This is shown in the following figure:



**FIGURE 11.** Example 2

Here, `MOD.U1.sync` is the hierarchical name of synchronizer output.

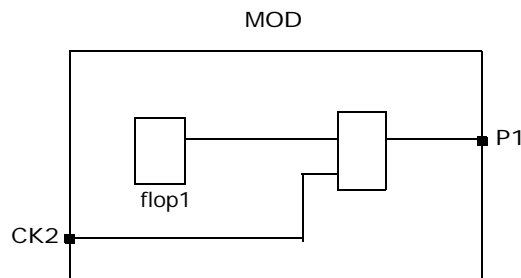
In this case, the port can act as a synchronized enable at a higher hierarchy for other crossings.

### Example 3

Consider the following example:

```
abstract_port -module MOD -ports P1 -clock Ck2 -scope cdc
```

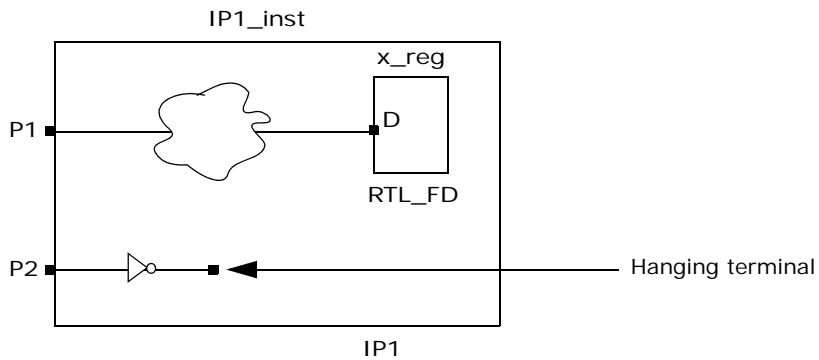
The above example implies that the P1 output port of the MOD module is driven by an unsynchronized crossing with the destination clock Ck2. This is shown in the following figure:



**FIGURE 12.** Example 3

**Example 4**

Consider the following figure:



**FIGURE 13.**

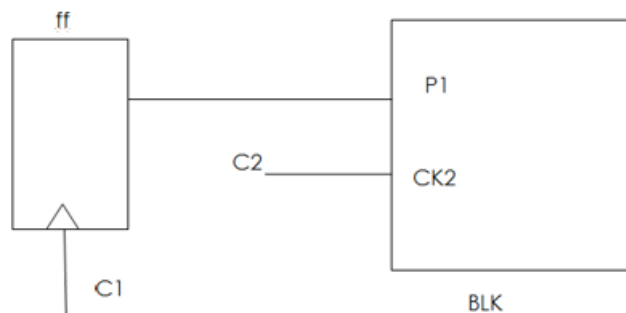
The following constraint, indicating that all the fan-out of the P2 input port are hanging, is generated by SpyGlass in this case:

```
abstract_port -module IP1 -ports P2 -ignore
```

**Example 5**

Consider the following example:

```
abstract_port -ports P1 -clock CK2 -start
```



**FIGURE 14.**

In the above example, the validation check is performed for port P1 and data domain mismatch is reported by the Ac\_abstract\_validation01 rule.

### Example 6

Consider the following example:

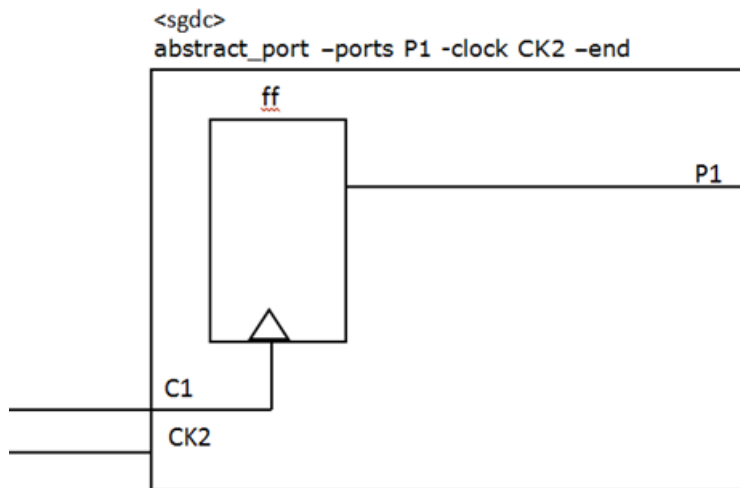


FIGURE 15.

In the above example, the Ac\_unsync01 rule reports a violation between the ff flop and P1 port.

**NOTE:** Note that no validation check is performed for output ports when the block is instantiated at the SoC level.

### Example 7

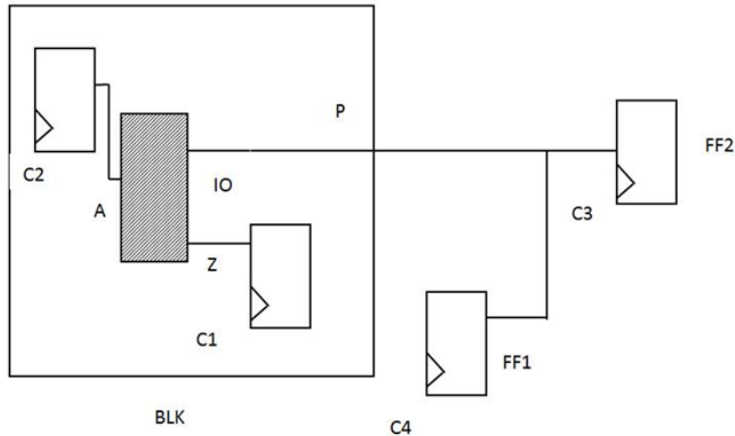
Consider a scenario where two constraints, one with -start and another with -end is defined for an inout port.

In this case, if the inout port is driven by a flop, then both validation and verification is performed (assuming the rules are enabled) with regard to -start and -end respectively.

Similarly if the inout port is feeding a flop, then both validation and

verification is performed because the `-start/-end` arguments are specified on the same port. Therefore, it is possible that SpyGlass CDC reports three violations for the same port because validation check is not performed for the `-end` argument on output or inout ports.

Consider the following schematic:



**FIGURE 16.**

In the above example, the P port is an inout port of the BLK block. The abstraction of the BLK block generates the following constraints:

```
abstract_port -ports P -clock C2 -start
```

At SoC level, the validation check is performed for the `abstract_port` constraint with the `-start` argument and it reports domain mismatch for the C4 clock because the FF1 flop is driving the P inout port. Similarly, a crossing is reported for C3 (where the P port is the source of the crossing) because the FF2 flop is driven by the P port.

### Example 8

Refer to the schematic shown in [Figure 17](#). The figure shows the various output ports and the corresponding `abstract_port` constraint generated by SpyGlass CDC for each output port.

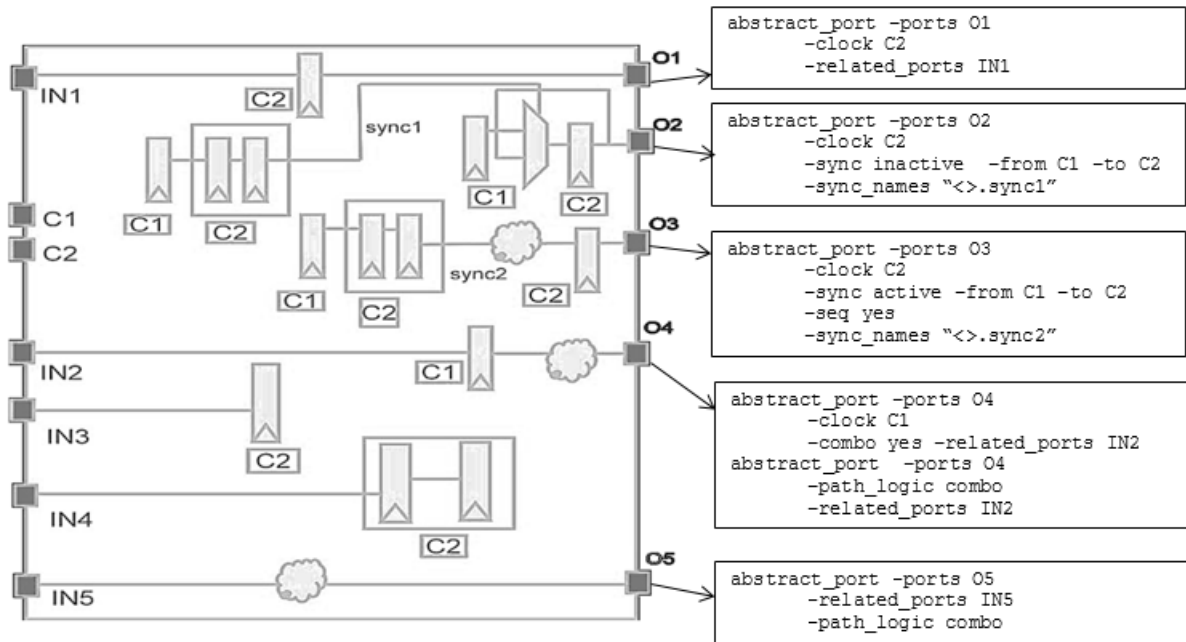


FIGURE 17.

**Example 9**

Refer to the schematic shown in [Figure 18](#). The figure shows the various input ports and the corresponding `abstract_port` constraint that you can specify for each input port.

## SpyGlass Design Constraints

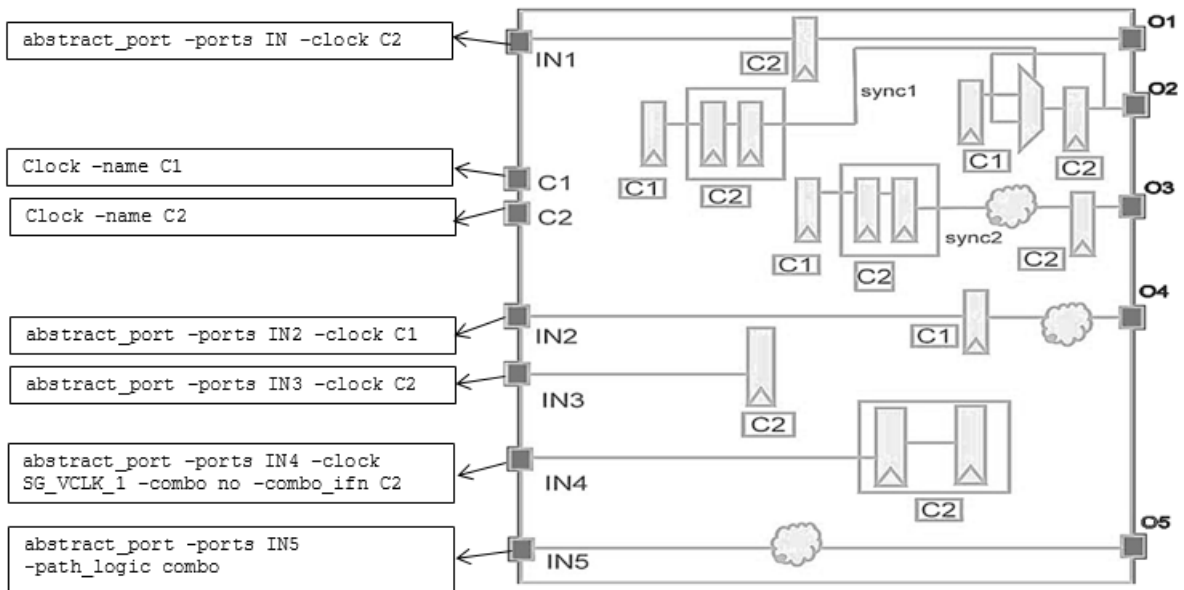


FIGURE 18.

## Rules

The `abstract_port` constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>		
Clock synchronization rules (except Clock_sync05)	Block constraint validation rules	Ac_abstract01
Ac_blksgdc01	Reset_sync02	Reset_sync03
Ar_unsync01	Ar_sync01	Ar_asyncdeassert01
Ar_syncdeassert01	Ar_resetcross01	Ar_resetcross_matrix01
<b>SpyGlass DFT Solution</b>		
All rules		
<b>SpyGlass DFT DSM Solution</b>		
All rules		

## activity

### Purpose

Use the `activity` constraint to specify activity values and probability values of input signal used for the rules that report power/activity. The activity value for an individual non-clock signal or all non-clock signals contained in a specified instance can be specified.

The ways in which activity and probability of a net can be determined, in decreasing order of priority, are as follows:

1. The simulation file
2. The `-name` and `-instname` arguments of the activity constraint
3. The `-all_primary_input`, `-all_register_output`, `-all_blackbox_output`, `-all_register_enable`, or `-all_power_essential_signals` arguments of the activity constraint
4. The `-default_startpoints` argument of the activity constraint
5. The `-default` argument of the activity constraint

**NOTE:** *The `pe_activity_priority` parameter is by default set to 'sim'. If it is changed to 'sgdc', points 2 and 3 take higher priority over point 1. For more information on this parameter, refer to the SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide.*

Nets for which activity and probability are not specified by any of the above points, the values are propagated from the fan-in using the SpyGlass activity propagation engine.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the activity constraint is as follows:

```
current_design <du-name>
  activity
    -name <sig-name> | -instname <inst-name> | -
default_startpoints | -default
  -prob <p-value>
```



```
-activity <a-value>
-all_primary_input
-all_register_output
-all_blackbox_output
-clock <clk-name>
-clock_enable_instname <flip-flop-inst-name>
-all_register_enable
-all_power_essential_signals
```

## Arguments

### <du-name>

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

### -name <sig-name>

Name of the non-clock signal for which you are specifying the activity value.

### -instname <inst-name>

Name of the instance under which you are specifying the activity value for all non-clock signals. This field supports regular expressions.

The `-instname` field supports a hierarchical instance name and NOT leaf-level cell name or black boxes.

### -default\_startpoints

Indicates that the specified probability and activity values are applicable to all unannotated primary inputs and blackbox outputs.

When you specify this argument:

- You can specify only the `-activity` and `-prob` arguments.
- The `-instname` argument is optional because SpyGlass automatically takes the top of the design. If you specify the `-instname` argument, make sure the value is the name of the top of the design. Otherwise, a FATAL message is reported.

**-default**

Indicates the specified probability and activity values are applicable to all nets, which are non-clock nets, and for which these values have not been specified by:

- Any other argument of the `activity` constraint
- The simulation file

**-prob <p-value>**

Probability of a signal being high (a real value between zero and one).

**-activity <a-value>**

The toggle activity value (a real number representing the number of transitions from zero to one or from one to zero in a clock cycle).

**-all\_primary\_input**

When you specify this argument with `-instname` argument, the activity values are set for all the input/inout pins of the respective hierarchical instance. Otherwise, the activity values are set for all the primary input/inout pins of the top domain in the design.

**-all\_register\_output**

If this field is specified, it sets the activity values for all the nets driven by registers.

The scope of this field can be specified through the `-instname` field of `activity` constraint. If scope is not specified, it sets the activity values for all the nets driven by registers in the design. For example:

```
activity -instname top.a -activity 0.3 -prob 0.5  
-all_register_output
```

The above example sets the activity values of the nets driven by registers in the `top.a` hierarchy.

**-all\_blackbox\_output**

If this field is specified, it sets the activity values for all the nets that are driven by black boxes.

The scope of this field can be specified through the `-instname` field of `activity` constraint. For example:

```
activity -instname top.a -activity 0.3 -prob 0.5
-all_blackbox_output
```

The above example sets the activity values of the nets driven by black boxes in the `top.a` hierarchy.

If the scope is not specified, it sets the activity values for all the nets that are driven by the black boxes in the design.

### **-clock <clk-name>**

In general, all the activity values are considered with respect to the fastest clock in the design. However, if you want to specify the activity of a net with respect to a local clock of a block, use this argument.

Consider the following example:

```
clock -name "clk1" -period 10
activity -instname "Top" -activity 0.5 -prob 0.3
clock -name "INST1.clk2" -period 20
activity -name "INST1.n1" -activity 0.5 \
-clock INST1.clk2 -prob 0.4
```

Here, activity value for the `n1` net of the `INST1` block is considered with respect to the local clock `INST1.clk2` rather than `clk1`, which is the fastest clock in the design. The last command of the above specification is equivalent to the following:

```
activity -name "INST1.n1" -prob 0.40 -activity 0.25
```

The activity value for the `n1` net of the `INST1` block remains the same as it is considered with respect to `clk1` clock.

### **-clock\_enable\_instname <flip-flop-inst-name>**

Name of the flip-flop instance on whose enable you want to apply an activity value or a probability value.

For example, consider the following specification of the `activity` constraint:

```
activity -clock_enable_instname FF1 -activity 0.5 -prob 0.3
```

The above specification implies that the activity value 0.5 and the probability value 0.3 should be applied on the enable of the FF1 flip-flop instance.

### **-all\_register\_enable**

Sets the activity and probability values for all register enables.

The scope of this argument can be specified through the `-instname` argument of the `activity` constraint. The scope includes the module and all the underlying instances within the module. If the scope is not specified, it sets the activity and probability for all register enables in the design.

In the following example, activity and probability values are set:

```
activity -instname top.M1 -prob 0.9 -activity 0.1
-all_register_enable
activity -instname top.M2 -prob 0.9 -activity 0.1
-all_register_enable
```

### **-all\_power\_essential\_signals**

Sets the activity and probability values for all essential signals.

The scope of this argument can be specified through the `-instname` argument of the `activity` constraint. The scope includes the module and all the underlying instances within the module. If the scope is not specified, it sets the activity and probability for all Power-Essential signals in the design.

In the following example, the activity values for all Power-Essential signals of module M1 have the probability of 0.9 and activity of 0.1:

```
activity -instname top.M1 -prob 0.9 -activity 0.1
-all_power_essential_signals
```

## **Rules**

The `activity` constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	PESTR03	PESTR05
PESTR06	PESTR13	poweraudit	

## activity\_data

### Purpose

Specifies a simulation file for rules in the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

The `activity_data` constraint specifies the VCD, FSDB, and SAIF file that is to be translated to SpyGlass activity format used by rules in the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

The `activity_data` constraint takes a set of information that was previously supported by some `spyParameters`. However, some new features like SAIF file input and multiple VCD are not supported by these `spyParameters`. Therefore, it is recommended that you use the `activity_data` constraint for all simulation inputs.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `activity_data` constraint is as follows:

```
current_design <du-name>
  activity_data
    -format <format> -file <file-name>
    -starttime <start-time>
    -endtime <end-time>
    -weight <value> -sim_topname <simulation top>
    -instname <instance-name>
    -sim_rtl_design_nl
    [-parallel_saif <saif-file-name>]
    [-parallel_saif_topname <saif-top-name>]
```

### Arguments

**<du-name>**

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**-format <format>**

Name of the activity data format.

This argument can have any of the following values: "SAIF", "VCD", or "FSDB", depending on the type of the specified simulation file.

**-file <file-name>**

Name of the VCD/FSDB/SAIF file.

**NOTE:** *You can specify compressed VCD file that has been generated by using the gzip utility.*

**-starttime <start-time>**

(Optional) Start time for VCD/FSDB file analysis with a time unit. The allowed time units are s, ms, us, ns, ps, and fs. In case, you do not specify a time unit, the timescale value provided in the VCD/FSDB file is used to infer the time value.

For example, if you specify the <start-time> as three with no time unit, and the corresponding VCD file is as shown below:

```
#####
$date
    Dec 18, 2009 17:48:09
$end
$timescale
    1ns
$end
#####
```

Then, the time value will be inferred as 3ns.

If you do not specify this argument, the start time given in VCD/FSDB file is used for analysis.

**-endtime <end-time>**

(Optional) End time for VCD/FSDB file analysis with a time unit. If you do not specify a time unit, the timescale value provided in the VCD/FSDB file

is used to infer the time value.

If you do not specify the `<end-time>` argument, the end time given in VCD/FSDB file is used for analysis.

#### **-weight <value>**

The percentage value (an integer number between 0 and 100) denoting weights assigned to the multiple VCD/FSDB files.

When using the `<value>` argument, you should specify weights for all the VCD/FSDB files. In case you do not specify this argument, the weights are considered based on simulation times of each VCD/FSDB file.

The sum of the `<value>` arguments specified for all the `activity_data` constraints should be 100.

If multiple VCD/FSDB files are specified and the weight is not specified, the average power estimation is done based on the weighted average of the duration of each VCD/FSDB file. For example:

If VCD1 has simulation values from 0 to 500 ns and VCD2 has simulation values from 200 ns to 450 ns, the average power is calculated, assuming that in a complete duration of

$(500 \text{ ns} - 0) + (450 \text{ ns} - 200 \text{ ns}) = 750 \text{ ns}$ ,  
VCD1 runs for 500 ns and VCD2 runs for 250 ns.

Therefore, the weight of VCD1:VCD2 will be 2:1.

However, if weights are specified for VCD/FSDB files, the duration of VCD/FSDB file is not considered while taking weighted average for power estimation.

#### **-sim\_topname <simulation top>**

Name of the instantiated top module in the VCD/FSDB/SAIF file.

If you do not specify `<simulation top>`, SpyGlass attempts to automatically infer the hierarchy in the VCD/FSDB/SAIF file corresponding to the top design.

#### **-instname <instance-name>**

(Optional) Name of the hierarchical instance for which the simulation file is applicable. This field is specified when horizontal simulation file flow is used.



## SpyGlass Design Constraints

**-sim\_rtl\_design\_n1**

(Optional) This field should be specified if simulation file is used with gate level design.

**-parallel\_saif <saif-file-name>**

This is an optional argument and is used to specify the Switching Activity Interchange Format (SAIF) file name with the simulation file formats VCD/FSDB.

During the annotation process, if any signal is not found in the VCD/FSDB files, that signal's activity is retrieved from the parallel SAIF when the signal is present in the parallel SAIF file.

**-parallel\_saif\_topname <saif-top-name>**

This is an optional argument and is used to specify the parallel SAIF top name. If you do not specify the parallel SAIF top name, the -sim\_topname argument is considered as the parallel SAIF top name.

**Rules**

The `activity_data` constraint is used by the following rules in the SpyGlass Power Estimation and SpyGlass Power Reduction solutions:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	PESAE02	PESAE03
poweraudit	PESTR03	PESTR05	PESTR06
PESTR13	PESAE04	PESAE06	

## add\_fault

### Purpose

The `add_fault` constraint is used to specify faults to be considered while calculating fault/test coverage. When `add_fault` is used then all faults that are not specified as `add_fault` are treated as `no_fault` and ignored while calculating fault/test coverage.

### Product

SpyGlass DFT solution

### Syntax

The syntax to specify the `add_fault` constraint is as follows:

```
add_fault
  -name <du-name> | <inst-name>
  [- fault <hier_pin_names> ]
  [- net <hier_net_names> ]
  [- net_input <hier_net_names> ]
  [- net_output <hier_net_names> ]
  [- clock_control <hier_net_names> ]
  [- set_control <hier_net_names> ]
  [- reset_control <hier_net_names> ]
  [- register_suffix ]
  [- instance_suffix ]
  [- module_suffix ]
```

**NOTE:** *The `add_fault` constraint supports wildcard characters. Using wildcards, expression is expanded only within the hierarchy.*

### Arguments

The `add_fault` constraint has the following arguments:

**-name <du-name> | <inst-name>**

Specifies name of a module or instance that needs to be marked as `add_fault`.

You can specify a single or space-separated list of design unit names / hierarchical instance names:

You can specify design unit names, hierarchical instance names, or a combination of both.

**-fault <hier\_pin\_names>**

(Optional) Space-separated list of hierarchical names of pins or ports.

Do not use this argument in case of RTL design because pin names will contain generated names and will fail SGDC sanity check at the RTL.

**-net <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults in the direct fanin or fanout of the net as add\_fault.

**-net\_input <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults in the direct fanin of the net as add\_fault.

**-net\_output <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults in the direct fanout of the net as add\_fault.

**-clock\_control <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults associated with the registers, where clock pin is topologically driven by the specified clock, as add\_fault.

**-set\_control <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the fault associated with the registers, where set pin is topologically driven by the specified set signal, as add\_fault.

**-reset\_control <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the fault associated with the registers, where reset pin is topologically driven by the specified reset signal, as add\_fault.

**-register\_suffix <suffixes>**

Space-separated list of suffixes to be specified as `add_fault`. The `-register_suffix` argument should not be used along with other arguments of the `add_fault` constraint, that is, `-name`, `-clock_control`, `-set_control`, or `-reset_control`.

If the value of the `dft_treat_suffix_as_pattern` 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.

If the value of the `dft_check_path_name_for_register_suffix` parameter is on, the value of the `-register_suffix` field will be matched with the register name along with the path in which the register is present.

**-instance\_suffix <suffixes>**

Define this field to use suffix based pattern match for all instance names.

**-module\_suffix <suffixes>**

Define this field to use suffix based pattern match for all module names.

If the value of the `dft_treat_suffix_as_pattern` parameter is on, the value of the `-module_suffix` field will be matched with the module name along with the path in which the module is present.

**Rules**

The `add_fault` constraint is used by the following rules:

**SpyGlass DFT Solution**

Info_coverage	Coverage_audit
---------------	----------------

**SpyGlass DFT DSM Solution**

Info_transitionCoverage	Info_transitionCoverage_audit
-------------------------	-------------------------------

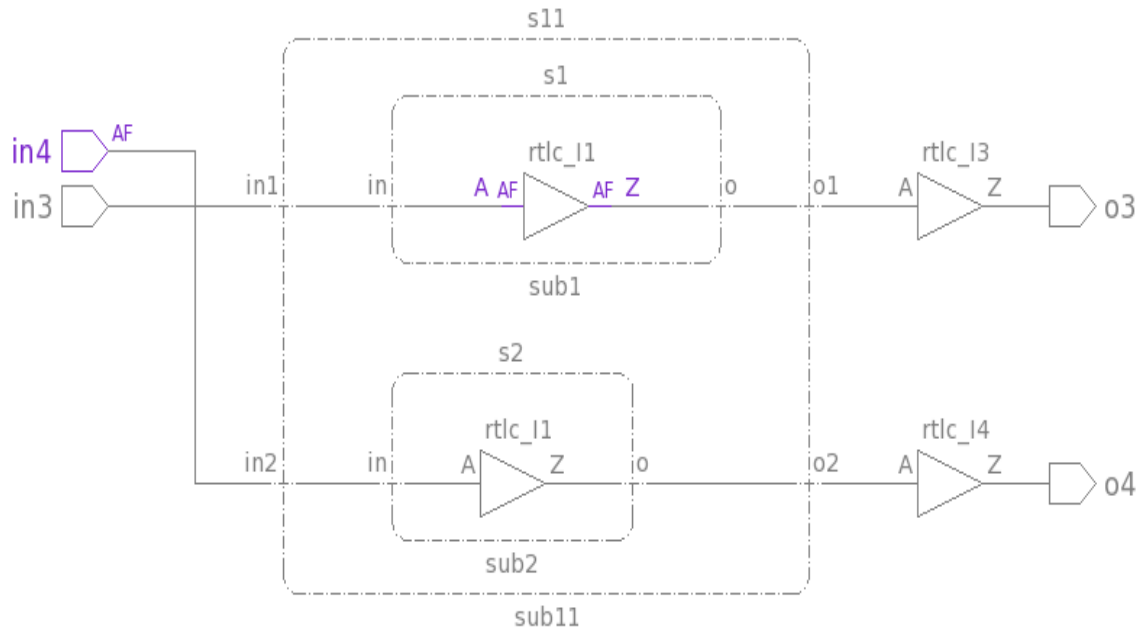
**Example****Example 1**

Consider the following `add_fault` definition:

## SpyGlass Design Constraints

```
add_fault -name sub1 -fault top.in4
add_fault constraint definition:
```

Now, consider the schematic for the same (*Figure 19*):



**FIGURE 19.** Terminals marked as add\_fault

In the above schematic, instance, in4, and terminals inside sub1 are marked as add\_fault. The faults of other terminals are not used for coverage calculations.

### Example 2: Specifying list of suffixes using the -register\_suffix argument

Consider the following example:

```
R1 (register 1) name: top.u_ctrl.u2.u1.ff1_ctrl
R2 (register 2) name: top.u_ctrl.u2.u1.ff1_state
R3 (register 3) name: top.u_core.u2.u1.ff1_state_ctrl
```

R4 (register 4) name: top.u\_ctrl\_state.u2.u1.ff1\_ctrl\_state

Now, consider the following *add\_fault* descriptions:

```
add_fault -register_suffix ctrl
add_fault -register_suffix state
```

The following table lists the results when combination of values are used for the *dft\_treat\_suffix\_as\_pattern* and *dft\_check\_path\_name\_for\_register\_suffix* parameters:

**TABLE 4** Pattern Matching for the -register\_suffix argument

Value of <i>dft_treat_suffix_as_pattern</i>	Value of <i>dft_check_path_name_for_register_suffix</i>	Value of -register_suffix	Matched Registers
off	off	ctrl	R1, R3
		state	R2, R4
off	on	ctrl	R1, R2, R3
		state	R2, R4
on	off	ctrl	R1, R3, R4
		state	R2, R3, R4
on	on	ctrl	R1, R2, R3, R4
		state	R2, R3, R4

### Example 3: Specifying list of suffixes using the -instance\_suffix argument

Consider the following example:

```
R1 (register 1) name: top.u_ctrl.u2.u1.ff1_ctrl
R2 (register 2) name: top.u_ctrl.u2.u1.ff1_state
R3 (register 3) name: top.u_core.u2.u1.ff1_state_ctrl
R4 (register 4) name: top.u_ctrl_state.u2.u1.ff1_ctrl_state

I1 (instance 1) name: top.u_ctrl.u2.u1.inst1_ctrl
I2 (instance 2) name: top.u_ctrl.u2.u1.inst1_state
I3 (instance 3) name: top.u_core.u2.u1.inst1_state_ctrl
I4 (instance 4) name: top.u_ctrl_state.u2.u1.inst1_ctrl_state
```

Now, consider the following *add\_fault* descriptions:

```
add_fault -instance_suffix ctrl
add_fault -instance_suffix state
```

The following table lists the results when combination of values are used for the *dft\_treat\_suffix\_as\_pattern* and *dft\_check\_path\_name\_for\_instance\_suffix* parameters:

**TABLE 5** Pattern Matching for the -instance argument

Value of <i>dft_treat_suffix_as_pattern</i>	Value of <i>dft_check_path_name_for_instance_suffix</i>	Value of - <i>instance_suffix</i>	Matched Registers/Instances
off	off	ctrl	R1, R3, I1, I3
		state	R2, R4, I2, I4
off	on	ctrl	R1, R2, R3, I1, I2, I3
		state	R2, R4, I2, I4
on	off	ctrl	R1, R3, R4, I1, I3, I4
		state	R2, R3, R4, I2, I3, I4
on	on	ctrl	R1, R2, R3, R4, I1, I2, I3, I4
		state	R2, R3, R4, I2, I3, I4



## allow\_combo\_logic

### Purpose

The `allow_combo_logic` constraint allows combinational logic between crossings only if the logic is within the modules specified using this constraint.

**NOTE:** *The `allow_combo_logic` constraint specifications will be applicable to all the clock-domain crossings in the design.*

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `allow_combo_logic` constraint is as follows:

```
current_design <du-name>
allow_combo_logic
  [ -name <space-separated-list> ]
  [ -all ]
  [ -none ]
```

### Arguments

#### **-name <space-separated-list>**

(Optional) Specifies a space-separated list of modules and/or library cells. You can also use wildcard characters while specifying module/library cell names. For example, you can specify names, as shown in the following examples:

```
allow_combo_logic -name MD1 MD2 MD3
allow_combo_logic -name "MD*"
allow_combo_logic -name "MD*" "AB*" PQR
```

#### **-all**

(Optional) Specifies that all the combinational logic should be allowed in a crossing path.

**-none**

(Optional) Specifies that no combinational logic should be allowed in the crossing path.

If the `allow_combo_logic` constraint is specified multiple times, the order of preference (starting from the highest priority) will be as follows:

1. `allow_combo_logic -all`
2. `allow_combo_logic -none`
3. `allow_combo_logic -name <list>`

**NOTE:** Do not use the `-all` and `-none` arguments together in the same `allow_combo_logic` constraint.

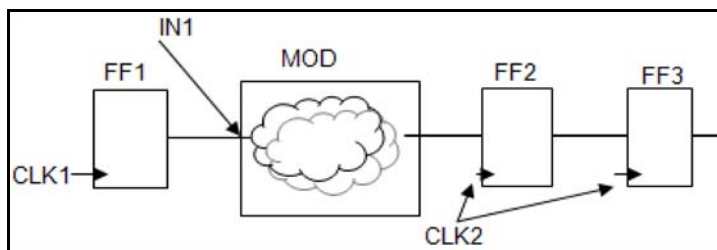
**Rules**

The `allow_combo_logic` constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_sync01	Ac_sync02	Ac_unsync01	Ac_unsync02
Ac_glitch03	Ac_conv01	Ac_conv02	Ac_conv03
Clock_sync09			

**Example**

Consider an example, as shown in the following figure:



**FIGURE 20.** Ac\_unsync01/Ac\_unsync02 Rule Example

In the above example, by default, the `Ac_unsync01/Ac_unsync02` rule is

reported for the crossing between FF1 and FF2. Now consider that you specify the `allow_combo_logic` constraint, as given below:

```
allow_combo_logic -name MOD
```

In the above case, the combinational logic will be allowed if it is inside the module, MOD, between the crossing path. Therefore, the crossing will be reported as synchronized by the *Ac\_sync01/Ac\_sync02* rule.

## allow\_test\_point

### Purpose

The `allow_test_point` constraint is used to specify modules or instance which should be considered for suggesting test points.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `allow_test_point` constraint is as follows:

```
current_design <du-name>
allow_test_point
  -name <module-name | instance_list>
```

### Arguments

**-name <module-name | instance\_list>**

Name of the module or the list of instances to be considered for suggesting the test points.

### Rules

The `allow_test_point` constraint is used by the `Info_random_resistance` rule.

## always\_on\_buffer

### Purpose

The `always_on_buffer` constraint is used to specify always-on buffers.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `aonbuffer`.

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `always_on_buffer` constraint is as follows:

```
current_design <du-name>
  always_on_buffer
    -name <cell-name>
    [ -vddcpin <vddc-pin> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the always-on buffer.

### -name <cell-name>

Name of the buffer cell. You can use wildcard characters while specifying the cell name.

### -vddcpin <vddc-pin>

Name of the VDDC pin of the always-on buffer as used by LPPLIB11 and LPPLIB06 rules of the SpyGlass Power Verify solution.

## Rules

The `always_on_buffer` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM08	LPSVM09	LPSVM10	LPSVM26
LPSVM28	LPSVM31	LPSVM40	LPSVM47
LPSVM48	LPSVM52	LPSVM53	LPSVM56
LPSVM57	LPSVM59	LPPLIB06	LPPLIB11
LPPLIB15	LPPLIB17		

## always\_on\_cell

### Purpose

The `always_on_cell` constraint is used to specify library cell names that should be instantiated in always-on domains only (that is, they should not be instantiated in switchable domains).

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `always_on_cell` constraint is as follows:

```
current_design <du-name>
  always_on_cell
    -name <cell-name-list>
    [ -locate <domain-type> ]
```

### Arguments

#### **<du-name>**

Name of the design unit under which you are specifying the always-on cells.

#### **-name <cell-name-list>**

Space-separated name list of always-on cells.

#### **-locate <domain-type>**

It can be AON (Always-On Domain), OFF (power domain), or BOTH (any domain).

**NOTE:** *You can use wildcard characters while specifying the cell names using the `-name` argument.*

### Rules

The `always_on_cell` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM09	LPSVM10	LPSVM28	LPSVM31
LPSVM47	LPSVM54	LPSVM57	LPSVM59
LPPLIB17			

## always\_on\_pin

### Purpose

The `always_on_pin` constraint is used to specify always-on cell pins.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `always_on_pin` constraint is as follows:

```
current_design <du-name>
  always_on_pin
    -cell <name>
    -pin <pin-name-list>
```

### Arguments

#### <du-name>

Name of the design unit under which you are specifying the always-on pins.

#### -cell <name>

Name of the design unit for which always-on pins are being specified.

#### -pin <pin-name-list>

Space-separated name list of always-on input/inout pins of the design unit <name>.

**NOTE:** *You can use wildcard characters while specifying the cell name using the `-cell` argument.*

## Rules

The `always_on_pin` constraint is used by the following rule.

---

**SpyGlass Power Verify Solution**

---

LPSVM53

---

## always\_on\_path

### Purpose

The `always_on_path` constraint is used to specify paths that should be always-on. There should not be any element that is driven by a switchable supply along the paths specified as always-on path.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `always_on_path` constraint is as follows:

```
current_design <du-name>
  always_on_path
    -from <start-point-name>
    -to <end-point-name>
```

### Arguments

#### <du-name>

Name of the design unit under which you are specifying the always-on cells.

#### -from <start-point-name>

Specifies the start point of `always_on_path`.



**-to <end-point-name>**

Specifies the end point of `always_on_path`.

**NOTE:** Here, <start-point-name>/<end-point-name> can be:

- ▢ *Top-level port (like TOP.in)*
- ▢ *Any hierarchical signal name (like TOP.u1.sig1)*
- ▢ *Pin of a hierarchical leaf level instance (like TOP.u1.LIB1.Z)*
- ▢ *Any hierarchical port of user defined module (like TOP.u1.clk\_out)*

**Rules**

The `always_on_pin` constraint is used by the following rule.

---

**SpyGlass Power Verify Solution**


---

LPAON02

---

**antenna\_cell****Purpose**

The `antenna_cell` constraint is used to specify the antennae protection cells (diode cells) that should be ignored.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `apcell`.

**Product**

SpyGlass Power Verify solution

**Syntax**

The syntax to specify the `antenna_cell` constraint is as follows:

```
current_design <du-name>
  antenna_cell
    -name <cell-name-list>
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the cells.

**-name <cell-name-list>**

Space-separated list of the cell names. You can use wildcard characters while specifying the cell name using the `-name` argument.

## Rules

The `antenna_cell` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM04A	LPSVM04B	LPSVM04C	LPSVM08
LPSVM09	LPSVM10		

## aon\_buffered\_signals

### Purpose

The `aon_buffered_signals` constraint is used to specify signals that should be driven by an always-on buffer.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `aonbufferedsignals`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `aon_buffered_signals` constraint is as follows:

```
current_design <du-name>
  aon_buffered_signals
```

```

-names <sig-name-list>
[ -terminatingcells <cell-pin-name-list> ]
[ -ignorecells <ignorecell-name-list> ]

```

## Arguments

### <du-name>

Name of the design unit under which you are specifying signals.

### -names <sig-name-list>

Space-separated name list of signals driven by an always-on buffer (specified by using the *always\_on\_buffer* constraint).

### -terminatingcells <cell-pin-name-list>

Space-separated cell-pin pair names list of terminating cells and their terminating pins.

**NOTE:** *You can use wildcard expressions while specifying this list.*

### -ignorecells <ignorecell-name-list>

Space-separated name list of cells to be ignored while traversing the fan-out of the specified signals. You can use wildcard characters while specifying cells to be ignored.

## Rules

The `aon_buffered_signals` constraint is used by the following rules:

SpyGlass Power Verify Solution	
LPSVM40	LPPLIB11

## assertion\_signal

### Purpose

Specifies Power On Reset (POR) signals and low power signals.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `lpsignal`.

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `assertion_signal` constraint is as follows:

```
current_design <du-name>
  assertion_signal
    -name <sig-name>
    -value <0 | 1>
    -type <POR | PORCHECK>
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the special signals.

**-name <sig-name>**

Name of the special signal.

**-value <0 | 1>**

The `-value` argument specifies whether the signal is active high (1) or active low (0).

**-type <POR | PORCHECK>**

Specify the `-type` argument with value `POR` for POR signals and with value `PORCHECK` for low-power signals.

## Rules

The `assertion_signal` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

LPSVM42

---

## associate\_lib

### Purpose

The `associate_lib` constraint is used to associate the library names and library cells with the domains declared in the UPF.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `associate_lib` constraint is as follows:

```
associate_lib
  -domain <domain-name>
  -lib <list of library-name>
  -cell <list of cell-name>
```

### Arguments

**-domain <domain-name>**

Name of a domain as declared in the UPF.

**-lib <list of library-name>**

List of one or more library names as defined in the `.lib` file with keyword:

```
library(library-name){
}
```

**-cell <list of cell-name>**

List of one or more cell names as defined in the `.lib` file with keyword:

```
cell (cell-name){
}
```

**NOTE:** *Wildcards are supported for cell names. Both '-lib' and '-cell' fields are optional and can be specified together. The -domain argument is mandatory.*

## Rules

The `associate_lib` constraint is used by the following rule:

SpyGlass Power Verify Solution	
LPPLIB20	SGDC_lowpower118

## assume\_waveform

### Purpose

The `assume_waveform` constraint is used to specify a user-defined waveform on a design object. This waveform is then used during the formal verification of false path (`set_false_path`) and multicycle path (`set_multicycle_path`) constraints in SpyGlass TXV.

You can specify the waveform in terms of edge list corresponding to a particular SDC `create_clock` or `create_generated_clock`.

### Product

SpyGlass TXV solution

### Syntax

The syntax of the `assume_waveform` keyword in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>
assume_waveform
    -clock <sdc-clock-name>
    -edgelist <list-of-edges>
    -object <port | pin | net>
```

### Arguments

**-clock <sdc-clock-name>**

The name of an SDC `create_clock`/`create_generated_clock` as

specified in the `-name` argument of the SDC `create_clock/`  
`create_generated_clock` command.

**-edgelist** <list-of-edges>

The waveform with reference to the SDC clock specified in the `-clock` argument.

**-object** <port-name | pin-name | net-name>

The design object where this waveform needs to be applied during verification of false path or multicycle path constraints. You can either specify a port, pin, or net name.

## Example

If the SDC clock is specified as:

```
create_clock -name clk1 -period 10 clk
```

The corresponding `assume_waveform` can be specified as:

```
assume_waveform
  -clock "clk1"
  -edge_list "{1 3 5}"
  -object "in1"
```

## Rules

The `assume_waveform` constraint is used by the following rules:

---

### SpyGlass TXV Solution

---

Txv_MCP01	Txv_FP01
-----------	----------

---

## assume\_path

## Purpose

The `assume_path` constraint is used to specify the paths that exist between the input pins and the output pins of black boxes.

**NOTE:** *The `assume_path` constraint will be deprecated in a future SpyGlass release. Use the [abstract\\_port](#) constraint instead of this constraint.*

**NOTE:** *Use this constraint if you want SpyGlass rules to consider combinational path from an input to output ports of a black box, whereas use the [abstract\\_port](#) constraint if you want to associate or specify a clock on an input or output port.*

## Product

SpyGlass CDC solution, SpyGlass Constraints solution, SpyGlass latch product, SpyGlass DFT solution

## Syntax

The syntax of using the `assume_path` keyword in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>
assume_path -name <bbdu-name>
            -input <input-pin-name>
            -output <output-pin-name>
```

## Arguments

**-name <bbdu-name>**

A black box module name (for Verilog designs) or a black box entity name (for VHDL designs).

For VHDL designs, this constraint is applied to all architectures of the specified entity.

**-input <input-pin-name>**

The name of an input pin of the black box design unit *<bbdu-name>*.

**-output <output-pin-name>**

A list of output pin names of the black box design unit *<bbdu-name>*.



## Examples

Consider the following example:

```
assume_path -name BBOX -input d -output q qbar
```

This constraint indicates that paths exist between input pin `d` and output pins `q` and `qbar` of black box design unit `BBOX`.

### For SpyGlass CDC Solution

For the SpyGlass CDC solution, clock domain propagation through black boxes is normally limited, because the exact relationship between input pins and output pins is not known. If only one clock signal is propagated to a black box instance, it is assumed that all pins of the black box instance are in the domain of that clock signal and further clock domain propagation is performed. However, if more than one clock signal propagates to the instance, it is not possible to assume any clock domain information.

One way to extend the clock domain propagation through a black box instance is to specify which output pins belong to the same clock domain as a particular input pin.

The `Ac_unsync01/Ac_unsync02` and `Ac_sync01/Ac_sync02` rules allow you to specify that paths exist between input pins and output pins of black boxes and, thus, clock domain propagation can traverse through these black boxes. These paths can be specified using the `assume_path` keyword in a SpyGlass Design Constraints file.

### For SpyGlass Constraints Solution

For the *SpyGlass Constraints solution*, `assume_path` constraint helps in determining different paths available, for a black box module, between the specified list of input pins and output pins.

## Rules

The `assume_path` constraint is used by the following rules:

---

#### SpyGlass CDC Solution

---

All rules

---

#### SpyGlass Constraints Solution

---

---

All rules

---

**SpyGlass latch Product**

---

LatchFeedback

---

**SpyGlass DFT Solution**

---

Clock\_04

---

## atspeed\_clock\_frequency

### Purpose

The `atspeed_clock_frequency` constraint is used to specify frequencies associated with a test clock. The `atspeed_clock_frequency` constraint might potentially affect the way SpyGlass infers at-speed clock domain.

**NOTE:** *Prior to the SpyGlass 4.4.0 release, the names of this constraint were `testclock_frequency` and `testclockFrequency`.*

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `atspeed_clock_frequency` constraint is as follows:

```
current_design <du-name>
  atspeed_clock_frequency
    -name <testclock-node>
    -freqList <freq-symbol-list>
    [-enables <assign-node-paths>
    -values <combination1 combination2>]
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the instance hierarchies.

**-name <testclock-node>**

Path to the test clock node. This node must be declared as a test clock.

**-freqList <freq-symbol-list>**

List of frequency symbols with which the test clock is associated. These can be actual numbers like 100, 200 or alphanumeric symbols like F1, f2, etc.

**-enables <assign-node-paths>**

(Optional) Paths to nodes that enable a particular frequency.

**-values <combination1 combination2>**

(Optional) Binary strings (0s and 1s) that specify a combination to be applied to the enable pins. The length of the combination should be equal to the number of enable pins.

## Examples

Consider the following example to specify frequencies (symbols/numeric) at nodes of interest that are declared as test clocks:

```
clock -name top.u2.clockout -testclock -atspeed
atspeed_clock_frequency -name top.u2.clockout -freqList 100
200 400 800 -enables s1 s2 s3 s4 -values 1000 0100 0010 0001
```

**NOTE:** Each symbol or value in the example has a corresponding set of enabling conditions.

You can infer the following from the example:

- The test clock can have frequencies: 100, 200, 400, and 800.
- These frequencies are selected by signals on the pins s1, s2, s3, and s4.
- Frequency 100 is produced when s1, s2, s3, & s4 are 1000.

- Frequency 200 is produced when s1, s2, s3, & s4 are 0100.
- Frequency 400 is produced when s1, s2, s3, & s4 are 0010.
- Frequency 800 is produced when s1, s2, s3, & s4 are 0001.

## Rules

The `atspeed_clock_frequency` constraint is used by the following rules.

---

**SpyGlass DFT DSM Solution**

---

All rules that use clock domain

---

## balanced\_clock

### Purpose

The `balanced_clock` constraint specifies points within the clock distribution system where all clocks fed from that point are to be considered in the same clock domain.

### Product

SpyGlass DFT solution

### Syntax

The syntax of using the `balanced_clock` keyword in a SpyGlass Design Constraints file is as follows:

```
balanced_clock
  -name <hier-pin-name-list>
  [ -domain <domain-name> ]
```

### Arguments

**-name <hier-pin-name-list>**

Hierarchical name of a clock pin.

The pin can be a primary pin as well as an internal pin.

You can specify a single hierarchical pin name or a space-separated list of hierarchical pin names. When you specify a clock pin list, all flip-flops triggered by these clock pins are assumed to be in the same clock domain by the rules that check for clock domains.

**-domain <domain-name>**

(Optional) Specifies the domain of the specified balanced clock.

The domain name can be any valid string.

When the domain name is not specified, a default domain name is created and all balanced clocks specified in one `balanced_clock` constraint are assumed to be in this domain.

When multiple `balanced_clock` constraints are used with the same domain name, all clocks specified in these `balanced_clock` constraints are assumed to be in this (common) domain.

Thus, the following specifications create four domains: `clkA`, `clkB`, `clkC`, and `clkD U clkE`:

```
balanced_clock -name clkA
balanced_clock -name clkB
balanced_clock -name clkC
balanced_clock -name clkD -domain big
balanced_clock -name clkE -domain big
```

## Examples

Consider the following example:

```
module top(...)
...
m1 u1 (clock, ...);
...
m2 u2 (... ,clock, ...);
...
endmodule
```

For example, when `top.u1.clock` is declared in a `balanced_clock` constraint, all flip-flops inside instance `u1` will be considered to be in the

same clock domain and not in the same clock domain as the flip-flops in instance u2.

Another example of the `balanced_clock` constraint is as follows:

```
balanced_clock -name top.inst1.clkPort
```

The above specification indicates that the `clkPort` pin of the instance `inst1` in the top-level design unit `top` is the balanced clock pin.

```
balanced_clock -name clk1
```

The above specification indicates that the `clk1` port of the top-level design unit is the balanced clock pin.

## Rules

The `balanced_clock` constraint is used by the following rules:

SpyGlass DFT Solution		
Clock03	Clock_10	Scan_22 (optional)

## blackbox\_power

### Purpose

The `blackbox_power` constraint is used for modeling the black boxes during power estimation.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `set_black_box_power`.*

Three models are supported for the black box modeling. Refer to the *Modeling Black Boxes in Power Estimation* section in the *SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide* for details.

Different arguments of this constraint are used for a different purpose for each model, as explained below:

- Refer to the Case 1 of the *Modeling Black Boxes in Power Estimation* section for the description of this scenario.

```
current_design <du-name>
  blackbox_power
    -modname <mod-name> | -instname <inst-name> }
    -leakage <lfloat> -internal <ifloat>
    -switching <sfloat>
    -input_pin_cap <ipcfloat>
```

Where:

- *<du-name>*: Module name (for Verilog designs) or design unit name in *<entity-name>.<arch-name>* format (for VHDL designs)
  - *<mod-name>*: Name of the black box module
  - *<inst-name>*: Hierarchical instance name of the black box
  - *<lfloat>*: Leakage power of the black box (in Watts)
  - *<ifloat>*: Internal power of the black box (specified in Watts)
  - *<sfloat>*: Switching power of the black box (specified in Watts)
  - *<ipcfloat>*: Average input pin capacitance of the each pin of the black box (in Farad).
- Refer to the Case 2 of the *Modeling Black Boxes in Power Estimation* section for the description of this scenario:

```
current_design <du-name>
  blackbox_power
    -modname <mod-name> | -instname <inst-name>
    -leakage <lfloat> -internal <ifloat_list>
    -switching <sfloat_list>
    -clocks <clk_pin_list>
    -input_pin_cap <ipcfloat>
```

Where:

- *<du-name>*: Module name (for Verilog designs) or design unit name in *<entity-name>.<arch-name>* format (for VHDL designs)
- *<mod-name>*: Name of the black box module

- ❑ `<inst-name>`: Hierarchical instance name of the black box
  - ❑ `<lfloat>`: Leakage power of the black box (in Watts)
  - ❑ `<ifloat_list>`: Space separated list of internal power in `Watts/Hz` contributed by the logic on clocks `<clk_pin_list>`
  - ❑ `<sfloat_list>`: Space separated list of Switching power in `Watts/Hz` contributed by the logic on the clock `<clk_pin_list>`
  - ❑ `<clk_pin_list>`: List of the clock pins in the black box
  - ❑ `<ipcfloat>`: Average input pin capacitance of the each pin of the black box (in Farad).
- Refer to the Case 3 of the *Modeling Black Boxes in Power Estimation* section for the description of this scenario:

```
current_design <du-name>
  blackbox_power
    -modname <mod-name> | -instname <inst-name>
    -leakage <lfloat> -internal <ifloat_list>
    -switching <sfloat_list>
    -input_pin_cap <ipcfloat> -activity <afloat>
    -clocks <clk_pin_list>
    -equiv_nand2_count <gate-count>
    -register_count <reg-count>
```

Where:

- ❑ `<du-name>`: Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs)
- ❑ `<mod-name>`: Name of the black box module
- ❑ `<inst-name>`: Hierarchical instance name of the black box
- ❑ `<lfloat>`: Leakage power of the black box (in Watts)
- ❑ `<ifloat-list>`: Space separated list of percentage of internal power contributed by the logic on the clocks `<clk_pin_list>`
- ❑ `<sfloat_list>`: Space separated list of percentage of switching power contributed by the logic on the clocks `<clk_pin_list>`
- ❑ `<ipcfloat>`: Average input pin capacitance of the each pin of the black box (in Farad)



- ❑ *<afloat>*: Average activity of the black box
- ❑ *<clk\_pin\_list>*: List of the clock pins in the black box
- ❑ *<gate-count>*: NAND gate equivalent for combinational gates of the black box
- ❑ *<reg-count>*: Register count of the black box.

## Rules

The `blackbox_power` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction solutions	
PEPWR01	PEPWR02

## block

### Purpose

The `block` constraint is used to specify the blocks (design units) under the `current_design` unit.

### Product

SpyGlass Constraints solution

### Syntax

The syntax of the `block` constraint is as follows:

```
current_design <du_name>
block -name <block1-name> <block2-name>
```

or

```
current_design <top_design_name>
block -name <block1-name>
block -name <block2-name>
```

## Arguments

**<du\_name>**

Name of the top design unit

**-name <block-name>**

Name of the design partition to be synthesized.

Each *<block-name>* should name a module (for Verilog) or entity (for VHDL) appearing in the design that should be treated as a top-level partition. More than one block constraints may appear, or all blocks may be specified in one constraint.

**NOTE:** *The top design is also considered as a block and all block-level rules are run on it unless explicitly mentioned otherwise.*

## Rules

The block constraint is used by the following rules:

SpyGlass Constraints			
Clk_Lat01	Clk_Lat02	Clk_Lat03	Clk_Uncert01
High_Fan04	High_Fan05	Clk_Consis05	Clk_Gen13
Clk_Gen17	Clk_Gen18	Clk_Gen21	Clk_Lat09
Check_Timing04	Clk_Lat06	Clk_Lat07	Clk_Trans04
Clk_Trans05	Clk_Trans06	Clk_Trans07	Clk_Trans08
Clk_Trans16	Clk_Trans17	Clk_Trans12	Clk_Trans11
Clk_Uncert05	Combo_Paths01	Combo_Paths02	Combo_Paths03
Combo_Paths04	Const_Struct01	Const_Struct02	Const_Struct04a
Const_Struct04b	Const_Struct05	Const_Struct09	IO_Consis02
Dont_Touch05	Block05	Block11	Clk_Gen01a
Clk_Gen01b	Clk_Gen33	Clk_Gen03	Clk_Gen06
Clk_Gen07	Clk_Gen14	Clk_Gen23	Clk_Gen24
Clk_Gen26	Clk_Gen27	Check_Timing03	Clk_Gen29
Clk_Gen30	Clk_Gen31	Clk_Gen32	Show_Clock_Propagation
Clk_Uncert03	Clk_Uncert08	Clk_Gen09	Clk_Gen02

## SpyGlass Design Constraints

Disable_Timing02	Block10	Combo_Paths06	SDC_Methodology29
Show_Case_Analysis	SDC_DnStrm08	DomainAnalysis	DomainInfo
DomainError	Domain_SGDC_Consis	Dont_Touch02	Dont_Touch03
False_Path07	False_Path08	CheckMCP	False_Path04a
False_Path04	False_Path03	High_Fan10	High_Fan02
Inp_Del03a	Inp_Del03b	Inp_Trans05	Inp_Trans01
Inp_Del01a	Inp_Del01b	Inp_Del01c	IO_Consis07
TE_Consis01	MCP04a	MCP04	MCP03
TE_Consis02	SDC_Methodology65	Op_Del01a	Op_Del01b
Op_Del01c	Op_Del03a	Op_Del03b	SDC_Case_Sanity01
SDC_Methodology66	SDC_Methodology67	SDC_Methodology68	Check_Timing02
Load02b	Load02a	Disable_Timing01	SDC_Methodology30
SDC_Methodology31	SDC_Methodology44_MG	SDC_Methodology47_MG	High_Fan06
High_Fan07	Test_Rules05	Test_Rules06	Clk_Uncert07
Clk_Gen08	High_Fan12	High_Fan01a	High_Fan01
High_Fan09	High_Fan14	High_Fan15	High_Fan08
Inp_Del05	Inp_Del07a	Inp_Del07	Inp_Trans07
Inp_Trans06	Inp_Trans01a	IO_Consis01	SDC_Methodology25
SDC_Methodology62	High_Fan11	SDC_Methodology26	SDC_Methodology63
Op_Trans01	MCP06	SDC_Methodology28	Block06
Op_Del05	Op_Del07a	Op_Del07	SDC_Methodology01
SDC_Methodology03	SDC_Methodology05a	SDC_Methodology06	SDC_Methodology07

SDC_Methodology09	SDC_DnStrm05	SDC_DnStrm06	SDC_Methodology10
SDC_Methodology11	SDC_Misc_Setup01	SDC_DnStrm07	SDC_Methodology22
SDC_Methodology23	SDC_Methodology24	SDC_Report04	SDC_Misc_Power01
SDC_Misc_Command01	Clk_Trans09	Test_Rules01	Test_Rules02
Test_Rules03	Test_Rules04	Clk_Gen05	High_Fan03a
High_Fan03b	Dont_Touch04	Inp_Del02	MCP05
Op_Del09	Op_Del02	SDC_DnStrm04a	SDC_DnStrm04
SDC_Report01	SDC_Misc_WLM01	SDC_DnStrm01	SDC_DnStrm02
SDC_DnStrm03	SDC_Methodology02	SDC_DnStrm04	SDC_DnStrm04a
SDC_Methodology12	SDC_Methodology13	SDC_Report03	SDC_Methodology27
SDC_Methodology70	SDC_Report01	Load01	XBuf01
Inp_Del14	Op_Del14		

## blocksize

### Purpose

The `blocksize` constraint is used to define the maximum or minimum allowed block size for each synthesizable block. The SpyGlass Constraint solution reports any block that does not lie in these limits. By default, the maximum block size is 100000 and the minimum is 5000.

### Product

SpyGlass Constraints solution

## Syntax

The syntax of the block constraint is as follows:

```
current_design <du_name>
block -name <block1-name> <block2-name>
blocksize -min <minimum-blocksize> -max <maximum-blocksize>
```

## Arguments

**<du\_name>**

Name of the top design unit

**-min <minimum-blocksize>**

(Optional) Specifies the minimum block size for the blocks

**-max <maximum-blocksize>**

(Optional) Specifies the maximum block size for the blocks

## Examples

The following code snippet sets the minimum block size to 20000 and the maximum block size to 120000.

```
current_design top
    sdc_data -type top.sdc
    block -name lower1 lower2 lower3
    blocksize -min 20000 -max 120000
```

# breakpoint

## Purpose

The `breakpoint` constraint is used to specify breakpoints in a design.

Using the `breakpoint` constraint, you can specify internal signals of a design to be considered as input ports of the design during functional analysis. The functional analysis will stop at the breakpoints.

## Product

SpyGlass Auto Verify solution, SpyGlass CDC solution

## Syntax

The syntax of the `breakpoint` constraint is as follow:

```
current_design <du-name>
breakpoint
  -name <pin-name-list>
```

## Arguments

**-name <pin-name-list>**

Space-separated list of hierarchical pin names.

## Example

The following constraint defines signals `sig1` and `sig2` in the design unit `top` as breakpoints:

```
breakpoint -name top.sig1 top.sig2
```

## Rules

The `breakpoint` constraint is used by the following rules:

---

### SpyGlass CDC Solution

---

Ac\_cdc01a

Ac\_cdc01b

Ac\_cdc01c

Ac\_cdc08

---

Ac_fifo01	Ac_handshake01	Ac_handshake0 2	Clock_sync03a
Ac_conv02			
<b>SpyGlass Auto Verify Solution</b>			
All rules			

## bypass

### Purpose

The `bypass` constraint is used to specify the design units (black boxes) that must be bypassed.

The intention is to declare design units that will not contain internal scan wrappers. The `bypass` constraint specifies modules whose output pins must be isolated from scan registers.

Use the `Scan_20` rule of the SpyGlass DFT solution to check that such design units are so connected.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `bypass` constraint is as follows:

```
bypass -name <du-name>
```

**NOTE:** *The `bypass` constraint supports wildcard characters.*

### Arguments

**-name <du-name>**

The name of the design unit to be bypassed.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are treated as bypassed.

You can specify a single design unit name or a space-separated list of design unit names.

## Rules

The `bypass` constraint is used by the following rule:

---

**SpyGlass DFT Solution**

---

Scan\_20

---

## cdc\_attribute

### Purpose

The `cdc_attribute` constraint is used to specify mutually exclusive and unrelated signals such that:

- Convergence-related violations are suppressed for such signals, and
- Glitch issues are filtered if the specified signals are on a control crossing.

### Product

SpyGlass CDC solution

### Syntax

The syntax of the `cdc_attribute` constraint is as follows:

```
cdc_attribute  
  [ -exclusive <mutually-exclusive-signal-names> ] |  
  [ -unrelated <unrelated-signal-names> ]
```



## Arguments

### **-exclusive** <mutually-exclusive-signal-names>

Space-separated list of mutually exclusive source or destination signals, such as hierarchical net, hierarchical terminal, or port.

Such signals cannot toggle at the same time, that is, they are gray encoded.

### **-unrelated** <unrelated-signal-names>

Space-separated list of unrelated source or destination signals, such as hierarchical net, hierarchical terminal, or port.

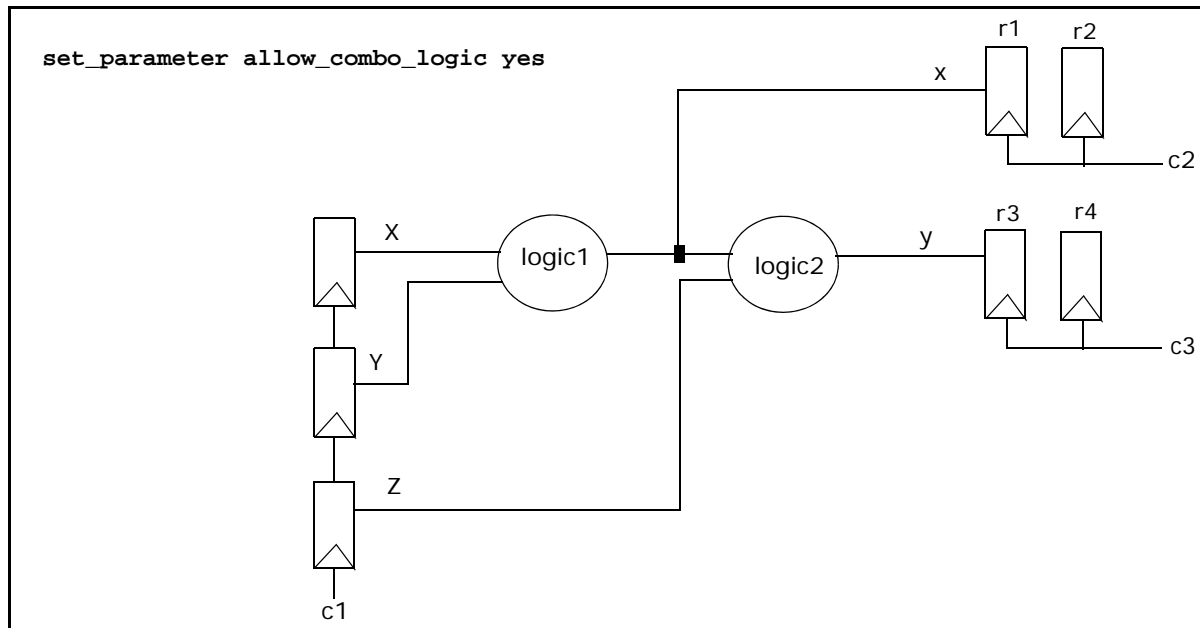
Such signals have no timing relationship (for example, interrupts).

**NOTE:** *For the Ac\_glitch03 rule, SpyGlass CDC does not consider sources specified through -unrelated argument because it assumes that the design uses custom techniques to avoid the glitch for these sources.*

## Examples

### **Example 1**

Consider the scenario shown in the following figure:

**FIGURE 21.**

In the above scenario, the *Ac\_glitch03* rule reports convergence issues at the *r1* and *r3* destination for the *X*, *Y*, and *Z* sources that are mutually exclusive due to gray encoding.

To suppress the *Ac\_glitch03* rule violations due to *X*, *Y*, and *Z*, specify the following constraint:

```
cdc_attribute -exclusive X Y Z
```

## Example 2

Consider the following files specified for SpyGlass analysis:

```

// test.v
module top(rst, in1, in2, in3, in4, clk1,
           clk2, out1);
  input rst, in1, in2, in3, in4, clk1, clk2;
  output out1;
  wire w1,w2;
  DES des(.d0(in1), .d1(in2), .d2(in3),
          .d3(in4),.c1(clk1), .out(w1));
  DFF fReg1(.clk(clk2), .d(w1), .q(w2));
  SYNC sync(.clk(clk2), .rst(rst),
            .in(w2), .out(out1));
endmodule
module DES(d0, d1,d2, d3,c1, out);
  input d0, d1, d2, d3,c1;
  output out;
  wire f1, f2, f3, w1;
  DFF fReg1(.clk(c1), .d(d0), .q(f1));
  DFF fReg2(.clk(c1), .d(d1), .q(f2));
  DFF fReg3(.clk(c1), .d(d3), .q(f3));
  MUX mux_1(.in1(f1) , .in2(f2),
            .sel(d2), .out(w1));
  assign out = w1 & f3;
endmodule

module MUX(in1 , in2, sel, out);
  input in1, in2, sel;
  output out;
  wire out;
  assign out = (sel) ? in1: in2;
endmodule

module DFF(clk, d, q);
  input clk;
  input d;
  output q;
  reg q;
  always @(posedge clk ) begin
    input clk;

```

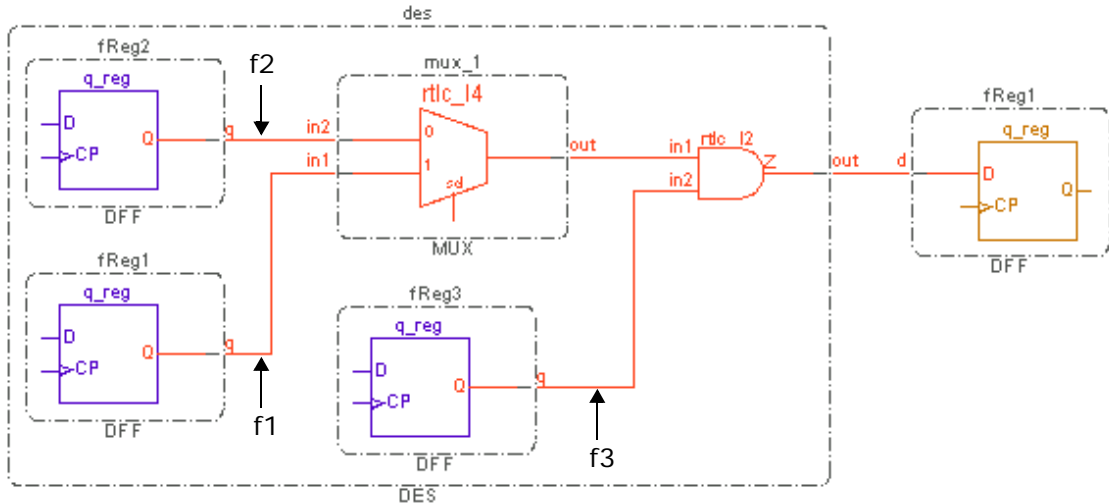
```

// constr.sgdc
current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d2

// Project File
set_parameter allow_combo_logic yes

```

For the above example, the *Ac\_glitch03* rule reports the following crossing:



### Violation Message:

Glitch check performed on destination flop 'top.fReg1.q' clocked by 'top.clk2' (3 source(s), 1 domain(s)). Multi-source toggling check : 'FAILED'

FIGURE 22.

In the above crossing, the *f1*, *f2*, and *f3* signals are mutually exclusive. If you do not want to report the crossings containing these signals, specify the following constraint:

```
cdc_attribute -exclusive top.des.f1 top.des.f2 top.des.f3
```

## Rules

The *cdc\_attribute* constraint is used by the following rule:

SpyGlass CDC Solution			
Ac_glitch03	Ac_conv01	Ac_conv02	Ac_conv03
Ac_conv04			

## cdc\_check\_glitch

### Purpose

The `cdc_check_glitch` constraint is used to generate glitch expressions for any net in the design.

### Product

SpyGlass CDC solution

### Syntax

Use the `cdc_check_glitch` constraint as follows to generate glitch expressions:

```
cdc_check_glitch -name <net-name> -type <type>
```

### Arguments

**-name <net-name>**

Specifies the hierarchical name of the net/pin.

**-type <type>**

Can be any subset {010, 101}

### Examples

The following specification builds a glitch expression for the `top.A` net for the 010 glitch:

```
cdc_check_glitch -name top.A -type 010
```

The following specification builds a glitch expression for the `top.B` net for both 101 and 010 glitch:

```
cdc_check_glitch -name top.B -type 010 101
```

## cdc\_define\_transition

## Purpose

The `cdc_define_transition` constraint is used to specify the permitted transitions of any net.

## Product

SpyGlass CDC solution

## Syntax

Use the `cdc_define_transition` constraint as follows to specify the permitted transitions:

```
cdc_define_transition -name <net-name> -type <type>
```

## Arguments

**-name <net-name>**

Specifies the hierarchical name of the net/pin.

**-type <type>**

Can be any subset {00, 11, 01, 10, 010, 101}, excluding {00, 11}

## Examples

The following specification specifies the permitted transitions of the top.A net:

```
cdc_define_transition -name top.A -type {00, 11, 01, 10}
```

## reset\_sense

### Purpose

The `reset_sense` constraint is used for stopping reset propagation at specified point in reset path.

### Product

SpyGlass CDC solution

## Syntax

Use the `reset_sense` constraint as follows to stop reset propagation:

```
reset_sense -pins <object_list>
```

## Arguments

**-object\_list** <list\_of\_objects>

Declares pins, hierarchical pins, or cell pins on which reset propagation should stop.

## Examples

The following specification stops reset propagation:

```
current_design top
    reset -name top.rst1 -value 0
    reset_sense -name top.pin1
    reset_sense -name top.M1.pin2
```

## cdc\_false\_path

### Purpose

The `cdc_false_path` constraint is used to specify false paths so that clock-domain crossings along these paths are ignored for rule checking. Such crossings are reported in *Section C* of *The Clock-Reset-Detail Report* of the SpyGlass CDC.

Note the following points:

- Use this constraint instead of waivers for SpyGlass CDC rules.
- If you want to suppress rule-checking on crossings that contain static signals, you can also use the [quasi\\_static](#) constraint.
- The internal rule `FalsePathSetup` checks for valid `cdc_false_path` constraints.

- Virtual clocks can be specified in the `-from` and `-to` arguments only of the `cdc_false_path` constraint. Note the following points for virtual clocks in the `cdc_false_path` constraint:
  - Only one virtual clock can be specified in the `-from` or `-to` arguments. For example, `cdc_false_path -from <virtual_clk1 virtual_clk2>` is not supported:
  - You can specify a virtual clock with real clock. For example, `cdc_false_path -from <virtual_clock> -to <real_clock>` is supported.
  - If a virtual clock is specified in any of the two arguments and non-clock object is specified in the other argument, the `SGDC_cdc_false_path06` rule reports a sanity Error.

### Advantages of using the `cdc_false_path` Constraint

Following are the advantages:

- Targets violations of the SpyGlass CDC solution with `-to`, `-through`, `-from` that accommodate various schemes of filtering for crossings, for example, `-from clk1` will take out all crossings initiated at flip-flops clocked by `clk1`.
- The constraint is understood by rules (waivers are applied post process), and as a result crossings as well as convergence issues may be filtered based on the `cdc_false_path` specification.
- More portable as it does not refer to rule names or messages.
- Reduces the size of the `*.vdb` file.
- As compared to the *quasi-static* constraint, the `cdc_false_path` constraint provides more flexibility in specifying the type of crossing through various options (`-from`, `-to`, and `-through`).

If `one_cross_per_dest` parameter set to `yes` and if a crossing (`s1->d1`) is waived off using `cdc_false_path` constraint (source `s1` for a destination `d1`), the other crossing (`s2->d1`) of the same destination having another source `s2` is reported. In case of waivers specified for one source (`s1->d1`), none of the crossings (`s1->d1`, `s2->d1`) will be reported for the destination.



## Product

SpyGlass CDC solution

## Syntax

Use the `cdc_false_path` constraint as follows to specify the false paths:

```
current_design <du-name>
  cdc_false_path
    [ -from <obj1-name> ]
    [ -to <obj2-name> ]
    [ -through <obj3-name> ]
    [ -from_type <obj1-type> ]
    [ -to_type <obj2-type> ]
    [ -from_obj <src-pin-name> ]
    [ -to_obj <des-pin-name> ]
    [ -from_clk <src-clk-name> ]
    [ -to_clk <des-clk-name> ]
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs)

### -from <obj1-name>

The name of a clock (real or virtual) source or the net connected to the clock pin of sequential elements, a clock tag (specified by the `-tag` argument of the `clock` constraint), a master design unit, a flip-flop output net, a port of a master design unit of the source, or an input/inout port.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

Wildcard expressions are matched to clock tag names only if no nets are present in the design with the same name. If a net is found in the design, clock tag name will not be inferred.

**-to <obj2-name>**

The name of a clock (real or virtual) source or the net connected to the clock pin of sequential elements, a clock tag (specified by the `-tag` argument of the `clock` constraint), a master design unit, a flip-flop output net, or a port of a master design unit of the destination.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

**-through <obj3-name>**

Name of an internal net, master design unit, or a port of the master design unit.

Note that specifying `cdc_false_path -through X -through Y` is equivalent to specifying `cdc_false_path -through X Y`.

You can use regular expressions and wildcard characters ('\*' and '?') while specifying names. For details, refer to the *Using Regular Expressions and Wildcard Characters* topic of the *Atrenta Console User Guide*.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression with the hierarchies. For example, the `top.*.n1` expression is matched to `top.u1.n1` and `top.u1.u2.n1`. By default, the `cdc_false_path` constraint matches only `top.u1.n1`.

Setting the value of the `hier_wild_card` parameter to `yes` runtime performance of the `cdc_false_path` constraint is impacted.

**-from\_type <obj1-type>**

Type of the signal specified in the `-from <obj1-name>` argument for which the crossings need to be filtered out.

This argument accepts any of the following values:

clock	data	all (default)
-------	------	---------------

**NOTE:** Use this argument only when you specify the `-from <obj1-name>` argument.

**-to\_type <obj2-type>**

Type of the signal specified in `-to <obj2-name>` for which the crossings need to be filtered out.

This argument accepts any of the following values:

clock	data	all (default)
-------	------	---------------

**NOTE:** Use this argument only when you specify the *-to <obj2-name>* argument.

#### **-from\_obj**

The name of the output pin or net connected to the source of the crossing. The *-from\_obj* argument cannot be used with the *-from*, *-to*, *-through*, *-from\_type*, or the *-to\_type* arguments.

#### **-to\_obj**

The name of the output pin or net connected to the destination of the crossing. The *-to\_obj* argument cannot be used with the *-from*, *-to*, *-through*, *-from\_type*, or the *-to\_type* arguments.

#### **-from\_clk**

Name of the object or the tag names of clocks that drive the source of the crossing. Note that virtual clocks are not supported. The *-from\_clk* argument cannot be used with the *-from*, *-to*, *-through*, *-from\_type*, or the *-to\_type* arguments.

#### **-to\_clk**

Name of the object or the tag names of clocks that drive the destination of the crossing. Note that virtual clocks are not supported. The *-to\_clk* argument cannot be used with the *-from*, *-to*, *-through*, *-from\_type*, or the *-to\_type* arguments.

**NOTE:** If objects specified with the *-from\_obj/to\_obj* arguments are not driven by the clocks specified with the *-from\_clk/-to\_clk* arguments, the *SGDC\_cdc\_false\_path09* rule reports an error.

## **Specifying Arguments of the *cdc\_false\_path* Constraint**

Please note the following points:

- You must specify at least one of the *-from*, *-to*, and *-through* arguments.

- When you specify a clock name (hierarchical net name) with the `-from` or `-to` argument, it should be a hierarchical net name. Then the applicable paths are paths originating from or terminating at flip-flops triggered by the specified clock.
- When a master design unit is specified with the `-from` or `-to` arguments, the applicable paths are paths originating from or terminating at all flip-flops in all instances of the specified design unit. When you specify a master design unit name with the `-through` argument, the applicable paths are all paths passing through all instances of the specified design unit.
- When you specify a flip-flop output net (hierarchical net name) with the `-from` or `-to` arguments, the applicable paths are paths originating from or terminating at the corresponding flip-flop.
- When you specify a pin name (of a master design unit) (in `<du-name>/<pin-name>` format) with the `-from` or `-to` arguments, the applicable paths are paths originating from or terminating at all flip-flops connected to the specified pin in all instances of the corresponding master design unit. When you specify a pin name (of a master design unit) with the `-through` argument, the applicable paths are all paths passing through the specified pin in all instances of the corresponding master design unit.
- When you specify an internal net (hierarchical net name) with the `-through` argument, the applicable paths are paths containing the specified net.
- When you specify a clock net with the `-from` or `-to` arguments, only those flip-flops that are exclusively triggered by the clock net are considered. Flip-flops where this clock net and some other clock net(s) are converging are not considered.

## Specifying Object Names to the `cdc_false_path` Constraint

The following points describe different ways of specifying different objects to the `cdc_false_path` constraint:

- Specify net names as hierarchical names with respect to top level. The top-level name is optional. For example, you can specify `top.U1.net1` or `U1.net1`.

- Specify design units (sub modules) as simple design unit names, such as myDU1.
- Specify top-level port names as simple names. For example, in1.  
Specify port of a design unit (submodule) as `<du-name>/<port-name>` or `<du-name>.<port-name>`. You should **not** specify instance names or top-level design unit name.

## Wild Card Support for the `cdc_false_path` Constraint

You can use the following wild card characters while specifying object names:

- Asterisk (\*): To match none, one, or multiple object names  
For example, `top.*.n1` matches with `top.U1.n1`.
- Question mark (?): To match none or one object name

### Handling Escaped Names

You cannot specify wildcards with escaped names directly. In such cases, replace the escape character (`\`) with a wildcard character.

For example, for the source flip-flop output `top.\rdPtr1` and `top.\rdPtr2`, you cannot specify the following constraint:

```
cdc_false_path -from "top.\rdPtr*"
```

Instead, you should specify the following:

```
cdc_false_path -from "top.*rdPtr*"
```

### Handling an Array of Instances

For an array of instances in a design, escaped character is appended as part of the hierarchy name. Therefore, wildcard in such names should also be specified as mentioned above.

For example, if the design `top` has an array of instances `U_INST[1:10]` and `n1` is the net inside the module of these instances. In this case, SpyGlass generates the following nets:

```
top.\U_INST[1].n1
top.\U_INST[2].n1
...
```

Now, to declare `cdc_false_path` for the `n1` net inside any of these instances, you cannot specify the name as `top.\U_INST[*].n1`. Instead, you should specify this as follows:

```
cdc_false_path -to "top.*U_INST*.n1"
```

**NOTE:** *The `hier_wild_card` parameter is set to yes and the `cdc_false_path` constraint matches all hierarchies based on the specified wildcard expression. For example, if you specify the wildcard expression `top.*.n1`, the `cdc_false_path` constraint matches the hierarchies `top.u1.n1` and `top.u1.u2.n1`. However, by default, the `cdc_false_path` constraint matches only `top.u1.n1`. In this example, if you specify the wildcard expression as `top.*.*.n1`, the `cdc_false_path` constraint matches `top.u1.u2.n1`.*

## Vector Name Support in the `cdc_false_path` Constraint

You can specify vector object names as simple object names, object names with width specification, part-select, or bit-select.

You must use the *Verilog bus-width specification* format for specifying vector object names even for VHDL designs. For example, `in1(7 downto 0)` must be specified as `in1[7:0]` or `in1[0:7]`. Similarly, a part-select should be written as say `in1[6:3]` or `in1[3:6]`. A bit-select should be written as say `in1[4]`.

You can also use wildcards in vector names. `in1[*]` matches any bit of `in1`.

The bit-width specification using wildcards for part-select is assumed to be higher bit first. Therefore, the order of bits is important. `in1[*:3]` matches `in1[7:3]`. `in1[3:*` matches `in1[3:0]`. `in1[0:*` does not match anything!

## Scope of False Paths specified by the `cdc_false_path` Constraint

The scopes for different object types are described below:

- **Clock name (hierarchical net name) with the `-from` or `-to` argument:** Applicable paths are considered to originate from or terminate at the flip-flops triggered by the specified clock.

- **Master design unit specified with the `-from` or `-to` arguments:**  
Applicable paths are considered to originate from or terminate at all flip-flops in all instances of the specified design unit.
- **Master design unit name with the `-through` argument:**  
Applicable paths are considered as all paths passing through all instances of the specified design unit.
- **Flip-flop output net (hierarchical net name) with the `-from` or `-to` arguments:** Applicable paths are considered as the paths originating from or terminating at the corresponding flip-flop.
- **Net Objects:** While specifying a net object, all nets directly connected to that net object (anywhere in the top-level design) are implicitly considered, in addition to the specified net object.

For example, `top.U1.net1` implicitly specifies directly connected net `top.net1`. This effect is more profound and often comes as a surprise when combined with a wildcard net specification. For example, if you specify `top.U1.*` with the intention to match all nets connected to output pins of flip-flops within `top.U1`, that will ultimately result in matching of every net within `top.U1` and every net present anywhere in the design unit name, `top`, that is directly connected to any net within `top.U1`. The most unexpected result is that this will match any clock net specified at higher levels that are directly connected to nets within `top.U1`. The best way to match all sources of crossings (or destinations of crossings) within a sub-module is to specify the design unit name of the sub-module directly. For example, `cdc_false_path -from A` command matches all crossings that originate within the sub-module `A`.

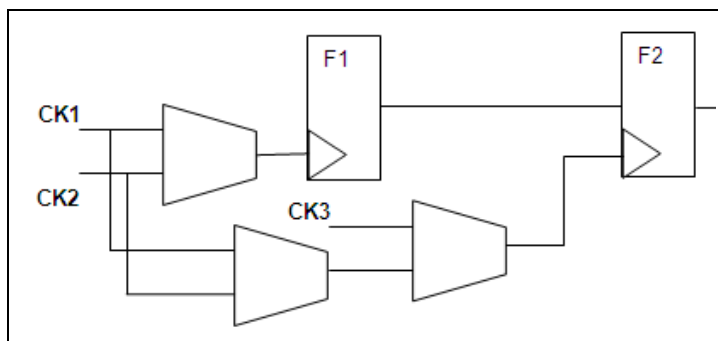
- **Pin name with the `-from` or `-to` arguments:** When you specify a pin of a master design unit in the `<du-name>/<pin-name>` format) with the `-from` or `-to` arguments, the applicable paths are the paths originating from or terminating at all the flip-flops connected to the specified pin in all instances of the corresponding master design unit.
- **Pin name with the `-through` argument:** When you specify a pin of a master design unit with the `-through` argument, the applicable paths are all paths passing through the specified pin in all instances of the corresponding master design unit.

- **Internal net (hierarchical net name) with the `-through` argument:** Applicable paths are the paths containing the specified net.

## Handling Merged Clocks with the `cdc_false_path` Constraint

When multiple clocks reach a clock pin of a flip-flop or a latch, such clocks are called merged clocks.

To waive clock crossing involving a flip-flop or a latch triggered by merged clocks, specify each clock pair to the `cdc_false_path` constraint. For example, consider the following figure:



**FIGURE 23.** Example of merged clocks

In the above case, specify the following constraints to waive the crossing:

```
cdc_false_path -from CK1 -to CK2
cdc_false_path -from CK1 -to CK3
cdc_false_path -from CK2 -to CK1
cdc_false_path -from CK2 -to CK3
```

The crossing is reported if you do not specify all combinations of clock pairs. The violation message reports the clock pair that is not specified by using the `cdc_false_path` constraint is reported.

**NOTE:** *This is not supported for crossings that involve black boxes or primary ports. In such cases, specify all the clocks in the same constraint.*

Alternatively, you can specify names of all clocks from a merged set as a space-separated list by using the `-from` or `-to` argument, as shown in



the following example:

```
cdc_false_path -from clk1 clk2 ...
```

You can specify the above constraint in the following manner:

```
cdc_false_path -from CK1 CK2 -to CK1 CK2 CK3
```

Clock crossing involving merged clocks are not waived if one of the clocks is not specified.

You cannot specify multiple objects that are not merged clocks. Such specification does not match any clock crossing and the `Fal sePathSetup` rule reports a violation. For example, the following specification is invalid:

```
cdc_false_path ... -to top.U1.Q clk2
```

### Impact of one\_cross\_per\_dest Parameter on cdc\_false\_path Constraint

Setting the `one_cross_per_dest` rule parameter to `yes` (default value) results in only the first found clock crossing of a destination object to be reported.

If you set a valid `cdc_false_path` constraint on this first found clock crossing, then the next found clock crossing for the same destination object is reported. For example, there are three clock crossings between sources `s1`, `s2`, and `s3` and the destination `d1` and these clock crossings are found as say, `s1->d1`, `s2->d1`, and `s3->d1`. Normally, the crossing `s1->d1` will be reported. If you set a valid `cdc_false_path` constraint on this path, the second found crossing `s2->d1` is reported:

```
cdc_false_path -from s1 -to d1
```

If you have only specified the `-to` argument (without `-from` argument) in a valid `cdc_false_path` constraint, all clock crossing involving the destination object are waived.

### cdc\_false\_path Sanity Checking Rules

Following are the sanity checking rules for the `cdc_false_path`

constraint:

Rule	Checks
SGDC_cdc_false_path01	Existence of object specified with the <code>-from</code> argument (without wildcards)
SGDC_cdc_false_path02	Existence of object specified with the <code>-to</code> argument (without wildcards)
SGDC_cdc_false_path03	Existence of object specified with the <code>-through</code> argument (without wildcards)
SGDC_cdc_false_path04	Existence of object specified with the <code>-from/-to/-through</code> arguments (with wildcards)
FalsePathSetup	Flags when a <code>cdc_false_path</code> constraint specification does not waive any clock crossing

## Example

### Example 1

```
cdc_false_path -from clk1 -to top.lower.q
```

The above specification suppresses all the paths originating from flip-flops triggered by clock `clk1` and terminating at flip-flop `top.lower.q`.

### Example 2

```
cdc_false_path -through LOWER/out1
```

The above specification suppresses all the paths passing through pin `out1` of all instances of master design unit `LOWER`.

### Example 3

```
cdc_false_path -from "top.b1.*" -to "block1" -from_type clock
```

The above specification suppresses all the paths that are:

- Originating from the flip-flops clocked by the clock that reaches the `b1` instance.
- Terminating at the `block1` block.

**Example 4**

```
cdc_false_path -from "top.b1.*" -to "block1" -from_type data
```

The above specification suppresses all the paths originating from the data path of the `b1` instance and terminating at the `block1` block.

In this case, the clocks reaching the `b1` instance are not honored for suppressing the crossings.

**Example 5**

```
cdc_false_path -from "block1" -to "top.b1.*" -to_type clock
```

The above specification suppresses all the paths originating from the `block1` block and terminating at flip-flops that are clocked by the same clock reaching the `b1` instance.

**Rules**

The `cdc_false_path` constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Clock_sync03a	Clock_sync03b	Clock_sync08	Clock_sync08a
Clock_sync09	Ac_cdc01a	Ac_cdc08	Ac_handshake01
Ac_cdc01b	Ac_cdc01c	Ac_conv02	Ac_conv03
Ac_handshake0	Ac_conv01	Ac_unsync01	Ac_unsync02
2			
Ac_conv04	Ac_conv05	Ac_crossing01	Ac_datahold01a
Ac_sync01	Ac_sync02	Ac_meta01	Ac_glitch01
Ac_glitch02	Ac_glitch03		

## cdc\_filter\_coherency

### Purpose

The `cdc_filter_coherency` constraint is used to specify points at or beyond which no convergence of signals should be reported. Filtering convergence this way helps in debugging convergences-related issues.

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `cdc_filter_coherency` constraint is as follows:

```
cdc_filter_coherency
  [ -stop_points <stop-points> ]
  [ -conv_gates <conv-points> ]
  [ -unrelated <list-of-synchronizers> ]
```

### Arguments

#### **-stop\_points <stop-points>**

Space-separated list of hierarchical net names, hierarchical terminals, ports, or instances beyond which convergence should not be propagated.

See [Example 2](#).

#### **-conv\_gates <conv-points>**

Space-separated list of hierarchical net names, hierarchical terminals, ports, or instances at which convergence should not be reported.

See [Example 1](#).

#### **-unrelated <list-of-synchronizers>**

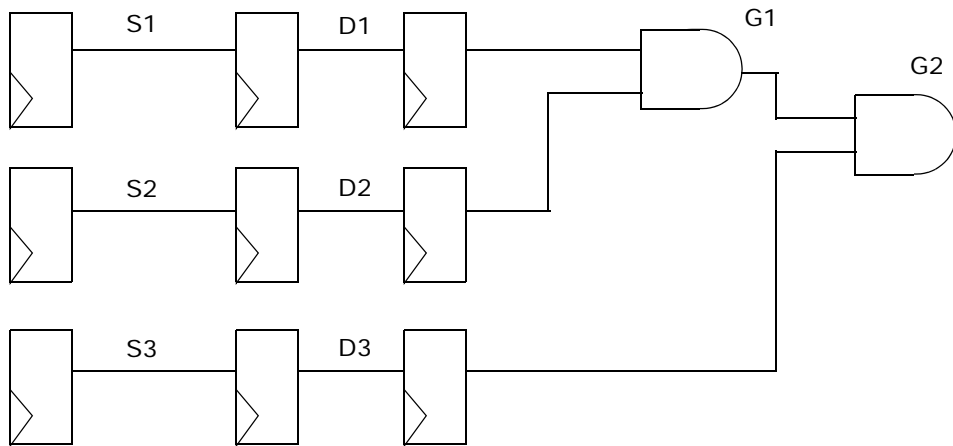
Space-separated list of synchronizers such that the `Ac_conv01`, `Ac_conv02`, `Ac_conv03`, and `Ac_conv04` rules do not report violations on the convergences involving these synchronizers.

**NOTE:** Use the `cdc_attribute` constraint instead of the `cdc_filter_coherency` constraint with the `-unrelated` argument. If the `cdc_filter_coherency -unrelated` constraint is used together with the `cdc_attribute` constraint, SpyGlass CDC ignores the `cdc_filter_coherency` constraint and honors the `cdc_attribute` constraint.

See [Example 3](#).

## Example

Consider the scenario shown in the following figure:



**FIGURE 24.**

[Example 1](#) and [Example 2](#) shows how you can filter coherence shown in the above scenario.

### Example 1

Consider the following constraint specification:

```
cdc_filter_coherency -conv_gates top.G2
```

On specifying the above constraint, no convergence will be reported on G2 shown in [Figure 24](#). However, convergence on G1 will get reported, which was not reported earlier.

### Example 2

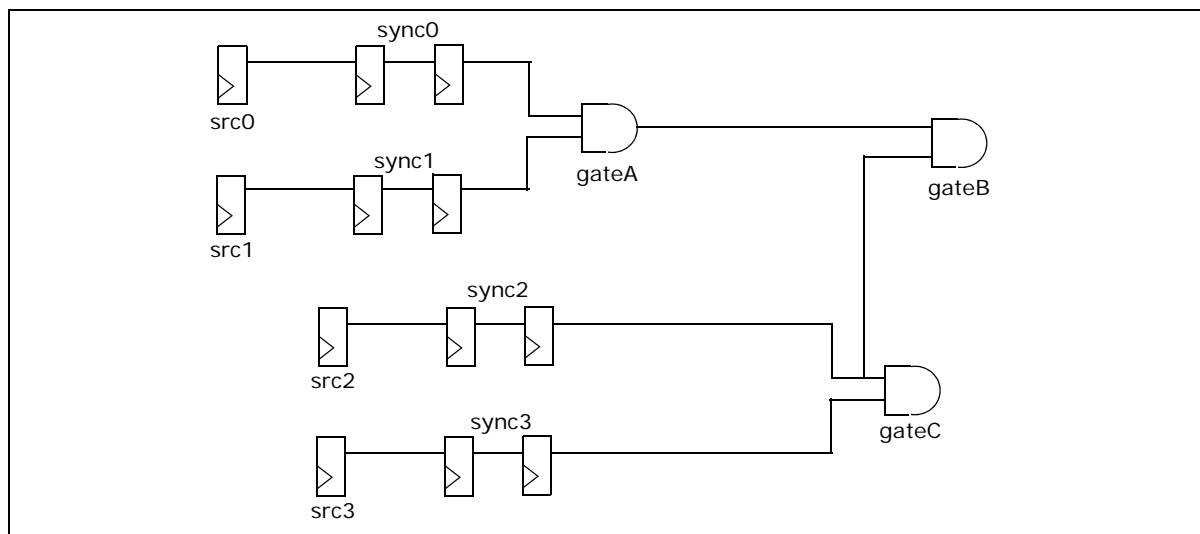
Consider the following constraint specification:

```
cdc_filter_coherency -stop_points top.G1
```

On specifying the above constraint, propagation from S1 and S2 will be stopped on G1 shown in [Figure 24](#). Therefore, no convergence will be reported on G2. However, convergence will be reported at G1 for S1 and S2.

### Example 3

Consider the following figure:



**FIGURE 25.** Example of -unrelated Argument of cdc\_filter\_coherency Constraint

In the above figure, if you do not want the *Ac\_conv01*, *Ac\_conv02*, *Ac\_conv03*, or *Ac\_conv04* rules to report convergence on gateB, specify the *sync0*, *sync1*, and *sync2* synchronizers as unrelated by specifying any of the following constraints:

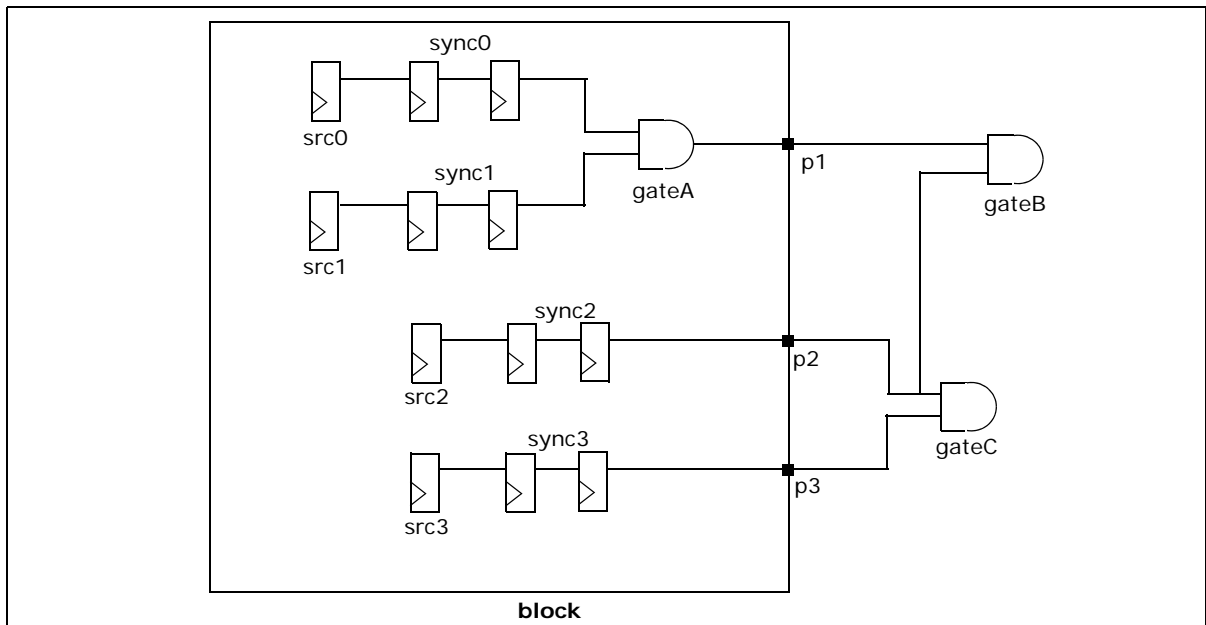
## SpyGlass Design Constraints

```
cdc_filter_coherency -unrelated sync0 sync1 sync2
```

or

```
cdc_filter_coherency -unrelated src0 src1 src2
```

However, note that the value specified by the `-unrelated` argument is not migrated to the block-level boundary during SoC-level verification. In such cases, specify the constraint manually at the top level. For example, consider the following figure:



**FIGURE 26.**

In the above scenario, consider that you have given the following constraints:

```
cdc_filter_coherency -unrelated sync0 sync1 sync2
```

In this case, the above constraint will not automatically migrate to the block boundary. Therefore, you must manually specify the following constraint to suppress the `Ac_conv01`, `Ac_conv02`, `Ac_conv03`, or `Ac_conv04` violations at gateB:

```
cdc_filter_coherency -unrelated p1 p2
```

**NOTE:** The `cdc_attribute -unrelated` constraint performs the same check as the `cdc_filter_coherency -unrelated` constraint. For example, the following two constraints perform the same checks:

```
cdc_filter_coherency -unrelated sync0 sync1 sync2
cdc_attribute -unrelated sync0 sync1 sync2
```

## Rules

The `cdc_filter_coherency` constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_conv01	Ac_conv02	Ac_conv03	Ac_conv04

## cdc\_filter\_path

### Purpose

The `cdc_filter_path` constraint allows you to specify paths so that the `Ac_unsync01/Ac_unsync02` and `Ac_sync01/Ac_sync02` rules of the *SpyGlass CDC* solution do not consider clock crossings along these paths.

The `cdc_filter_path` constraint is the same as [cdc\\_false\\_path](#).

### Product

SpyGlass CDC solution

### Rules

The `cdc_filter_path` constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_sync01	Ac_sync02	Ac_unsync01	Ac_unsync02



## cdc\_matrix\_attributes

### Purpose

The `cdc_matrix_attributes` constraint is used to set a limit for SpyGlass-CDC attributes during the SpyGlass-CDC setup stage.

To check if an attribute exceeds the specified limit, view the report generated by the `Setup_req01` rule, and fix the issue that is causing the limit to exceed. This way, you can ensure that the design statistics (in the form of attributes) are good enough to proceed with SpyGlass CDC verification.

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `cdc_matrix_attributes` constraint is as follows:

```
current_design <du-name>
cdc_matrix_attributes
  -src_clock_limit <int>
  -gen_clock_limit <int>
  -sync_reset_limit <int>
  -async_reset_limit <int>
  -domain_limit <int>
  -crossing_limit <int>
  -src_per_dest_limit <int>
  -crossing_per_clock_pair_limit <int>
```

**NOTE:** If you specify "-1" to any of the above arguments, the `Setup_req01` rule does not perform checking for the limit corresponding to that argument. You must specify -1 in double quotes (" ").

## Arguments

**<du-name>**

Name of the design unit.

**-src\_clock\_limit <int>**

Specifies the maximum number of source clocks.

By default, this argument is set to 200.

**-gen\_clock\_limit <int>**

Specifies the maximum number of generated clocks.

By default, this argument is set to 1000.

**-sync\_reset\_limit <int>**

Specifies the maximum number of synchronous resets.

By default, this argument is set to 500.

**-async\_reset\_limit <int>**

Specifies the maximum number of asynchronous resets.

By default, this argument is set to 500.

**-domain\_limit <int>**

Specifies the maximum number of clock domains.

By default, this argument is set to 200.

**-crossing\_limit <int>**

Specifies the maximum number of clock-domain crossings in a design.

By default, this argument is set to 5000.

**-src\_per\_dest\_limit <int>**

Specifies the maximum number of sources that reach a particular destination.

By default, this argument is set to 50.

**-crossing\_per\_clock\_pair\_limit <int>**

Specifies the maximum number of crossings between each clock pair.

By default, this argument is set to 500.

**Example**

The following example sets a limit for various SpyGlass-CDC attributes:

```
cdc_matrix_attributes -src_clock_limit "-1"  
-gen_clock_limit 2 -sync_reset_limit 0 -async_reset_limit 0  
-domain_limit 1 -crossing_limit 8 -src_per_dest_limit 0  
-crossing_per_clock_pair_limit 0
```

Refer to the documentation of the CDC matrix report to see the information generated in this report based on the limit set by the `cdc_matrix_attributes` constraint.

**Rules**

The `cdc_matrix_attributes` constraint is used by the following rules:

---

**SpyGlass CDC Solution**

---

Setup\_req01

---

## cell\_hookup

### Purpose

The `cell_hookup` constraint is used to specify the special cell hookups as used by the LPSVM44 rule of the *SpyGlass Power Verify* solution.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `cellhookup`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `cell_hookup` constraint is as follows:

```
current_design <du-name>
  cell_hookup
    -signame <sig-name>
    [ -names <cell-pin-name-list> ]
    [ -ignorecells <ignorecell-name-list> ]
```

### Arguments

#### **<du-name>**

Name of the design unit under which you are specifying the special cells.

#### **-signame <sig-name>**

Name of the signal to be checked for connections.

#### **-names <cell-pin-name-list>**

Space-separated cell-pin pair name list of special cells and their pins. If you need to specify more than one pin for a cell, specify the cell-pin pair more than once as in the following example:

```
... -names cell1 p1 cell1 p2 ...
```

You can use wildcard characters while specifying the cell-pin pair names using the `-names` argument.

**-ignorecells <ignorecell-name-list>**

Space-separated list of cells to be skipped while traversing the fan-out of the specified signals. You can use wildcard characters while specifying cells to be ignored using the `-ignorecells` argument.

**Rules**

The `cell_hookup` constraint is used by the following rules:

SpyGlass Power Verify Solution	
LPSVM44	LP_SGDC_CHECKS (optional)

**cell\_pin\_info****Purpose**

Specifies pin-to-supply name pairs for multi-supply cells.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `set_cell_pin_info`.*

**Product**

SpyGlass Power Verify solution

**Syntax**

The syntax to specify the `cell_pin_info` constraint is as follows:

```
current_design <du-name>
  cell_pin_info
    -cellname <cell-name>
    -pin_name_supply <pin-sname-pair-list>
```

**Arguments****<du-name>**

Name of the design unit under which you are specifying the pin-to-supply

name pairs for multi-supply cells.

**-cellname** <cell-name>

Simple cell name or a regular expression.

**-pin\_name\_supply** <pin-sname-pair-list>

Space-separated pair list of pin name and its supply name as in the following example:

```
cell_pin_info -cellname "INVX2*"
  -pin_name_supply VSS VSS1 VDD VDDC IN VDDC OUT VDDC
```

## Rules

The `cell_pin_info` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

LPSVM49

---

## cell\_tie\_class

### Purpose

Specifies tie conditions for multi-power and multi-ground cells.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `set_cell_tie_class`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `cell_tie_class` constraint is as follows:

```
current_design <du-name>
  cell_tie_class
    -cell <cell-name> | -instance <inst-name>
```

```

-tie1 <power-pin-net-name>
-tie0 <ground-pin-net-name>
[ -no_tie <no-tie-pin-name-list> ]
[ -exception <exception-list> ]

```

## Arguments

### <du-name>

Name of the design unit under which you are specifying tie conditions.

### -cell <cell-name>

Name of the cell for which you are specifying tie conditions.

### -instance <inst-name>

Name of the cell instance for which you are specifying tie conditions.

You can use wildcard characters while specifying cell names (using the -cell argument) and instance names (using the -instance argument).

For example,

```

cell_tie_class -cell "RSHL*" -tie1 vdd -tie0 vss -no_tie A
E

```

### -tie1 <power-pin-net-name>

Name of the power pin of the cell <cell-name> when the cell is a multi-power, multi-ground cell or name of the net connected to the power pin of cell instance <inst-name> when the cell instance is an instance of a multi-power, multi-ground cell. The field accepts a scalar net or bit select of the vector net.

### -tie0 <ground-pin-net-name>

Name of the ground pin of the cell <cell-name> when the cell is a multi-power, multi-ground cell or name of the net connected to the ground pin of cell instance <inst-name> when the cell instance is an instance of a multi-power, multi-ground cell. The field accepts a scalar net or bit select of the vector net

**-no\_tie <no-tie-pin-name-list>**

(Optional) Specifies a space-separated name list of pins of cell *<cell-name>* or nets connected to pins of cell instance *<inst-name>* that should not be connected to the power/ground pins/nets specified using the *-tie1/-tie0* arguments.

**-exception <exception-list>**

(Optional) Specifies the pins of cell *<cell-name>* or nets connected to pins of cell instance *<inst-name>* that should not be connected to the power/ground pins/nets specified using the *-tie1/-tie0* arguments and should be connected to the specified power/ground pins/nets.

*<exception-list>* is a space-separated list of the following tuples for each exception pin/net:

*<pin-net-name>* *<pwr-pin-net-name>* *<gnd-pin-net-name>*

For example:

```
-exception Z top.VSS_3 top.VSS_5
```

## Rules

The `cell_tie_class` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

---

LPPLIB14

---

## clock

**NOTE:** *The clock SGDC command is not supported in Tcl shell. Instead, you can use the `create_clock`, `create_clock_attribute`, and `create_generated_clock` commands to specify clock and its attributes in Tcl shell.*

## Purpose

The `clock` constraint is used to define clocks of a design.

The definition of design clocks is of high importance for proper functionality



of associated products. Wrong assumptions on clock periods/edges may cause undetected bugs or false rule violations.

## Product

SpyGlass Auto Verify solution, SpyGlass Constraints solution, SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions, SpyGlass ERC product, SpyGlass Power Verify solution, and SpyGlass CDC solution.

## For SpyGlass CDC solution, SpyGlass Constraints solution, and SpyGlass Auto Verify solution

### Syntax

The syntax of the `clock` constraint is as follows:

```
current_design <du-name>
  clock
    -name <clk-name> | -tag <logical-clock-name>
    [ -period <period> ]
    [ -edge <edge-list> ]
    [ -domain <domain-name> ]
    [ -add ]
```

### Arguments

#### <du-name>

Specifies any of the following:

- Module name for Verilog designs
- Design unit name in the `<entity-name>.<arch-name>` format for VHDL designs

#### -name <clk-name>

The clock port/pin name.

You can specify a single port/pin name or a space-separated list of port/

pin names.

For top-level port/pin, `<clk-name>` can be the port's/pin's full hierarchical name or its simple name. For other pins, `<clk-name>` must be the pin's full hierarchical name.

If the clock name is an escape name, it should be enclosed in `q%` and `%`, as shown in the following example:

```
clock -name q%top.\ipcie/txbclk[0]%
```

**NOTE:** *If you define multiple clocks on multiple terminals that are derived by the same net, such clocks are considered as different clocks by SpyGlass CDC solution.*

### **-tag <logical-clock-name>**

(Optional) Logical name of the clock.

Based on the specified logical name (tag), the spreadsheet of the following rules displays the source and destination clock tag names:

Ac_glitch01	Ac_glitch02	Ac_sync02	Ac_sync01
Ac_unsync02	Ac_unsync01	Ac_cdc01a	Ac_cdc01b
Ac_cdc01c			

SpyGlass reports a violation if multiple clocks have the same tag name but different characteristics, such as domain.

If you do not specify the `-name` argument of the `clock` constraint, the name specified by the `-tag` argument is considered as the virtual clock name.

### **-period <period>**

(Optional) Clock period (in nanoseconds) to be used in clock domain crossing checks.

You can specify the clock period as any positive floating-point number. SpyGlass Auto Verify solution rounds off the specified number to the nearest multiple of 0.5.

If you do not specify the clock period, SpyGlass Auto Verify solution assumes the clock period to be 10.

**-edge <edge-list>**

(Optional) Clock-edge value list.

By default, clock edges are assumed 0, 50, 100, and so on.

The following command describes a 100MHz clock with posedge at zero ns and negedge at five ns.

```
clock -name top.clk -period 10 -edge {0 5}
```

**-domain <domain-name>**

(Optional) Clock domain name.

The domain name can be any valid string not containing whitespace characters.

If the domain name is not specified, the clock domain name is the same as the clock name.

**NOTE:** *The -domain argument of the clock constraint is ignored by the SpyGlass Constraints solution.*

You must provide all clock sources of a design along with their attributes (period and edges). The associated products first analyze the clocks of a design prior to any functional analysis. At least one source clock is expected for any register clock pin. If no source clock could be identified for a register, an error message is issued and no functional analysis is performed.

The clock definitions are reported through the `Av_clkinf01` rule of the SpyGlass Auto Verify solution or the `Propagate_Clocks` rule of the SpyGlass CDC solution. You can visualize and explore the clocks definitions through schematic highlight and RTL back-annotations. You can determine the missing clocks and provide their definition and re-run the analysis again.

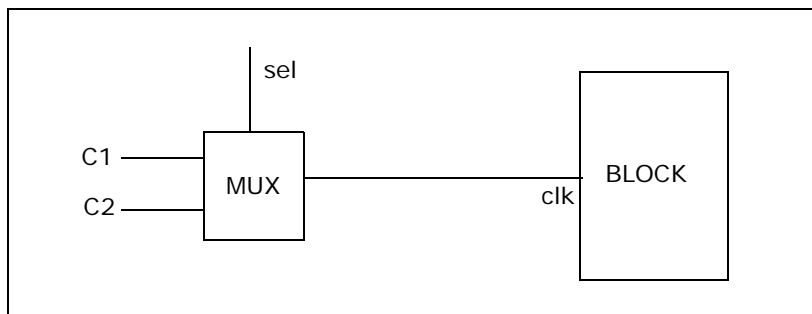
If you are unsure of the clock domains in your design, use the SpyGlass CDC solution to identify clock sources of a design.

The associated products support multiple asynchronous clocks and gated clocks. The source clocks are supposed to be linked to the register clock pin through combinational or sequential gates. SpyGlass Auto Verify solution analyzes the clock circuitry while analyzing the functionality of a design. In particular, clock dividers are participating in functional analysis.

**-add**

(Optional) Use this argument to specify multiple clocks on the same object. You can specify multiple clock constraints on the same object (specified by the *-name <clk-name>* argument) only if you specify the *-add* argument. This argument overrides the unique check on the *-name <clk-name>* argument.

Consider the scenario shown in the following figure:



**FIGURE 27.**

In the above scenario, the block pin *clk* can have any of the clock, *C1* or *C2*, coming through the MUX depending upon the MUX select line. This means that this block can have two different values for the *clk* clock. Therefore, it is necessary to define multiple clocks on the same object in the above case.

In such cases, use the *-add* argument in the next clock constraint specification on the same object, as shown in the following example:

```
clock -name top.clk -tag C1 -domain A -period 10.0
clock -name top.clk -tag C2 -domain B -period 10.0 -add
```

In the above example, the *-name* argument is the unique key of the clock constraint. That is, only one clock can be applied on a single object (in this case, *clk*). To add another clock with the same *-name* argument, the *-add* argument is used.

## Primary Clocks

A primary clock is a source clock defined using the `clock` constraint. A primary clock may be directly connected to a clock pin of a register or it may pass through combinational and/or sequential logic before reaching the clock pin of a register. You should define the following potential primary clock sources:

- Primary inputs of a design
- Black box outputs (for example, PLL outputs)

In addition, you can define an intermediate signal of a design as a primary clock. For instance, if a MUX is selecting between two clock sources running at 33MHz and 66MHz, and if you want to check the functionality of a design for 66MHz, the output of the MUX can be defined as a primary clock with 66MHz frequency. Alternatively, the 66MHz source clock can be defined as a clock and a constraint can be set for the MUX select pin to select the 66MHz clock only.

## Derived Clocks

A derived clock is a clock that is generated internally from a primary clock. The primary clock could pass through a combinational logic or sequential logic or both to generate a derived clock. Assume a clock port traversing a clock divider, the clock port should be defined as a primary clock and the output of clock divider feeding some registers' clock is a derived clock (assuming you have not defined it as a primary clock).

The concept of derived clocks is important for some specific rules where the relative frequencies of the derived clocks are used to determine if the crossing is from a fast clock domain to a slow clock domain. For structural analysis (needed by some rules), the frequency seen by the derived clock is the maximum frequency of all source clocks in the fan-in cone of the derived clock. If this assumption is not appropriate, you can define a primary clock for the internal signal that is directly controlling the register.

The SpyGlass Auto Verify solution and SpyGlass CDC solution support complex gated clocks such that they can analyze the functionality of a design in the presence of such complex gated clocks. However, these products do not attempt to determine the exact phase and frequency of such clocks and do not report them. For instance, if these products carry out functional analysis on the design illustrated in the figure shown below. However, the clock report (the `Av_clkinfo01` rule of the SpyGlass Auto Verify solution or the `Propagate_Clocks` rule of the SpyGlass CDC solution) will

only report the clocks defined by you that are ck1, ck2, and ck3.

### Default clock

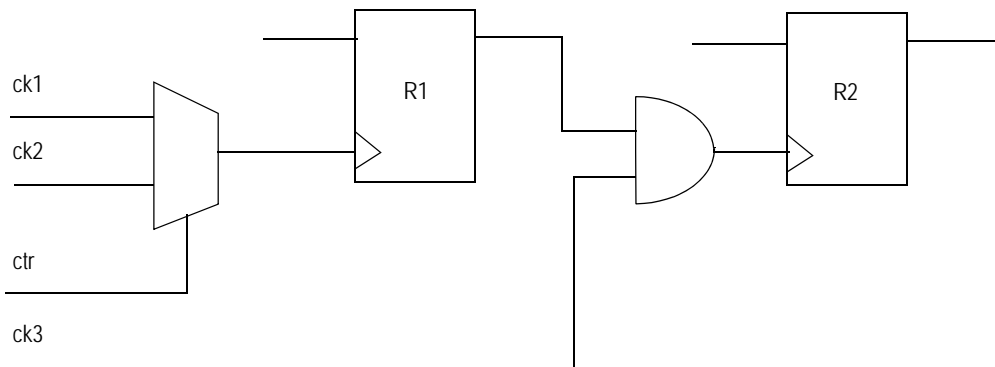
If the clock definitions of primary clocks provided by you are missing the period or edge information, default clock attributes will be assigned to them.

A default clock has a period of 10ns (100 MHz) with a rising edge at 0 and a falling edge at 5ns. While the default clock can be used with a design with a single clock feeding all registers with a single phase, it may lead to erroneous failure or success of the rules of the associated products for gated clock or multi-clock designs.

### Clock Definition Impact on Functional Analysis

The associated products determine the source clocks for all registers in a design. The functional analysis is carried out by evaluating each register at the active edge of its controlling clocks.

For example, in the figure shown below, clocks ck1, ck2, and ck3 are defined as clocks using the `clock` constraint. Register R1 is seeing ck1 and ck2 as clocks, while register R2 is seeing ck1, ck2, and ck3 as clocks. Therefore, register R1 is evaluated at the active edges of ck1 as well as the active edges of ck2, while register R2 is evaluated at the active edges of all three clocks — ck1, ck2, and ck3.



**FIGURE 28.** Clock Definition Impact

## Rules

The clock constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Clock_check01	Clock_check02	Clock_check03	Clock_check04
Clock_check05	Clock_check06a	Clock_check06b	Clock_check07
Clock_converge01	Clock_delay01	Clock_delay02	Clock_glitch01
Clock_glitch02	Clock_glitch03	Clock_info02	Clock_info03a
Clock_info03b	Clock_info03c	Clock_info05	Clock_info05a
Clock_info06	Clock_info07	Clock_info14	Clock_info15
Clock_info16	Clock_Reset_check01	Clock_Reset_Info01	Clock_sync03a
Clock_sync03b	Clock_sync05	Clock_sync06	Clock_sync08
Clock_sync08a	Clock_sync09	Propagate_clocks	Ar_resetcross01
Reset_sync01	Reset_sync02	Reset_sync03	Reset_sync04
Ac_cdc01a	Ac_cdc01b	Ac_cdc01c	Ac_cdc08
Ac_fifo01	Ac_conv01	Ac_conv02	Ac_conv03
Ac_sync01	Ac_sync02	Ac_unsync02	Setup_quasi_static01
Ar_resetcross_matrix01	Ac_conv04	Ac_conv05	Ac_handshake01
Ac_handshake02			
<b>SpyGlass Auto Verify Solution</b>			
All rules			
<b>SpyGlass Constraints Solution</b>			
SDC_GenerateIncr			

## For SpyGlass DFT solution, SpyGlass DFT DSM solution

### Purpose

The clock constraint declares the clock pins used as test clocks.

## Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

## Syntax

```
clock
    -name <pin-name> ]
    [ -sysclock | -testclock ]
    [ -atspeed ]
    [ -value <value> ]
    [ -fflimit <limit> ]
    [ -fflimit_percentage <percentage> ]
    [ -domain <dom-name> ]
    [ -pll_reference ]
    [ -frequency <frequency> ]
    [ -period ]
    [ -scanshift]
    [ -capture]
```

## Arguments

### **-name <pin-name>**

Complete hierarchical name of a clock port/pin.

The pin can be a primary pin as well as an internal pin.

You can specify a single port/pin's full hierarchical name or a space-separated list of full hierarchical port/pin names.

For primary ports, you can also specify the simple port name, as in the following example:

```
current_design top
    clock -name in15 ...
```

When you specify this argument with the *clock* constraint, all the clocks specified irrespective of the presence/absence of the *-testclock* and *-atspeed* modifiers, are considered as functional clocks.

### **-sysclock**

Specifies that the clock is a system clock.



**NOTE:** *By default, the clock pin specified with the `clock` constraint is considered as a system clock. Therefore, you need not use the `-sysclock` argument unless required for clarity.*

#### **-testclock**

Specifies that the clock is a test clock.

When you specify this argument with the `clock` constraint, all the clocks specified irrespective of the presence/absence of the `-atspeed` modifier are considered as slow clocks. Such clocks are used during scan shift and stuck-at capture.

**NOTE:** *You should use this option for all the rules of the SpyGlass DFT DSM solution. Otherwise, the clock pin specified with the `clock` constraint is considered as a system clock. Therefore, the coverage reported is very low.*

#### **-atspeed**

Specifies that the clock operates at the system frequency.

When you specify this argument with the `clock` constraint, all the clocks specified irrespective of the presence/absence of the `-testclock` modifier are considered as `atspeed` clocks. Such clocks are used during `atspeed` test.

**NOTE:** *This option is not applicable for the SpyGlass DFT solution. Use this option for all the DSM rules. Otherwise, the coverage reported will be very low.*

#### **-value <value>**

Determines the edge (first or second) on which the capture will occur.

You can specify the value as `rto` or `rtz`. If the value is not specified, `rtz` is assumed.

For information on `rtz` and `rto` clocks, refer to *Test clocks* section in *SpyGlass DFT Rules Reference Guide*.

#### **-fflimit <limit>**

Specifies the maximum number of flip-flops that can be driven by one test clock, as checked by the `Clock_25` rule of the *SpyGlass DFT* solution.

**NOTE:** *This option is not applicable for the SpyGlass DFT DSM solution.*

**-fflimit\_percentage <percentage>**

This signifies the upper limit on the percentage of total flip-flops driven by a given test clock. If the specified limit is crossed, the Clock\_25 rule generates a violation message.

**NOTE:** *This option is not applicable for the SpyGlass DFT DSM solution.*

**-domain <dom-name>**

Specifies the domain name. This option provides a mechanism to merge two or more clocks into the same domain.

**NOTE:** *This option is not applicable for the SpyGlass DFT solution. This option is only applicable for the at-speed clocks. All the at-speed clocks not merged in the same domain are considered as asynchronous.*

**-pll\_reference**

Specifies that the clock should be used only as a pll reference clock and should not be used for any other purpose.

**NOTE:** *Ensure that the -pll\_reference modifier is specified for the PLL\_03 rule to run.*

**-frequency <frequency>**

Specifies frequencies (in MHz) or a logical frequency name (as a string) associated with a test clock.

**-period**

Specifies period, in nanoseconds, associated with a test clock.

**-scanshift**

(Optional) Indicates that the clock is only required during the scan shifting operations.

**NOTE:** *You can specify both the -scanshift argument and the -capture argument together. However, you can not specify -scanshift and -testclock arguments together.*

**-capture**

(Optional) Indicates that the clock is only required during the capture operations.

**NOTE:** *You can specify both the -scanshift argument and the -capture argument*

together. However, you can not specify `-capture` and `-testclock` arguments together.

## Notes

- If the same pin is used as a system clock and a test clock, use two clock constraints — one with `-sysclock` or no argument and the other with the `-testclock` argument.
- If a clock constraint is a proper subset of another clock constraint, then it is recommended to ignore the sub-set clock constraint.

Consider the following example:

```
clock -name clk -testclock
clock -name clk -testclock -atspeed // ignore above
```

In the example above, you can ignore the first clock definition as it a sub-set of the second one.

Using multiple definitions for the same clock constraint, generates multiple messages for the same clock.

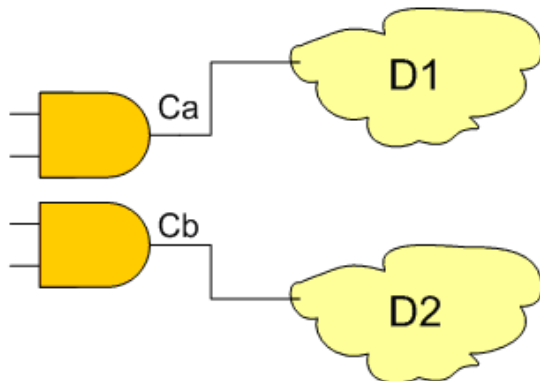
## Example

### For SpyGlass DFT DSM solution:

#### *Example 1*

If the following constraints are used as shown in the diagram in the illustration shown below, all flip-flops in D1 and in D2 will be considered as the same at-speed domain.

```
clock -name Ca -testclock -atspeed -domain CC
clock -name Cb -testclock -atspeed -domain CC
```



**FIGURE 29.** Flip-Flops in same at-speed domain

### **Example 2**

Consider the following example:

```
clock -name clk1 -atspeed -testclock -period 10
```

In the above example, the clock, `clk1`, is assumed to be running with a 10 nanosecond period in at-speed mode.

### **Example 3**

The following table lists some examples on various definitions of the clock constraint.

<b>Example</b>	<b>Description</b>
<code>clock -name</code>	Used as a functional clock
<code>clock -name clk</code> <code>-testclock</code>	Used as a functional and a slow clock
<code>clock -name clk -atspeed</code>	Used as a functional and an atspeed clock
<code>clock -name clk</code> <code>-testclock -atspeed</code>	Used as a functional, slow, and an atspeed clock

## **Rules**

The `clock` constraint is used by the following rules.

<b>SpyGlass DFT DSM Solution</b>			
Atspeed_01	Atspeed_02	Atspeed_03	Atspeed_04
Atspeed_05	Atspeed_06	Atspeed_07	Atspeed_08
Atspeed_09	Atspeed_11	Diagnose_01	Diagnose_03
Diagnose_04	Info_transitionCoverage	Info_transitionCoverageAudit	Info_atSpeedClock
PLL_01	PLL_02	Atspeed_12	Atspeed_13
Atspeed_14	PLL_03	Atspeed_26	Atspeed_27
Info_enabledFlops			
<b>SpyGlass DFT Solution</b>			
clock_01	clock_02	clock_03	clock_04
clock_05	clock_06	clock_08	clock_14
clock_18	clock_21	clock_26	Info_blackboxDriver
Latch_04	Topology_11		

## For SpyGlass Power Verify solution, SpyGlass ERC Product, and SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Purpose

The `clock` constraint declares the clocks used in the design.

### Product

SpyGlass Power Verify solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions, and SpyGlass ERC product

### Syntax

The syntax to specify the `clock` constraint is as follows:

```
current_design <du-name>
  clock
```

```
-name <clk-name>
[ -period <period> ]
```

**NOTE:** *The SpyGlass ERC product uses only the -name argument of the clock constraint to specify the names of the clock pin, ports, or nets. The -period argument is ignored by the SpyGlass ERC product.*

## Arguments

**<du-name>**

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

**-name <clk-name>**

Name of the clock port, pin, or net.

**-period <period>**

The clock period (in nanoseconds).

You can specify multiple clock constraints.

## Rules

The clock constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction solutions</b>			
poweraudit	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	PESTR03	PESTR05
PESTR06	PESTR07	PESTR08	PESTR09
PESTR10	PESTR11	PESTR12	PESTR13
<b>SpyGlass ERC Product</b>			
clockPinsConnectedToClockNets			
<b>SpyGlass Power Verify Solution</b>			
LPSVM42	LPSVM43		

**NOTE:** *The information provided by the clock constraint overrides the simulation*

information provided by the [activity\\_data](#) constraint.

## clock\_buffer

### Purpose

Specifies a buffer library cell that should be used in clock lines while estimating a clock tree.

If you do not specify any buffer using the `clock_buffer` constraint, the related rules heuristically infer the best buffer cell to be used.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `clock_buffer` constraint is as follows:

```
current_design <top-du-name>
  clock_buffer
    -cellname <cell-name>
    [ -libname <lib-name> ]
    [ -maxfanout <num> ]
```

### Arguments

#### <top-du-name>

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

#### -cellname <cell-name>

Name of the buffer cell of library `<lib-name>`.

**NOTE:** *The -libname argument is optional.*

Specify a library name using the optional `-libname` argument only when a cell with the same name is available in more than one library.

**NOTE:** *If you do not use the `-libname` argument, the library that contains the specified cell name will be taken.*

### **-maxfanout** <num>

<num> is the maximum allowed fan-out for the specified buffer cell. Then, SpyGlass creates cascade of sets of buffer cell instances, each set having the specified number of buffer cell instances, to create the required buffering. If you do not specify the `-maxfanout` argument, SpyGlass creates the required buffering based on the `max_capacitance` attribute value for the buffer cell.

## Example

Consider the following example, the `clock_buffer` constraint is specified for a net having a fan-out of 500:

```
clock_buffer -cellname BUFX4 -maxfanout 16,
```

Then SpyGlass will assume  $500/16 = 31.25 \sim 32$  (rounded off) buffers at the first stage.

At the second stage, it will assume  $32/16 = 2$  buffers.

At the third stage, it will assume one root buffer. Therefore, a total of  $32 + 2 + 1 = 35$  buffers will be assumed.

## Rules

The `clock_buffer` constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## clock\_group



## Purpose

The `clock_group` constraint is used to specify the clock relationship (domain, synchronous/asynchronous, exclusive). In this document, the domain term is used to denote the clock relationship. The wildcard support is provided for the `clock_group` constraint.

**NOTE:** The *domain* constraint has been renamed to *clock\_group*. For backward compatibility, the *domain* constraint is currently still available.

## Product

SpyGlass Constraints solution

## Syntax

The `clock_group` constraint is used in the following syntax:

```
current_design <du-name>
  clock_group -name <clock_group-name>
  [ -clock_pin <clk-pin-name-list> ]
  [ -clock <SDC-clk-name-list> ]
```

## Arguments

### <du-name>

The top-level module name (for Verilog designs) or the top-level entity name (for VHDL designs) or a synthesis partition name specified using the `block` keyword.

### -name <clock\_group-name>

(Mandatory) Specifies a clock group name. It can be any valid string.

### -clock\_pin <clk-pin-name-list>

Specifies space-separated list of design clock pin names.

If this argument is used, all clocks specified on the clock pin are considered synchronous.

### -clock <SDC-clk-name-list>

Specifies a space-separated list of SDC clock names.

If both options are specified, then the `clock_group` will include all the clocks specified with `-clock` option and all the clocks defined on the clock pin specified with the `-clock_pin` option.

## Example

Let the clock group information be provided as follows:

```
current_design top
  clock_group -name d1 -clock { C1 C2 }
  clock_group -name d2 -clock { C3 }
```

In the above example, clocks C1 and C2 are synchronous, and clocks C1 and C3, C2 and C3 are asynchronous. You cannot specify the same clock in two different `clock_group` constraints. If you do so then the `DomainSanityCheck` rule reports a FATAL message.

If the same `clock_group` is specified in multiple lines of an SGDC file, the union of clocks (specified in multiple lines) will be in the same `clock_group`. For example:

```
clock_group -name d1 -clock {C1 C2}
clock_group -name d1 -clock {C4}
```

Here, clocks C1, C2, C4 will be assumed as synchronous.

All create clocks and their derived clocks will be assumed to be synchronous if you have specified the `clock_group` constraint among create clocks and their derived clocks. You can even specify clock and its derived clocks in a different `clock_group`. For example:

C1 -> GC1 -> GC2 where GC2 is derived from GC1 and so on.

```
// C1, GC1, GC2 will be assumed to be synchronous unless the
// user overrides it
clock_group -name d1 -clock C1
```

```
// User specified clock GC2 with a different group name
clock_group -name d2 -clock GC2
```

The `domainsanitycheck` rule reports a violation if you specify create clock and its derived clock in different clock groups.

**NOTE:** *All SGDC-dependent rules will be impacted.*

## Rules

The `clock_group` constraint is used by the following rules:

<b>SpyGlass Constraints Solution</b>			
Clk_Gen05	Clk_Lat03	Clk_Uncert03	False_Path07
False_Path08	Domain_SGDC_Consis		

## clock\_path\_wrapper\_module

### Purpose

The `clock_path_wrapper_module` constraint is used to exclude modules from the checks performed by the `Clock_hier01`, `Clock_hier02`, and `Clock_hier03` rules.

### Product

SpyGlass CDC solution

### Syntax

The syntax of the `clock_path_wrapper_module` constraint is as follows:

```
current_design <du-name>  
  clock_path_wrapper_module -names <module-list>
```

### Arguments

The `clock_path_wrapper_module` constraint has the following arguments:

#### <du-name>

The name of the design unit from which you want to exclude modules from the checks performed by the `Clock_hier01`, `Clock_hier02`, and `Clock_hier03` rules.

#### -names <module-list>

Space separated list of module names.

### Example

If you want to exclude specific modules, specify the module names in a space separated list, as shown in the following:

```
clock_path_wrapper_module -names MOD1 MOD2
```

In this example, the checks performed by the `Clock_hier01`, `Clock_hier02`,

and *Clock\_hier03* will exclude MOD1 and MOD2.

## Rules

The `clock_path_wrapper_module` constraint is used by the following rules:

---

**SpyGlass CDC Solution**

---

Clock_hier01	Clock_hier02	Clock_hier03
--------------	--------------	--------------

---

## clock\_pin

### Purpose

The `clock_pin` constraint is used to specify black box pins that should be assumed clock pins.

Then the `Clock_11` rule, which uses clock source analysis, treats a pin specified with the `clock_pin` constraint just as it does clock pins on flip-flops. The source of such a pin must be a test clock controlled just as any other clock source.

This is particularly useful to ensure that the clock logic connected to a black box is designed for SpyGlass DFT solution even before the box itself is available.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `clock_pin` constraint is as follows:

```
clock_pin
  -name <du-name>.<port-name>
  [ -value <value> ]
```

### Arguments

The `clock_pin` constraint has the following arguments:

#### <du-name>

The name of the design unit (black box) for which you are specifying the clock pin.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name `<du-name>` can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are processed.

You can specify a single design unit name or a space-separated list of

design unit names.

#### **<port-name>**

Name of the clock port on the design unit (black box).

You can specify only a single port name.

#### **-value <value>**

(Optional) The active value for this clock port *<port-name>*. This argument can take one of the following values: *rtz* or *rto*

For information on *rtz* and *rto* clocks, refer to the *Test clocks* section in *SpyGlass DFT Rules Reference Guide*.

## Rules

The `clock_pin` constraint is used by the following rules:

SpyGlass DFT Solution	
Clock_11	Clock_11_capture

## clock\_root

### Purpose

The `clock_root` constraint is used to specify the starting node of a clock tree for checking whether clock tree is balanced.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `clockgen`.*

### Product

SpyGlass DFT DSM solution

### Syntax

```
clock_root
  -name <lst_signal_names>
```

## Arguments

The `clock_root` constraint has the following arguments:

**-name <lst\_signal\_names>**

List of signal names

## Rules

The `clock_root` constraint is used by the following rule:

---

**SpyGlass DFT DSM Solution**

---

Atspeed\_08

---

## clock\_sense

### Purpose

Use the `clock_sense` constraint to stop propagation of clocks from the specified pins.

### Product

SpyGlass CDC solution

### Syntax

```
clock_sense
  -pins <pins-list>
  [ -tag <tag-names> ]
```

## Arguments

The `clock_sense` constraint has the following arguments:

**-pins <pins-list>**

Specifies the pins at which clock propagation should stop.



**-tag <tag-names>**

Specifies tags so that the propagation of the clocks associated with the specified tags is stopped at the pins specified by the *-pins <pins-list>* argument.

If you do not specify this argument, SpyGlass stops the propagation of all the clocks at the pins specified by the *-pins <pins-list>* argument.

**Example**

Consider the following constraints specification:

```
current_design top
clock -name top.clk1 -domain D1 -tag C1
clock -name top.clk1 -domain D2 -tag C2
clock_sense -pins AND.in1 -tag C1
clock_sense -pins AND.in2
```

In this case:

- Propagation of the `clk1` clock is stopped at the `in1` pin of the AND gate.
- Propagation of all the clocks is stopped at the `in2` pin of the AND gate.

**Rules**

The `clock_sense` constraint is used by the following rule:

---

**SpyGlass CDC Solution**


---

All clock-checking rules

---

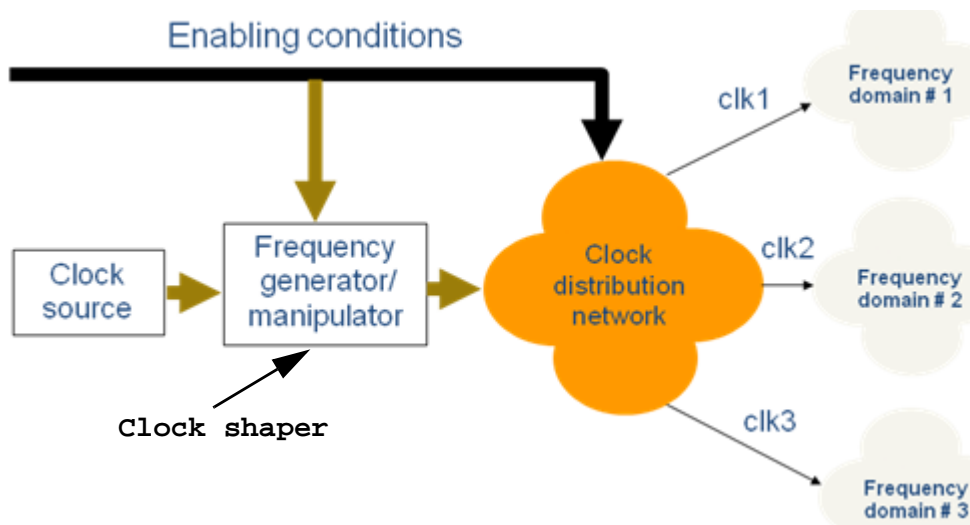
**clock\_shaper****Purpose**

The `clock_shaper` constraint is a module that controls clock pulse propagation in a design by enabling/disabling it under certain conditions or by modifying frequencies. Here, clock shaping means changing the

frequency, that is, pulse shape, duty cycle, and enabling condition of the individual clock propagation unit.

However, a clock gating cell (CGC) can only enable or disable a clock. It is not responsible for clock shaping.

Consider the following figure:



**FIGURE 30.** A clock shaper

The above figure represents a clock distribution network. The clock shaper is used as a key element in the network.

The structure and the use model of a clock shaper can be very versatile, ranging from a simple clock buffer to a complex programmable structure. The following are some uses of a clock shaper:

- PLL (clock generator)
- Frequency multiplier
- Frequency divider
- Flip-flop based divide-by-2 counter
- Crossbar switch between set of input and output clock pins with possible frequency manipulation

A module must be declared as `clock_shaper` and proper constraint options must be given for proper functioning in this way. Clock shapers are treated as black boxes by simulation.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `clockshaper`.*

## Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

## Syntax

```
clock_shaper
  -name <module-name>
  [-clkkin <clkkin-pin-list>]
  [-clkout <clkout-pin-list>]
  [-reset <reset-pin> -resetvalue <reset-value>]
  [-enable <en-pin-list> -envalue <en-value-list>]
  [-freqcontrols <freq-ctrl-pins> -freqcontrolvalues
<values_string>]
  [-freqmult <mult-values>]
  [-clockout_reference_inputclock <clkout pins, mapped
clkkin pins>]
  [-clockout_frequency_multiplier <clkout pins, mapped freq
multiplier>]
  [-register]
  [-pll]
  [-divider]
  [-maskcaptureATspeed]
  [-scan_clock <name>]
  [-scan_clock_value <value>]
  [-scan_set <name>]
  [-scan_set_value <name>]
  [-scan_reset <name>]
  [-scan_reset_value <value>]
```

**NOTE:** *The `clock_shaper` constraint supports wildcard characters.*

**NOTE:** *See [Arguments used by the `dftDsmConstraintCheck\_01` rule](#) to view the arguments used by the `dftDsmConstraintCheck_01` rule.*

## Arguments

### General Arguments

The `clock_shaper` constraint has the following arguments:

**-name <module-name>**

The name of the module to be declared as a clock shaper. You can also specify an instance name as an input, if you have specified `-register` argument. This implies that instance-based `clock_shapers` are allowed only for flip-flops. Module names specified using this argument, are allowed for any user module. However, if the module is not a black box, then software forces it to be treated as black box and you would see the *InfoAnalyzeBBox* rule violation for that module.

**-clkkin <clkkin-pin-list>**

List of clock input pins of the clock shaper. Do not specify this argument, if you have specified the `-register` argument.

**-clkout <clkout-pin-list>**

List of clock output pins of the clock shaper. Do not specify this argument, if you have specified the `-register` argument.

**-reset <reset-pin>**

(Optional) Reset pin of the clock shaper. When this is activated, the clock shaper does not propagate any clock pulse through its output.

**-enable <en-pin-list>**

(Optional) List of control pins of the clock shaper. These pins should be activated for the clock shaper to propagate clock pulses. By default, a clock shaper is considered enabled.

**-freqcontrols <freq-ctrl-pins>**

(Optional) Pins that control or modify frequency of the clock pulse(s) propagating through the clock shaper.

**-freqmult <mult-values>**

(Optional) Multiplying values for a frequency of clock pulses propagating

through a clock shaper.

**-resetvalue <reset-value>**

A single 0 or 1 value. The value specifies the signal that activates the corresponding reset pin.

**-envalue <en-value-list>**

List of a space-separated single 0,1, or X (don't care) values where each value is for corresponding enable pin given through the -enable switch. For such values, clock shaper is enabled when each enable get its corresponding value.

You can also specify a list of space-separated string of values where each value string is of size of number of enable pins given. For such values, clock shaper is enabled when value at the enable pins satisfies any one value string.

**-freqcontrolvalues <values\_string>**

Binary strings (0s and 1s) that specify a combination to be applied to their respective pins. The length of the combination should be equal to the number of pins that it refers.

**-clockout\_reference\_inputclock <clkout pin, related clkin pins>**

Specifies a relation of clkout pin with clkin pins, based on frequency control values in case of multi-pin support.

**-clockout\_frequency\_multiplier <clkout pins, mapped freq multiplier>**

Specifies a multiplicative factor to an input frequency based on frequency control value so that clkout has a modified frequency.

**-register**

Applies the clock\_shaper constraint on hierarchical flip-flop instances or output net of the flip-flops. When you specify this option, you do not need to provide -clockin and -clockout fields because the clock pin of the flip-flop is treated as clockin and the output pin of the flip-flop is treated as clockout. In addition, the default value of the multiplication factor is 0.5. You can change the value of the multiplication factor by using the -clockout\_frequency\_multiplier option along with the

-register option. Also, you need to specify this argument to specify the instance name for the -name argument.

**-pll**

(Optional) Enables the clock shaper to work as a PLL. All the PLL checks are applied to this constraint.

**-divider**

(Optional) Enables the clock shaper to work as a clock divider.

**-scan\_clock <name>**

Specifies the name of clock shaper pin driving clock pin of flops inside clock shaper.

**-scan\_clock\_value <value>**

Accepts either `rising_edge` or `falling_edge`, as values, to specify clock pin phase.

**-scan\_set <name>**

Specifies the name of the clock shaper pin, which is driving the set pin of flip-flops inside clock shaper.

**-scan\_set\_value <value>**

Accepts either `high` or `low`, as values, to specify set pin phase.

**-scan\_reset <name>**

Specifies the name of the clock shaper pin, which is driving the reset pin of flip-flops inside clock shaper.

**-scan\_reset\_value <value>**

Accepts either `high` or `low`, as values, to specify set pin phase.

For more information on support for clock shaper with scannable flip-flops, refer to *Support for clock shaper with scannable flops* section in the *SpyGlass DFT Rules Reference Guide*.

## Arguments used by the `dftDsmConstraintCheck_01` rule

### **-testmodepins**

Specifies the pins that determine whether the clockshaper is enabled in a given test mode. The values of these pins for given test modes should be passed under the `scanshift` or `capture` mode.

### **-scanshift**

Binary strings (0s and 1s) that specify a combination to be applied to their respective pins in the `scanshift` mode. The length of the combination should be equal to the number of pins that it refers.

### **-capture**

Binary strings (0s and 1s) that specify a combination to be applied to their respective pins in the `capture` mode. The length of the combination should be equal to the number of pins that it refers.

### **-captureATspeed**

Binary strings (0s and 1s) that specify a combination to be applied to their respective pins in the `CaptureATspeed` mode. The length of the combination should be equal to the number of pins that it refers.

### **-maskcaptureATspeed**

This option masks the test mode pin values under `captureATspeed`. For any test mode pin, the enabling condition is not checked if the corresponding bit is one in `maskcaptureATspeed`.

This argument is applicable only to the `Atspeed_17_captureatspeed` rule. It does not impact `atspeed` clock propagation.

## Rules

The `clock_shaper` constraint is used by the following rules:

---

### **SpyGlass DFT Solution**

---

All rules

---

---

**SpyGlass DFT DSM Solution**


---

All rules	Specifically used by:	Atspeed_12, Atspeed_13, Atspeed_17_shift, Atspeed_17_capture and Atspeed_17_captureatspeed, dftDsmConstraintCheck_01
-----------	-----------------------	--

---

## Examples

Consider the following examples:

### Clock Shaper Without an Enable Pin

Consider the following `clock_shaper` constraint specification having a clock input pin and a clock output pin:

```
clock_shaper -name cg_cell
  -clkin clk -clkout clko
```

In this case, the `clk` pin is connected to the `clko` pin with frequency multiplier 1.

### Clock Shaper With a Single Enable Pin

Consider the following `clock_shaper` constraint specification having a single enable pin:

```
clock_shaper -name cg_cell
  -clkin clk -clkout clko
  -enable en -envalue 11000
```

Here, the `clk` pin is connected to the `clko` pin with frequency multiplier 1 when the `en` pin has the specified value. If the enable value is not satisfied, the `clko` pin remains at X (don't care).

### Clock Shaper With Multiple Enable Pins

Example 1

Consider the following `clock_shaper` constraint specification having multiple enable pins:

```
clock_shaper -name cg_cell
  -clkin clk -clkout clko
```



```
-freqcontrols en1 en2 en3 en4 en5
-freqcontrolvalues 111xx XX011
-freqmult 1 1
```

In this case, the clock shaper is enabled in the following two cases:

- The en1, en2, and en3 pins are equal to 111 irrespective of the values on the en4 and en5 pins.
- The en3, en4, and en5 pins are equal to 011 irrespective of the values on the en1 and en2 pins.

If neither of the above cases is satisfied, the device is disabled so that the output remains at X (don't care).

The above example also illustrates the use of don't care conditions on the enable pins.

### **Example 2**

Consider the following example:

```
clock_shaper -name cs1
  -clkin clk_in1
  -clkout clk_out1 clk_out2 clk_out
  -enable en1 en2 en3 -envalue 0 1 0
  -freqcontrols freq_cnt1 freq_cnt2 freq_cnt3 freq_cnt4
  -freqcontrolvalues x000 1001 1111
  -freqmult 1 1 1 1 1 1 1 1 1
```

In the above example, clock shaper is enabled when value at en1, en2 and en3 is 0, 1 and 0 respectively.

### **Example 3**

Consider the following example:

```
clock_shaper -name cs2
  -clkin clk_in1
  -clkout clk_out1 clk_out2 clk_out3
  -enable en1 en2 en3 -envalue 011 lxx
  -freqcontrols freq_cnt1 freq_cnt2 freq_cnt3 freq_cnt4
  -freqcontrolvalues x000 1001 1111
  -freqmult 1 1 1 1 1 1 1 1 1
```

In the above example, clock shaper is enabled when value at en1, en2

and en3 is 0, 1 and 1 respectively or value at en1 is 1

### **Clock Shaper With Multiple Clock-input and Clock-output Pins**

A clock shaper can have multiple clock-input and clock-output pins. The input to output mapping is illustrated in the following example:

```
clock_shaper -name cg_cell
  -clkin clk1 clk2 clk3 -clkout clko1 clko2 clko3
```

In the above example, the connections are parallel, that is, the clk1 pin is connected to the clko1 pin, the clk2 pin is connected to the clko2 pin, the clk3 pin is connected to the clko3 pin, with frequency multiplier 1.

**NOTE:** *If the length of the clkin pin list is not equal to the length of the clkout pin list, the clock\_shaper constraint specification is ignored and a warning message is reported.*

### **Clock Shaper With Frequency Division**

Consider the following divide-by-2 clock\_shaper constraint specification:

```
clock_shaper -name shaper
  -clkin clk -clkout clko -freqmult 0.5
```

In this case, the frequency of the clko pin is half of that of the clk pin.

Now consider the following clock\_shaper constraint specification which produces the same result as the above specification:

```
clock_shaper -name shaper
  -clkin clk -clkout clko
  -clockout_frequency_multiplier clko 0.5
```

**NOTE:** *If the length of the clkin pin list is not equal to the length of the clkout pin list, the clock\_shaper constraint specification is ignored and a warning message is reported.*

The advantage of using the -clockout\_frequency\_multiplier argument of the clock\_shaper constraint over the -freqmult argument is that it provides flexibility to handle more complex clock shapers.

**NOTE:** *The -clockout\_frequency\_multiplier argument of the clock\_shaper constraint has higher priority than the -freqmult argument.*

### Clock Shaper With Frequency Multiplication

Consider the following `clock_shaper` constraint specification:

```
clock_shaper -name cg_cell
  -clkin clk -clkout clko -freqmult 2.0
```

In this case, the frequency of the `clko` pin is twice that of the `clk` pin.

### Clock Shaper With Multiple Clock-output Pins With the Same Frequency Multiplication

Consider the following divide-by-2 `clock_shaper` constraint specification:

```
clock_shaper -name cg_cell
  -clkin clk1 clk2 clk3 -clkout clko1 clko2 clko3
  -freqmult 0.5
```

In this case, the `cg_cell` clock shaper has multiple clockin to clockout paths and the same divide-by-2 occurs on all the paths.

**NOTE:** *If the length of the `clkin` pin list is not equal to the length of the `clkout` pin list, the `clock_shaper` constraint specification is ignored and a warning message is reported.*

### Clock Shaper With Multiple Clock-output Pins With Different Frequency Multipliers

Consider the following `clock_shaper` constraint specification:

```
clock_shaper -name cg_cell
  -clkin clk1 clk2 clk3 -clkout clko1 clko2 clko3
  -clockout_frequency_multiplier clko1 1.2
  -clockout_frequency_multiplier clko2 1.5
  -clockout_frequency_multiplier clko3 1.7
```

In the above example, the three clock-output pins, `clko1`, `clko2`, and `clko3`, have different frequency multiplications, that is, 1.2, 1.5, and 1.7, respectively.

**NOTE:** *The clock-output pin that does not have a frequency multiplier specified, is assigned a multiplier of 1.0 by default.*

### Clock Shaper With a Single Clock-out Pin and Selectable Frequency

## Multipliers

Consider the following clock shaper with the selectable frequency multiplication capability:

```
clock_shaper -name var_multiplier
  -clkin cin -clkout cout
  -freqcontrols fc0 fc1 -freqcontrolvalues 00 01 10
  -freqmult 1.0 0.5 0.25
```

In this case, the behavior of the clock shaper is described in the following table:

fc0	fc1	Output
0	0	cout frequency = cin frequency
0	1	cout frequency = cin frequency * 0.5
1	0	cout frequency = cin frequency * 0.25
1	1	The clock_shaper constraint is disabled

**NOTE:** *If the length of the freqcontrols pin list is not equal to the length of the freqcontrolvalues pin list, the clock\_shaper constraint specification is ignored and a warning message is reported.*

## Clock Shaper With Multiple Clock-input/Clock-output Pairs

The frequency control syntax is also used for specifying the mapping between the input and output clocks for clock shapers that can handle multiple clocks simultaneously.

The following table lists the arguments used to define the clock shaper control port names, the active values on these ports, and the clockin to clockout mapping:

Argument of the clock_shaper constraint	Defines
-freqcontrols	The control pins on a clock shaper
-freqcontrolvalues	A list of values on the control pins
-clockout_reference_inputclock	An output clock pin and a list of input clock pins that can be mapped to this output clock

Please note that the number of sets of control values must match the number of input clocks. In addition, each set of values must be unique because SpyGlass searches the value list in the left to right order.

Now consider the following `clock_shaper` constraint specification with the selectable frequency multiplication capability:

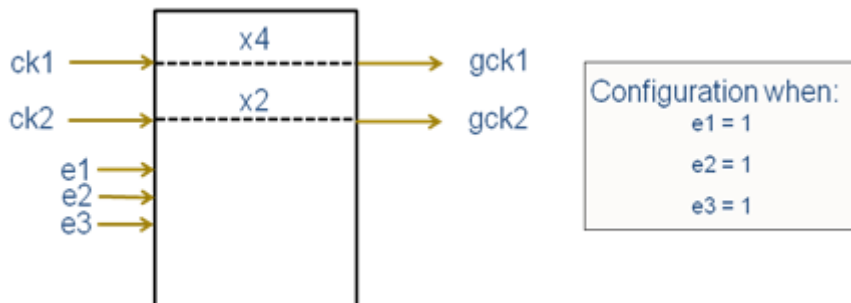
```
clock_shaper -name cg_cell
  -clkin clk1 clk2 clk3 -clkout clko1 clko2 clk03
  -freqmult 1.2 1.5 1.7
```

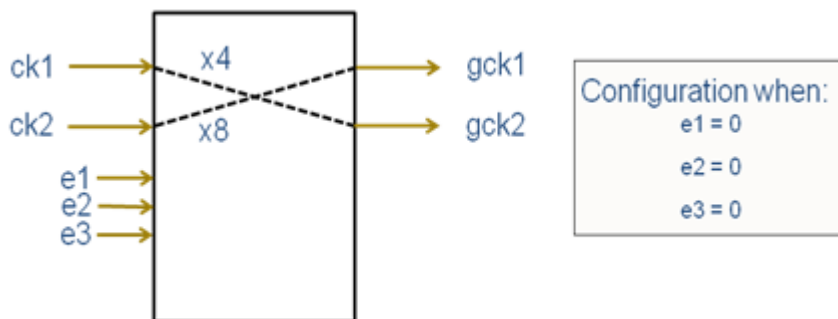
In the above example, a list of multipliers is specified. All frequency multiplications are applied in parallel, that is, the  $n^{\text{th}}$  multiplier in the list is applied to the  $n^{\text{th}}$  clockin/clockout pair.

**NOTE:** *If the lengths of the clkin pin list, the clkout pin list, and the freqmult list are not equal, the clock\_shaper constraint specification is ignored and a warning message is reported.*

### Clock Shaper With Selectable Mapping of Clock-input Pins to Clock-output Pins

```
clock_shaper -name CCN
  -clkin ck1 ck2 -clkout gck1 gck2
  -freqcontrols e1 e2 e3 -freqcontrolvalues 111 000
  -clockout_reference_inputclock gck1 ck1 ck2
  -clockout_reference_inputclock gck2 ck2 ck1
```





**NOTE:** *If the length of the `clockout_reference_inputclock` pin list is not equal to the length of the `clkout` pin list, the `clock_shaper` constraint specification is ignored and a warning message is reported.*

### Clock Shaper With Selectable Clock-input/Clock-output Pairs and Different Frequencies

The syntax for clock shapers can be used to describe complex devices that not only have selectable clockin to clockout mapping but also have different frequency multipliers for each selection.

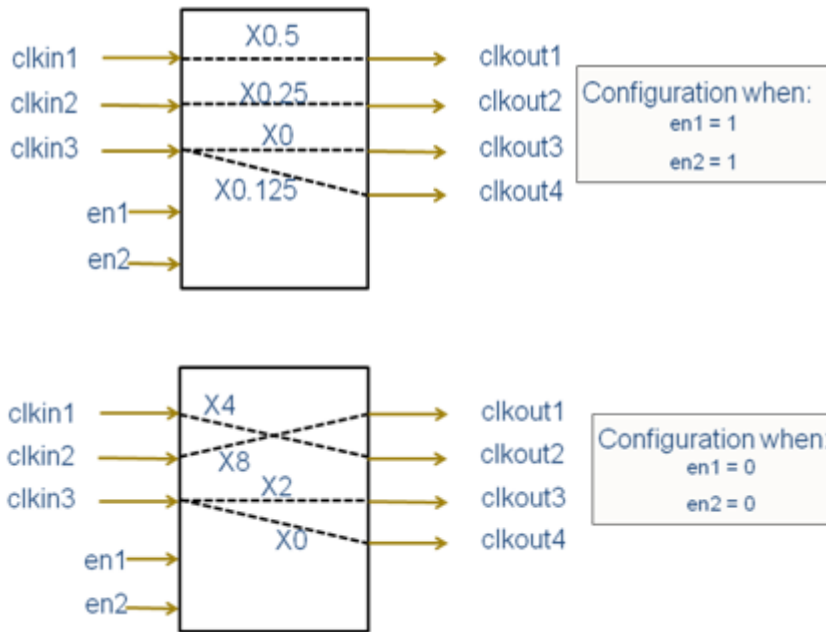
The syntax of the `-clockout_frequency_multiplier` argument of the `clock_shaper` constraint specifies a clockout pin and a list of frequency multipliers to be used in the same order as the values specified in the `-freqcontrolvalues` argument.

Consider the following `clock_shaper` constraint specification in which the input clocks are mapped to the output clocks with different frequency multipliers:

```
clock_shaper -name OCC
  -clkin clkin1 clkin2 clkin3
  -clkout clkout1 clkout2 clkout3 clkout4
  -clockout_reference_inputclock clkout1 clkin1 clkin2
  -clockout_reference_inputclock clkout2 clkin2 clkin1
  -clockout_reference_inputclock clkout3 clkin3 clkin3
  -clockout_reference_inputclock clkout4 clkin3 clkin3
  -clockout_frequency_multiplier clkout1 0.5 4
  -clockout_frequency_multiplier clkout2 0.25 8
  -clockout_frequency_multiplier clkout3 0 2
```

## SpyGlass Design Constraints

```
clkout4 0.125 0
-freqcontrols en1 en2 -freqcontrolvalues 11 00
```



### Clock Shaper With PLL Support

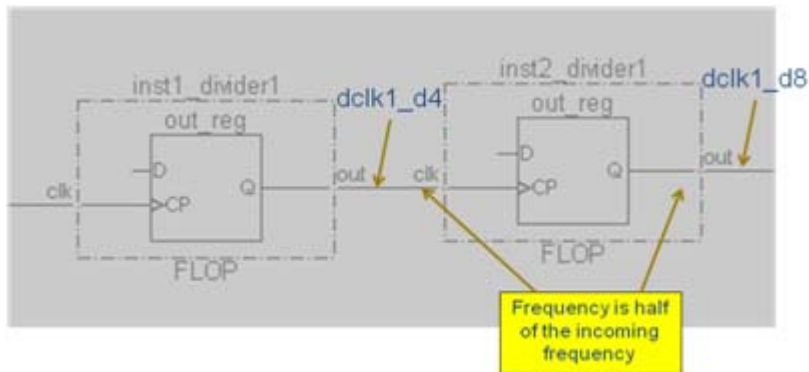
Consider the following `clock_shaper` constraint specification in which a clock shaper is specified as a PLL:

```
clock_shaper -pll -name myPLL
  -clkin ref_clk -clkout clk1 clk2
  -reset reset -resetvalue 0
```

The SpyGlass DFT DSM solution allows the use of PLLs. Please refer to rules, such as `PLL_01` and `PLL_02` for details.

### Clock Shaper With Register Support

Consider the following conventional flip-flop based divide-by-2 counter:



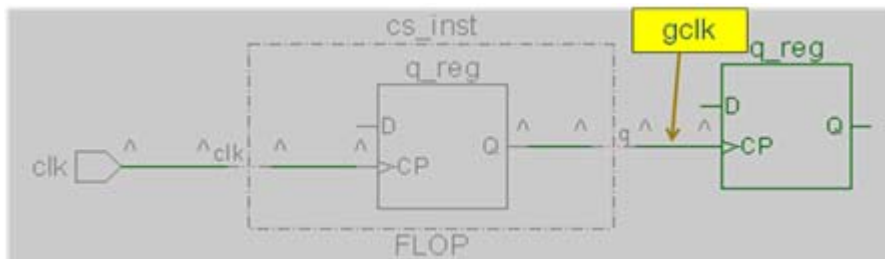
The above counter can be constrained by using the `-register` argument of the `clock_shaper` constraint as shown below:

```
clock_shaper -name dclk1_d4 -register
clock_shaper -name dclk1_d8 -register
```

**NOTE:** Please note the following points:

- 📖 The `-register` argument of the `clock_shaper` constraint specifies that the module is a flip-flop based frequency divider.
- 📖 The default multiplication value is `0.5`, which can be overridden.
- 📖 The name for the clock shaper is the net connected to the output of the flip-flop (the flip-flop is not explicitly instantiated in the RTL).
- 📖 In case of an instantiated technology library cell, the constraint can be applied on the instance.
- 📖 The `clock_shaper` constraint cannot be applied on a level-sensitive latch.

Consider the following schematic generated by the `Info_testclock` rule:





## SpyGlass Design Constraints

The clock\_shaper constraint specification in this case is as follows:

```
clock_shaper -name gclk -register
```

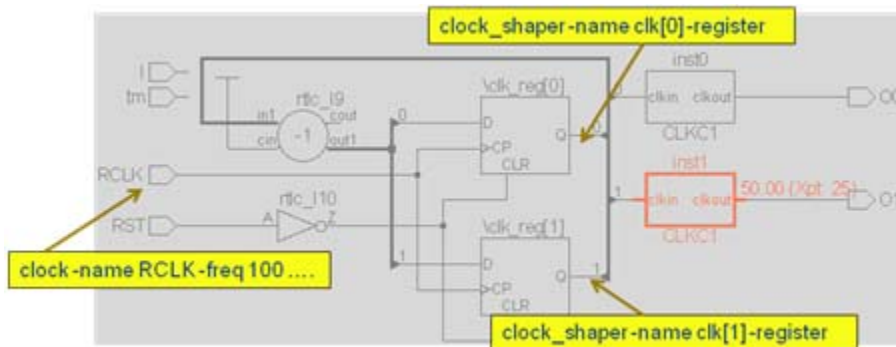
In the above schematic, the clock propagation through the flip-flop-based clock shapers is represented by the carrot (^) symbol.

Now consider the following multiplier with frequency multiplication value different from 0.5:

```
always @(posedge RCLK or negedge RST) begin
    if(!RST)
        clk <= 2'h0;
    else
        clk <= clk - 1'b1;
end
```

To model such a multiplier, use the `-clockout_frequency_multiplier` argument of the `clock_shaper` constraint.

The following figure specifies a countdown counter that can manipulate the frequency functionally:



The output frequencies of the above counter are as follows:

Clock Pin	Output Frequency of the Counter
clk[0]	50
clk[1]	25

The above result can be modeled in the following two ways:

```
clock_shaper -name clk[1] -register
-clockout_frequency_multiplier 0.25
```

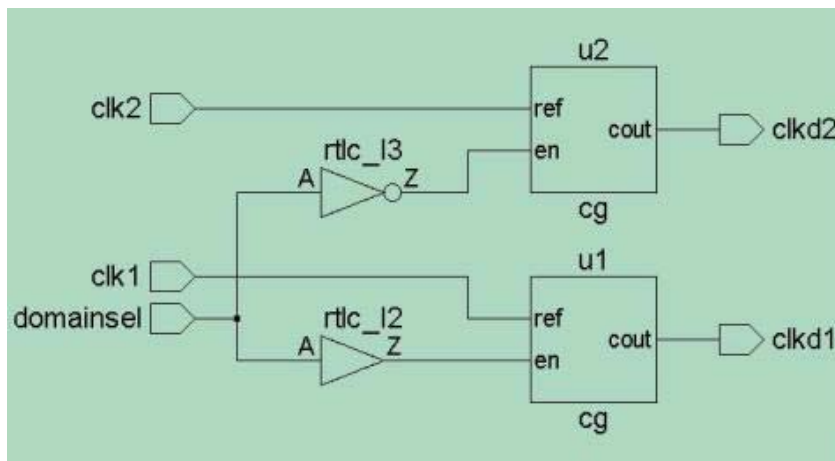
Or

```
clock_shaper -name clk[1] -register
-freqmult 0.25
```

**NOTE:** *If both the -clockout\_frequency\_multiplier and -freqmult arguments of the clock\_shaper constraint are specified, the -clockout\_frequency\_multiplier is used and the -freqmult argument is ignored.*

### Clock Shaper With External Enable Decode

```
clock_shaper -name cg -clkkin ref -clkout cout
-enable en -envalue 1
```



**FIGURE 31.** Clock shaper with external enable decode

### Clock Shaper With Multiple Clock-out and Clock-in Pins

```
clock_shaper -name CST -clkkin clkkin1 clkkin2 clkkin3 -
clkout clkout1 clkout2 clkout3
-clockout_reference_inputclock clkout1 clkkin2 clkkin1 clkkin3
-clockout_reference_inputclock clkout2 clkkin1 clkkin3 clkkin3
```

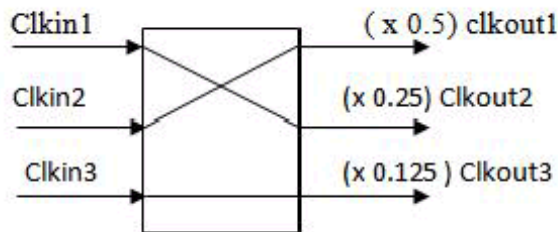
## SpyGlass Design Constraints

```

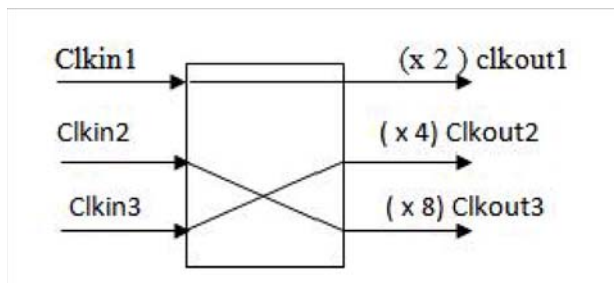
-clockout_reeference_inputclock clkout3 clkin3 clkin2 clkin1
-clockout_frequency_multiplier clkout1 0.5 2 2
-clockout_frequency_multiplier clkout2 0.25 4 4
-clockout_frequency_multprier clkout3 0.125 8 0
-freqcontrols en1 en2 en3
-freqcontrolvalues 110 001 111

```

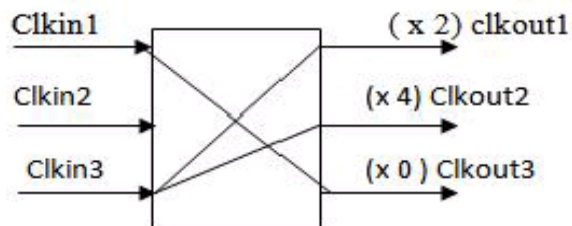
The following figure illustrates the input-output mapping of clock shaper, when -freqcontrols pins gets 110 under capture at-speed mode:



The following figure illustrates the input-output mapping of clock shaper, when -freqcontrol pins get 001 under capture at-speed mode:



The following figure illustrates the input-output mapping of clock shaper, when -freqcontrol pins get 111 under capture at-speed mode:



## clockgating

### Purpose

The `clockgating` constraint is used to specify gating conditions for test clocks.

**NOTE:** *Three test mode options are supported: `scanshift`, `capture`, and `captureATspeed`. Because these categories are not at-speed clock specific, additional data is needed to specify the gating signals that are specific to an at-speed test clock. See rule `Atspeed_07` for an example.*

### Product

SpyGlass DFT DSM solution

### Syntax

```
clockgating
  -name <clk-name>
  -pin <lst_signal_names>
  -value <values>
```

### Arguments

The `clockgating` constraint has the following arguments:

**-name <clk\_name>**

The name of the test clock, which is gated.

**-pin <lst\_signal\_names>**

List of signal names that act as a gating signal for the specified test clock.

**-value <values>**

List of values corresponding to the gating signals.

## Rules

The `clockgating` constraint is used by the following rule:

---

**SpyGlass DFT DSM Solution**

Atspeed\_07

---

## complex\_cell

### Purpose

A complex cell is a module that controls clock pulse propagation in a design, by enabling/disabling it under different test mode conditions. Complex cells are instances of `clock_shaper`, and are treated as black boxes by simulation.

**NOTE:** *The `complex_cell` constraint will be deprecated in a future SpyGlass release. Please make use of [clock\\_shaper](#) constraint in place of this constraint.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

```
complex_cell
  -name <cell-name>
  -tclkinport <clkin-pin>
  -tclkoutport <clkout-pins>
  -testmodepins <tm-pins>
  -scanshift <values>
  -capture <values>
```

```
-captureStatic <values>  
-captureATspeed <values>  
[ -maskcaptureATspeed <values> ]
```

## Arguments

The `complex_cell` constraint has the following arguments:

**-name <cell-name>**

Name of the module to be declared as a complex cell.

**-tclkinputport <clkinput-pin>**

Clock input pin of the complex cell (can be one pin only).

**-tclkoutputport <clkoutput-pins>**

Clock output pin of the complex cell (can be one pin only).

**-testmodepins <tm-pins>**

Pins that determine whether the complex cell is enabled in a given test mode. Values of these pins for given test modes should be passed under the arguments, such as `-scanshift` and `-capture`.

**<values>**

Binary strings (0 and 1) that specify a combination to be applied to their respective pins. The length of the combination should be equal to the number of pins that it refers.

For example, if there are three test mode pins, arguments, such as `-scanshift` and `-capture`, should have values of length three.

## Rules

The `complex_cell` constraint is used by the following rules:

---

**SpyGlass DFT Solution**

---

All rules

---

---

**SpyGlass DFT DSM Solution**


---

All rules	Specifically used by:	Atspeed_12, Atspeed_13, Atspeed_17_shift, Atspeed_17_capture, and Atspeed_17_captureatspeed
-----------	-----------------------	--

---

## compressor

### Purpose

This constraint is used to specify test data compressors and data output pins.

The compressor constraint is used when data from internal chains is driving an on-chip compressor.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax of the `compressor` constraint is as follows:

```
compressor
  -name <mod-name| instance>
  -dataout <list of compressor module ports>
```

### Arguments

**-name <mod-name| instance>**

Name of an instance or a module. If the `-name` argument is a module name, then all instances of that name will be defined as compressors with the same data output port names. The `-dataout` ports are assumed as the compressed scan out data.

**-dataout <list of compressor module ports>**

List of output ports of compressor driving primary outputs.

## Examples

If the `-name` field contains a module name, then the constraint information will be applied to all instances of this name unless another compressor constraint refers to a specific instance of the same module name. For example, consider the SGDC listed below in which `top.u1` is an instance of module `compA`:

```
compressor -name compA -datain depins[31:0]
compressor -name top.u1 -datain depins[15:0]
```

In this example, `top.u1` is intended to have 16 tail registers whereas all other instances of `compA` have 32 head registers.

## Rules

The `compressor` constraint is used by the following rules:

SpyGlass DFT DSM Solution				
TC_01	TC_02	TC_03	TC_04	TC_05

## dbist

### Purpose

The `dbist` constraint specifies the set of conditions, both pins and values, that when simulated will force the circuit into a state for DBIST mode.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `Dbist`.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `dbist` constraint is as follows:

```
dbist
  -name <name>
```



-value <value>

**NOTE:** *The dbist constraint supports wildcard characters.*

## Arguments

The dbist constraint has the following arguments:

### -name <name>

Complete hierarchical name of DBIST port/pin.

The pin can be a primary pin as well as an internal pin.

You can specify a single port/pin's full hierarchical name or a space-separated list of full hierarchical port/pin names. The port/pin name supports wildcard.

For primary ports, you can also specify the simple port name as in the following example:

```
current_design top
  dbist -name in15 ...
```

### -value <value>

Value list for the DBIST pin.

The value list is the sequence of one or more values (each value being 0, 1, X, Z, or a combination) that when applied to the DBIST pin, will cause the circuit to enter DBIST mode.

## Rules

The dbist constraint is used by the following rules:

---

### SpyGlass DFT Solution

---

Latch_16	Tristate_17
----------	-------------

---

## decompressor

### Purpose

This constraint is used to specify a test data decompressor and its data input pins.

## Product

SpyGlass DFT DSM solution

## Syntax

The syntax of the `dbist` constraint is as follows:

```
decompressor
  -name <mod-name| instance>
  -datain <list of decompressor module ports>
```

## Arguments

**-name <mod-name| instance>**

Name of an instance or a module. If the `-name` field contains a module name, the constraint information is applied to all instances of this name unless another decompressor constraint refers to a specific instance of the same module name.

**-datain <list of decompressor module ports>**

List of input ports of decompressor driven by primary inputs.

## Examples

Consider the following example:

```
Decompressor -name decompA -datain depins[31:0]
Decompressor -name top.u1 -datain depins[15:0]
```

In the SGDC listed above, `top.u1` is an instance of module `decompA`.

## Rules

The `decompressor` constraint is used by the following rules:

SpyGlass DFT DSM Solution				
TC_01	TC_02	TC_03	TC_04	TC_05

## define\_clock\_tree

### Purpose

The `define_clock_tree` constraint is used to approximate the buffering in the clock tree and to apply the correction factor on the clock pin capacitance of the flop.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax of the `define_clock_tree` constraint is as follows:

```
define_clock_tree
  -name <clock-netname>
  -leaf_fanout <fvalue>
  -intermediate_fanout <fvalue>
  -intermediate_buffer <im-buffer/inverter-cellname>
  -leaf_buffer <leaf-buffer/inverter-cellname>
  [-leaf_cap_per_fanout <fvalue> ]
  [ -wire_cap_correction_factor <fvalue> ]
  [ -pin_cap_correction_factor <fvalue> ]
  [ -root_buffer <root-buffer/inverter-cellname> ]
  [ -root_buffer_lib <root-buffer/inverter-libname> ]

  [ -intermediate_buffer_lib <im-buffer/inverter-libname>
]
  [ -leaf_buffer_lib <leaf-buffer/inverter-libname> ]
  [ -icgc_fanout <int> ]
  [ -root_fanout <fvalue> ]
  [ -icgc_cell <icgc-cellname> ]
  [ -icgc_cell_lib <icgc-libname> ]
```

**NOTE:** *The `icgc_cell` and `icgc_cell_lib` arguments will be supported in a future release.*

## Arguments

### **-name <clock-netname>**

Specifies the RTL net name of the clock source.

**NOTE:** *This argument supports wildcard characters. Refer to [Example 4](#) for more information.*

### **-leaf\_fanout <fvalue>**

Specifies a floating-point value as the average fan-out of the leaf-level buffer.

The specified value must be greater than or equal to 2.

**NOTE:** *You must specify the -leaf\_fanout argument if the define\_clock\_tree constraint is used to calculate the buffering in the clock tree.*

### **-leaf\_cap\_per\_fanout <fvalue>**

(Optional) Specifies the capacitance per fanout of nets connecting the leaf buffer to flip-flops. Also, range of fanout multiple is 0 to INF.

### **-wire\_cap\_correction\_factor <fvalue>**

(Optional) Specifies a correction factor (floating-point value) to scale the capacitance obtained from the wireload model.

The default value is 1 . 0.

### **-pin\_cap\_correction\_factor <fvalue>**

(Optional) Specifies a correction factor (floating-point value) to scale the capacitance of the fan-out clock pin of the net. Note that the fan-out clock pin of the net is also the fan-in of the flop.

The default value is 1 . 0.

### **-root\_buffer <root-buffer/inverter-cellname>**

(Optional) Specifies the cell name to be used as buffer/inverter at the root level.

**NOTE:** *You must specify the -root\_fanout argument of the*

`define_clock_tree` constraint with the `-root_buffer` argument of the constraint.

**-root\_buffer\_lib <root-buffer/inverter-libname>**

(Optional) Specifies the library name of the buffer/inverter at the root level.

**NOTE:** *You must specify the `-root_buffer` argument of the `define_clock_tree` constraint with the `-root_buffer_lib` argument of the constraint.*

**-intermediate\_buffer <im-buffer/inverter-cellname>**

Specifies the cell name to be used as buffer/inverter at the intermediate levels.

**NOTE:** *You must specify the `-intermediate_fanout` argument of the `define_clock_tree` constraint with the `-intermediate_buffer` argument of the constraint.*

**-intermediate\_buffer\_lib <im-buffer/inverter-libname>**

(Optional) Specifies the library name of the buffer/inverter at the intermediate levels.

**NOTE:** *You must specify the `-intermediate_buffer` argument of the `define_clock_tree` constraint with the `-intermediate_buffer_lib` argument of the constraint.*

**-leaf\_buffer <leaf-buffer/inverter-cellname>**

Specifies the cell name to be used as buffer/inverter at the leaf level.

**NOTE:** *You must specify the `-leaf_fanout` argument of the `define_clock_tree` constraint with the `-leaf_buffer` argument of the constraint.*

**-leaf\_buffer\_lib <leaf-buffer/inverter-libname>**

(Optional) Specifies the library name of the buffer/inverter at the leaf level.

**NOTE:** *You must specify the `-leaf_buffer` argument of the `define_clock_tree` constraint with the `-leaf_buffer_lib` argument of the constraint.*

**NOTE:** *If a root buffer or inverter is not specified, SpyGlass uses an intermediate buffer/inverter specified in the `define_clock_tree` constraint.*

**-icgc\_fanout <int>**

(Optional) Specifies the maximum fan-out of all the ICGCs, where all fan-outs are going to sink, in the netlist. If there are no such ICGCs, the `leaf_fanout` is taken as the `icgc_fanout`.

**-root\_fanout <fvalue>**

(Optional) Specifies a floating-point value as the average fan-out of the root level buffers/inverters

**-intermediate\_fanout <fvalue>**

Specifies a floating-point value as the average fan-out of the intermediate level buffers/inverters

## Examples

### Example 1

Consider the following `define_clock_tree` constraint specification:

```
define_clock_tree
  -name CLKNET
  -leaf_fanout 16.00
  -wire_cap_correction_factor 2.0
  -root_fanout 4.00
  -intermediate_fanout 4.00
  -intermediate_buffer BUFI
  -leaf_buffer BUFL
```

In the above `define_clock_tree` constraint, the root-level buffer/inverter is not specified. In this case, SpyGlass uses the intermediate-level buffer/inverter (BUFI) as the root-level buffer/inverter.

Therefore, the buffers/inverters at each level will be as follows:

Buffer/Inverter at Level	Buffer/Inverter
Intermediate-level(s)	BUFI
Leaf-level	BUFL
Root-level	BUFI

### Example 2

Consider a clock tree having 1000 pins with the following specification of the `define_clock_tree` constraint:

```
define_clock_tree
  -name CLKNET
  -leaf_fanout 16.00
  -wire_cap_correction_factor 2.0
  -intermediate_fanout 4.00
  -root_fanout 4.00
  -root_buffer BUFR
  -intermediate_buffer BUFI
  -leaf_buffer BUFL
```

In this case, the number of drivers required at each level will be as follows:

- **Level 0:** Number of BUFL drivers =  $1000/16 = 63$
- **Level 1:** Number of BUFI drivers =  $63/(16/4) = 16$
- **Level 2:** Number of BUFI drivers =  $16/4 = 4$
- **Level 3:** Number of BUFR driver =  $4/4 = 1$

Now, consider that the wireload for nets with fan-out as 16 is 4 pF and the wire capacitance correction factor is 2. In this case, the capacitance of nets will be  $4 * 2 = 8$  pF.

### Example 3

Consider a clock tree having 1000 pins with the following specification of the `define_clock_tree` constraint:

```
define_clock_tree
  -name CLKNET
  -leaf_fanout 16.00
```

```
-wire_cap_correction_factor 2.0  
-root_fanout 4.00  
-intermediate_fanout 4.00  
-root_buffer BUFR  
-intermediate_buffer BUFI  
-leaf_buffer BUFL  
-pin_cap_correction_factor 3.5
```

In this case, the specified pin cap correction for the CLKNET is 3.5.



**Example 4**

Consider the following example that shows the extracted `define_clock_tree` constraint from a netlist by the PECLKTREE rule:

```
define_clock_tree
  -name "*"
  -root_buffer BUFX8
  -intermediate_buffer BUFX2
  -leaf_buffer BUFXL
  -leaf_fanout 16.00
  -intermediate_fanout 8.00
  -root_fanout 2.00
  -leaf_cap_per_fanout 0.100
```

In this example, the clock tree in the netlist design has on average 16.00 as leaf fan-out and the average capacitance of every fan-out of the leaf nets is 0.10pf. The netlist has on average an intermediate fan-out of 8.00 and the root fan-out of 2.00. In addition, the buffer names specified in this example are determined from the clock tree of the netlist.

**Rules**

SpyGlass Power Estimation and SpyGlass Power Reduction solutions				
PEPWR01	PEPWR02	PEPWR03	PEPWR05	PEPWR13
PEPWR14	poweraudit			

**define\_illegal\_input\_values****Purpose**

If illegal values are specified on a set of inputs, all input values not specified for that set of inputs are considered legal.

**Product**

SpyGlass DFT solution

## Syntax

The syntax of the `define_illegal_input_values` is as follows:

```
define_illegal_input_values -name  
  -name <node_names>  
  -value <values>
```

## Arguments

### **-name <node\_names>**

Specifies the list of node names. The names may refer to internal nodes or top-level ports.

### **-value <values>**

Specifies the list of illegal value combinations. It is the only set of invalid simulation sequence.

## Examples

Consider the following example:

```
define_illegal_input_values -name I1 I2 -values 00 11
```

In the above example, for nodes, I1 and I2, the invalid set of value combination is 00 and 11. All the values except 00 and 11 are considered valid.

## Rules

The `define_illegal_input_values` constraint is used by the following rules:

---

**SpyGlass DFT Solution**

---

Async\_06

---

## define\_legal\_input\_values

## Purpose

If legal input values are specified for a set of inputs, all input values not specified for that set of inputs are considered illegal.

## Product

SpyGlass DFT solution

## Syntax

The syntax of the `define_legal_input_values` is as follows:

```
define_legal_input_values -name  
    -name <node_names>  
    -value <values>
```

## Arguments

### **-name <node\_names>**

Specifies the list of node names. The names may refer to internal nodes or top-level ports.

### **-value <values>**

Specifies the list of legal value combinations. It is the set of valid simulation sequence.

## Examples

Consider the following example:

```
define_legal_input_values -name I1 I2 -values 10 01
```

In the above example, for nodes, I1 and I2, only the value combinations, 10 or 01, are allowed. All the values except 10 and 01 are considered invalid.

## Rules

The `define_legal_input_values` constraint is used by the following

rules:

---

**SpyGlass DFT Solution**

---

Async\_06

---

## define\_library\_group

### Purpose

Defines the groups of Synopsys technology libraries at different PVT values. Library groups need to be defined because same cell definition is possible in multiple libraries at different PVT values.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `define_library_group` constraint is as follows:

```
current_design <top-du-name>
  define_library_group
    -name <lib-grp-name>
    -libname <lib-name-list>
```

### Arguments

**<top-du-name>**

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**-name <lib-grp-name>**

Name of the library group. The specified name should be unique.

**-libname <lib-name-list>**

Name of the library/libraries that you want to add in the specified library group.

**NOTE:** You can also use the `-libname` argument as `-libnames` to specify a single library or a group of libraries.

Consider the following example, for the libraries: L1, L2, L3, L4 and L5, use the `define_library_group` constraint as:

```
define_library_group -name G1 -libname L1 L2
```

```
define_library_group -name G2 -libname L3 L4
```

**NOTE:** Refer to the SGDC command [use\\_library\\_group](#) to specify library groups to various design hierarchies.

**Rules**

The `define_library_group` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

**define\_macro**

Enables you to create dynamic macros in SpyGlass DFT. You can set your own set of macros (User-Defined Macros or UDMs) using this constraint.

Macros created using this constraint are similar to other dynamic macros, that is, Tcl-based UDMs.

All the macros defined using this constraint can be further specified through `-type`, `-from_type`, `-to_type`, and/or `-except_type` arguments of other constraints (`test_mode` & `require*`). It is similar to other macros (static-macros and Tcl-based UDMs) being provided as input.

**Product**

SpyGlass DFT solution

## Syntax

The following is the syntax for the `define_macro` constraint:

```
define_macro
  -macro <udm_name>
  [ -name <node_name> ]
  [ -type <design_object_type> ]
  [ -except <except_node_name> ]
  [ -except_type <except_design_object_type> ]
  [ -filter_in_name <include_node_name> ]
  [ -filter_in_type <include_design_object_type> ]
  [ -cmt <cmt_expression> ]
  [ -filter_in_cmt <include_cmt_expression> ]
  [ -ignorecase ]
```

## Arguments

### **-macro <udm\_name>**

(Mandatory) Name of user-defined macro (UDM)

Name of macro should be unique, that is, it should meet the following conditions :

- It should not match with any static-macro (such as INPUT\_PORTS, OUTPUT\_PORTS) or tcl-based macro name
- Multiple specifications of macro are not allowed

### **-name <node\_name>**

(Optional) The name can be a top-module port, or any internal net name, or terminal name.

**NOTE:** *Wildcard and scoping is also allowed. For example, "my\_module:\*visa\*", "\*\*powergood\*". Also, it is recommended to specify at least one of the following fields: -name, -type or -cmt.*

### **-type <design\_object\_type>**

(Optional) Same as <node\_name> but it takes only macros as input.

**NOTE:** *Both static-type and dynamic-type macros are supported*

**-except <except\_node\_name>**

(Optional) Same as <node\_name> but defines design nodes that are to be excluded.

**-except\_type <except\_design\_object\_type>**

(Optional) Same as <design\_object\_type> but defines design nodes that are to be excluded.

**-filter\_in\_name <include\_node\_name>**

(Optional) Same as <node\_name> but defines design nodes that are to be included.

**-filter\_in\_type <except\_design\_object\_type>**

(Optional) Same as <design\_object\_type> but defines design nodes that are to be included.

**-cmt <cmt\_expression>**

(Optional) Same as <node\_name> but takes constraint\_message\_tag expression as input nodes for which expression holds true.

**-filter\_in\_cmt <include\_cmt\_expression>**

(Optional) Same as <cmt\_expression> but includes design nodes for which constraint\_message\_tag expression holds true

**-ignorecase**

(Optional) to ignore the case for <nodename> specified as -name/-except/-filter\_in\_name

**NOTE:** *It applies on all fields which take design\_node\_name as input*

**Examples****Example 1**

Consider below constraint specifications:

```
define_macro -macro my_macro -name "*powergood*" -
ignorecase test_mode -type my_macro -value
```

In the above example, my\_macro is the user-specified macro which is

passed as `-type` field input for the `test_mode` constraint.

### Example 2 : OR of `-name` and `-type`

Consider the following constraint description:

```
define_macro -macro my_macro1 -name "*powergood*" -type
INPUT_PORTS -ignorecase
```

In the above example, `my_macro1` contains all input ports and objects matching with `*powergood*` (case-insensitive). This signifies an OR operation.

### Example 3: AND (through `filter_in`) of `-name` and `-type`

Consider the following constraint description:

```
define_macro -macro my_macro2 -filter_in_name
"*powergood*" -type INPUT_PORTS -ignorecase
```

```
define_macro -macro my_macro3 -name "*powergood*" -
filter_in_type INPUT_PORTS -ignorecase
```

In the above example, `my_macro2` and `my_macro3` will contain all input ports matching with `*powergood*` (case-insensitive). This signifies an AND operation.

### Example 4: Selection based on `-cmt`

Consider the following constraint description:

```
define_macro -macro my_macro4 -cmt "CHK_1:PASS &&
CHK_2:FAIL || CHK_3:PASS"
```

In the above example, `my_macro4` contains all design objects corresponding to the given tag expression, that is, `- CHK_1:PASS && CHK_2:FAIL || CHK_3:PASS`, whereas, `CHK_1`, `CHK_2` and `CHK_3` are the `constraint_message_tags`.

### Example 5: AND (through `filter_in`) of `-type` and `-cmt`

Consider the following constraint description:

```
define_macro -macro my_macro5 -type INPUT_PORTS -
filter_in_cmt "CHK_1:PASS || CHK_2:PASS"
```

In the above example, `my_macro5` contains all such input ports for which



the tag expression, that is, - CHK\_1:PASS || CHK\_2:FAIL holds true.

### Example 6: Defining a UDM from already defined UDM

Consider the following constraint description:

```
define_macro -macro my_macro6 -name "*powergood*" -
ignorecase
```

```
define_macro -macro my_macro7 -type my_macro6 -name
"my_module::*visa"
```

In the above example, my\_macro7 contains all design objects matching with \*powergood\* (case-insensitive) and objects corresponding to my\_module::\*visa\*.

### Example 7: Exclusion based on type

Consider the following constraint description:

```
define_macro -macro my_macro8 -name "*powergood*" -
except_type "INPUT_PORTS" "OUTPUT_PORTS" -ignorecase
```

In the above example, my\_macro8 contains all internal design objects matching with \*powergood\* (case-insensitive).

### Example 8: Usage of UDM in test\_mode

Consider the following constraint description:

```
define_macro -macro my_macro -name "*powergood*" -
filter_in_type INPUT_PORTS -ignorecase
```

```
test_mode -type my_macro -value 1
```

A value 1 is defined (as test\_mode) on all input ports for which name matches with \*powergood\* (case-insensitive).

### Example 9: Mandatory field -macro missing

Consider the following constraint description:

```
define_macro -name obj1
```

In the above example, an SGDC syntax error for missing mandatory field, -macro, is reported.

**Example 10: Invalid macro name**

Consider the following constraint description:

```
define_macro -macro "" -name obj1 ## Missing macro name
```

```
define_macro -macro INPUT_PORTS -name obj2 ## static macro
name used
```

```
define_macro -name "M&" -name obj3 ## macro name can only
contain 'a-z', 'A-Z', '0-9' and '_'
```

**Example 10: Undefined macro**

Consider the following constraint description:

```
define_macro -macro my_macro
```

In the above example, the `define_macro` constraint must have at least one of the following fields defined: `-name`, `-type`, and `-cmt`

**Example 11: Unique key violation**

Consider the following constraint description:

```
define_macro -macro my_macro1 -name obj1
define_macro -macro my_macro1 -name obj2
```

In the above example, the `define_macro` constraint is already specified for the `my_macro1` object, which is of `UNIQUE` type field, `-macro`.

**Rules**

The `define_macro` constraint is used by the following rules:

**SpyGlass DFT solution**

dftSGDCDefineMacroCh eck_01	dftSGDCDefin eMacroCheck _02
--------------------------------	------------------------------------

**define\_reset\_order**

## Purpose

Specifies the reset order, which determines the flow of data from one reset to another reset.

## Product

SpyGlass CDC solution

## Syntax

The syntax to specify the `define_reset_order` constraint is as follows:

```
current_design <du-name>
define_reset_order
  -from <from-rst-list>
  -to <to-rst-list>
```

## Arguments

### **-from <from-rst-list>**

Specifies the name of the source resets in the reset ordering. You can specify a space-separated list of hierarchical reset net names in this argument.

This argument supports wildcard characters.

**NOTE:** *Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.*

### **-to <to-rst-list>**

Specifies the name of the destination resets in the reset ordering. You can specify a space-separated list of hierarchical reset net names in this argument.

This argument supports wildcard characters.

**NOTE:** *The resets specified in the above arguments should match with one of the asynchronous resets specified in the `reset` constraint or inferred through the `use_inferred_resets` parameter.*

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression with the hierarchies. For example, the `top.*.rst1` expression is matched to `top.u1.rst1` and `top.u1.u2.rst1`. By default, the `define_reset_order` constraint matches only `top.u1.rst1`. Setting the value of the `hier_wild_card` parameter to `yes` runtime performance of the `define_reset_order` constraint is impacted.

## Examples

Consider the following examples:

### Example1

```
define_reset_order -from R1 -to R2
define_reset_order -from R1 -to R3
```

The above example defines the flow of data from R1 to R2 and from R1 to R3.

### Example2

```
define_reset_order -from RST1 -to RST2
define_reset_order -from RST2 -to RST1
```

The above example shows bidirectional reset ordering. Such cases are reported by the `SGDC_05` rule.

### Example3

Consider an example in which the `f1` flip-flop has the `R1` reset pin, and the `f2` flip-flop has the `R2` reset pin. Also, consider that `R1` comes from primary resets, `r1`, `r2`, and `r3`, and `R2` comes from `r1`, `r2`, `r4`, and `r5`. In this case, the design is fine if every reset that exists in fan-in of `R1` but not `R2` (`R1-R2`) happens before (through a `define_reset_order` constraint) every reset in the fan-in of `R2`.

This guarantees that asynchronously resetting flip-flop, `f1`, does not cause metastability on flip-flop, `f2`. If the specified condition is not met, a violation will be reported. For example, the reset ordering is fine in the following case:

```
define_reset_order -from r3 -to r1
```

```
define_reset_order -from r3 -to r2
define_reset_order -from r3 -to r4
define_reset_order -from r3 -to r5
```

**NOTE:** A violation is reported if the first flip-flop has preset/clear pins and the second flip-flop does not have any of them.

#### Example4

The following example specifies a reset crossing from top.RST1 to top.RST2:

```
define_reset_order -from top.RST1 -to top.RST2
```

#### Example5

The following example specifies reset ordering from top.RST1 to top.RST2 and top.RST3:

```
define_reset_order -from top.RST1 -to top.RST2 top.RST3
```

#### Example6

The following example specifies a reset ordering between flip-flops where source receives RST1 and RST2 and destination receives RST3 and RST4:

```
define_reset_order -from RST1 -to RST3 RST4
define_reset_order -from RST2 -to RST3 RST4
```

#### Example7

The following example specifies a reset ordering between flip-flops where source receives RST1 and RST2 and destination receives RST2 and RST3:

```
define_reset_order -from RST1 -to RST2 RST3
define_reset_order -from RST2 -to RST3
```

## Rules

The `define_reset_order` constraint is used by the following rules:

---

#### SpyGlass CDC solution

---

Ac_resetcross01	Ar_resetcross01	Ar_resetcross_ma trix01
-----------------	-----------------	----------------------------

---

## define\_tag

### Purpose

The `define_tag` constraint is used to define a named condition for application of certain stimulus at the top port or an internal node. The connectivity checks are made by applying one or more of the user-defined conditions such that circuit can be simulated into the desired mode.

The `define_tag` constraint can be used in the following different ways:

1. Provide initial state of a design

To provide an initial state for a design, each register needs to be initialized using the `define_tag` constraint. The following example initializes a register `foo` of design unit `top` to the value 0:

```
define_tag -tag initState -name top.foo -value 0
```

The following example initializes a vector register `foo[7:0]` of design unit `top` to the value 0:

```
define_tag -tag initState -name top.foo[7:0]
           -value "0"
```

2. Provide an initialization vector

A simulation vector that would initialize registers of a design. The following example initializes registers by applying three different vectors on three inputs of a design:

```
define_tag -tag initSeq -name top.reset1
           -value 1 1 1 x x x
define_tag -tag initSeq -name top.reset2
           -value x x x 1 1 1
define_tag -tag initSeq -name top.d
           -value 0 0 0 0 0 0
```

In this example, `reset1` is asserted for three cycles to initialize some set of registers, `reset2` is asserted to initialize another set of registers while some other registers are initialized by providing 0 on a data line "d" for six cycles. Note that `reset1` and `reset2` can be asserted simultaneously if they can reset registers independently.

The value can be 0, 1, or x (case-insensitive). Note that each value is

separated by a blank.

Vectors can also be initialized as in the following example:

```
define_tag -tag initSeq -name top.foo[0:3]
          -value {h 0 6}
```

The above specification indicates that the vector `top.foo` would be assigned 0 (in Hex) or 0000 (in binary) in the first cycle and 6 (in Hex) or 0110 (in binary) in the second cycle.

**NOTE:** *When you specify an initial sequence using the `define_tag` constraint, other initial values specified with the `reset` constraints are ignored.*

### 3. Specifying the bit-stuck values for nets

**NOTE:** *The `define_tag` constraint is used for specifying the bit-stuck values only for SpyGlass Auto Verify solution. This feature is not used by SpyGlass CDC solution.*

Bit-stuck values for nets are checked by the `Av_bitstuck01` rule of the *SpyGlass Auto Verify* solution.

The `Av_bitstuck01` rule of the *SpyGlass Auto Verify* solution checks only those nets that are specified with the `-name` argument of a `define_tag` constraint defining a tag named `netBitStuck` as in the following example:

```
define_tag -tag netBitStuck -name top.var[0]
          -value 0
```

Then, the `Av_bitstuck01` rule checks the stuck-at condition based on the value of the `-value` argument. If you do not specify the `-value` argument, both stuck-at 0 and stuck-at 1 conditions are checked.

## Using Wildcards

You can use wildcards with the `-value` argument while specifying the `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 initSeq -name top.rst1 -value 0 0 1 1
```

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

```
define_tag -tag initSeq -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).
- The one set of operations with the wildcard character must be separated by a blank space with 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, sequence 101.

## Product

SpyGlass Auto Verify solution, SpyGlass CDC solution, SpyGlass DFT solution, and SpyGlass DFT DSM solution

## For SpyGlass Auto Verify solution and SpyGlass CDC solution

### Syntax

The syntax of the `define_tag` constraint is as follows:

```
current_design <du-name>

define_tag
  -tag <initState | initSeq | netBitStuck>
  -name <flop-name> | <net-name>
  -value <value> | <value-list>
```

**NOTE:** `netBitStuck` is used for the `-tag` argument only for the SpyGlass Auto Verify solution. This feature is not used by the SpyGlass CDC solution.



## Arguments

**-tag <initState | initSeq | netBitStuck>**

The `-tag` argument accepts only a predefined list of values (case-sensitive).

**-name <flop-name> | <net-name>**

The `-name` argument accepts a hierarchical net name (scalar or vector).

**-value <value> | <value-list>**

The `-value` argument accepts values as follows:

Value of -tag argument	Signal Type	Allowed value of -value argument
initState, netBitStuck	Scalar	1, 0, or x
	Vector	Binary or hexadecimal number
initSeq	Scalar	Sequence of 1,0, and x
	Vector	Sequence of Binary or Hexadecimal numbers

**NOTE:** *The first row is only applicable for SpyGlass Auto Verify solution. This feature is not used by SpyGlass CDC solution.*

## Rules

The `define_tag` is used by the following rules:

SpyGlass CDC Solution			
Ac_cdc01a	Ac_cdc01b	Ac_cdc01c	Ac_cdc08
Ac_fifo01	Ac_handshake01	Ac_handshake02	Clock_sync03a
Ac_conv02			
SpyGlass Auto Verify Solution			
Av_initstate01	Av_bitstuck01		

## For SpyGlass DFT solution and SpyGlass DFT DSM solution

### Purpose

The `define_tag` constraint is used to define a named condition for application of certain stimulus at the top port or an internal node. The connectivity checks are made by applying one or more of the user-defined conditions such that circuit can be simulated into the desired mode.

This condition is defined in one of the following methods:

1. Application of logic value on a specified node

This method is used to define a named condition under which given value is applied to specified node (top module port or internal node) for one or more LE simulation cycles. For example:

```
define_tag -tag s1 -name top.EN -value 0101
```

This statement defines a condition by name `s1` to mean that logic value (or sequence) `0101` be applied on pin `top.EN`. The semantics of the `define_tag` constraint is similar to the `test_mode` constraint.

Multiple pin-value pairs can be associated under the same condition name. For example,

```
define_tag -tag s1 -name top.EN1 -value 0101
```

```
define_tag -tag s1 -name top.EN2 -value 0100
```

However, please note these should not be mutually conflicting as shown below.

```
define_tag -tag s1 -name top.EN1 -value 0101
```

```
define_tag -tag s1 -name top.EN1 -value 0100
```

In addition, there is no precedence between various pin-value specifications under the same condition name, and all of these are simulated in parallel by SpyGlass LE engine.

Name can be a top-module port, any internal net name, or terminal name. It may also be bit-select or part-select.

You can specify repeat sequences for the `define_tag` constraint.

For fields that require repeat sequence, specify the values as `<I*S>`.

Here, `S` is any string that does not contain the `<`, `>`, and `*` characters. However, `S` can contain another `<I*S>` expression. `I` is an integer that is always interpreted as a decimal value. The expression `<I*S>` means

that the sequence S will be repeated I number of times.

**NOTE:** *Tagging for nesting is not allowed. For example, the following `define_tag` statements are not allowed:*

```
define_tag -name sub_seq -value <5*01>
define_tag -name main_seq -value <100*sub_seq>
```

However, you can achieve the same result by using the `setvar` command.

## 2. Group various conditions together to create a new condition name

This method is used to group one or more previously defined conditions under a new condition name. The semantics of this method are equivalent to creating a new condition name to represent all pin-value specifications defined under the conditions being merged. For example:

```
define_tag -tag smode100
  -merge <list-of-previously-defined-conditions>
```

This statement defines a condition by name `smode100` to mean the grouping of all individual pin-value specifications as specified after the `-merge` argument. It should be noted that condition names require a 'define before use' model and therefore, use of any undefined condition name will report a syntax error in reading SGDC file.

The affect of merging multiple condition names into a single condition is the same as defining multiple pin-value specifications under the same condition name. As previously described, these should not be mutually conflicting. In addition, there is no precedence between various pin-value specifications coming from the same or different condition names, and all of these are simulated in parallel by SpyGlass LE engine.

The above two methods of defining a condition cannot be used simultaneously. You can either define a condition by pin-value specification or by merge specification. A condition created by merge method can be defined only once.

Further, a condition cannot be modified (or appended to with additional pin-value specification) after it has been used. This limitation has been defined to avoid any ambiguity in SGDC specification, and ensure WYSIWYG advantage for end users.

The condition creation by the merge method can be used for aliasing purposes also by specifying only one condition name after the `-merge` argument. This is a very useful feature because now all subsequent

connectivity check commands can use the alias name and the alias name can be changed to point to different condition group(s) by a very simple edit at one location only.

Now, you can run SpyGlass analysis under a different condition group by making just one small change in the SpyGlass Design Constraints file.

**NOTE:** *The scope of a condition is restricted to the current\_design specification only.*

You can also specify the `define_tag` constraint for a bus signal as shown below.

```
define_tag -tag T1 -name top.in[2:0] -value {b 001}
```

Where `in` is the `2:0` vector net.

### ***When to use the define\_tag constraint***

You should use `define_tag` constraint for the connectivity specific (Soc\_xx rules) and scan chain-specific rules only. The idea is that if different parts of a circuit operate under a different set of simulation conditions, you can do the connectivity checks under a different simulation condition in a single run of SpyGlass analysis.

You should also use `define_tag` constraint when the test initialization sequence is complex (such as, a TAP controller controlling the scan chains). In such cases, setting up proper test mode may take multiple runs of SpyGlass analysis. Thus, `define_tag` constraint enables you to try out different initialization sequences using different `define_tag` statements, so that you can evaluate and infer which initialization sequence is appropriate in a single run of SpyGlass using the `require_value` constraint.

You should use `test_mode` constraint to define the final set of simulation condition under which chip is tested. The `define_tag` constraint is used by all rules except the SoC rules.

## **Syntax**

The syntax of the `define_tag` constraint is as follows:

```
define_tag
  -tag <any-string-name>
  -name <net-name>
  -value <value> | <value-list>
```

```
-merge <tag> | <tag-list>
```

## Arguments

**-tag <any-string-name>**

The `-tag` argument accepts any string name.

**-name <net-name>**

The `-name` argument accepts a hierarchical net name | port | hierarchical terminal name (scalar or vector).

**-value <value> | <value-list>**

The `-value` argument accepts values of any string constituted using 1, 0, X, Z.

**-merge <tag> | <tag-list>**

The `-merge` argument accepts tag name or a list of tag names. Please note that tags specified as a value to `-merge` field must be defined using `define_tag`.

## Examples

Consider the following examples:

### Example1

```
define_tag -name abc -value "<5*10>"
```

The above example will be expanded as follows:

```
define_tag -name abc -value 1010101010
```

### Example2

```
define_tag -name abc -value "11<5*10>010"
```

The above example will be expanded as follows:

```
define_tag -name abc -value 111010101010010
```

### Example3

```
define_tag -name abc -value "<50*11<5*10>>010"
```

The above example will be expanded as follows:

```
define_tag -name abc -value 111010101010...(repeated 50 times
followed by 010)
```

You can also set a variable using the command `setvar` to obtain the above result as follows:

```
setvar x 11<5*10>
```

```
define_tag -name abc -value "<50*${x}>010"
```

The above example will be expanded as follows:

```
define_tag -name abc -value 111010101010...(repeated 50 times
followed by 010)
```

Consider the following example that has three conditions (C1, C2, and C3) and three condition groups (CG1, CG2, and CG3):

```
define_tag -tag C1 ...
define_tag -tag C2 ...
define_tag -tag C3 ...
define_tag -tag CG1 -merge C1 C2
define_tag -tag CG2 -merge C2 C3
define_tag -tag CG3 -merge C3 C1
```

Now, you want to apply the condition groups one-by-one. Normally, you would need to create the constraints of each group separately. However, you can create an alias condition (say TM) with the first condition group CG1 as follows:

```
define_tag -tag TM -merge CG1
```

Use the alias condition TM in subsequent constraints and run SpyGlass analysis. Next, modify the alias condition TM to the next condition group CG2 as follows:

```
define_tag -tag TM -merge CG2
```

#### Example 4

Consider the following sample input values:

```
define_tag -name vec[3:0] -value { b 1 0 1 0 }
```

where `vec` is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "1010"
define_tag -name vec[1] -value "0000"
define_tag -name vec[2] -value "0000"
define_tag -name vec[3] -value "0000"
```

### Example 5

Consider the following sample input values:

```
define_tag -name vec[3:0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "0"
define_tag -name vec[1] -value "1"
define_tag -name vec[2] -value "0"
define_tag -name vec[3] -value "1"
```

### Example 6

Consider the following sample input values:

```
define_tag -name vec[3:0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "1"
define_tag -name vec[1] -value "0"
define_tag -name vec[2] -value "0"
define_tag -name vec[3] -value "0"
```

### Example 7

Consider the following sample input values:

```
define_tag -name vec -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "1010"
define_tag -name vec[1] -value "0000"
define_tag -name vec[2] -value "0000"
```

```
define_tag -name vec[3] -value "0000"
```

### Example 8

Consider the following sample input values:

```
define_tag -name vec -value { b 1010 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "0"
```

```
define_tag -name vec[1] -value "1"
```

```
define_tag -name vec[2] -value "0"
```

```
define_tag -name vec[3] -value "1"
```

### Example 9

Consider the following sample input values:

```
define_tag -name vec -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "1"
```

```
define_tag -name vec[1] -value "0"
```

```
define_tag -name vec[2] -value "0"
```

```
define_tag -name vec[3] -value "0"
```

### Example 10

Consider the following sample input values:

```
define_tag -name vec[0] -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "1010"
```

### Example 11

Consider the following sample input values:

```
define_tag -name vec[0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:



```
define_tag -name vec[0] -value "0"
```

**Example 12**

Consider the following sample input values:

```
define_tag -name vec[0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "1"
```

**Example 13**

Consider the following sample input values:

```
define_tag -name sclr -value { b 1 0 1 0 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
define_tag -name sclr -value "1010"
```

**Example 14**

Consider the following sample input values:

```
define_tag -name sclr -value { b 1010 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
define_tag -name sclr -value "0"
```

**Example 15**

Consider the following sample input values:

```
define_tag -name sclr -value { b 1 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
define_tag -name sclr -value "1"
```

**Example 16**

Consider the following sample input values:

```
define_tag -name vec -value { h 6 }
```

where `vec` is the 3:0 vector net

The above input is expanded as shown below:

```
define_tag -name vec[0] -value "0"
define_tag -name vec[1] -value "1"
define_tag -name vec[2] -value "1"
define_tag -name vec[3] -value "0"
```

## Rules

The `define_tag` constraint is used by the following rules:

<b>SpyGlass DFT Solution</b>			
Info_schain	Scan_22	Scan_24	Scan_25
Scan_26	Scan_29	Diagnose_Scan Chain	
<b>SpyGlass Connectivity Verify</b>			
Soc_01	Soc_02	Soc_04	
<b>SpyGlass DFT DSM Solution</b>			
Atspeed_21	Info_Atspeed_21		

## delay\_buffer

### Purpose

The `delay_buffer` constraint is used to specify the delay buffers.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `delay_buffer` constraint is as follows:

```
current_design <du_name>
delay_buffer
  -names <cell-name-list>
```

## Arguments

**<du\_name>**

Name of the design unit under which you are specifying the cells.

**-names <cell-name-list>**

Space-separated list of the cell names. You can use wildcard characters while specifying the cell name using the `-names` argument.

## Rules

The `delay_buffer` constraint is used by the following rules:

---

**SpyGlass Power Verify Solution**

---

LPPSW03

LPPSW04

---

## deltacheck\_ignore\_instance

### Purpose

Specifies the instance to be ignored for delta delay value checking.

Then, the DeltaDelay01 rule does not report the delta delay values for all flip-flops/latches in the specified instance. However, the actual delta delay value for the specified instance is used for further calculations.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `deltacheck_ignore_instance` constraint is as follows:

```
current_design <du-name>
  deltacheck_ignore_instance
    -name <inst-name>
```

### Arguments:

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the instance to be ignored.

#### -name <inst-name>

The hierarchical name of the instance to be ignored.

You can specify multiple instances to be ignored using multiple `deltacheck_ignore_instance` constraints.

**NOTE:** For VHDL design units, instance names are case-insensitive.

### Rules

The `deltacheck_ignore_instance` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

---

DeltaDelay01

---

## deltacheck\_ignore\_module

### Purpose

Specifies the design unit to be ignored for delta delay value checking.

Then, the DeltaDelay01 rule does not report the delta delay values for all flip-flops/latches in all instances of the specified design unit. However, the actual delta delay values for these instances are used for further calculations.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `deltacheck_ignore_module` constraint is as follows:

```
current_design <du-name>  
  deltacheck_ignore_module  
    -name <ignr-du-name>
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the design unit to be ignored

#### -name <ignr-du-name>

The name of the design unit to be ignored.

You can specify multiple design units to be ignored using multiple

deltacheck\_ignore\_module constraints.

**NOTE:** For VHDL design units, names of the design units to be ignored are case-insensitive.

## Rules

The `deltacheck_ignore_module` constraint is used by the following rule:

---

### SpyGlass CDC Solution

---

DeltaDelay01

---

## deltacheck\_start

### Purpose

Specifies the start points (clock ports, clock pins, or clock nets) for checking by the DeltaDelay01 rule.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `deltacheck_start` constraint is as follows:

```
current_design <du-name>
  deltacheck_start
    -name <sig-name>
    [ -value <value> ]
```

### Arguments

#### <du-name>

Module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the start point clock port/pin/net.

**-name <sig-name>**

Hierarchical name of a clock port/pin/net.

**-value <value>**

Expected delta delay value at all flip-flops/latches triggered by the clock port/pin/net specified with the `-name` argument.

You can specify multiple start point signals using multiple `deltacheck_start` constraints.

## Examples

The following example specifies net `clk1` of design unit `top` as the start point:

```
current_design top
  deltacheck_start -name top.clk1
```

If you do not specify any `deltacheck_start` constraints, the `DeltaDelay01` rule infers all clock source ports/nets/pins and uses them as the start points.

The `-value` argument is an optional argument used for clock pin balancing. When you do not specify the `-value` argument, the `DeltaDelay01` rule expects the delta delay value for all flip-flops/latches triggered by the specified clock to be the same. When you specify the `-value` argument, the `DeltaDelay01` rule expects the delta delay value for all flip-flops/latches triggered by the specified clock to be the same and equal to the specified value.

The following example specifies net `clk1` of design unit `top` as the start point and expects all flip-flops/latches to have a delta delay value of 1:

```
current_design top
  deltacheck_start -name top.clk1 -value 1
```

When you want all flip-flops/latches in the design to have the same delta delay value, specify the (same) expected value with the `-value` argument for all specified clocks.

## Rules

The `deltacheck_start` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

---

DeltaDelay01

---

## deltacheck\_stop\_instance

### Purpose

Specifies the instance where the DeltaDelay01 rule should stop further traversal along the clock tree when the clock pin of the specified instance is hit.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `deltacheck_stop_instance` constraint is as follows:

```
current_design <du-name>
  deltacheck_stop_instance
    -name <inst-name>
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the stop point instance

#### -name <inst-name>

The hierarchical name of the stop point instance.

You can specify multiple stop point instances using multiple `deltacheck_stop_instance` constraints.

**NOTE:** For VHDL design units, instance names are case-insensitive.



## Examples

The following example specifies instance I21 as the stop point instance under the design unit top:

```
current_design top
  deltacheck_stop_instance -name top.I21
```

Here, the DeltaDelay01 rule stops further traversal along the clock tree when the clock pin of instance top.I21 is hit.

The clock traversal automatically stops at instances of a stopped design unit (stopped using the `set_option stop <du-name>` command in the project file or the equivalent in the GUI). Therefore, you need not specify such design unit instances with the `deltacheck_stop_instance` constraint.

## Rules

The `deltacheck_stop_instance` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

DeltaDelay01

---

## deltacheck\_stop\_module

### Purpose

Specifies the design unit where the DeltaDelay01 rule should stop further traversal along the clock tree when the clock pin of an instance of the specified design unit is hit.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `deltacheck_stop_module` constraint is

as follows:

```
current_design <du-name>
  deltadelay_stop_module
    -name <stp-du-name>
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the stop point design unit.

### -name <stp-du-name>

The name of the stop point design unit.

You can specify multiple stop point design units using multiple `deltacheck_stop_module` constraints.

**NOTE:** For VHDL design units, names of stop point design units are case-insensitive.

## Examples

The following example specifies design unit `m1` as the stop point design unit under the design unit `top`:

```
current_design top
  deltacheck_stop_module -name m1
```

Here, the `DeltaDelay01` rule stops further traversal along the clock tree when the clock pin of an instance of design unit `m1` is hit.

The clock traversal automatically stops at instances of a stopped design unit (stopped using the `set_option stop <du-name>` command in the project file or the equivalent option in the Atrenta Console GUI).

Therefore, you need not specify such design units with the `deltacheck_stop_module` constraint.

## Rules

The `deltacheck_stop_module` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

---

DeltaDelay01

---

## deltacheck\_stop\_signal

### Purpose

Specifies the design points (ports, pins, or nets) where the DeltaDelay01 rule should stop further traversal along the clock tree.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `deltacheck_stop_signal` constraint is as follows:

```
current_design <du-name>
  deltacheck_stop_signal
    -name <sig-name>
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the stop point.

#### -name <sig-name>

The hierarchical name of the end point port, pin, or net.

You can specify multiple stop points using multiple `deltacheck_stop_signal` constraints.

**NOTE:** For VHDL design units, signal name is case-insensitive.

## Examples

The following example specifies pin `d1` of instance `I31` in design unit `top` as the stop point:

```
current_design top
  deltacheck_stop_signal -name top.I31.d1
```

Here, the `DeltaDelay01` rule stops further traversal along the clock tree when the pin `top.I1.d1` is hit.

The clock traversal automatically stops at the design points (ports, pins, or nets) that belong to instances of a stopped design unit (stopped using the `set_option stop <du-name>` command in the project file or the equivalent option in the Atrenta Console GUI). Therefore, you need not specify such design points with the `deltacheck_stop_signal` constraint.

## Rules

The `deltacheck_stop_signal` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

---

DeltaDelay01

---

## design\_map\_info

### Purpose

The `design_map_info` constraint can be used to improve RTL to RTL simulation file annotation as well as to improve netlist to RTL simulation file annotation. You can specify either of the following while using the `design_map_info` constraint:

- The `match_point_rtl` and the `match_point_gate` arguments to map netlist to RTL simulation data (`sim_file` is optional).

The `design_map_info` constraint is used for mapping RTL matchpoints with the corresponding gate-level matchpoints. Currently,

only sequential cell outputs are considered for matching. Use the `match_point_rtl` argument to specify the name of sequential cell output used in the RTL. To specify the name of the corresponding sequential cell output used in the gate-level netlist, use the `match_point_gate` argument.

Please note that it is recommended to use matchpoints instead of net names to match RTL and gate-level netlist.

- The `gatenet` and the `rtlnet` arguments to map netlist to RTL simulation data

In addition, the `design_map_info` constraint is also used to specify a mapping between the RTL net names and gate-level net names. You can use the `design_map_info` constraint to manually map a simulation hierarchy with a design hierarchy. SpyGlass PE considers both the simulation and design hierarchies and tries to create a mapping between the simulation hierarchy and the design hierarchy. However, if there is a mismatch in name or a change in the hierarchy, the signals present in this hierarchy are not mapped automatically.

- The `sim_hier_name`, `design_hier_name` arguments to map RTL to RTL simulation data (`sim_file` is optional)

The `design_map_info` constraint is also used for mapping the hierarchical instance name in the RTL with the corresponding hierarchical name in the simulation file. Use the `design_hier_name` argument to specify the hierarchical instance name in the RTL and the `sim_hier_name` argument to specify the corresponding hierarchical name in the simulation file. If you have multiple simulation files for the same design and if the mapping is applicable to only one simulation file, use the `sim_file` argument to specify the simulation file. Note that the `sim_file` argument is optional.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `set_design_map_info`.*

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `design_map_info` constraint is as follows:

```
current_design <du-name>
  design_map_info
    -match_point_rtl <matchpoints_rtl>
    -match_point_gate <matchpoints_gate-level>
```

OR

```
current_design <du-name>
  design_map_info
    -rtl原因 <rtl-netname>
    -gatenet <gate-netname>
```

OR

```
design_map_info
  -sim_hier_name <simulation_hierarchy>
  -design_hier_name <design_hierarchy>
  -sim_file <simulation_file>
```

## Arguments

### `<matchpoints_rtl>`

Specifies the hierarchical name of sequential cell output as used in the RTL.

### `<matchpoints_gate-level>`

Specifies the hierarchical name of sequential cell output as used in the gate-level netlist.

### `<rtl-netname>`

Specifies the hierarchical RTL net name.

### `<gate-netname>`

Specifies the gate-level net name corresponding to the RTL net name specified in the `<rtl-netname>` field.

**<simulation\_hierarchy>**

Specifies the name of the simulation hierarchy to be mapped to the design hierarchy specified in the `design_hierarchy` argument.

**<design\_hierarchy>**

Specifies the name of the design hierarchy to be mapped to the simulation hierarchy specified in the `simulation_hierarchy` argument.

**< simulation\_file >**

Specifies the name of the simulation file. Relative or absolute paths can be used to specify the file name. The name specified in this argument should be any of the names specified in the `-file` argument of the `activity_data` constraint.

If this name is not provided, SpyGlass searches for the simulation hierarchy in all the simulation files specified by the `activity_data` constraint. In case no match is found, SpyGlass exits with a FATAL error. If multiple matches are found, the mapping is applied to all the matching files.

**Rules**

The `design_map_info` constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	PEPWR20	PEPWR21
PEPWR22	PEPWR23	PEPWR24	PEPWR25
PEPWR28			

**dftmax\_partition****Purpose**

The `dftmax_partition` constraint is used to specify partitions in the design targeted for DFTMAX Ultra codexes to automate the planning and configuration of DFTMAX Ultra in an SoC.

## Product

SpyGlass DFT Product

## Syntax

The syntax for `dftmax_partition` constraint is as follows:

```
dftmax_partition
  [ -name <partition_name> ]
  [ -blocklist <list_of_blocks> ]
  [ -type <streaming_compression_type> ]
  [ -inputs <SI_count> ]
  [ -outputs <SO_count> ]
  [ -chain_count <number_of_chains> ]
```

## Arguments

**-name <partition\_name>**

**NOTE:** (Mandatory) Specifies a label for each DFTMAX Ultra codec partition. **NOTE :** Specified name should be unique, that is, no two codec partitions can have same label. Refer to Example 1 for more information.

**-blocklist <list\_of\_blocks>**

(Mandatory) Specifies the list of blocks (top-level instances) to be included in codec partition.

**NOTE:** This argument supports wildcard characters. Refer to example 3 for more information.

**-type <streaming\_compression\_type>**

(Optional) Specifies the streaming compression type, only ultra/ULTRA is supported for now.

**-inputs <SI\_count>**

(Optional) Specifies the number of scan-in pins.

If the `-inputs` argument is not specified, then the allocation of scan-in pins is done by SpyGlass.



**-outputs <SO\_count>**

(Optional) Specifies the number of scan-out pins.

If the `-outputs` argument is not specified, then the allocation of scan-out pins is done by SpyGlass.

**-chain\_count <number\_of\_chains>**

(Optional) Specifies the number of scan chains.

If the `-chain_count` argument is not specified, then the allocation of scan chains is done by SpyGlass.

**Examples**

The following examples show the usage of `dftmax_partition` constraint:

**Example 1**

Consider the following constraint definition:

```
dftmax_partition -name partition1 -blocklist u1 u2
```

The above example indicates that block (top-level instances), `u1` and `u2`, are considered in codec partition, `partition1`.

**Example 2**

Consider the following constraint definitions:

```
dftmax_partition -name partition1 -blocklist u1 -inputs 2
dftmax_partition -name partition2 -blocklist u2 -outputs 4
dftmax_partition -name partition3 -blocklist u1 -chain_count
4
dftmax_partition -name partition4 -blocklist u3 -inputs 4 -
outputs 6
```

The above example considers:

- 2 scanin pins for codec partition, `partition1`
- 4 scanout pins for codec partition, `partition2`
- 4 scan chains for codec partition, `partition3`, and
- 4 scanin pins & 6 scanout pins for codec partition, `partition4`.

**Example 3**

Consider the following constraint definition:

```
dftmax_partition -name partition1 -blocklist "u*"
```

The above example uses wildcard expression to indicate that all the blocks (top-level instances) starting with the string, u, are considered in codec partition, partition1.

#### Example 4

Consider the following constraint definitions:

```
dftmax_partition -name partition1 -blocklist u1  
dftmax_partition -name default_partition -blocklist u2
```

The above example considers block u1 in codec partition, partition1 and block u2 in default codec partition.

In the absence of the dftmax\_partition command for default codec partition, SpyGlass considers all unallocated blocks (top-level instances) for default codec partition.

## Rules

The dftmax\_partition constraint is used by the Info\_dftmax\_configuration rule of the SpyGlass DFT product.

## dft\_report\_fault\_list

### Purpose

Use this constraint to define the modules and/or instances for which the fault list needs to be generated.

### Product

SpyGlass DFT Solution

### Syntax

The syntax to specify the dft\_report\_fault\_list constraint is as follows:

```
dft_report_fault_list  
    [-module < module_name> ]  
    [-instance < hier_instance_name> ]
```

## Arguments

### `<module_name>`

Specifies the module names for which fault data is included in the fault report.

### `<hier_instance_name>`

Specifies the hierarchical path of the instances for which fault data is included in the fault report.

## Rules

All rules of the SpyGlass DFT solution use this constraint.

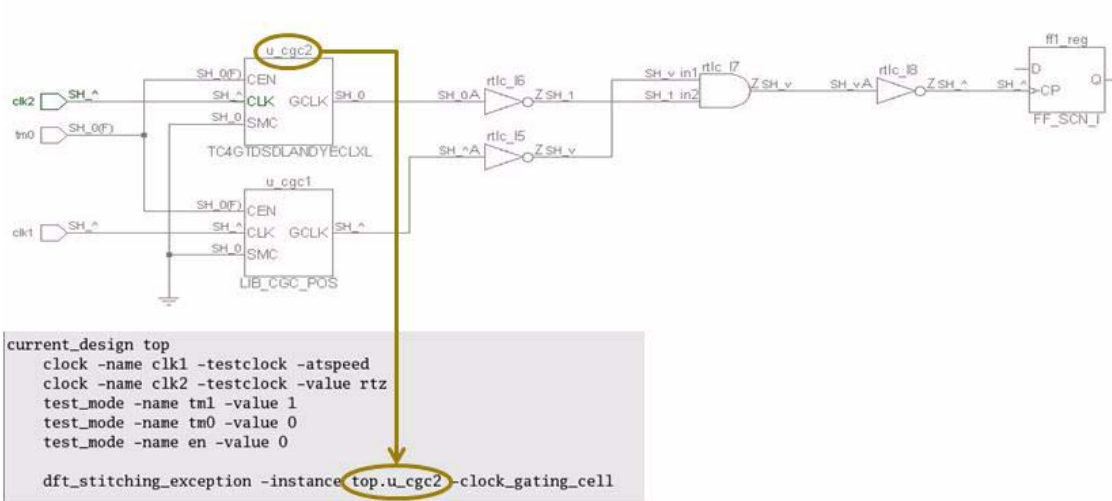
## `dft_stitching_exception`

### Purpose

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 shift mode. In this case it is assumed that after scan stitching (`-dftDesignState=post_scan_stitched`), the grounded `test_enable` pin of the CGC will be properly connected with the `SCAN_MODE` signal.

For the SoC designs containing both `pre_scan_stitched` and `post_scan_stitched` IP blocks, set the `dftDesignState` parameter to `pre_scan_stitched` and use the `dft_stitching_exception` constraint to designate the design areas as `post_scan_stitched`. For such exceptions, the CGC is treated as a regular CGC and if its `test_enable_pin` is tied to ground, the test clock does not propagate through it during shift mode.

For example, consider the following figure where one CGC output allows clock to propagate but other CGC output is used to drive a constant value.



## Product

SpyGlass DFT solution

## Syntax

The syntax to specify the `dft_stitching_exception` constraint is as follows:

```

dft_stitching_exception
  -clock_gating_cell
  [-module < module_name> ]
  [-instance < hier_instance_name> ]

```

## Arguments

### `-clock_gating_cell`

Specifies the need to apply the stitching exception to the clock gating cells that are part of the modules/instances specified using the `-module/-instance` arguments.

### `<module_name>`

Specifies the name of the design unit, CGC module, or the module containing all the ignored CGCs.

**<hier\_instance\_name>**

Specifies the hierarchical path of:

- The instance containing all the ignored CGCs
- The CGC instance

**Rules**

All rules of the SpyGlass DFT and SpyGlass DFT DSM solution use this constraint.

**dft\_report\_coverage****Purpose**

The dft\_report\_coverage constraint is used to specify list of modules and instances, which needs to be included in the fault coverage report.

**Product**

SpyGlass DFT product

**Syntax**

```
dft_report_coverage
  -module <module_names>
  -instance <instance_names>
```

**Arguments**

**-module <module\_names>**

List of module names. This argument supports wildcard characters.

**-instance <instance\_names>**

List of full hierarchical name of instances. This argument supports wildcard characters.

## Examples

```
dft_report_coverage -module mymod mymod2 -instance
top.l1l1.inst1
```

## Rules

The `dft_coverage_report` constraint is used by the following rules of the SpyGlass DFT Product:

---

Info_coverage	Info_transition_coverage
---------------	--------------------------

---

## disable\_timing

### Purpose

The `disable_timing` constraint is used to disable the timing arcs between the specified pins of a library cell.

### Product

All SpyGlass products

### Syntax

```
disable_timing
  -name <name>
  -from <from-pin>
  -to <to-pin>
```

### Arguments

**-name <name>**

Name of the library cell or an instance.

**-from <from-pin>**

Name of the pin from which the timing arc should be disabled. The pin should be a single-bit or multi-bit pin.

**-to <to-pin>**

Name of the pin till which the timing arc starting from the *-from <from-pin>* should be disabled. The pin should be a single-bit or multi-bit pin.

**Rules**

All rules of all the SpyGlass products.

**disallow\_modification\_type****Purpose**

The `disallow_modification_type` constraint is used to specify the type of modification to be disallowed in the RTL generated by the AutoFix feature.

**Product**

SpyGlass Power Reduce

**Syntax**

```
disallow_modification_type
  [ -port_insertion ]
  [ -generate_unroll ]
  [ -max_hier <int> ]
  [ -bus_split ]
```

**Arguments****-port\_insertion**

Specifies that the AutoFix feature should not perform RTL modification causing insertion of a port.

**-generate\_unroll**

Specifies that the AutoFix feature should not perform RTL modification resulting in unroll of generate statements.

**-max\_hier <int>**

Specifies the maximum number of hierarchies so that any signal that requires routing in the modified RTL beyond the specified hierarchy is not auto fixed.

**-bus\_split**

Specifies that the AutoFix feature should not perform RTL modification that requires bus split of destination register.

**Examples****Example 1**

Consider the following constraint:

```
disallow_modification_type -port_insertion
```

In this example, the AutoFix feature does not perform RTL modification that results in the insertion of a port.

**Example 2**

Consider the following constraint:

```
disallow_modification_type -max_hier 3
```

In this example, any power reduction opportunity that requires routing of a signal through more than three hierarchical boundaries are not auto fixed.

**Rules**

The `disallow_modification_type` constraint is used by the following rules:

PEPWR20	PEPWR21	PEPWR22	PEPWR23
PEPWR24	PEPWR25	PEPWR28	PEPWR29
PESTR20	PESTR21	PESTR22	PESTR23
PESTR24	PESTR25	PESTR28	PESTR29



## disallow\_upf\_command

### Purpose

This command is used to allow/disallow specified UPF commands and their options.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `disallow_upf_command` constraint is as follows:

```
disallow_upf_command -name <command-name> -options <option-list>
```

### Arguments

#### **-name <command-name>**

(Mandatory) Specifies the name of the UPF 1.0/2.0 command. The command name is checked against UPF 1.0/2.0 for sanity checking of the constraint in the *SGDC\_lowpower119* rule.

#### **-options <option-list>**

(Optional) Specifies the arguments of the UPF command. The arguments specified in the `-options` argument are not supported and all other options of that command will be supported. If you provide `disallow_upf_command <command-name>` only, without specifying `-options` argument, then the whole command will be disallowed. The specified arguments are checked against UPF 1.0/2.0 for sanity checking of the constraint in the *SGDC\_lowpower119* rule.

### Rules

The `disallow_upf_command` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**


---

SGDC_lowpo wer119	UPFSEM_42	UPFSEM_43
----------------------	-----------	-----------

---

## domain

### Purpose

The `domain` constraint is used to specify the clock domain information in the design. The wildcard support is provided for the `domain` constraint.

**NOTE:** *The domain constraint has been renamed to `clock_group`. For backward compatibility, the domain constraint is currently still available.*

### Product

SpyGlass Constraints solution

### Syntax

The `domain` constraint is used in the following syntax:

```
current_design <du-name>
  domain -name <domain-name>
  [ -clock_pin <clk-pin-name-list> ]
  [ -clock <SDC-clk-name-list> ]
```

### Arguments

#### <du-name>

The top-level module name (for Verilog designs) or the top-level entity name (for VHDL designs) or a synthesis partition name specified using the `block` keyword.

#### -name <domain-name>

(Mandatory) Specifies the clock domain name. It can be any valid string.

**-clock\_pin <clk-pin-name-list>**

Specifies a space-separated list of design clock pin names. If this option is used then all clocks specified on the clock pin will be considered in the same domain.

**-clock <SDC-clk-name-list>**

Specifies a space-separated list of SDC clock names.

If both options are specified, then the domain will include all the clocks specified with `-clock` option and all the clocks defined on the clock pin specified with the `-clock_pin` option.

**Example**

Let the clock domain information be provided as follows:

```
current_design top
  domain -name d1 -clock { C1 C2 }
  domain -name d2 -clock { C3 }
```

In the above example, clocks C1 and C2 are synchronous, and clocks C1 and C3, C2 and C3 are asynchronous. You cannot specify the same clock in two different domains. If you do so, the `DomainSanityCheck` rule reports a FATAL message.

If the same domain is specified in multiple lines of an SGDC file, the union of clocks (specified in multiple lines) will be in the same domain. For example:

```
domain -name d1 -clock {C1 C2}
domain -name d1 -clock {C4}
```

Here, clocks C1, C2, C4 will be assumed as synchronous.

All create clocks and their derived clocks will be assumed to be synchronous if you have specified the domain among create clocks and their derived clocks. You can even specify clock and its derived rules in a different domain. For example:

C1 -> GC1 -> GC2 where GC2 is derived from GC1 and so on.

```
// C1, GC1, GC2 will be assumed in same domain
domain -name d1 -clock C1
// User specified clock GC2 to d2 domain
```

```
domain -name d2 -clock GC2
```

No `domainsanitycheck` violation will be thrown if you specify create clock and its derived clock in a different domain.

**NOTE:** *All SGDC-dependent rules will be impacted.*

## Rules

The `domain` constraint is used by the following rules:

SpyGlass Constraints Solution			
Clk_Gen05	Clk_Lat03	Clk_Uncert03	False_Path07
False_Path08	Domain_SGDC_Consis		

## domain\_inputs

### Purpose

Specifies expected values of various inputs of a power domain under a power-down condition.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `powerdomaininputs`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `domain_inputs` constraint is as follows:

```
current_design <du-name>
  domain_inputs
    -name <pd-condition-name>
    [-value <input-value-pairs>]
    [-default <0 | 1 | 2> ]
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the power domain input values under power-down condition.

**-name <pd-condition-name>**

Name of the power-down condition specified by using the *voltage\_domain* constraint (the `-inputs` argument).

**-value <input-value-pairs>**

Space-separated list of input-value pairs where each pair is a space-separated pair of power domain input signal name and its expected power-down value (0 for active low and 1 for active high).

**-default <0 | 1 | 2>**

Specifies the expected steady state value of power domain inputs or outputs. Specify 0 for active low, 1 for active high, and 2 for hold specified value.

## Examples

You can specify vector inputs in one specification if all bits of the vector input are expected to attain the same value under power-down condition, as in the following example:

```
domain_inputs -name V3_PD_COND -value in[3:0] 1
```

## Rules

The `domain_inputs` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

LPSVM47

---

## domain\_outputs

## Purpose

The `domain_outputs` constraint is used to specify the values of various signals under the steady state condition specified in [voltage\\_domain](#) constraint.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `powerdomainoutputs`.*

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `domain_outputs` constraint is as follows:

```
current_design <du-name>
  domain_outputs
    -name <ss-condition-name>
    [ -default <0 | 1 | 2> ]
    [ -dest ]
    [ -value <signal-value-pairs> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the power domain output values under steady state.

### -name <ss-condition-name>

Name of the steady state condition specified using the [voltage\\_domain](#) constraint.

### -default <0 | 1 | 2>

Specifies the expected steady state value for all signals except those specified with the `-value` argument. Specify 0 for active low, 1 for active high, and 2 for hold earlier value.

**-dest**

If `-dest` argument is specified, then isolation cells are placed in destination domain hierarchy. Otherwise, isolation cells are placed in top hierarchy.

**-value <signal-value-pairs>**

Space-separated list of signal-value pairs *<signal-value-pairs>* where each pair is a space-separated pair of power domain output signals and its expected steady state value (0 for active low, 1 for active high, and 2 for hold earlier value).

**Rules**

The `domain_outputs` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**


---

 LPSVM23
 

---

**domain\_signal****Purpose**

Specifies the power-up/power-down signals.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `pdsignal`.*

**Product**

SpyGlass Power Verify solution

**Syntax**

The syntax to specify the `domain_signal` constraint is as follows:

```
current_design <du-name>
  domain_signal
    -name <sig-name>
    -value <0 | 1>
```

```
[ -clocks <clk-name-list> ]  
[ -seqsignals <sig-name-list> ]  
[ -seqvalue <value-seq> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the power-up/power-down signals.

### -name <sig-name>

Name of the power-up/power-down signal.

### -clocks <clk-name-list>

Specifies a space-separated name list of clock sources associated with power-up/power-down.

### -value <0 | 1>

Specifies whether the signal is active high (1) or active low (0).

### -seqsignals <sig-name-list>

Specifies a space-separated name list of signals to be checked for value sequence specified by the -seqvalue argument.

### -seqvalue <value-seq>

Specifies a space-separated list of value sequence for signals specified with the -seqsignals argument. The width of the sequence must be the same as the number of signals specified with the -seqsignals argument and should describe the signal values at all transitions of all signals during the power-up/power-down. In addition, use only 0 and 1 to described the sequence.

## Examples

Consider the following example:

```
domain_signal -name N1 -value 0 -clocks C1  
-seqsignals N2 N3 -seqvalue 00 01 10 11
```



Here, there are two nets N2 and N3 and the expected sequence is a space-separated list of pairs of two values (0/1) indicating the expected value for each signal, respectively. Thus, both nets are expected to have a value of 0 at power-down. Then, the net N3 is expected to change from 0 to 1 at a later time while net N2 is expected to remain at 0. Then, the net N2 is expected to change from 0 to 1 and net N3 is expected to change from 1 to 0 later. Lastly, both nets are expected to attain a value of 1 before power-up. If these nets do not follow the specified sequence, the LPSVM43 rule of the *SpyGlass Power Verify* solution reports a violation when the first mismatch is encountered.

## Rules

The `domain_signal` constraint is used by the following rule:

---

### SpyGlass Power Verify Solution

---

LPSVM43

---

## dont\_touch

### Purpose

The `dont_touch` constraint specifies the modules/nets that are not considered for AutoFix.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `dont_touch` constraint is as follows:

```
dont_touch
  -module <module-name>
  -net <net-name>
  -auto_fix
```

**NOTE:** *The dont\_touch constraint supports wildcard characters.*

## Arguments

The dont\_touch constraint has the following arguments:

**-module** <module-name>

The name of the module that should not be considered for AutoFix.

**-net** <net-name>

Name of the net (complete hierarchical net name) that should not be considered for AutoFix.

## Example

```
dont_touch -module m1 -auto_fix
```

```
dont_touch -net net1 -auto_fix
```

**NOTE:** *The -module and -net arguments should not be used together. In addition, you should always specify the -auto\_fix argument.*

## Rules

The dont\_touch constraint is used by the following rules:

SpyGlass DFT Solution			
Async_07	Clock_11	Latch_05	Latch_08
TA_06			

## expect\_frequency

### Purpose

The `expect_frequency` constraint is used to specify that a certain frequency value is expected at a node in the design. This constraint uses MHz as the frequency unit.

**NOTE:** `require_frequency` is an alias for the `expect_frequency` constraint used by the SpyGlass DFT DSM solution.

There could be a hard macro with no clockshaper inside the hard macro. Therefore, a macro top-level pin directly connects to flip-flops. In such a case, the macro designer has the `expect_frequency` constraint at this macro-top-level pin.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `expectFrequency`.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `expect_frequency` constraint is as follows:

```
current_design <du-name>
  expect_frequency
    [-name <path> ]
    [-except <path to cellport | top_level_pin_name> ]
    [-type <DO_objlist> ]
    [-except_type <ExceptDO_obj_type> ]
    -freqList <freq-symbol-list>
    [-multiplier <freq multiplier> ]
    [-constraint_message_tag <value>]
```

### Arguments

**<du-name>**

(Optional) Name of the design unit under which you are specifying the

instance hierarchies.

**-name <path>**

(Optional) Path to the node where the given frequency is expected, otherwise respective violation occurs.

**-except <path to cellport | top\_level\_pin\_name>**

(Optional) Same as <path> but defines design nodes whose path need not be specified.

**-type <DO\_objlist>**

(Optional) Same as <path> but it takes only macros as inputs.

**-except\_type <ExceptDO\_obj\_type>**

(Optional) Same as <DO\_obj\_type> but it takes only macros as inputs.

**-freqList <freq-symbol-list>**

List of frequency symbols with which the test clock is associated. These can be actual numbers like 100, 200 or alphanumeric symbols like F1, f2, and so on.

**-multiplier <freq multiplier>**

(Optional) Specifies the frequency multiplier for all frequencies given through the `-freqList` argument.

If different multipliers are required, use multiple *expect\_frequency* constraints for the same design node.

**-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**Supported Macros**

(Optional) To view the list of macros supported by the `expect_frequency` constraint, see [Supported Macros](#).

## Examples

### Example 1

Consider the following examples to specify expected frequencies (symbols/numeric) at nodes of interest:

```
expect_frequency -name u1.u7 -freqList F6
expect_frequency -name u2.u3.clk -freqlist 256
```

You can ensure the following:

- Node of interest in the design achieves 'expected' frequency.
- Node of interest in the design does not attain any other frequency.

### Example 2

Consider the following example:

```
expect_frequency -name q1_reg.CP -freqList f0 f1 -multiplier
0.25
```

Above constraint implies that `q1_reg.CP` should get  $f0*0.25$  and  $f1*0.25$  frequencies only.

### Example 3

Consider the following examples:

```
expect_frequency -name q1_reg.CP -freqList f0
-multiplier 0.25
```

```
expect_frequency -name q1_reg.CP -freqList f1 -multiplier 1.5
```

Above constraints imply that `q1_reg.CP` should get  $f0*0.25$  and  $f1*1.5$  frequencies only.

## Rules

The `expect_frequency` constraint is used by the following rule:

---

**SpyGlass DFT DSM Solution**

---

Atspeed\_13

---

## false\_path

### Purpose

The `false_path` constraint enables you to input false and multi-cycle paths that are to be excluded from at-speed testing.

**NOTE:** *You can convert the `set_false_path` SDC command to `false_path` SGDC command using the SDC to SGDC conversion. For details, refer to *Atrenta Console Reference Guide*.*

### Product

SpyGlass DFT DSM solution

### Syntax

```
false_path
  [-from <from_list>]
  [-through <through_list>]
  [-to <to_list>]
```

**NOTE:** *You should specify at least one of these options for the `false_path` constraint.*

### Arguments

The `false_path` constraint has the following arguments:

#### **-from <from\_list>**

Specifies a list of objects that act as the start point for the multi-cycle path. A valid start point is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell. If a clock is specified, all registers related to the clock are considered as start points.

#### **-to <to\_list>**

Specifies a list of objects that act as endpoint for the multi-cycle path. A valid endpoint is a primary output or inout port, a sequential cell, a data pin of a sequential cell.

**-through <through\_list>**

Specifies a list of pins or nets that you want to disable.

**NOTE:** *The DFT DSM policy does not support the -through option.*

**Rules**

The `false_path` constraint is used by the following rules:

SpyGlass DFT DSM Solution			
Info_transitionCoverage	Atspeed_05	Atspeed_06	Diagnose_03
Info_transitionCoverageAudit			

**fifo****Purpose**

The `fifo` constraint provides a mechanism to provide FIFO information so that SpyGlass can perform complete recognition and verification of FIFOs.

**Product**

SpyGlass CDC solution

**Syntax**

The syntax of using the `fifo` keyword in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>
fifo
  [ -memory <memory_name> ]
  [ -rd_data <read_data> -wr_data <write_data> ]
  [ -rd_ptr <read_pointer> -wr_ptr <write_pointer> ]
```

## Arguments

**-memory <memory\_name>**

Allows you to provide the memory usage of the fifo. Memory can be a black box/library cell name, hierarchical net name, hierarchical instance name, or module name of the hierarchical instance containing memory.

**-rd\_data <read\_data> / -wr\_data <write\_data>**

Can be hierarchical nets or hierarchical instance pins. These should be specified as vector signals. Scalar signals are not considered. Both *<read\_data>* and *<write\_data>* options should be specified together.

You can use a combination of wildcard characters ('\*' and '?') when specifying hierarchical net names.

**-rd\_ptr <read\_ptr> / -wr\_ptr <write\_ptr>**

Can be hierarchical nets or hierarchical instance pins. These should be vector signals. Scalar signals are not considered. Both *<read\_ptr>* and *<write\_ptr>* options should be specified together.

You can use a combination of wildcard characters ('\*' and '?') when specifying hierarchical net names.

## Rules

The `fifo` constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Clock_sync03a	Clock_sync03b	Clock_sync08	Clock_sync08a
Clock_sync09	Ac_fifo01	Ac_unsync01	Ac_unsync02
Ac_sync01	Ac_sync02		



## force\_no\_scan

### Purpose

The `force_no_scan` constraint is used to exclude black box, latches, and flip-flops from being declared scannable even if they so qualify.

The `force_no_scan` constraint may be used when there is no intention for scan insertion in a module or circuit or when sequential ATPG tools are planned (in conjunction with the `seq_atpg` constraint.).

Specifying this constraint impacts the reported scan flip-flop percentage. That is, if you specify this constraint, specified and inferred flip-flops are excluded from the denominator while computing the scannable flip-flop percentage.

**NOTE:** *Prior to SpyGlass 5.4.0 release, the name of this constraint was `no_scan`*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `force_no_scan` constraint is as follows:

```
force_no_scan
-name <du-name> | <net-name> | <hier-inst> |
-clock_control <signal-name> |
-set_control <signal-name> |
-reset_control <signal-name>
-register_suffix <suffixes>
-module_suffix <suffixes>
  [ -black_box ]
  [ -latch ]
  [ -flip-flop ]
```

**NOTE:** *The `force_no_scan` constraint supports wildcard characters. Using wildcards, expression is expanded only within the hierarchy.*

### Arguments

The `force_no_scan` constraint has the following arguments:

**-name <du-name>**

The name of the design unit from which scan should be excluded.

You can specify design units that are single flip-flops or design units where one or more flip-flops are described besides other logic. Then, all flip-flops in the specified design unit are excluded from scan.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered.

You can specify a single design unit name or a space-separated list of design unit names.

**-name <net-name>**

The name of a net that is connected to the output pin of a flip-flop.

Then, the corresponding flip-flop is excluded from scan.

You can specify a simple net name or a hierarchical net name. The net specified as simple net name is searched at the top-level.

You can specify a single net name or a space-separated list of net names.

**-name <hier-inst>**

The name of the hierarchical instance names that should be excluded from scan.

**-clock\_control <signal-name>**

Flip-flops whose clock control pin is driven by the specified signal are excluded from the scan.

**-set\_control <signal\_name>**

Flip-flops whose set control pin is driven by the specified signal are excluded from the scan.

**-reset\_control <signal\_name>**

**NOTE:** *The traversal involved in this is structural only.*

**-register\_suffix <suffixes>**

Space-separated list of suffixes to be specified as noscan. The

-register\_suffix argument should not be used along with other arguments of the force\_no\_scan constraint, that is, -name, -clock\_control, -set\_control, or -reset\_control.

If the value of the dft\_treat\_suffix\_as\_pattern 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.

If the value of the dft\_check\_path\_name\_for\_register\_suffix parameter is on, the value of the -register\_suffix field will be matched with the register name along with the path in which the register is present.

#### **-module\_suffix <suffixes>**

Define this field to use suffix based pattern match for all module names.

If the value of the dft\_treat\_suffix\_as\_pattern parameter is on, the value of the -module\_suffix field will be matched with the module name along with the path in which the module is present.

#### **-black\_box**

Marks only black boxes as no scan.

#### **-latch**

Marks only latches as no scan.

#### **-flip\_flop**

Marks only flip-flops as no scan.

**NOTE:** *If you do not specify either -blackbox, -latch, or -flip-flop options, then all black boxes, latches, and flip-flops are marked as no scan.*

## **Examples**

You can use the force\_no\_scan constraint in the following ways:

### **Specifying only the design unit names with the -name argument**

By specifying a design unit name using the -name argument only, all instances of this design unit are considered non-scannable. The following

`force_no_scan` constraint indicates that all flip-flops within all instances of `modName1` will not be considered scannable:

```
force_no_scan -name modName1
```

### Specifying only the net names with the `-name` argument

By specifying a net name using the `-name` argument only, the corresponding flip-flop will be considered non-scannable. The following `force_no_scan` constraint indicates that the flip-flop whose output pin is connected to net `reg_123` (at the top-level) will not be considered scannable:

```
force_no_scan -name reg_123
```

### Specifying only the hierarchical instance names with the `-name` argument

By specifying a hierarchical instance name using the `-name` argument only, all the flip-flops inside the given hierarchy will be considered as non-scannable. The following `force_no_scan` constraint indicates that the flip-flop that lies inside the hierarchy `top.inst1` will not be considered scannable:

```
force_no_scan -name top.inst1
```

**NOTE:** *The `Scan_08` and `Scan_16` rules ignore flip-flops specified in a `force_no_scan` constraint. The effects of a `force_no_scan` constraint specification are visible with the `TA_01` or `TA_02` testability analysis rules.*

**NOTE:** *The `force_no_scan` constraint overrides the `force_scan` constraint.*

### Specifying list of suffixes using the `-register_suffix` argument

Consider the following example:

```
R1 (register 1) name: top.u_ctrl.u2.u1.ff1_ctrl  
R2 (register 2) name: top.u_ctrl.u2.u1.ff1_state  
R3 (register 3) name: top.u_core.u2.u1.ff1_state_ctrl  
R4 (register 4) name: top.u_ctrl_state.u2.u1.ff1_ctrl_state
```

Now, consider the following `force_no_scan` descriptions:

```
force_no_scan -register_suffix ctrl  
force_no_scan -register_suffix state
```

The following table lists the results when combination of values are used for the `dft_treat_suffix_as_pattern` and `dft_check_path_name_for_register_suffix` parameters:

**TABLE 6** Pattern Matching for the `-register_suffix` argument

Value of <code>dft_treat_suffix_as_pattern</code>	Value of <code>dft_check_path_name_for_register_suffix</code>	Value of <code>-register_suffix</code>	Matched Registers
off	off	ctrl	R1, R3
		state	R2, R4
off	on	ctrl	R1, R2, R3
		state	R2, R4
on	off	ctrl	R1, R3, R4
		state	R2, R3, R4
on	on	ctrl	R1, R2, R3, R4
		state	R2, R3, R4

## Rules

The `force_no_scan` constraint is used by the following rules:

---

### SpyGlass DFT Solution

---

All rules

---

## force\_ta

### Purpose

The `force_ta` constraint specifies control and/or observe values for ports/pins/nets.

Use the `force_ta` constraint when it is known that various ports/pins/nets are to be connected or tied off at the next higher level of assembly in order to evaluate the effect of coverage within the current design.

If some pins of this design cannot be fully controlled or observed when instantiated in a larger design and the `force_ta` constraint is not used, the coverage for this design is reported as unrealistically high.

When `test_mode`, `force_ta`, or `test_point` constraints are specified on the same node, following is the priority among different constraints:

- `Test_mode`
- User-specified specific *force\_ta* / *test\_point*
- Effect of *dft\_treat\_primary\_inputs\_as\_x\_source* and *dft\_treat\_primary\_outputs\_as\_unobservable* parameters

For example, if `test_mode 1` and `test_point control` are applied on the same node then the `test_mode` constraint will be considered.

Also, if the `test_mode`, `force_ta`, or `test_point` constraints are found in the fanout of each other, following is the priority among different constraints:

- The constraint in the fanout gets the priority
- Fanin effect is blocked by the specified / resolved constraint on the node

For example, consider that `test_mode 1` is applied on the input of buffer and `test_point control` is applied on the output of the same buffer. In this case, input will have simulation value 1 and nyn controllability but output will have yyn controllability and no simulation value.

**NOTE:** *The `force_ta` constraint impacts the observability for the fan-in logic-cone of the net where `force_ta` is applied. However, it does not impact the terminal. Similarly, it impacts the controllability of the fan-out logic cone of the net where `force_ta` is applied. However, it does not impact the terminal. See [Figure 55](#) and [Figure 56](#) for more information on the application of the `force_ta` constraint.*

## Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

## Syntax

The syntax for `force_ta` constraint is as follows:

```
force_ta
  -name <net-name>
    -control <ctrl-value> | -observe <obs-value>
```

## Arguments

### **-name <net-name>**

The port/pin/net on which control or observe value is being set.

You can specify a single port/pin/net name or a space-separated list of port/pin/net names.

The port names can be simple names while pin names and net names should be hierarchical names.

### **-control <ctrl-value>**

The control value for a port/pin/net.

The possible control values are as follows:

<b>&lt;ctrl-value&gt;</b>	<b>Purpose</b>
nnn	Not controllable
ynn	Controllable to 0
nyn	Controllable to 1
nyy	Controllable to Z
yyn	Controllable to 0 and 1
yny	Controllable to 0 and Z
nyy	Controllable to 1 and Z
yyy	Controllable to 0, 1, and Z

**-observe <obs-value>**

The observe values for a port/pin/net.

The possible observe values are as follows:

<b>&lt;cvalue&gt; or &lt;obs-value&gt;</b>	<b>Purpose</b>
n	Not observable
y	Observable

**Examples**

Consider the following example:

```
force_ta -name pin1[3:2] -value ynn
force_ta -name outPin -observe n
force_ta -name inPin[1:2] -control nyy
```

Here, the port `pin1[3:2]` is being made controllable to just '0', port `outPin` is being made unobservable, and port `inPin[1:2]` is being made controllable to '1' and 'Z'.

**Rules**

The `force_ta` constraint is used by the following rules:

<b>SpyGlass DFT Solution</b>			
Bist_04	Info_uncontrollable	Info_unobservable	Info_undetectCause
Info_path	Info_coverage	Info_untestable	Coverage_audit
TA_01	TA_02	TA_06	
<b>SpyGlass DFT DSM Solution</b>			
All rules			



## force\_probability

### Purpose

The `force_probability` constraint is used to specify the probability of controllability and observability of a design node.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax of the `force_probability` is as follows:

```
force_probability
  -name <node_name>
  [ -control_one <ctrl1_prob> ]
  [ -control_zero <ctrl0_prob> ]
  [ -observe <obs_prob> ]
```

### Arguments

**-name <node\_name>**

Name of the pin, port, or net for which probability is specified.

**-control\_one <ctrl1\_prob>**

(Optional) Probability of controllability to 1 for the design node.

**-control\_zero <ctrl0\_prob>**

(Optional) Probability of controllability to 0 for the design node.

**-observe <obs\_prob>**

(Optional) Probability of observability for the design node.

### Rules

The `force_probability` constraint is used by the

Info\_random\_resistance rule of the SpyGlass DFT DSM solution:

## Examples

The following examples show the usage of the `force_probability` constraint:

### Example 1

```
force_probability -name in1 -control_zero 0.4 -control_one  
0.6
```

### Example 2

```
force_probability -name Q -observe 0.05
```

### Example 3

```
force_probability -name top.inst.in1 -control_zero 0.4  
-control_one 0.6 -observe 0.5
```

## formal\_analysis\_filter

### Purpose

The `formal_analysis_filter` constraint is used to specify the modules or hierarchies on which formal analysis should be ignored or performed.

### Product

SpyGlass Auto Verify

### Syntax

The syntax of the `formal_analysis_filter` constraint is as follows:

```
formal_analysis_filter
  -module_names <design-unit-names>
  [ -rules <rule-names> ]
  [ -hierarchical <yes | no> ]
  [ -analyze <yes | no> ]
```

### Arguments

#### **-module\_names <design-unit-names>**

Specifies the design units on which formal analysis should be ignored or performed.

For Verilog design, a design unit name is a module name.

For VHDL design, a design unit name is an entity name (`<entity-name>`) or an architecture name (`<entity-name>.<architecture-name>`).

#### **-rules <rule-names>**

Specifies the SpyGlass Auto Verify rules on which this constraint is applicable.

By default, this constraint is applicable to all the SpyGlass Auto Verify rules.

**-hierarchical <yes | no>**

Set this argument is set to `yes` so that the nets present directly inside the specified module (`-module_names <design-unit-names>`) are only considered for matching with the *Start Nets* of a property.

By default, this argument is set to `yes` so that the nets present within the specified module (`-module_names <design-unit-names>`) and also its submodules are considered for matching with the *Start Nets* of a property.

**-analyze <yes | no>**

By default, this argument is set to `no` so that property analysis by formal engines is ignored when all the *Start Nets* of that property belong to the specified block (`-module_names <design-unit-names>`) or hierarchy (`-hierarchical <yes | no>`).

For example, consider the user-specified modules M1 and M2. In this case, by default:

- If the P1 property contains the start nets N1 (present in M1) and N2 (present in M2), P1 is ignored.
- If the P2 property contains start net N1 (present in M1) and N2 (present in M3), P2 is not ignored.

Set this argument to `yes` to start property analysis by formal engines when any one of the start nets of that property belong to the specified block or hierarchy.

**Start Nets**

Start nets are the nets from where the property/assertion checking should start.

For SpyGlass Auto Verify RTL rules, such as `Av_deadcod01` and `Av_range01`, start nets belong to the RTL block in which a property is modeled.

For SpyGlass Auto Verify flat rules, such as `Av_bus01`, `Av_bus02`, and `Av_staticnet01`, the driver of the net on which the property is modeled is considered as the start net.

**Rules**

The `formal_analysis_filter` constraint is used by the following

rules:

---

### SpyGlass Auto Verify

---

All implicit property rules

---

## fsm

### Purpose

The `fsm` constraint is used to:

- Modify or remove the FSMs, FSM states, and FSM transitions.
- Add FSMs, FSM states, and FSM transition.

### Product

SpyGlass Auto Verify

### Syntax

The syntax of the `fsm` constraint is as follows:

#### Usage 1

```
fsm
  -name <fsm-logical-name>
  [ -state_value <state-values> ]
  [ -from_state_value <from-state-values> ]
  [ -to_state_value <to-state-values> ]
  [ -append | -remove ]
```

#### Usage 2

```
fsm
  -module <module-name>
  -state_variables <FSM-state-variables> ]
  [ -state_value <state-values> ]
  [ -from_state_value <from-state-values> ]
  [ -to_state_value <to-state-values> ]
  [ -append | -remove ]
```

## Arguments

### **-name <fsm-logical-name>**

Specifies the name of the FSM to be added, modified, or deleted.

### **-module <module-name>**

Specifies the module in which the FSM is encoded.

It can be a top-level module or a sub module/entity in the design.

### **-state\_variables <FSM-state-variables>**

Specifies a space-separated list of synthesized nets of a module.

Some examples are as follows:

- *-state\_variables state[4:0]*
- *-state\_variables state*
- *-state\_variables state[4] state[3] state[2] state[1] state[0]*

Note that a space-separated list is considered as concatenation. Therefore, check for the order of values specified to this argument. Consider the following example:

```
fsm -module M1
  -state_variables curr_state0 curr_state1
  -state_value 00 01
```

The equivalent RTL for the above fsm constraint is as follows:

```
case (curr_state0, curr_state1)
  2'b00: ...
  2'b01: ...
endcase
```

### **-state\_value <state-values>**

Specifies a space-separated list of state values for the FSM.

The allowed values to this argument are same as the allowed values to the `-value` argument of the `set_case_analysis` constraint. For details, refer to the documentation of the `set_case_analysis` constraint.

Some examples of the values to this argument are as follows:

- 1010101010
- "d 5" or "D 5"
- "h AB" or "H AB"

**NOTE:** *When the width of a state variable mismatches with the state value, redundant bits are removed or non-specified bits in the state value are considered as zero.*

**-from\_state\_value <state-values> -to\_state\_value <state-values>**

The arguments together specify the transition of an FSM.

The format of the <state-value> arguments is same as described in [-state\\_value <state-values>](#).

If you specify a new state value for an FSM, which is not automatically-detected, or if you specify a state value that is not already specified to the [-state\\_value <state-values>](#) argument, SpyGlass adds that state value to the [-state\\_value <state-values>](#) argument.

**-append | -remove**

-append is the default argument over -remove.

Use -append to modify the RTL of the automatically-detected FSM or create a new FSM.

Use -remove to remove the specified FSM state, state values (and its corresponding transition), or transition from an FSM.

## Examples

### Example 1

The following constraint removes the FSM detected on s[4:0] from the M1 module:

```
fsm -module M1 -state_variables s[4:0] -remove"
```

### Example 2

The following constraint removes the FSM state 0000 and its corresponding transition with the FSM state variable s [4 : 0] from the M1 module:

```
fsm -module M1 -state_variables s[4:0] -state_value 0000 -
```

remove

### Example 3

Consider the following constraint:

```
fsm -module M1 -state_variables s[4:0] -from_state_value  
0000 -to_state_value 0001 -append
```

When you specify the above constraint, one of the following occurs based on a condition:

- **Condition:** When an FSM with the `s [4 : 0]` state variable is not detected in the M1 module  
**Result:** SpyGlass adds a new FSM with the specified state values transition.
- **Condition:** When an FSM with the `s [4 : 0]` state variable is detected in the M1 module but either of states, 0000 or 0001, is not detected  
**Result:** SpyGlass adds the specified state values and the new transition to the FSM.
- **Condition:** When an FSM with the `s [4 : 0]` state variable is detected in the M1 module along with both the states, 0000 and 0001  
**Result:** SpyGlass adds the specified transition to the FSM.
- **Condition:** When an FSM with the `s[4:0]` state variable is detected in the M1 module but both the states, 0000 and 0001, and their transition is automatically-detected  
**Result:** SpyGlass reports the `SGDC_fsm_Setup01` violation.

## gating\_cell

### Purpose

Specifies the user-defined clock-gating cell.

By default, a 2 input gate on the clock line or an integrated clock-gating is detected as a clock-gating cell.

You should use the `gating_cell` constraint to specify complex clock-gating cells.



## Key Points

- For a white box model, `gating_cell` constraint specification takes precedence over the synthesizable logic.
- If the `gating_cell` `sgdc` command is specified on a lib cell then its internal functionality is completely ignored and the cell is treated like a black box with the `gating_cell` constraint.
- For SpyGlass DFT or SpyGlass DFT DSM solution, the `gating_cell` constraint is not necessary when the white box model resembles a latch-based standard integrated clock gating library cell. This automatic inference is controlled by the `dft_infer_clock_gating_cell` parameter. For details of this parameter, please refer to SpyGlass DFT Rules Reference Guide or SpyGlass DFT DSM Rules Reference Guide.
- Prior to SpyGlass 4.3.0 release, the name of this constraint was `gatingcell`.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions, SpyGlass DFT solution, SpyGlass DFT DSM solution

## Syntax

The syntax to specify the `gating_cell` constraint is as follows:

```
current_design <du-name>
  gating_cell
    -name <cell-name>
    [-clkInTerm <clk-in-pin>]
    [-clkOutTerm <clk-out-pin>]
    [-enTerm <enable-pin>]
    [-enValue <enable-value> ]
    [-testenTerm <test-enable-pin>]
    [-testenValue <test-enable-value>]
    [-cgEdgeType <positive | negative>]
    [-obsTerm <obs-pin>]
```

**NOTE:** *The `gating_cell` constraint supports wildcard characters.*

## Arguments

**<du-name>**

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**-clkInTerm <clk-in-pin>**

Name of the clock input terminal, which is a single-bit pin.

**-clkoutTerm <clk-out-pin>**

Name of the output clock terminal, which can be either single-bit or multi-bit.

**-enTerm <enable-pin>**

Name of the input enable terminal through which a clock-gating cell can be enabled or disabled. This can be either a single-bit or a multi-bit pin. The size of `<enable_pin>` should match the size of `<clk_out_pin>`. Also, if you want to enable the  $i^{\text{th}}$  bit of the `<clk_out_pin>`, ensure that the  $i^{\text{th}}$  bit of the `<enable_pin>` is set. To understand the relationship between the multi-bit system-enable and clock-out pins, refer to [Figure 32](#).

**NOTE:** *If `<enable-pin>` is not specified, it implies that a clock-gating cell is always enabled. Therefore, a clock reaching `<clk-in-name>` always passes through `<clk-out-name>`.*

**-enValue <enable-value>**

**NOTE:** *This option is not applicable for the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.*

Value on the enable pin that disables the gating cell and allows a clock reaching the input terminal to pass through the clock output terminal.

**NOTE:** *If `<enable-pin>` is specified and `<value>` is not specified, it implies that the signal value is active high (value 1).*

**-testenTerm <test-enable-pin>**

Name of the input enable terminal in the test mode. The pin through which a clock-gating cell can be enabled or disabled in the test mode.

**-testenValue <test-enable-value>**

**NOTE:** This option is not applicable for the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

Value on the enable pin in the test mode that disables the gating cell and allows a clock reaching the input terminal to pass through the clock output terminal.

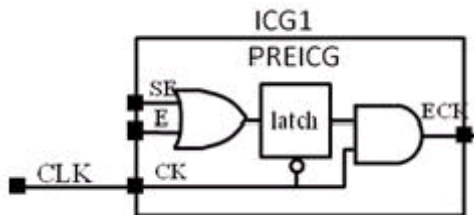
**-cgcEdgeType <positive | negative>**

Used to define the edge of the clock gating cell. By default, edge type is positive. You can also set the value of this argument as negative.

**Positive Edge**

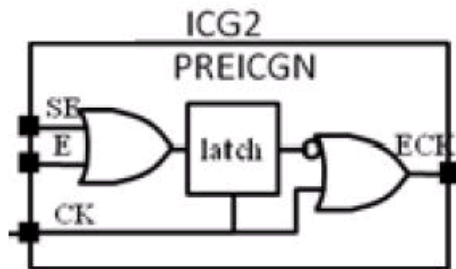
Positive edge type signifies that latch-enable pin gets the inverted clock (clock input to gating\_cell).

The following figure illustrates a positive edge type:

**Negative Edge**

Negative edge type signifies that latch-enable pin gets the clock (clock input to gating\_cell) without inversion.

The following figure illustrates a negative edge type:



**-obsTerm <obs-pin>**

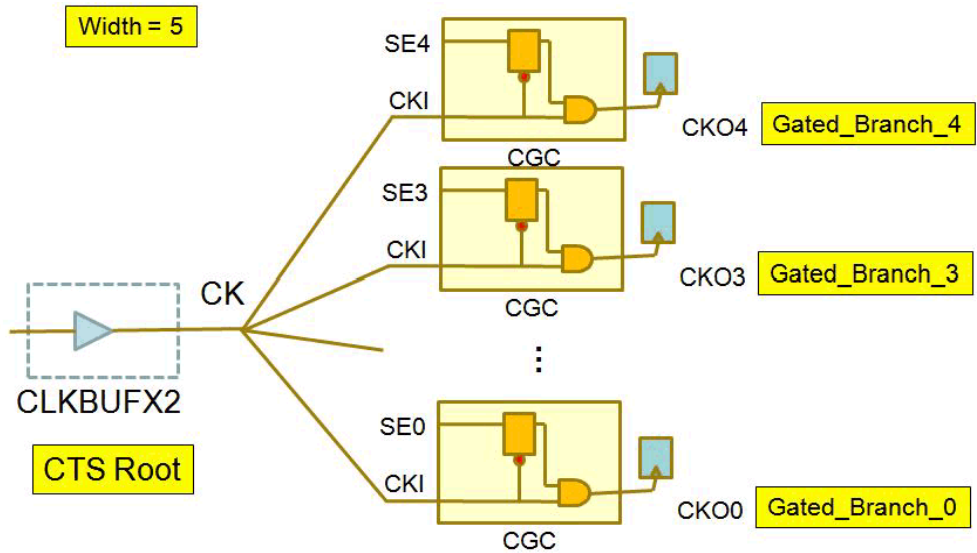
Name of the observation pin using which we observe the enable signal of the cell.

Consider the following example:

```
module dpICG(I, SE, Z, EN);
    parameter width = 1;
    output [width -1:0] Z;
    input [width-1:0] EN;
    input I;
    input SE;
```

```
endmodule
```

The following figure shows the schematic for the above code:



**FIGURE 32.** Relation Between Multi-Bit System-Enable Pin and Clock-Out Pin

The above code and figure explain the relationship between the multi-bit system-enable and clock-out pins.

## Rules

The `gating_cell` constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	PESTR03	PESTR05
PESTR06	PESTR07	PESTR08	PESTR09
PESTR10	PESTR11	PESTR12	PESTR13
poweraudit			
<b>SpyGlass DFT Solution</b>			
All rules			
<b>SpyGlass DFT DSM Solution</b>			
All rules			

## gating\_cell\_enable

### Purpose

The `gating_cell_enable` is used to specify the gating cell enable signal.

**NOTE:** *Prior to SpyGlass 4.4 release, the name of this constraint was `gatingcell_enable`.*

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `gating_cell_enable` constraint is as follows:

```
gating_cell_enable
  -name <signal-name>
  -type phased_clock_enable
  -pipeline_depth <depth> |
  -pipeline_depth_range <min max>
```

### Arguments

**-name <signal-name>**

Signifies root level input ports, which are used to drive the test enable pin of a clock-gating cell.

**-type phased\_clock\_enable**

Specifies that the phased clocks would be used in the LSSD design style.

**-pipeline\_depth <depth>**

Signifies the pipeline stage that is expected on `gating_cell_enable` signal. You can specify a positive integer value as an input.

**NOTE:** *Specify either `-pipeline_depth` or `pipeline_depth_range` argument. Do not use both the arguments together.*

**-pipeline\_depth\_range <min max>**

Specifies the range of pipeline stages expected on gating\_cell\_enable signal.

**NOTE:** *Specify either -pipeline\_depth or pipeline\_depth\_range argument. Do not use both the arguments together.*

**Rules**

The gating\_cell\_enable constraint is used by the following rule:

---

**SpyGlass DFT DSM Solution**


---

CG\_04

---

**generated\_clock****Purpose**

The generated\_clock constraint is used to specify the clock that traverses from the output (hierarchical pin or net) of a sequential element.

By default, when a clock reaches a sequential element, it propagates beyond that sequential element irrespective of whether the output of the element is reaching a clock pin.

Use this constraint to stop propagation of such clocks beyond sequential elements such that the clock specified by this constraint is propagated beyond the sequential element. For details, see [Examples](#).

If the generated\_clock constraint is defined at the output of a combo logic that is placed just after first sequential element receiving the master clock, the generated\_clock constraint is honored. For details, see [Example 3](#).

This constraint is read only if the enable\_generated\_clocks parameter is set to yes.

**NOTE:** *During SDC-to-SGDC translation, the create\_generated\_clocks SDC command is saved as the generated\_clock constraint.*

## Product

SpyGlass CDC solution

## Syntax

The syntax to specify the `generated_clock` constraint is as follows:

```
current_design <du-name>
generated_clock -name <clk-obj-name>
    -source <source-obj-name>
    [ -tag <tag-name> ]
    [ -divide_by <divide-factor> ]
    [ -multiply_by <mult-factor> ]
    [ -master_clock <source-clock-tag-name> ]
    [ -add ]
    [ all ]
```

## Arguments

### **-name <clk-obj-name>**

Specifies the name of the clock pin or net present on the output of a sequential element so that the generated clock propagates from that pin or net.

See [Example 1](#).

### **-source <source-obj-name>**

Specifies the clock reaching the sequential element so that propagation of this clock beyond the sequential element is stopped when the `generated_clock` constraint is specified.

See [Example 1](#).

### **-tag <tag-name>**

Specifies the tag of the generated clock.

See [Example 1](#).



**-divide\_by <divide-factor>**

Specifies the frequency division factor.

For example, if this argument is 2, the generated clock period is twice as long as the master clock period.

**-multiply\_by <mult-factor>**

Specifies the frequency multiplication factor.

For example, if this argument is 3, the generated clock period is one-third as long as the master clock period.

**-master\_clock <source-clock-tag-name>**

Specifies the tag name of the clock that is the master of the source clock specified by the *-source <source-obj-name>* argument.

See [Example 1](#).

**-add**

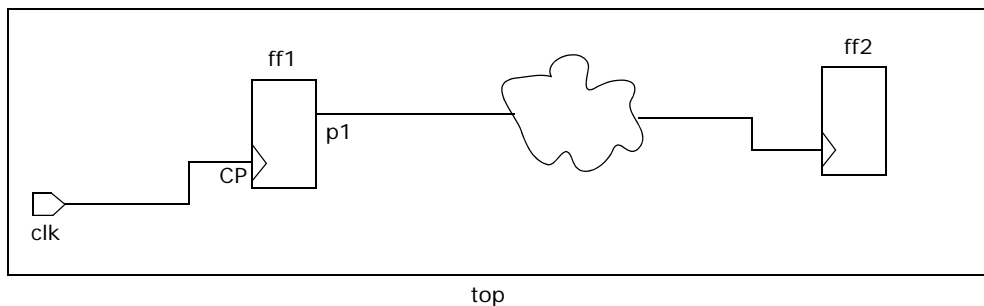
Specify this argument to add multiple `generate_clock` constraints on a single object.

See [Example 2](#).

## Examples

### Example 1

Consider the following figure:



```
// SGDC File:
clock -name top.clk -period 10 -domain d1 -tag T1
```

**FIGURE 33.** Example 1 - generated clock

In the above scenario, by default, the `clk` clock propagates beyond the `ff1` flip-flop.

To stop the propagation of `clk` beyond `ff1` and propagate another clock from the output `p1` of `ff1`, perform the following steps:

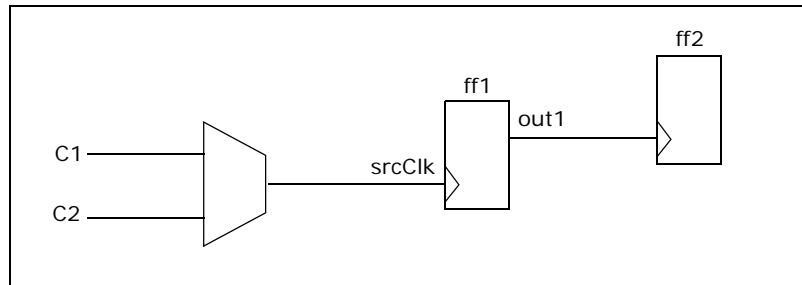
1. Set the `enable_generated_clocks` parameter to `yes`.
2. Specify the following constraint to generate a clock on the output `p1` of the `ff1` flip-flop, where the source clock `CP` is derived from the master clock `top.clk` (tag `T1`):

```
generated_clock -name top.ff1.p1 -source top.ff1.CP
-tag GT1 -divide_by 2 -master_clock T1
```

After performing the above steps, a generated clock propagates from the `p1` pin of the `ff1` flip-flop.

## Example 2

Consider the following figure:



```
// SGDC File:
clock -name C1 -period 10 -domain d1 -tag T1
clock -name C2 -period 15 -domain d2 -tag T2
```

**FIGURE 34.** Example 2 - generated clock

In the above scenario, multiple master clocks converge on the source clock `srcClk`. In this case, to stop propagation of `srcClk` beyond `ff1` and enable a generated clock traverse from `out1` of `ff1`, perform the following steps:

1. Set the `enable_generated_clocks` parameter to `yes`.
2. Specify the following constraints:

```
generated_clock -name out1 -source srcClk -master_clock C1
-tag T1 -divide_by 2
```

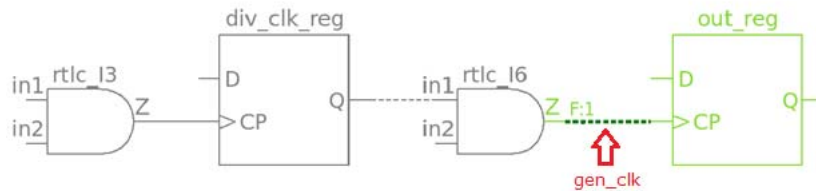
```
generated_clock -name out1 -source srcClk -master_clock C2
-tag T2 -add -divide_by 2
```

You should specify a `generated_clock` constraint with respect to each master clock of the source clock.

Alternatively, select one master clock by defining a mode and then specify one `generated_clock` constraint with respect to the selected master clock.

### Example 3

Consider the following figure:

**//SGDC File:**

```
generated_clock -name test.gen_clk -source test.div_clk_reg.CP
-master_clock T1 -divide_by 2 -tag T4
```

**FIGURE 35.** Example 3 - generated clock

In the above example, if the `generated_clock` constraint is defined on the `gen_clk` net with source as `div_clk_reg.CP`, SpyGlass CDC honors this constraint and the generated clock is created on the `gen_clk` net.

## glitch\_free\_module

### Purpose

The `glitch_free_module` constraint is used to specify the modules where if a constant value is reaching the data pin of a flop in the module, it will be propagated to the output of the flop even if preset/clear pins are present.

### Product

SpyGlass CDC Solution

### Syntax

The syntax to specify the `glitch_free_module` constraint is as follows:

```
glitch_free_module -name <du-name>
```

## Arguments

**-name** <du-name>

Name of the module (for Verilog designs) and <entity-name> or <entity-name>.<arch-name> (for VHDL designs).

## Example

```
glitch_free_module -name MOD
```

## Rules

The `glitch_free_module` constraint is used by the following rules:

---

### SpyGlass CDC Solution

---

All CDC rules

---

## gray\_signals

### Purpose

The `gray_signals` constraint is used to specify the signals that should be gray encoded.

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `gray_signals` constraint is as follows:

```
gray_signals -name <signal-name-list>
```

### Arguments

**-name** <signal-name-list>

Specifies a space-separated list of signals, such as hierarchical net names,

hierarchical terminals, and ports.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

## Rules

The `gray_signals` constraint is used by the following rules:

SpyGlass CDC Solution		
Ac_conv05	SGDC_gray_signals01	SGDC_gray_signals02
SGDC_gray_signals03		

## ignore\_clock\_gating

### Purpose

The `ignore_clock_gating` constraint is used to disable generation of clock-gating logic for:

- Enabled flip-flops driven by the specified clock nets.
- Enabled flip-flops present within the specified modules.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `ignore_clock_gating` constraint is as follows:

```
ignore_clock_gating
  -clock <clock-nets> | -module <module-list>
```

### Arguments

**-clock <clock-nets>**

List of clock nets so that clock-gating logic is not generated for the enabled

flip-flops driven by these clock nets. The type of clock nets supported are:

- Port clocks
- Clocks driven by black boxes
- Clocks specified using the `clock` SGDC constraint

#### `-module <module-list>`

List of modules so that clock-gating logic is not generated for the enabled flip-flops present within these modules.

## Examples

Consider the top design that has:

- The `clk1` and `ckl2` clocks.
- The `middle1` and `middle2` modules.

For the above design, consider you specify the following constraints:

```
ignore_clock_gating -clock "top.clk1"
ignore_clock_gating -module "middle1"
```

On specifying the above constraints, SpyGlass will not use ICGC for enabled flops present either in middle or driven by the `top.clk1` clock. It uses the ICGC only for the enabled flip-flops inside `middle2` and driven by the `top.clk2` clock.

## Rules

The `ignore_clock_gating` constraint is used by the following rules:

---

### SpyGlass Power Estimation and SpyGlass Power Reduction solutions

PEPWR01	PEPWR02	PEPWR03	PEPWR14	PEPWR20
PEPWR21	PEPWR22	PEPWR23	PEPWR24	PEPWR25
PEPWR28	PESTR03	PESTR06	PESTR08	PESTR12
PESTR13	PESTR20	PESTR21	PESTR22	PESTR23
PESTR24	PESTR25	PESTR28	poweraudit	

---

## ignore\_crossing

### Purpose

The `ignore_crossing` constraint is used to specify power domain to voltage domain crossings and power domain to power domain crossings that should be ignored by the LPSVM08, LPSVM09, LPSVM10, LPSVM23, and LPSVM47 rules of the SpyGlass Power Verify solution.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `ignorepdcrossing`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `ignore_crossing` constraint is as follows:

```
current_design <du-name>
  ignore_crossing
    -from <src-pd-name>
    -to <dest-pd-name> | <vd-name>
```

### Arguments

**<du-name>**

Name of the design unit under which you are specifying the crossings to be ignored

**-from <src-pd-name> | -to <dest-pd-name>**

Name of the power domain (source, destination)

**<vd-name>**

Name of the voltage domain (only destination)



## Rules

The `ignore_crossing` constraint is used by the following rules:

<b>SpyGlass Power Verify Solution</b>			
LPSVM08	LPSVM09	LPSVM10	LPSVM22
LPSVM23	LPSVM47		

## ignore\_supply\_pin

### Purpose

The `ignore_supply_pin` constraint is used to specify specific pins of a cell on which the LPPLIB06 or LPPLIB15 rules should not report a violation.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `ignore_supply_pin` constraint is as follows:

```
current_design <du-name>
  ignore_supply_pin
    -cell <cell-name>
    -pin <pin-name-list>
```

### Arguments

**<du-name>**

Name of the design unit

**-cell <cell-name>**

Name of a cell. This field supports wildcards.

**-pin <pin-name-list>**

Name of specific pins of a cell. This field supports wildcards.

### Rules

The `ignore_supply_pin` constraint is used by the following rules:

---

**SpyGlass Power Verify Solution**

---

LPPLIB06

LPPLIB15

---

## illegal\_constraint\_message\_tag

### Purpose

The *illegal\_constraint\_message\_tag* constraint checks 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.

### Product

SpyGlass DFT Product

### Syntax

The syntax to specify the *illegal\_constraint\_message\_tag* constraint is as follows:

```
illegal_constraint_message_tag
  [-name <nodename>]
  [-except <except_nodename>]
  [-except_type <exceptDo-nodename>]
  [-type <DO_nodename>]
  [-constraint_message_tag_expression
  <constraint_message_tag_expression>]
```

### Argument

**[-name] <nodename>**

Specifies the name of the top-module port, or any internal net or terminal or leaf instance for which the specified tag expression must be satisfied. A module name specified expands to list of all its instantiations (full hierarchical name). When at least one of the pins of the instance gets required expression, a FAIL status is generated. However, if none of the pins of the instance gets required expression, a PASS status is generated.

For more information, see [Example 2](#).

**[-except <except\_nodename>]**

Specifies the name of the top-module port, any internal net, terminal, or leaf instance name, which needs to be excluded from rule checking.

**-except\_type <exceptDo-nodename>**

Specifies the name of macro for which the specified tag expression must be satisfied.

**-type <DO\_nodename>**

Specifies the name of the macro, which needs to be excluded from rule checking.

**-constraint\_message\_tag\_expression <constraint\_message\_tag\_expression>**

Signifies the message tag expression specified using logical '||' and '&&' operator and their combinations and :PASS and :FAIL values of tags. You can also use braces ('(',')') when specifying message tag expression.

**Examples****Example 1**

Consider the following example:

```
-from "top.cgc_1.clkout" -to_type
FLIP_FLOrequire_pathP_CLOCK LATCH_ENABLE
-constraint_message_tag CGC_CHECK_1

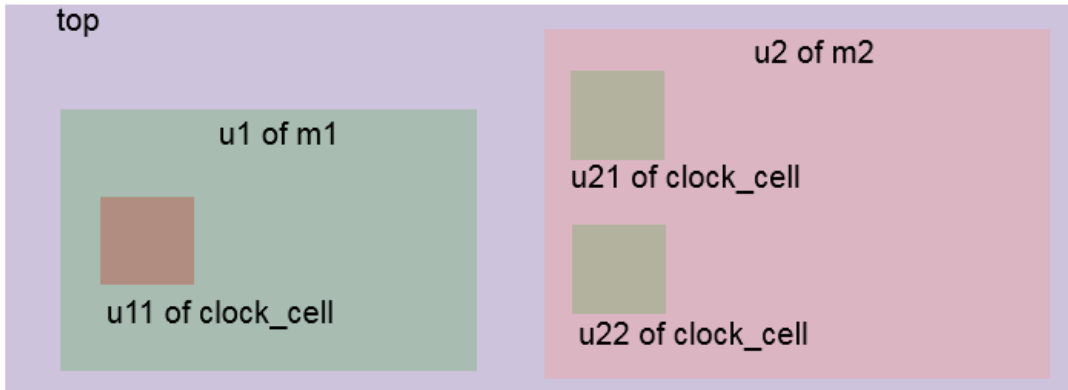
require_path -from "top.cgc_2.clkout" -to_type
FLIP_FLOP_CLOCK LATCH_ENABLE -constraint_message_tag
CGC_CHECK_2

illegal_constraint_message_tag -type ICG -
constraint_message_tag_expression "CGC_CHECK_1:PASS ||
CGC_CHECK_2:FAIL"
```

In the above example, the `illegal_constraint_message_tag` reports violation if `CGC_CHECK_1` does not have any violation or `CGC_CHECK_2` have any violation.

**Example 2**

Consider the following top instantiation:



**FIGURE 36.** Top Instantiation

Now, consider the following SGDC command:

```
illegal_constraint_message_tag -name clock_cell <other options>
```

The above SGDC command implies:

```
illegal_constraint_message_tag -name top.u1.u11 top.u2.u21  
top.u2.u22 <other options>
```

## illegal\_path

### Purpose

The `illegal_path` constraint is used to define nodes between which path should not exist.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `illegal_path` constraint is as follows:

```
illegal_path
  [ -from <from_node_list> ] [ -to <to_node_list> ]
  [ -except_from <except_from_list> ]
  [ -except_to <except_to_list> ]
  [ -from_type <from_type_list> ]
  [ -from_one_of <fromoneof_pinlist> ]
  [ -from_one_of_type <fromoneof_DO_pinlist> ]
  [ -exact_sequential_depth <sequential_depth> ]
  [ -sequential_depth <value> ]
  [ -except_from_type <except_from_type_list> ]
  [ -to_type <to_type_list> ]
  [ -except_to_type <except_to_type_list> ]
  [ -to_one_of <tooneof_pinlist> ]
  [ -to_one_of_type <tooneof_DO_pinlist> ]
  [ -tag <tag_name> | -use_shift | -use_capture |
    -use_captureATspeed ]
  [ -path_type <buffered | sensitized | topological |
  sensitizable> ]
  [-constraint_message_tag <value>]
  [-report_failure_as_info]
  [-min_to_paths <value>]
  [-max_to_paths <value>]
  [ -min_from_paths ]
  [ -max_from_paths ]
  [-filter_in_cmt_from
```

```

<constraint_message_tag_expression>]
  [-filter_in_cmt_to <constraint_message_tag_expression>]
  [-named_association]
  [-positional_association]
  [-instance_association]
  [-instance_filter_in_cmt_from
<constraint_message_tag_expression>]
  [-instance_filter_in_cmt_to
<constraint_message_tag_expression>]
  [ -filter_in_from <include_from_pinlist> ]
  [ -filter_in_to <include_to_pinlist> ]
  [ -filter_in_type_from <include_from_DO_pinlist> ]
  [ -filter_in_type_to <include_to_DO_pinlist> ]
  [ -ignorecase ]

```

## Guidelines for Using Arguments

The following combination of arguments are not allowed for the `illegal_path` constraint:

- The `-from_one_of` argument can not be used with the `-to_one_of` argument
- The `-min_from_paths` argument cannot be used with the `-max_from_paths` argument
- The `-min_to_paths` argument cannot be used with `-max_to_paths` argument

Also, The `-min_to_paths`, `max_to_paths`, `min_from_paths`, and `max_from_paths` arguments should have the corresponding `-from/from_type/from_one_of/ from_one_of_type` arguments, as well as, the `-to/to_one_of/to_type /to_one_of_type` arguments.

## Arguments

**-from <from\_node\_list>**

(Optional) This is a list of top-module port names, internal net names, or

terminal names from which path should not exist to nodes specified through <to node list> and <to type list>, if both <to type list> and <to node list> are not given then this implies that nodes given in <from node list> should not be driving any leaf cell/top module port.

**-to <to\_node\_list>**

(Optional) This is a list of top-module port names, internal net names, or terminal names to which path should not exist from nodes specified through <from node list> and <from type list>, if both <from type list> and <from node list> are not given then this implies that nodes given in <to node list> should not be driven by any leaf cell/top module port.

**-exact\_sequential\_depth <sequential\_depth>**

(Optional) Defines the exact sequential depth. This argument takes an integer as an input.

Note that you can not use this argument with the `-sequential_depth` argument.

**-sequential\_depth <value>**

Specifies the number of sequential elements between end points specified on an illegal path. This means that the *illegal\_path* check will go through the specified number of sequential elements. You can specify an integer value as an input to this argument.

It is recommended to use this argument when the path type is set to sensitizable. For other path\_types, that is, direct, buffered, and sensitized, a sequential element is a stop-point. This is so because path through sequential cell is neither buffered nor sensitized path.

**-except\_from <except\_from\_list>**

(Optional) Nodes specified in this list are effectively removed from list of nodes specified in <from node list>

**-except\_to <except\_to\_list>**

(Optional) Nodes specified in this list are effectively removed from list of nodes specified in <to node list>



**-from\_type <from\_type\_list>**

(Optional) Same as <from node list> but it takes only macros as inputs.

**-from\_one\_of <fromoneof\_pinlist>**

(Optional) Specifies that no error message is reported, if there is at least one success case among the specified nodes.

**-from\_one\_of\_type <fromoneof\_DO\_pinlist>**

(Optional) Same as -from\_one\_of <fromoneof\_pinlist> but it takes only macros as inputs.

**-except\_from\_type <except\_from\_type\_list>**

(Optional) This is a list of macros, nodes specified through this list are effectively removed from list of nodes specified through <from type list>.

**-to\_type <to\_type\_list>**

(Optional) Same as <to node list> but it takes only macros as inputs.

**-to\_one\_of <tooneof\_pinlist>**

(Optional) Specifies that no error message is reported, if there is at least one success case among the specified nodes.

**-to\_one\_of\_type <tooneof\_DO\_pinlist>**

(Optional) Same as -to\_one\_of <tooneof\_pinlist> but it takes only macros as inputs.

**-except\_to\_type <except\_to\_type\_list>**

(Optional) This is a list of macros, nodes specified through this list are effectively removed from the list of nodes specified through <to type list>.

**-tag <tag name> | -use\_shift | -use\_capture | -use\_captureATspeed**

(Optional) These arguments are used to specify simulation condition under which path is searched between from nodes and to nodes. If none of them are specified then path is searched after simulating all power-ground connections. Please note that only one of these arguments can be used to specify simulation condition in the illegal\_path constraint.

<tag name> is a condition name that has been previously defined by using the `define_tag` constraint.

If `-use_shift`, `-use_capture` or `-use_captureATspeed` is specified then `shift`, `capture` or `captureATspeed` mode is simulated respectively.

**-path\_type** <buffered | sensitized | topological | sensitizable>

The `-path_type` argument accepts only predefined list of values: `buffered`, `sensitized`, `topological`, and `sensitizable`. The default value of this qualifier is `sensitizable`. This argument determines the type of path that is searched between from and to nodes.

See for more information, see [Example 6](#) and [Example 7](#).

**-constraint\_message\_tag** <value>

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

### Supported Macros

To view the list of macros supported by the `require_strict_path` constraint, see [Supported Macros](#).

**-report\_failure\_as\_info/-report\_failures\_as\_info**

Reports all the failures as info severity message.

**-min\_to\_paths** <value>

(Optional) Specifies minimum number of expected successful paths. You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**-max\_to\_paths** <value>

(Optional) Specifies maximum number of expected successful paths.

The following are the rules for using this argument:

- If you are using both the `-min_to_paths` and `-max_to_paths` arguments, then the value of the `-max_to_paths` argument should be greater than the `-min_to_paths` argument.

- You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**`-min_from_paths <value>`**

(Optional) Specifies minimum number of expected successful paths. You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**`-max_from_paths <value>`**

(Optional) Specifies maximum number of expected successful paths. The following are the rules for using this argument:

- If you are using both the `-min_from_paths` and `-max_from_paths` arguments, then the value of the `-max_from_paths` argument should be greater than the `-min_from_paths` argument.
- You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**`-named_association`**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same name, from the expanded from-to-set. Individual checks are then performed on each such sub-group.

**`-positional_association`**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same position, from the expanded from-to-set. Individual checks are then performed on each such sub-group.

**`-instance_association`**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same instance, from the expanded from-to-set. Individual checks are then performed on each such sub-group. This argument is useful while looking for a self-loop type structure.

**`-filter_in_cmt_from <constraint_message_tag_expression>, -filter_in_cmt_to <constraint_message_tag_expression>`**

(Optional) Filters the starting-point and end-point when the `constraint_message_tag_expression` holds TRUE for the specified node.

**NOTE:** *You can not use `-filter_in_cmt_from` argument with the `-instance_filter_in_cmt_from` argument and `-filter_in_cmt_to` argument with the `-instance_filter_in_cmt_to` argument.*

**`-instance_filter_in_cmt_from <constraint_message_tag_expression>, instance_filter_in_cmt_to <constraint_message_tag_expression>`**

(Optional) Filters the starting-point and end-point when the `constraint_message_tag_expression` holds TRUE for the associated instance of the specified node.

**NOTE:** *You can not use `-instance_filter_in_cmt_from` argument with the `-filter_in_cmt_from` argument and `-instance_filter_in_cmt_to` argument with the `-filter_in_cmt_to` argument.*

**`-filter_in_from <include_from_pinlist>, -filter_in_to <include_to_pinlist>`**

(Optional) Same as the `-from` and `-to` arguments but defines design nodes that are to be included.

**`-filter_in_type_from <include_from_DO_pinlist>, -filter_in_type_to <include_from_DO_pinlist>`**

(Optional) Same as `-from_type` and `-to_type` but defines design nodes that are to be included.

**`-ignorecase`**

(Optional) Ignores the case for the nodename specified using the `-from`, `-to`, `-except_from`, `-except_to`, `-filter_in_name_from`, and `-filter_in_name_to` arguments.

**NOTE:** *This is applicable on all fields which take design-node name as an input.*

## Rules

The `illegal_path` constraint is used by the following rules:

---

**SpyGlass Connectivity Verify solution**

Soc\_09

---

**SpyGlass DFT Solution**

Conn\_09

---

## Examples

### Example 1

```
illegal_path -from top.clk1 -to_type module1:FLIP_FLOP_CLOCK  
-use_shift -path_type sensitized
```

The above example implies that there should be no sensitized path from `top.clk1` to clock pin of any flip-flop inside instances of `module1` in shift mode.

### Example 2

```
illegal_path -from_type top.inst1:FLIP_FLOP_OUT -to_type  
top.inst2:FLIP_FLOP_DATA -tag tag1 -path_type sensitizable
```

The above example implies that there should be no sensitizable path from output pin of any flip-flop inside `test.inst1` to data pin of any flip-flop inside `top.inst2` in tag `tag1`.

### Example 3

```
illegal_path -from top.ip1
```

The above example implies that `top.ip1` should not be driving any leaf cell or top module port.

### Example 4

```
illegal_path -to top.op1
```

The above example implies that `top.op1` should not be driven by any leaf cell or top module port.

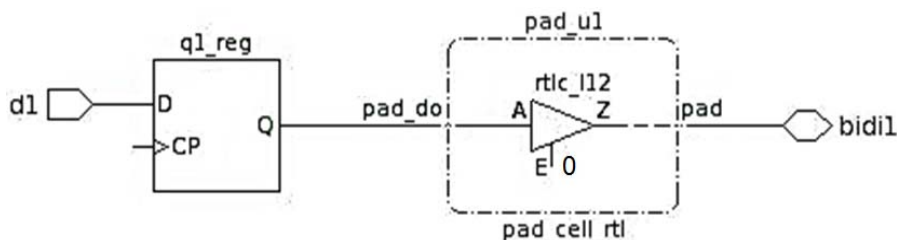
### Example 5

```
illegal_path -from top.ip1 -to_type FLIP_FLOP_DATA  
-except_to_type test.inst1:FLIP_FLOP_DATA -use_shift
```

The above example implies that there should be no sensitizable path from `top.ip1` to data pin of any flip-flop outside `test.inst1` in shift mode.

### Example 6

Consider the following schematic:

**FIGURE 37.**

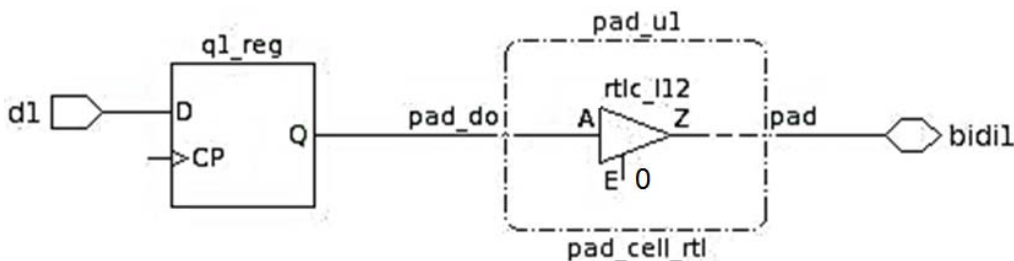
Now, consider the following constraint description:

```
illegal_path -from d1 -to bidi1 -path_type topological -
sequential_depth 1
```

For the above example, a topological path is traced through the flip-flop if the sequential depth greater than 0 is specified, even if the tristate buffer is disabled.

### Example 7

Consider the following schematic:

**FIGURE 38.**

Now, consider the following constraint description:

```
require_path -from clk1 -to clk_out1 -path_type topological
```

For the above example, a topological path is traced through the tristate buffer even though the path is blocked by value 0 on the enable pin of the tristate buffer.

**Example 8**

Consider the following constraint specification:

```
illegal_path -from_type FLIP_FLOP_OUTPUT -to sig1  
-filter_in_cmt_from "X1:PASS"
```

The above example implies that there should be no path to sig1 from flip-flop outputs, which have passed the check for constraint\_message\_tag, X1.

**Example 9**

Consider the following constraint specification:

```
illegal_path -from sig1 -to_type FLIP_FLOP_RESET  
-filter_in_cmt_to "X1:PASS"
```

The above example implies that there should be no path from sig1 to flip-flop reset pins, which have passed the check for constraint\_message\_tag X1.

**Example 10**

Consider the following constraint specification:

```
illegal_path -from sig1 -to sig2 -filter_in_cmt_from  
"X1:PASS" -filter_in_cmt_to "X2:PASS"
```

The above example implies that there should be no path from sig1 to sig2, where, sig1 has passed the check for constraint\_message\_tag, X1 and sig2 has passed the check for constraint\_message\_tag, X2.

**Example 11**

Consider the following constraint specification:

```
illegal_path -from_type FLIP_FLOP_OUTPUT -to sig1  
-instance_filter_in_cmt_from "X1:PASS"
```

The above example implies that there should be no path to sig1 from flip-flop outputs whose instances have passed the check for the constraint\_message\_tag, X1.

**Example 12**

Consider the following constraint specification:

```
illegal_path -from sig1 -to_type FLIP_FLOP_RESET
-filter_in_cmt_to "X1:PASS"
```

The above example implies that there should be no path from `sig1` from flip-flop reset pins whose instances have passed the check for the `constraint_message_tag, X1'`

### Example 13

Consider the following constraint specification:

```
illegal_path -from sig1 -to sig2 -filter_in_cmt_from
"X1:PASS" -filter_in_cmt_to "X2:PASS"
```

The above example implies that there should be no path from `sig1` to `sig2`, where, `sig1`'s instance have passed the check for `constraint_message_tag, X1` and `sig2`'s instance has passed the check for `constraint_message_tag, X2`.

### Example 14

Consider the following constraint specifications:

```
illegal_path -filter_in_from "in*" -from_type INPUT_PORTS -filter_in_to
"out*" -to_type OUTPUT_PORTS -ignorecase
```

```
illegal_path -filter_in_from "in*" -from_type INPUT_PORTS
-to "out*" -filter_in_type_to OUTPUT_PORTS -ignorecase
```

```
illegal_path -from "in*" -filter_in_type_from INPUT_PORTS
-filter_in_to "out*" -to_type OUTPUT_PORTS -ignorecase
```

```
illegal_path -from "in*" -filter_in_type_from INPUT_PORTS
-to "out*" -filter_in_type_to OUTPUT_PORTS -ignorecase
```

In the above example, each `illegal_path` constraint ensures that no path from input ports matching with `"in*" (case-insensitive)` to output ports matching with `"out*" (case-insensitive)`.

## illegal\_value



## Purpose

The `illegal_value` constraint checks for the presence of illegal value on a certain node when the circuit has been simulated using the condition specified by the `-tag` argument.

When `illegal_value` is used twice for the same node, `spyglass` interprets as neither 0 nor 1.

## Product

SpyGlass DFT solution

## Syntax

The syntax for the `illegal_value` constraint is as follows:

```
illegal_value
  [-name <nodename> ]
  [ -except <except_nodename> ]
  [ -type <DO_nodename> ]
  [ -except_type <exceptDO_nodename> ]
  [ -value <value> ]
  [ -value_type <type> ]
  [ -tag <condname> | -use_shift |
    -use_capture | -use_captureATspeed ]
  [ -matchNBits <num> ]
  [-constraint_message_tag <value>]
  [-report_failure_as_info]
  [-filter_in_cmt <constraint_message_tag_expression>]
  [-instance_filter_in_cmt
  <constraint_message_tag_expression>
  [ -filter_in_name <include_nodename> ]
  [ -filter_in_type <include_DO_nodename> ]
  [ -ignorecase ]
```

## Arguments

**-name <nodename>**

(Optional) The name can be a top-module port, or any internal net name,

or terminal name. More than one pin name can be specified, and it is effectively read as a concise description of as many individual value checks.

**NOTE:** *Specify either -name or -type argument.*

**-except <except\_nodename>**

(Optional) Same as <nodename> but defines design nodes that are not to be used as name.

**-type <DO\_nodename>**

Same as <nodename> but it takes only macros as inputs.

**NOTE:** *Specify either -name or -type argument.*

To view the list of macros supported by the `illegal_value` constraint, see [Supported Macros](#).

**-except\_type <exceptDO\_nodename>**

Same as <DO\_nodename> but it takes only macros as inputs.

**-value <value>**

The value is a logic value string of 0, 1, X, Z, 1\_or\_0, and 0\_or\_1. A single-bit value means check at end of complete simulation. The X value is treated as do-not-compare. A multi-bit value means check on cycle-by-cycle simulation basis. For specification of a vector value, SGDC multi-bit specification format (same as used for the `test_mode` constraint) should be used.

You can specify repeat sequences for the `illegal_value` constraint.

For fields that require repeat sequence, you can specify the values as <I\*S>. Here, S is any string that does not contain the <, >, and \* characters. However, S can contain another <I\*S> expression. I is an integer that is always interpreted as a decimal value. The expression <I\*S> means that the sequence S will be repeated I number of times.

**value\_type <type>**

Specify one the following values:

- **active:** Implies 1 for non-inverting clock-pin and 0 for inverting clock-pin.
- **inactive:** Implies 0 for non-inverting clock-pin and 1 for inverting clock-pin.

**NOTE:** *Specify either -value or -value\_type argument. Otherwise, the illegal\_value constraint is ignored for analysis.*

To view the list of macros supported by the -value\_type field of the require\_value and illegal\_value constraints, see [List of Macros Supported by the require\\_value and illegal\\_value Constraints](#).

- FLIP\_FLOP\_RESET
- SCAN\_FLIP\_FLOP\_RESET
- LATCH\_RESET
- FLIP\_FLOP\_SET
- SCAN\_FLIP\_FLOP\_SET
- LATCH\_SET
- FLIP\_FLOP\_ENABLE
- SCAN\_FLIP\_FLOP\_ENABLE
- LATCH\_ENABLE
- FLIP\_FLOP\_CLOCK
- SCAN\_FLIP\_FLOP\_CLOCK

**-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

**-tag <condname>**

(Optional) A condition previously defined by using the [define\\_tag](#) constraint. It describes a stimulation condition.

Note that only one condition name can be defined in a illegal\_value specification. However, simulation for a given condition name simulates all pin-value specifications simultaneously. The built-in power-ground connections are also simulated in this process.

**-use\_shift | -use\_capture | -use\_captureATspeed**

For any of these modifiers, `illegal_value` simulates test mode of that particular mode.

If `-use_shift`, `-use_capture`, or `-use_captureATspeed` argument is specified, the constraint simulates all, `shift`, `capture`, or `captureATspeed` test\_mode constraints, respectively.

**NOTE:** *If more than one of the `-tag`, `-use_shift`, `-use_capture`, or `-use_captureATspeed` arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with `illegal_value` constraint.*

**-matchNBits <num>**

(Optional) Specifies that only the `<num>` number of least significant bits are to be considered. If `<num>` is greater than `<value>` (specified with `-value` argument), the latter is padded with `X` to match the former's width.

**-report\_failure\_as\_info/-report\_failures\_as\_info**

Reports all the failures as info severity message.

**-filter\_in\_cmt <constraint\_message\_tag\_expression>**

(Optional) Filters the specified node when the `constraint_message_tag_expression` holds TRUE on the associated instance of the node.

**NOTE:** *You can not use this argument with the `-instance_filter_in_cmt` argument.*

**-instance\_filter\_in\_cmt <constraint\_message\_tag\_expression**

(Optional) Filters the specified node when the `constraint_message_tag_expression` is TRUE for the associated instance.

**NOTE:** *You can not use this argument with the `-filter_in_cmt` argument.*

**-filter\_in\_name <include\_nodename>**

(Optional) Same as the `-name` argument but defines design nodes that are to be included.

**-filter\_in\_type <include\_DO\_nodename>**

(Optional) Same as the -type argument but defines design nodes that are to be included.

**-ignorecase**

(Optional) Ignores the case for the nodename specified using the -name, -except\_to, and -filter\_in\_name arguments.

**NOTE:** *It is applicable to all the arguments, which take design-node name as input.*

**Examples**

Consider the following examples:

**Example1**

```
illegal_value -name abc -value "<5*10>"
```

The above example will be expanded as follows:

```
illegal_value -name abc -value 1010101010
```

**Example2**

```
illegal_value -name abc -value "11<5*10>010"
```

The above example will be expanded as follows:

```
illegal_value -name abc -value 111010101010010
```

**Example3**

```
illegal_value -name abc -value "<50*11<5*10>>010"
```

The above example will be expanded as follows:

```
illegal_value -name abc -value 111010101010...(repeated 50
times followed by 010)
```

You can also set a variable using the command setvar to obtain the above result as follows:

```
setvar x 11<5*10>
```

```
illegal_value -name abc -value "<50*${x}>010"
```

The above example will be expanded as follows:

```
illegal_value -name abc -value 111010101010...(repeated 50
times followed by 010)
```

**NOTE:** *Tagging for nesting is not allowed. For example, the following `illegal_value` statements are not allowed:*

```
illegal_value -name sub_seq -value <5*01>
illegal_value -name main_seq -value <100*sub_seq>
```

*However, you can achieve the same result by using the `setvar` command.*

### Example 5

```
illegal_value -tag s1 -name top.U1.U2.SEF
-value 1010 -matchNBits 2
```

### Example 6

Consider the following example:

```
-illegal_value -name p1 -value 0X110_or_110
```

The above constraint specification means:

- first bit must not be 0
- second bit is don't care
- third and fourth bits must not be 1
- fifth should be neither 0 nor 1
- sixth should not be 1
- seventh bit should not be 0

### Example 7

Consider the following sample input values:

```
illegal_value -name vec[3:0] -value { b 1 0 1 0 }
```

where `vec` is the 3:0 vector net

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "1010"
illegal_value -name vec[1] -value "0000"
illegal_value -name vec[2] -value "0000"
illegal_value -name vec[3] -value "0000"
```

**Example 8**

Consider the following sample input values:

```
illegal_value -name vec[3:0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "0"
illegal_value -name vec[1] -value "1"
illegal_value -name vec[2] -value "0"
illegal_value -name vec[3] -value "1"
```

**Example 9**

Consider the following sample input values:

```
illegal_value -name vec[3:0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "1"
illegal_value -name vec[1] -value "0"
illegal_value -name vec[2] -value "0"
illegal_value -name vec[3] -value "0"
```

**Example 10**

Consider the following sample input values:

```
illegal_value -name vec -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "1010"
illegal_value -name vec[1] -value "0000"
illegal_value -name vec[2] -value "0000"
illegal_value -name vec[3] -value "0000"
```

**Example 11**

Consider the following sample input values:

```
illegal_value -name vec -value { b 1010 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "0"  
illegal_value -name vec[1] -value "1"  
illegal_value -name vec[2] -value "0"  
illegal_value -name vec[3] -value "1"
```

### Example 12

Consider the following sample input values:

```
illegal_value -name vec -value { b 1 }  
where vec is the 3:0 vector net
```

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "1"  
illegal_value -name vec[1] -value "0"  
illegal_value -name vec[2] -value "0"  
illegal_value -name vec[3] -value "0"
```

### Example 13

Consider the following sample input values:

```
illegal_value -name vec[0] -value { b 1 0 1 0 }  
where vec is the 3:0 vector net
```

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "1010"
```

### Example 14

Consider the following sample input values:

```
illegal_value -name vec[0] -value {b 1010}  
where vec is the 3:0 vector net
```

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "0"
```

### Example 15

Consider the following sample input values:

```
illegal_value -name vec[0] -value { b 1 }  
where vec is the 3:0 vector net
```



The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "1"
```

### Example 16

Consider the following sample input values:

```
illegal_value -name sclr -value { b 1 0 1 0 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
illegal_value -name sclr -value "1010"
```

### Example 17

Consider the following sample input values:

```
illegal_value -name sclr -value { b 1010 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
illegal_value -name sclr -value "0"
```

### Example 18

Consider the following sample input values:

```
illegal_value -name sclr -value { b 1 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
illegal_value -name sclr -value "1"
```

### Example 19

Consider the following sample input values:

```
illegal_value -name vec -value { h 6 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
illegal_value -name vec[0] -value "0"
```

```
illegal_value -name vec[1] -value "1"
```

```
illegal_value -name vec[2] -value "1"
```

```
illegal_value -name vec[3] -value "0"
```

**Example 20**

Consider the following constraint specification:

```
illegal_value -type FLIP_FLOP_OUTPUT -value 0 -filter_in_cmt
"X1:PASS && X2:FAIL"
```

In the above example, the *illegal\_value* constraint ensures that there should be value 0 on any flip-flop output, which has passed the check for *constraint\_message\_tag*, X1 and failed the check for *constraint\_message\_tag*, X2.

**Example 21**

Consider the following constraint specification:

```
illegal_value -type FLIP_FLOP_OUTPUT -value 0
-instance_filter_in_cmt "X1:PASS && X2:FAIL"
```

In the above example, the *illegal\_value* constraint checks that none of the flip-flop output whose instance have passed the check for *constraint\_message\_tag* X1 and failed the check for *constraint\_message\_tag* X2, have a value 0.

**Example 22**

Consider the following constraint specifications:

```
illegal_value -filter_in_name "out*" -type OUTPUT_PORTS -
ignorecase -value 0
illegal_path -name "out*" -filter_in_type OUTPUT_PORTS -
ignorecase -value 0
```

In above cases, each *illegal\_value* constraint will check that none of the output ports matching with "out\*" (case-insensitive) have value 0

**Rules**

The *illegal\_value* constraint is used by the following rule:

---

**SpyGlass Connectivity Verify Solution**


---

Soc\_10

---

**SpyGlass DFT Solution**


---

Conn\_10

---

## initialize\_for\_bist

Currently, the `initialize_for_bist` constraint is merged with `test_mode` constraint and can be used as an argument to the `test_mode` constraint.

## initstate

### Purpose

The `initstate` command is used to specify the initial state sequence for the design.

You can specify the initial state sequence in a Tcl file or have the product read it from a VCD file.

**NOTE:** Use the `simulation_data` constraint instead of the `initstate` constraint because the `initstate` constraint may be deprecated in the future.

### Product

SpyGlass TXV solution

**NOTE:** The `initstate` command is not required for combinational analysis.

### Syntax

The syntax of the `initstate` constraint is as follow:

```
current_design <du-name>
  initstate
    -type <tcl | vcd>
    -file <file-name>
    [ -mode <mode-name> ]
    [ -time <value> ]
    [ -scopename <block-name> ]
    [ -modulename <module-name> ]
```

## Arguments

### **<du-name>**

Name of the design unit under which you are specifying the initial state sequence.

### **-type <tcl | vcd>**

The `-type` argument specifies the input file type as `tcl` for Tcl file or `vcd` for VCD file.

### **-file <file-name>**

The `-file` argument specifies the Tcl file or the VCD file.

### **-mode <mode-name>**

The `-mode` argument is optional and is used to specify the applicable mode (one of the mode values specified with the `-mode` argument of the `sdc_data` constraint.) If you do not specify the `-mode` argument, the SpyGlass TXV solution assumes that the specified initial state sequence is applicable for all modes.

### **-time <value>**

The `-time` argument is optional and is used to read the initial state by using a timestamp.

### **-scopename <block-name>**

The `-scopename` argument is used if the given VCD file is for a full chip but you want to perform verification for a sub-block module. The hierarchical path of the block module's instantiation (in VCD) must be specified to the `initstate` constraint.

For example, consider a VCD file with a top module `top` and a sub module `block2` that lies within another block `block1`. Therefore, while performing the verification of `block2`, the following must be defined to initialize the `block2` module from the VCD file generated for the full chip design:

```
current_design block2
```

```
initstate -type vcd -scopename
"top.block1_inst.block2_inst" -file chip.vcd
```

**NOTE:** For sequential analysis, it is recommended that you either provide sufficient reset and set\_case\_analysis constraints so that the SpyGlass TXV solution automatically reads the initial state sequence or specify the `initstate` constraint. If the number of sequential elements initialized is high, the quality of the result produced by the SpyGlass TXV solution is good.

**-modulename <module-name>**

The `-modulename` argument is used specify the top module.

## Examples

Consider the example below:

```
initstate -type vcd -modulename A -scopename B -file
design.vcd
```

In the above example, A is the name of the top module and B is the sub block module name. Here, the top module name A will replace the sub block module name B and initial state sequence is read for the sub block module.

An example of the Tcl File is as follows:

```
set sig1 0xffffffff
incr a
for {set i 1} {$i<=4} {incr i} {
    force xyz [incr a 4]
    simulate 2 -clk clk1
}
force sig2 [incr a]
simulate 2
```

## Rules

The `initstate` constraint is used by the following rules:

---

**SpyGlass TXV Solution**

---

All rules

---

# input

## Purpose

The `input` constraint is used to specify clock domain at input ports.

## Product

SpyGlass CDC solution

## Syntax

The syntax of using the `input` keyword in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>  
  clock -name <clk-name> -domain <domain-name>  
  input -name <input-name-list>  
    -clock <src-clk-name>
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs)

### -name <clk-name>

The clock signal name.

### -domain <domain-name>

The clock domain name.

### -name <input-name-list>

Port names in `<input-name-list>` can be scalar ports, bus ports, or wildcard names (matching against all top-level ports of appropriate type).

If you specify the same input port in multiple `input` constraint

specifications, SpyGlass considers the last input constraint specification. Consider the following example in which the same input port, `in1`, is specified in two different input constraint specifications:

```
input -name in1 -clock clk2
input -name in1 -clock clk4
```

In this example, SpyGlass considers the last input constraint specification, which is input `in1` clocked by clock `clk4`.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

### **-clock <src-clk-name>**

For *SpyGlass CDC* solution, this argument is the name of the effective source clock of a primary input port or a virtual clock specified by the `-tag <logical-clock-name>` argument of the `clock` constraint. For virtual clock specification, see [Example 2](#).

For *SpyGlass Auto Verify* solution, this argument is the name of the clock constraining the specified primary input ports.

**NOTE:** Do not specify clock domain names to this argument.

## **Examples**

### **Example 1**

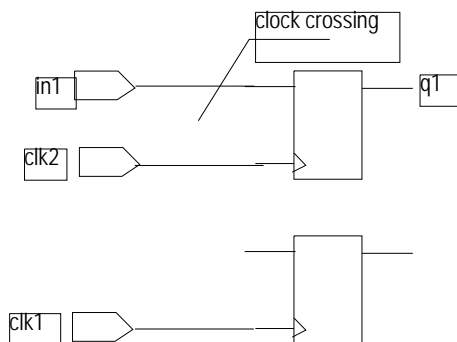
Consider the following input constraint specification:

```
input -name in1 -clock clk1
```

The above specification matches input port with the name `in1`

If you want to check synchronization for paths starting from input ports, you need to provide information about the source clocks associated with those ports. The input constraint is used for this purpose.

Consider the following example:



**FIGURE 39.** Clock Crossing

Assume that the input port `in1` is in the clock domain (say `D1`) of clock `clk1` and is connected to the data pin of flip-flop `top.q1` that is triggered by clock `clk2`. Therefore, there is clock domain crossing between primary input port `in1` (clock domain `D1`) and the flip-flop `top.q1` (clock domain `D2`). This situation can be specified as follows:

```
current_design top
  clock -name top.clk1 -domain D1
  clock -name top.clk2 -domain D2
  ...
  input -name in1 -clock top.clk1
  ...
```

**NOTE:** You must specify top-level simple (non-hierarchical) port names with the `-name` argument of the input and output constraints.

## Example 2

You can specify input constraints using virtual clocks if the clock domain of the input port is not known or is not present in the current design. For example, assume that the input port `in1` in [Example 1](#) is in the clock domain of the virtual clock `vclk`. This situation can be specified as follows:

```
current_design top
  clock -name top.clk2 -domain D2
  ...
```



```
input -name in1 -clock vclk
...
```

Note that there is no clock constraint for the virtual clock `vclk`.

In this case, there is clock domain crossing between primary input port `in1` (clock domain `vclk`) and the flip-flop `top.q1`.

If input constraint is defined for multiple ports with the same virtual clock, these ports are considered to be in the same domain.

You can use the match many (\*) and match one (?) wildcard characters with the `-name` argument of the `input` constraint by specifying the regular expression enclosed in double quotes ("").

However, do not specify a wildcard character if you want to apply the `input` constraint for the whole bus. For example, if you want to specify all bits of the vector input port, say `in[0:15]`, use the following constraint:

```
input -name "in" -clock clk
```

## Rules

The `input` constraint is used by the following rules:

---

### SpyGlass CDC Solution

---

All clock synchronization rules except the Clock_sync05 rule	Reset_sync01	Reset_sync02	Reset_sync03
Reset_sync04			

---

## input\_drive\_strength

### Purpose

Specifies the maximum capacitance (in picofarads) that any input port can drive.

The maximum capacitance value is used by these rules for estimating the clock buffers and high fan-out buffers on input lines.

In case `input_drive_strength` constraint is not specified, the primary ports of the design will be considered to have infinite drive strength. The value of `input_drive_strength` is used to estimate the buffers for ports nets that have a high load.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `pi_drive_strength`.*

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `input_drive_strength` constraint is as follows:

```
current_design <top-du-name>
  input_drive_strength
  -value <value>
```

## Arguments

### <top-du-name>

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

### -value <value>

The maximum capacitance value.

You can specify only one `input_drive_strength` constraint for one top-level design unit.

## Rules

The `input_drive_strength` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## input\_isocell

### Purpose

Specifies the isolation cells at inputs of a power domain.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was inisocell.*

**NOTE:** *Currently, a level shifter with an enable pin (clamp level shifter cell) is also treated as isolation cell.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `input_isocell` constraint is as follows:

```
current_design <du-name>
  input_isocell
    -names <cell-name-list>
    [ -pin <pin-name-list> ]
    [ -belongsto <pd-name> ]
    [ -input_pin <pin-name> ]
    [ -enable_pin <pin-name> ]
    [ -inhibit ]
```

### Arguments

**<du-name>**

Name of the design unit under which you are specifying the input-side isolation cells.

**-names <cell-name-list>**

Space-separated name list of input-side isolation cells. You can use wildcard characters while specifying the cell names using the `-names` argument.

**-pin <pin-name-list>**

Space-separated name list of pins that are allowed to be connected outside of a power domain.

**-belongsto <pd-name>**

Name of the power domain for which you are specifying the input-side isolation cells.

**-input\_pin|-enable\_pin <pin-name>**

The `-input_pin` and `-enable_pin` arguments specify the input pin name and the enable pin name for the cells specified with other arguments. Then, these pin names are used in isolation cell instances for named association.

**-inhibit**

Specifies that the specified cells should not be used as input-side isolation cells.

**Rules**

The `input_isocell` constraint is used by the following rules:

---

<b>SpyGlass Power Verify Solution</b>			
LPSVM48	LPSVM51	LPSVM52	LPSVM60

---

## instance\_trace

### Purpose

Specifies the instances or design units for which the PESAE04 and PESAE06 rules generate the activity trace or PEPWR01 and PEPWR02 with n-cycles generate the power trace.

By default, if the `instance_trace` constraint is not specified, the respective rules dump the activity/power trace for the top design in the activity/power browser. If you specify this constraint, the `pe_cycle_power` report lists the peak power consumed and the time interval for the specified instances.

**NOTE:** *For more information on the `pe_cycle_power` report, refer to the *SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide*.*

You should use the `instance_trace` constraint to specify additional hierarchies (using the `-instname` or `-meminst` argument) for which the activity/power trace needs to be created.

**NOTE:** *A design hierarchy specified with `-meminst` argument is only used to create a power trace.*

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `set_instance_for_activity_trace`.*

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax of the `instance_trace` constraint is as follows:

```
current_design <top-du-name>
  instance_trace
    -name <tag-name>
    [-instname <inst-name-list>]
    [-meminst <mem-inst-list>]
    [-clock <clk-name>]

    [-type <type-name>]
    [-component <component-name>]
```

## Arguments

### <top-du-name>

Module name (for Verilog designs) or design unit name in <entity-name> . <arch-name> format (for VHDL designs).

The Power Activity Product works only with designs having a single top-level design unit (checked by the PECHECK05 rule). You must specify the name of the top-level design unit.

**NOTE:** *The power/activity trace for the top-level design unit is always generated.*

### -name <tag-name>

Symbolic name for activity/power trace to be generated. The <tag-name> is a symbolic alphanumeric string used as the activity trace label in the SpyGlass Activity/Power Browser.

Each symbolic name specified by the -name argument can correspond to multiple instance names. If there are multiple instance names specified, the symbolic name for each instance name is derived by appending an ID with the symbolic name specified.

See *Viewing Results in Activity Browser* section in the *SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide* for more details.

### -instname <inst-name-list>

Name of the instances for which you want the activity/power trace.

This argument supports wildcards. Therefore, you can use wildcard characters while specifying the instance names. Currently, only the asterisk (\*) operator and question mark (?) are supported as wildcard characters. In addition, you must specify wildcard names enclosed in double quotes ("").

**NOTE:** *If you do not specify the -instname argument, respective rules generate the activity/power trace for the top-level design unit only.*

### -meminst <mem-inst-list>

Memory instances or design hierarchies, for which you want the power trace. For the design hierarchies, the trace is done for all the memories

instantiated under the specified hierarchy.

You can also use wildcards for specifying the memory instances.

In case of multiple instances, a power trace is done for all memory instances in a single graph.

**-clock <clk-name>**

List of root clocks of the specified instance for which the graph will be populated by PESAE06 rule.

**-type <type-name>**

Used to specify the type of power. It can have the following values: `leakage`, `internal`, `switching`, `all`, and `total`. The `total` value for `-type` attribute generates the total component power graph and it overwrites all other options. For more information, see the Viewing the Power Graph section in the *SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide*.

**-component <component-name>**

Used to specify the design component that you want to include in the graph. It can have following values: `combinational`, `sequential`, `clock`, `memory`, `blackbox`, `iopad`, and `all` values. For more information, see the Viewing the Power Graph section in the *SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide*.

## Rules

The `instance_trace` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction solutions			
PESAE04	PESAE06	PEPWR01	PEPWR02

## ip\_block

## For SpyGlass CDC solution and SpyGlass Auto Verify Solution

### Purpose

Your design may have IP blocks instantiated where you are only interested in clock domain crossing synchronization status at the interface to ensure that the IP block is hooked up correctly.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was IP\_block.*

### Product

SpyGlass CDC solution, SpyGlass Auto Verify solution

### Syntax

Use the `ip_block` constraint to specify the IP blocks in your design in the following format:

```
ip_block -name <du-name>
```

### Arguments

**-name <du-name>**

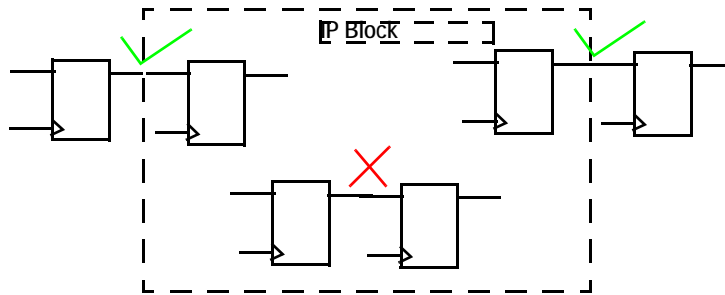
The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs). You can specify more than one IP Block design unit as a space-separated list.

Then, the clock synchronization checking rules will check for synchronization status of the clock crossing at the interface and ignore the clock crossings within the IP block.

**NOTE:** *Violations of any GuideWare rule of SpyGlass CDC solution are not reported for IP modules specified by the `ip_block` constraint.*

The following figure summarizes the above situation:





**FIGURE 40.** Synchronization Status Check

## Rules

The `ip_block` constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Ac_unsync01	Ac_unsync02	Clock_sync08	Clock_sync03a
Clock_sync03b	Ac_handshake01	Clock_sync08a	Clock_sync09
Ac_handshake02	Ac_cdc01a	Ac_cdc01b	Ac_cdc01c
Ac_cdc08	Propagate_Clocks	Ac_conv01	Ac_conv02
Ac_conv03	Ac_fifo01	Ac_sync02	Ac_sync01
<b>SpyGlass Auto Verify Solution</b>			
Av_bus01	Av_bus02	Av_deadcode01	Av_staticnet01
Av_case01	Av_case02	Av_bitstuck01	Av_fsm_analysis
Av_dontcare01	AV_setreset01	Av_staticreg01	Av_divide_by_zero

## For SpyGlass DFT solution and SpyGlass DFT DSM solution

### Purpose

Specifies the design units for which you want to generate the boundary information.

## Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

## Syntax

Use the `ip_block` constraint to specify the IP blocks in your design in the following format:

```
ip_block -name <du-name>
        [-create_constraints]
        [-create_report]
        [-clock]
        [-mode]
        [-control]
        [-observe]
```

## Arguments

### **-name <du-name>**

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs). You can specify more than one IP Block design units as a space-separated list.

### **-create\_constraints**

Generates boundary information for a constraint file in an SGDC format.

### **-create\_report**

Generates boundary information for a report file in a textual format.

### **-clock**

Lists clock-specific boundary information for Shift mode, Capture mode, and at-speed mode in the generated report.

### **-mode**

Lists test mode-specific boundary information for Shift mode, Capture mode, and at-speed mode in the generated report.

## SpyGlass Design Constraints

**-control**

Generates controllability-specific boundary information for the IP block in Capture mode.

**-observe**

Generates observable-specific boundary information for the IP block in Capture mode.

**Rules**

The `ip_block` constraint is used by the following rules:

**SpyGlass DFT Solution**

CreateDebugSGDC

**SpyGlass DFT DSM Solution**

Info\_IP\_Report

**isolation\_cell****Purpose**

The `isolation_cell` constraint is used to specify the isolation cells in power domains.

**NOTE:** *Currently, a level shifter with an enable pin (clamp level shifter cell) is also treated as isolation cell.*

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `isocell`.*

**Product**

SpyGlass Power Verify solution

**Syntax**

The syntax to specify the `isolation_cell` constraint for the LPSVM08, LPSVM09, LPSVM10, LPSVM22, LPSVM31, LPSVM35, LPSVM48, and LPSVM51 rules of the *SpyGlass Power Verify* solution is as follow:

```
current_design <du-name>
  isolation_cell
    -names <name-list>
    [ -belongsto <dom-name> ]
```

The syntax to specify the `isolation_cell` constraint for the LPSVM23 rule of the *SpyGlass Power Verify* solution is as follows:

```
current_design <du-name>
  isolation_cell
    [ -iso_enable_val <0 | 1> ]
    [ -and_cell <cell-name> ]
    [ -or_cell <cell-name> ]
    [ -latch_cell <cell-name> ]
    [ -not_cell <cell-name> ]
    [ -and_cell_for_isosig <cell-name> ]
    [ -or_cell_for_isosig <cell-name> ]
    [ -input_pin <pin-name> ]
    [ -output_pin <pin-name> ]
    [ -enable_pin <pin-name> ]
    [ -config 1 | 2 | 3 | 4 ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the level shifters with isolation capability.

### -names <name-list>

Space-separated name list of output-side isolation cells.

You can use wildcard characters while specifying the cell names using the `-names` argument. Currently, only the star (\*) operator for zero or more times and question mark (?) for zero or one time are supported. In addition, you must specify wildcard names enclosed in double quotes ("").

### -belongsto <dom-name>

Name of the power domain for which you are specifying the `<name-list>`.

If you do not specify the `-belongsto` argument, the specified cells are applicable for all power domains.

**`-iso_enable_val <0 | 1>`**

A value 0 or 1 of the `-iso_enable_val` argument specifies that the steady state of the power domain is active low or active high respectively.

**`-and_cell|-or_cell|-latch_cell <cell-name>`**

The cell name `<cell-name>` specified with the `-and_cell`, `-or_cell`, and `-latch_cell` arguments are the AND cell, OR cell, and Latch Cell to be used as isolation cells, respectively.

You should specify the `-and_cell`, `-or_cell`, and `-latch_cell` arguments if any of the power domain output signals has an expected steady state value of active low, active high, or hold, respectively.

**`-not_cell <cell-name>`**

The cell name `<cell-name>` specified with the `-not_cell` argument is the NOT cell to be used to change the polarity of the isolation cell.

**`-and_cell_for_isosig|-or_cell_for_isosig <cell-name>`**

If you specify multiple isolation signals with the `-isosig` argument of the `voltage_domain` constraint, specify the cell to be used for generating a single isolation signal by specifying AND or OR gates for the signals using the `-and_cell_for_isosig` argument or the `-or_cell_for_isosig` argument, respectively.

When you specify the `-and_cell_for_isosig` argument, the AND gate is applied to the signal. Similarly, when you specify the `-or_cell_for_isosig` argument, the OR gate is applied to the signals. You should specify only one of these arguments

**`-input_pin|-output_pin|-enable_pin <pin-name>`**

The `-input_pin`, `-output_pin`, and `-enable_pin` arguments specify the input pin name, the output pin name, and the enable pin name for the cells specified with other arguments. Then, these pin names are used in isolation cell instances for named association.

## **-config**

The `-config` argument specifies the order of values in the isolation cell instantiations as follows:

Value	Order
1	input-enable-output
2	output-input-enable
3	enable-input-output
4	output-enable-input

The same configuration without the enable part is also applicable for the NOT cell used for changing the polarity of the isolation signal. Thus, the configurations 1 and 3 indicate `input-output` and configurations 2 and 4 indicate `output-input`.

Configuration is useful when pin names are not provided and the instances do not have named association.

## **Rules**

The `isolation_cell` constraint is used by the following rules:

<b>SpyGlass Power Verify Solution</b>			
LPSVM08	LPSVM09	LPSVM10	LPSVM22
LPSVM23	LPSVM31	LPSVM35	LPSVM48
LPSVM51			

## **isolation\_wrapper**

### **Purpose**

The `isolation_wrapper` constraint specifies the isolation wrapper modules. The wrapper module name should be specified in the `-name` argument. Wildcard characters are also supported.

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the *isolation\_wrapper* constraint is as follows:

```
isolation_wrapper -name <isolation-wrapper-mod-name>
```

## Arguments

**-name <isolation-wrapper-mod-name>**

Use this argument to specify the wrapper module name.

## Rules

The *isolation\_wrapper* constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM08	LPSVM22	LPSVM47	LPSVM60

## keeper

### Purpose

The `keeper` constraint is used to specify bus-keeper design units or nets connected to a (virtual) bus-keeper design unit instance so that various rules can take appropriate action.

For example, the `Scan_21` rule uses the `keeper` constraint to check whether a scan flip-flop exists in the fan-in of the enable pin of a bus-keeper design unit. The `Tristate_06` rule and the `Tristate_09` rules ignore those floating tristate signals that are connected to an instance of the bus-keeper design unit or are connected to the specified net.

## Product

SpyGlass DFT solution

## Syntax

The syntax of the `keeper` constraint for the `Scan_21` rule is as follows:

```
keeper
  -name <du-name>
  [ -pin <pin-name> -value <value> ]
```

The syntax of the `keeper` constraint for the `Tristate_06` and `Tristate_09` rules is as follows:

```
keeper
  [ -name <du-name> -pin <pin-name> -value <value> ]
  | [ -name <net-name> ]
```

**NOTE:** *The `keeper` constraint supports wildcard characters.*

## Arguments

### **-name <du-name>**

The name of the bus-keeper design unit (black box).

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are treated as keeper design units.

You can specify a single design unit name or a space-separated list of design unit names.

### **-pin <pin-name>**

(Optional) The enable pin on the bus-keeper design unit (black box). You can specify only a single pin name.

If you do not specify the enable pin, the design unit (black box) is always assumed an enabled bus-keeper.

### **-value <value>**

(Optional) The expected value (0,1, X, or Z) on the enable pin *<pin-name>* under the shift mode.



You need to specify the value only if you have specified the pin name.

**-name <net-name>**

(Optional) The net that is connected to a tristate signal to be ignored.

You need to specify the hierarchical net name with respect to the top (with top name being optional).

## Rules

The `keeper` constraint is used by the following rules:

SpyGlass DFT Solution		
Scan_21	Tristate_06	Tristate_09

## latched\_port

### Purpose

The `latched_port` constraint is used to specify that a vector port is error-free.

### Product

SpyGlass Power Estimation Solution

### Syntax

The syntax to specify the `latched_port` constraint is as follows:

```
current_design <du-name>
  latched_port
    -port_name <name>
```

### Arguments

**<du-name>**

Name of the design unit under which you are specifying the latched port.

**-port\_name** <name>

Name of the vector port that you want to declare as error-free.

## Rules

The `latched_port` constraint is used by the PRARITH01 rule.

## levelshifter

### Purpose

The `levelshifter` constraint is used to specify the names of design units to be used as level shifters.

**NOTE:** *Currently, a level shifter with an enable pin (clamp level shifter cell) is also treated as isolation cell.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `levelshifter` constraint is as follows:

```
current_design <du-name>
  levelshifter
    -name <name-list>
    -from <vd-name>
    -to <vd-name>
    [ -inTerm <term-name> ]
    [ -enableTerm <term-name-list> ]
    [ -outTerm <term-name> ]
    [ -inSupplyTerm <term-name> ]
    [ -outSupplyTerm <term-name> ]
    [ -enableNet <en-net-name> ]
    [ -locate <vd-name> | src | dest ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the level shifters.

### -name <name-list>

Space separated list of names of the level shifter design units. You can use wildcard characters while specifying level shifter cell names using the -name argument. Currently, only the '\*' operator for zero or more times and '?' for zero or one time are supported. You must enclose wildcard-based names in double quotes ("").

### -to <vd-name>

Name of a voltage domain specified using the *voltage\_domain* (SpyGlass Power Verify solution) constraint.

### -inTerm <term-name> / -outTerm <term-name>

Name of the input/output terminal

### -enableNet <en-net-name>

Name of the net to which the enable terminal is to be connected.

### -enableTerm <term-name-list>

Space-separated name list of enable terminals.

When you specify the -enableTerm argument, SpyGlass assumes the level shifter to have isolation capability (and requires you to also specify such level shifters with the *isolation\_cell* constraint). Then, the LPSVM08, LPSVM09, and LPSVM10 rules of the SpyGlass Power Verify solution do not skip such level shifter instances (with isolation capability) at the output of power domains.

### -outTerm <term-name>

Use the -inSupplyTerm and -outSupplyTerm arguments to specify the input-side supply terminal and output-side supply terminal respectively as used by the LPPLIB05 rule of the SpyGlass Power Verify solution.

**-locate <vd-name> | src | dest**

The `-locate` argument specifies the location (voltage domain) of the level shifter instance. For LPSVM30 rule of the SpyGlass Power Verify solution, you can specify the actual voltage domain or specify `src` or `dest` (case-insensitive) to indicate that the level shifter instance should be placed in the source voltage domain or destination voltage domain, respectively. If the `-locate` argument is not specified, the LPSVM30 rule of the SpyGlass Power Verify solution places all level shifter instances at the top of the design hierarchy.

## Rules

The `levelshifter` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM04A	LPSVM04B	LPSVM04C	LPSVM04D
LPSVM08	LPSVM09	LPSVM10	LPSVM17
LPSVM30	LPPLIB04	LPPLIB05	LPPLIB06
LPPLIB07	LPPLIB15		

## lp\_ignore\_cells\_for\_erc

### Purpose

The `lp_ignore_cells_for_erc` constraint is used to specify the cells that should be ignored while checking of LPERC rules. You can use wildcard characters while specifying cell names using the `-names` argument. Currently, only the `*` operator is supported. You must enclose wildcard-based names in double quotes (`""`).

### Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `lp_ignore_cells_for_erc` constraint is as follows:

```
lp_ignore_cells_for_erc -names <cell-name-list>
```

## Arguments

**-names <cell-name-list>**

A space separated list of cell names is specified.

## Rules

The `lp_ignore_cells_for_erc` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPERC01A	LPERC01B	LPERC01C	LPERC02A
LPERC02B	LPERC03A	LPERC04A	LPERC04B

## make\_mandatory\_upf\_commands\_options

### Purpose

This command is used to make optional arguments of a UPF command as mandatory.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `make_mandatory_upf_commands_options` constraint is as follows:

```
make_mandatory_upf_commands_options -name <command-name> -  
options <option-list>
```

## Arguments

**-name** <command-name>

(Mandatory) Specifies the name of the UPF 1.0/2.0 command. The command name is checked against UPF 1.0/2.0 for sanity checking of the constraint in the *SGDC\_lowpower119* rule.

**-options** <option-list>

(Mandatory) Specifies the arguments of the UPF command. These arguments once specified become mandatory. The specified arguments are checked against UPF 1.0/2.0 for sanity checking of the constraint in the *SGDC\_lowpower119* rule.

## Rules

The `make_mandatory_upf_commands_options` constraint is used by the following rule:

---

### SpyGlass Power Verify Solution

---

SGDC_lowpo	UPFSTX_18
wer119	

---

## mapped\_pin\_map

### Purpose

When SpyGlass analyzes the RTL description, there is no knowledge about post-synthesis pin names. The design is synthesized internally, using the SpyGlass synthesis libraries that may use pin names different from the actual ones. Therefore, you need to define a pin name mapping for the different types of pins.

The `mapped_pin_map` constraint is used to define the mapping between the pin names specified in the user's library and the SpyGlass synthesis library.

## Product

SpyGlass Constraints solution

## Syntax

The syntax to specify the `mapped_pin_map` constraint is as follows:

```
current_design <du-name>
  mapped_pin_map
    [-clock <clock_pin_names_list>]
    [-enable <enable_pin_names_list>]
    [-data <data_pin_names_list>]
    [-out <output_pin_names_list>]
    [-preset <preset_pin_names_list>]
    [-clear <clear_pin_names_list>]
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in <entity-name>.<arch-name> format (for VHDL designs).

### -clock <clock\_pin\_names\_list>

(Optional) Specify the list of clock-pin names in the user's library.

### -enable <enable\_pin\_names\_list>

(Optional) Specify the list of enable-pin names in the user's library.

### -data <data\_pin\_names\_list>

(Optional) Specify the list of data-pin names in the user's library.

### -out <output\_pin\_names\_list>

(Optional) Specify the list of output-pin names in the user's library.

### -preset <preset\_pin\_names\_list>

(Optional) Specify the list of preset-pin names in the user's library.

**-clear <clear\_pin\_names\_list>**

(Optional) Specify the list of clear-pin names in the user's library.

**NOTE:** *The mapped\_pin\_map constraint is required only when analyzing RTL descriptions. For netlists, the information is obtained from the corresponding gate library (.sglib files or .lib files).*

For example, consider the following:

```
current_design top
  mapped_pin_map
    -clock CK CP -enable EN EX -data D -out Q QN
```

The above specification indicates that pins named CK or CP are clock pins in the user-specified technology library cells. Similarly, pins named EN or EX are enable pins, pins named D are data pins, and pins named Q or QN are output pins.

## Rules

The mapped\_pin\_map constraint is not rule-specific. It is used by the whole SpyGlass Constraints solution.

## mcp\_info

### Purpose

The mcp\_info SGDC constraint is used to specify enables and other attributes to verify the SDC multicycle path constraints.

**NOTE:** *The mcp\_info SGDC command gets priority over any parameters.*

### Product

SpyGlass TXV Solution

### Syntax

The syntax of the mcp\_info constraint is as follows:

```
current_design <du-name>
```



```

mcp_info
  -name <name>
  [ -multiplier <MUL> ]
  [ -launch_clock <LC> ]
  [ -launch_enable <LE> ]
  [ -launch_flop <LE> ]
  [ -capture_clock <CC> ]
  [ -capture_enable <CE> ]
  [ -capture_flop <CE> ]
  [ -cpath_enable ]
  [ -start/-end ]

  -sdc_file_name <file-name>
  -sdc_line <line-number>
  [ -force_ldce <0 | 1> ]
  [ -make_pi <list-of-signals> ]
  [ -IConstrs <expression> ]

```

## Arguments

### **-name <name>**

(Mandatory) Used to match the name of the `set_multicycle_path` constraint defined in the SDC file. You can specify the name of a `set_multicycle_path` constraint in the SDC file by using the comment argument:

```
set_multicycle_path 2 -from C1 -to C2 -comment {Atrenta -name
<name1>}
```

### **-multiplier <MUL>**

(Optional) Used to specify the path multiplier of the `set_multicycle_path` constraint.

### **-launch\_clock <LC>**

(Optional) Used to specify the SDC name of the clock that is traversing to the source point of the timing path constrained by the `set_multicycle_path` constraint.

**-launch\_enable <LE>**

(Optional) Used to specify the start-point enable expression in terms of user specified signals in the RTL.

**NOTE:** *Do not use this argument to specify clock path enables.*

**-launch\_flop <flop-list>**

(Optional) Use this argument to specify an enable for the launch flip-flop. This argument must be used with the `-launch_enable` argument.

**Apart from the `-launch_enable` argument, no `mcp_info` argument is supported with the `-launch_flop` argument.**

This argument supports wildcards.

See [Example 3: Specifying Enables for Launch/Capture Flip-Flops](#).

**-capture\_clock <CC>**

(Optional) Used to specify the SDC name of the clock that is traversing to the end-point of the timing path constrained by the `set_multicycle_path` constraint.

**-capture\_enable <CE>**

(Optional) Use to specify the end-point enable expression in terms of user specified signals in the RTL.

**NOTE:** *Do not use this argument to specify clock path enables.*

**-capture\_flop <flop-list>**

(Optional) Use this argument to specify an enable for the capture flip-flop. This argument must be used with the `-capture_enable` argument.

**Apart from the `-capture_enable` argument, no `mcp_info` argument is supported with the `-capture_flop` argument.**

This argument supports wildcards.

See [Example 3: Specifying Enables for Launch/Capture Flip-Flops](#).

**-cpath\_enable**

(Optional) Specify this argument to detect enables only in the clock path for both source and destination.

The following `mcp_info` arguments are not supported with this argument:

- `-capture_enable`
- `-launch_enable`
- `-force_ldce`
- `-launch_flop`
- `-capture_flop`

See [Example 4: Specifying `cpath\_enable`](#).

#### **`-start/-end`**

(Optional) If the multi-cycle information is relative to the period of the start clock, specify `-start`. Otherwise, specify `-end`. If neither is specified, `-end` is taken as default. A FATAL violation message is reported if you specify both `-start` and `-end` in the same `mcp_info` constraint.

#### **`-sdc_file_name <file-name>`**

(Mandatory, if either `force_ldce`, `make_pi`, or `lConstrs` is specified)  
Used to specify the name of the SDC file which contains the multicycle path specification. For this specification, the `force_ldce`, `make_pi`, or `lConstrs` arguments are applied.

The SDC file must exist in the project working directory. Do not specify the file name as an absolute or relative path.

#### **`-sdc_line <line-number>`**

(Mandatory, if either `force_ldce`, `make_pi`, or `lConstrs` is specified)  
Used to state the line number of the SDC file specified using the `sdc_file_name` argument. This is the line number of the multicycle path specification on which the `force_ldce`, `make_pi`, or `lConstrs` arguments are applied.

#### **`-force_ldce <0 | 1>`**

(Optional) This argument forces multicycle path verification on the basis of

data transition on the source side, instead of verification based on the source enable.

Set this argument to 1 to verify on the basis of data transition on the source side.

### Options Not Supported With `force_ldce`

The following arguments cannot be specified with the `force_ldce` argument: `-launch_enable`, `-launch_clock`, or `-capture_clock`. If any of the existing options is combined with `-force_ldce`, a violation is reported. See [Example 2: The `force\_ldce` Argument Violations](#).

#### **`-make_pi` <list-of-signals>**

(Optional) Use to specify the signals whose fan-in cone is not to be considered during multicycle path verification. The SpyGlass TXV solution cuts off the design present in the fan-in of the specified signals and treats these signals as primary inputs.

Specify the list of signals by separating the signal names by the "@" symbol. For example, `sig1@sig2@sig3`.

#### **`-lConstrs` <expression>**

(Optional) Initializes signals.

This argument takes an expression, which comprises the value of the signals, so that the final expression is always true. It can take simple expressions, such as `&`, `I`, and `~`.

## Example

This section contains the following examples:

- [Example 1: Flows of Multicycle Path Verification](#)
- [Example 2: The `force\_ldce` Argument Violations](#)
- [Example 3: Specifying Enables for Launch/Capture Flip-Flops](#)

### Example 1: Flows of Multicycle Path Verification

The flow of multicycle path verification is in two categories:

- Name specified in the `mcp_info` SGDC command matches with that in the SDC file
- Name specified in the `mcp_info` SGDC command does not match with that in the SDC file

### ***Name Specified in the SGDC File Matches***

In this example, the name `MCP1` matches with the name specified in the SDC file. Therefore, the values specified in the SGDC file are used to verify the `set_multicycle_path` constraint. You can use this approach to override some of the attributes or to provide missing information that SpyGlass is not able to infer.

#### **SDC**

```
set_multicycle_path 3 -from {ff1_reg/CP} -to {ff2_reg/D} -
comment {Atrenta -name <MCP1>; My comment}
```

#### **SGDC**

```
mcp_info -name MCP1 -launch_clock CLK -launch_enable
test.src_en -capture_clock CLK2 -capture_enable
test.dst_en
```

### ***Considerations***

The following considerations are applied during the verification:

- If either of the `-multiplier`, `-start/-end`, `-launch_clock`, `-launch_enable`, `-capture_clock`, or `-capture_enable` arguments is specified, the value specified in the SGDC constraint overrides that attribute for all the timing paths constrained through the `set_multicycle_path` constraint.
- For the SDC clocks specified in the `-launch_clock` and `-capture_clock` arguments, SpyGlass checks whether the clocks specified are traversing at either the source or destination. If not, those clocks are ignored.
- The enable expression specified in the `-launch_enable` or `-capture_enable` argument is not validated by SpyGlass. The verification is performed based on the assumption that these signals are the enabling condition for the start/end-points.

### ***Name Specified in the SGDC File Does Not Match***

In this example, the name MCP11 does not match with the name specified in the SDC file. You can use this approach when you do not have a `set_multicycle_path` constraint and want to validate the timing path formally.

### SDC

```
set_multicycle_path 3 -from {ff1_reg/CP} -to {ff2_reg/D} -  
comment {Atrenta -name <MCP1>; My comment}
```

### SGDC

```
mcp_info -name MCP11 -launch_clock CLK -launch_enable  
test.src_en -capture_clock CLK2 -capture_enable  
test.dst_en
```

### Considerations

The following considerations are applied during the verification:

- The `-name` argument would be a name that does not match with any of the existing `set_multicycle_path` constraints.
- The `-multiplier` argument is mandatory for this flow.
- If `-start` is specified, the `-launch_capture` argument is mandatory to determine the multi-cycle information relative to the period of the start clock.
- If `-end` is specified, the `-capture_clock` argument is mandatory to determine the multi-cycle information relative to the period of the end clock.
- Either the `-launch_enable` or `-capture_enable` arguments must be specified. The missing enable is taken as one.
- If either of `-launch_capture` or `-capture_clock` is not specified, the verification is done based on the multi-cycle information relative to the period of the start clock.

### Example 2: The `force_ldce` Argument Violations

This example shows the violation message that is reported when the `-force_ldce` argument is specified with unsupported arguments. Consider the following `mcp_info` SGDC specification.

```
mcp_info  
-name "MCP1"
```

```

-multiplier 2
-launch_clock "clk"
-capture_clock "pclk"
-launch_enable "(pclk)&pclk"
-capture_enable "pclk"
-start
-sdc_file_name "test.sdc"
-sdc_line 8
-force_ldce 1

```

The SpyGlass TXV solution reports the following violation because the `-force_ldce` argument cannot be specified with `-launch_enable`, `-launch_clock`, or `-capture_clock`:

For `mcp_info` command '`-force_ldce`' and '`-launch_enable`' are not supported together. '`-force_ldce`' and '`-launch_clock`' are not supported together. '`-force_ldce`' and '`-capture_clock`' are not supported together.

### Example 3: Specifying Enables for Launch/Capture Flip-Flops

This section contains the following examples:

- [Specifying launch\\_flop](#)
- [Specifying capture\\_flop](#)
- [Impact of Multiple mcp\\_info Commands](#)
- [Multiple launch\\_flop/capture\\_flop Options Specified for Same mcp\\_info Command](#)

#### ***Specifying launch\_flop***

For the launch flip-flop, specify the hierarchical name of the output terminal (Q terminal) of the launch flip-flop, as shown in the following `mcp_info` specification.

```

mcp_info -launch_flop "F1.Q, F2.Q, F3.Q" -launch_enable "e1"
-name MCP1

```

#### ***Specifying capture\_flop***

For the capture flip-flop, specify the hierarchical name of the input terminal

(falling on the data path) of the launch flip-flop, as shown in the following `mcp_info` specification.

```
mcp_info -capture_flop "F5.D, F6.D, F7.D" -capture_enable  
"e2" -name MCP1
```

However, if the `mcp_info` specification is:

```
mcp_info -capture_flop "F5.D, F6.D, F7.D" -name MCP1
```

The `Txv_mcp_info` rule reported the following violation:

```
TXV_SGDC_MCP: For mcp_info command '-capture_flop' cannot be  
used without '-capture_enable'.
```

### ***Impact of Multiple `mcp_info` Commands***

You can specify multiple `mcp_info` commands with the same `-name` argument.

In this example, the following snippet shows one `mcp_info` specification with `launch_enable/capture_enable`, while the other specification shows without `launch_flop/capture_flop`.

```
mcp_info -name "MCP1" -launch_flop "F1.Q" -launch_enable "e1"  
mcp_info -name "MCP1" -launch_enable "e2"
```

In this case, **e1** is the enable for flip-flop **F1**, while enable **e2** is for all other launch flip-flops.

### ***Multiple `launch_flop/capture_flop` Options Specified for Same `mcp_info` Command***

For the following `set_multicycle_path` SDC commands, multiple `launch_flop/capture_flop` arguments have been specified for the same flip-flop in the same `mcp_info` command:

```
set_multicycle_path 2 -from ff1_reg -to ff2_reg -comment {Atrenta -name <MCP1>}  
set_multicycle_path 2 -from ff1_reg -to ff3_reg -comment {Atrenta -name <MCP2>}
```



The `mcp_info` SGDC specification is as follows.

```

1 current_design test
2 sdc_data -file test.sdc
3
4 mcp_info -name "MCP1" -launch_flop "test.ff1_reg.Q" -launch_enable "en2"
5 mcp_info -name "MCP2" -launch_flop "test.ff1_reg.Q" -launch_enable "en3"
6 mcp_info -name "MCP2" -launch_enable "en2" -capture_flop "test.ff3_reg.D" -capture_enable "en2"
7 # Duplicate... should be ignored
8 mcp_info -name "MCP2" -multiplier 2 -launch_enable "en3" -start
9 # Valid...
10 mcp_info -name "MCP2" -multiplier 2 -capture_enable "en3" -start
11 # Duplicate... should be ignored
12 mcp_info -name "MCP2" -multiplier 2 -capture_enable "en3" -start
13 # Duplicate entry for -capture flop ... should be ignored
14 mcp_info -name "MCP2" -capture_flop "test.ff3_reg.D" -capture_enable "en2" -start

```

The `Txv_mcp_info` rule generates the following violation messages:

TXV\_SGDC\_MCP: Duplicate entry for test.sgdc: 8 "mcp -name MCP2 -multiplier 2 -launch\_enable en3 -start " ignored for sdc constraint

TXV\_SGDC\_MCP: Duplicate entry for test.sgdc: 12 "mcp -name MCP2 -multiplier 2 -capture\_enable en3 -start " ignored for sdc constraint

TXV\_SGDC\_MCP: Duplicate entry for -capture flop test.ff3\_reg.D test.sgdc: 14 "mcp -name MCP2 -multiplier 2 -capture\_enable en2 -capture\_flop test.ff3\_reg.D -start " ignored for sdc constraint.

#### Example 4: Specifying `cpath_enable`

The following `mcp_info` specification shows how to specify the `-cpath_enable` argument:

```
mcp_info -name MCP1 -cpath_enable
```

## Rules

The `mcp_info` constraint is used by the following rules:

---

#### SpyGlass Txv Solution

---

Txv\_MCP01

---

## memory

### Purpose

The memory constraint is used to specify information about memory cells in the design. This information is used by PEPWR18 rule for memory splitting and PESTR11 rule for identifying the enable signals of the memory cells.

You can specify the memory details like its address ports, data ports, enable signals and their polarity using this constraint. Half-sized equivalent and quarter-sized equivalent memories can also be specified for the memory cells.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax of the memory constraint is as follows:

```
memory
  -name <list-of-memory-cells>
  -address <list-of-address-ports>
  -data <list-of-data-ports>
  -halfsize <module-name1>
  -quartersize <module-name2>
  -enable <list-of-enables>
  -enable_polarity <en-polarity>
```

### Arguments

#### **-name <list-of-memory-cells>**

Name of a memory cell or a list of memory cells. You can also use wildcards for specifying the memory cell names.

While specifying multiple memory cells or wildcard specification with the `-name` argument, you cannot use any of `-address`, `-data`, `-halfsize` or `-quartersize` argument.

**-address <list-of-address-ports>**

List of address ports of a memory cell.

**NOTE:** For the `-address/-data` argument of the memory constraint, generally, the address/data bus that is defined in library as `A[0:32]` is specified as 'A'. For some libraries, where the address/data bus is in form of bits like `A1, A2, A3`, and so on, you can collectively specify 'A%d' for all the pins. The tool infers all such pins from the library as the pins of the specified address/data bus.

**-data <list-of-data-ports>**

List of data ports of a memory cell.

**-halfsize <module-name1>**

Half-sized equivalent for a memory cell.

**-quartersize <module-name2>**

Quarter-sized equivalent for a memory cell.

**-enable <list-of-enables>**

List of enable signals of the specified list of memory cells.

**-enable\_polarity <en-polarity>**

Polarity of the specified enable signals.

**Rules**

The memory constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction solutions			
PEPWR01	PEPWR02	PEPWR18	PESTR11
poweraudit			

**memory\_force**

Currently, the `memory_force` constraint is merged with [test\\_mode](#)

constraint and can be used as an argument to the `test_mode` constraint.

## memory\_port

### Purpose

The `memory_port` constraint is used to specify information about memory cells or submodules in the design. This constraint can be specified at the technology cell level and at the wrapper level. When you have an RTL module with MBIST and other peripheral logic contained inside a wrapper, it is recommended that you specify the complete wrapper as memory using the `memory_port` constraint. In such cases, you will specify the ports of the wrapper memory in the arguments, such as `-when` and `-clock`. Refer to the [Examples](#) section for more details.

Refer to the [Rules](#) section for a list of rules that use the information specified in the `memory_port` constraint to determine memory specifications.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax of the `memory_port` constraint is as follows:

```
memory_port
  -name <list-of-memory-cells>
  -data <data-port>
  -address <address-port>
  -operation <read|write|unwanted>
  -posedge|-negedge <clk_pin>
  -when <pin-state>
  -label <label-name>
  -async
  -retain_output <0|1|yes|no>
  -rom
  -cycle_count <0|1>
  -writethru
```

```

-data_out <data-out-port>
-write_mask <write-mask-port>
-write_mask_value <0|1>
-sleep_mode_when <pin-sleep-state>
-down_cycles <num-cycles-mem-down>
-wakeup_cycles <num-cycles-mem-wakeup>

```

## Arguments

### **-name <list-of-memory-cells>**

Name of a memory cell or a list of memory cells or sub-modules. A sub-module refers to a design unit that can act as a memory.

You can also use wildcards for specifying the name of memory cells and modules.

### **-data <data-port>**

Specifies the data port of a memory module.

### **-address <address-port>**

Specifies the address port of a memory module.

**NOTE:** For the `-address/-data` argument, the address/data bus that is defined in library as `A[0:32]` is specified as `A'`. For some libraries, where the address/data bus is in form of bits such as `A1, A2, A3`, you can collectively specify `A*` for all the pins. The tool infers all such pins from the library as the pins of the specified address/data bus.

### **-operation <read|write|unwanted>**

Specifies the type of operation to be performed on a memory module. The `unwanted` operation represents a condition, where the memory control pins are in an unexpected state. For example, if a clock edge is applied while the chip enable for the memory is turned off.

### **-when <pin-state>**

Lists conditions, comprising of states of pins required to trigger the specified operation on a memory module. In this option, you can specify

pins, which should be asserted high and pins, which must be asserted low (with a ! prefix). For example, if CE must be high and WENB must be low for the write operation, you need to specify: `-when "CE !WENB"`

To specify the condition, you can use any of the following operators:

`() & | ^ ! .`

#### **-posedge | -negedge <clk\_pin>**

Specifies whether the specified operation occurs on rising or falling transition of the associated clock pin.

#### **-label <label-name>**

Specifies a unique label to identify a `memory_port` constraint.

#### **-async**

Specifies that the memory is an asynchronous read memory.

When this argument is specified in the *PESAEO7* rule, the read frequency of the memory is calculated based on the change in the address bus.

**NOTE:** Do not specify the `-async` argument with the `-posedge` and `-negedge` arguments.

#### **-retain\_output <0|1|yes|no>**

Specifies whether the output retains its value or not when the memory is switched off.

Default value is 1.

Value	Condition
0   no	Data at the read output pin does not retain its value
1   yes	Data at the read output pin retains its value

#### **-rom**

Specifies that the memory is a read-only memory.

When this argument is specified, all PE rules, except PESTR28 and PEPWR28, treat the memory cell as a read-only memory. Therefore, all

write operations are ignored if the `-rom` argument is specified in a read operation.

**NOTE:** *Do not specify the `-rom` argument with the `-write` argument.*

**`-cycle_count <0|1>`**

Specifies the number of clock cycles for the data to be available at the memory output from the time the read condition is enabled.

Default value is 0.

**`-writethru`**

Specifies whether the memory is write-through or not.

Specifying this argument ensures that the data written to the data port (specified by the `-data` argument) is read and available at the data output port (specified by the `-data_out` argument).

**`-data_out <data-out-port>`**

Specifies the data output port corresponding to the data port, when the `-writethru` argument is specified.

**NOTE:** *The `-writethru` and `-data_out` arguments must be specified together.*

**`-write_mask <write-mask-port>`**

Specifies the write mask port for the data port.

The write data port is masked if the value on the port specified by the `-write_mask` argument is equal to the value specified by the `-write_mask_value` argument.

**`-write_mask_value <0|1>`**

Specifies the value of the write mask port (specified by the `-write_mask` argument) for which the write data port is masked.

For example, consider that the write mask port is `Q[0:63]` and the write mask value is 1. In this case, the write data port will be masked only if the value on the `Q[0:63]` port is equal to 1.

**NOTE:** *The `-write_mask_value` and `-write_mask` arguments must be specified*

together.

**NOTE:** The `-writethru`, `-data_out`, `-write_mask_value`, and `-write_mask` arguments must be specified for the write operation only. For the read operation, these arguments are ignored.

#### **-sleep\_mode\_when <pin-sleep-state>**

Specifies the states of pins required to trigger light sleep mode on a memory module.

In this option, you can specify the memory pin that should be asserted high. For example, if the LS pin must be high to trigger light sleep mode, specify the following expression:

```
-sleep_mode_when LS
```

#### **-down\_cycles <num-cycles-mem-down>**

Specifies the number of clock cycles required to assert the light sleep pin of the memory for it to enter light sleep mode.

#### **-wakeup\_cycles <num-cycles-mem-wakeup>**

Specifies the number of clock cycles required to de-assert the light sleep pin of the memory for it to recover from light sleep mode.

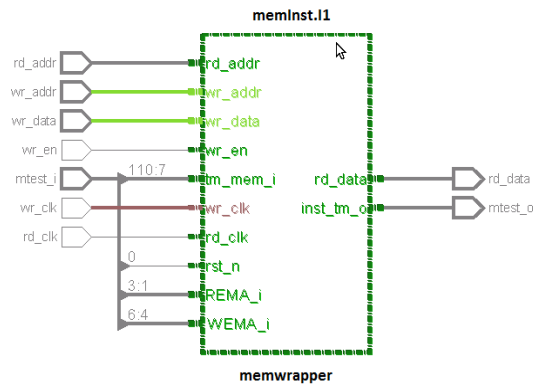
## **Examples**

This example illustrates how to use the `memory_port` constraint at wrapper (submodule) level.

The image below, `memInst.I1` is the hierarchical boundary in the design for the module (wrapper) `memwrapper`. The `mtest_i` port is the test mode data and control logic.



## SpyGlass Design Constraints



The memory\_port constraint specification for the memwrapper wrapper using the module ports as follows.

### **#-- Wrapper memwrapper**

#### **# Read operation**

```
memory_port -name " memwrapper " \
            -operation read \
            -label syncread \
            -posedge rd_clk \
            -when "!wr_en" \
            -address "rd_addr" \
            -data "rd_data" \
            -retain_output yes \
            -cycle_count 0
```

#### **# Write operation**

```
memory_port -name " memwrapper " \
            -operation write \
            -label syncwrite \
```

```
-posedge wr_clk \
-when "wr_en" \
-address "wr_addr" \
-data "wr_data"
```

## Rules

The `memory_port` constraint is used by the following rules of the SpyGlass Power Estimation and SpyGlass Power Reduction solutions:

PESAE07	PEPWR20	PEPWR21	PEPWR22	PEPWR23
PEPWR24	PEPWR25	PEPWR28	PESTR20	PESTR21
PESTR22	PESTR23	PESTR24	PESTR25	PESTR28
PEPWR02	PEPWR29	PESTR29		

## memory\_inst\_port

### Purpose

The `memory_inst_port` constraint is used to overwrite the value of the `retain_output` parameter of the `memory_port` constraint.

The `retain_output` argument of the `memory_port` constraint specifies if memory will retain its output value or not when memory is switched off. In some cases, it may happen that though the value of the `retain_output` argument specified in the `memory_port` constraint is '1' but for a particular instance of that memory, the user does not care about the output when memory is turned off. To capture this information, user can specify this constraint.

Consider the following `memory_port` constraint specifications:

```
memory_port -name mem_inst -posedge "CLKA" -data "QA" \
-when "!WEA MEA" -operation read -address "ADRA" -
retain_output 1
```

```
memory_port -name mem_inst -posedge "CLKA" -data "DA" \
-when "WEA MEA" -operation write -address "ADRA"
```

```
memory_inst_port -term "top.m1.QA" -retain_output x
```

In the above example, the memory cell is `mem_inst` as specified by the `-name` argument of the `memory_port` constraint. The value of the `-retain_output` argument for the QA read data bus of the m1 memory instance is overridden by the `memory_inst_port` constraint. For all other instances, the value of the `-retain_output` argument will remain 1.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax of the `memory_inst_port` constraint is as follows:

```
memory_inst_port
  -term <full-hierarchical-path-of-the-terminal>
  -retain_output <x|X>
```

## Arguments

### **-term <full-hierarchical-path-of-the-terminal>**

Specifies the name of the read data pin of the memory instance. You can use wildcards for specifying the memory terminal names. This should be the same as the name of the data bus specified in the read operation of the `memory_port` constraint.

### **-retain\_output <x|X>**

Used to override the value from '1' to 'x' for a particular instance.

**NOTE:** *This overriding of the value from '1' to 'x' is restricted to PESTR25 and PEPWR25 only*

## memory\_read\_pin

## Purpose

The `memory_read_pin` constraint is used to specify the read pin port name on a memory and the *inactive* value on that port.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `memoryreadpin`.*

## Product

SpyGlass DFT solution

## Syntax

The syntax of the `memory_read_pin` constraint is as follows:

```
memory_read_pin
  -memname <mem-name>
  -readport <readpin-name-list>
  -value <value>
```

## Arguments

The `memory_read_pin` constraint has the following arguments:

### **-memname <mem-name>**

The memory design unit (black box) type name (specified using a `memory_type` constraint).

You can specify a single design unit name or a space-separated list of design unit names.

### **-readport <readpin-name-list>**

Names of read pins in the specified memory module type.

You can specify only a single pin name or a list of pin names.

### **-value <value>**

The inactive value (0,1, X, or Z) for this read pin `<readpin-name>`.

**NOTE:** *The inactive value is a single value that when applied to the read pin on this memory will disable read from the memory.*

## Notes

- The `memory_read_pin` constraint requires use of a `memory_type` constraint with the same type specified. The specified pin is a read pin on this memory type.
- The `memory_read_pin` constraint may be used in conjunction with `test_mode` constraint.

## Rules

The `memory_read_pin` constraint is used by the following rule:

<b>SpyGlass DFT Solution</b>
RAM_06

## memory\_tristate

### Purpose

The `memory_tristate` constraint defines pins on memories and the values necessary to prevent the output of those memories from being a high impedance value.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `memory3s`.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `memory_tristate` constraint is as follows:

```
memory_tristate
  -memname <mem-name>
  -enableport <enable-pin>
  -value <value>
```

## Arguments

**-memname** <mem-name>

The memory design unit name.

**-enableport** <enable-pin1>

Name of the enable pin on the memory.

**-value** <value>

Value (string) of the enable pin.

## Rules

The `memory_tristate` constraint is used by the following rules:

---

**SpyGlass DFT Solution**

---

All Tristate rules

---

## memory\_type

### Purpose

The `memory_type` constraint specifies the memory design unit (black box) names. Then, all instances of these design units in the design are assumed to be memory instances.

**NOTE:** *In addition to `memory_type` constraint, memory instances are automatically inferred from library cells as well.*

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `memorytype`.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `memory_type` constraint is as follows:

```
memory_type -name <mem-name>
[-q_pin <q-pin-name> ]
[-d_pin <d-pin-name> ]
[-path_type <combinational | sequential>]
[-clock_pin <clk-pin-name>]
```

**NOTE:** *The memory\_type constraint supports wildcard characters.*

## Arguments

### **-name <mem-name>**

The memory design unit (black box) name.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are treated as memory design units.

You can specify a single design unit name or a space-separated list of design unit names.

**NOTE:** *The memory\_type constraint supports wildcard expressions. The supported meta-characters are \* (asterisk) and ? (question mark) where \* matches any number of characters and ? matches only one character. The wildcard support is applicable for non-escaped names only. If the meta-characters appear inside an escaped name, they are treated as literals. For example, in the expression "top.\mid\*\bottom", mid\* is considered as a literal and does not expand to "mid1, mid2, and so on. In addition, if you specify a hierarchical path using a wildcard, any sub-portion of this path that contains the wildcard 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 wildcard string. For example, the expression "top.mid\*.bottom" will expand to "top.mid1.bottom" and not to "top.mid2.bottom".*

**NOTE:** *The expression on which a wildcard is used should always be enclosed within double quotes. For example, "top.mid\*.bottom".*

**NOTE:** *The wildcard support is applicable for design objects only. For non-design objects, the support is not applicable.*

### **-q\_pin <q-pin-name>**

Q-pin of the memory design unit (black box) name.

**-d\_pin <d-pin-name>**

Data pin of the memory design unit (black box) name.

**-path\_type <combinational | sequential>**

When this argument is set as `sequential/combinational`, the path between `<d-pin-name>` to `<q-pin-name>` is treated as `sequential/combinational` path, respectively. By default, this is set as `sequential`.

**-clock\_pin <clk-pin-name>**

Clock pin of the memory design unit.

**Notes**

More than one `memory_type` constraint may be necessary. If more than one `memory_type` constraint is used, they will all be processed in parallel.

**Rules**

The `memory_type` constraint is used by the following rules:

SpyGlass DFT Solution			
RAM_01	RAM_02	RAM_03	RAM_04
RAM_05	RAM_06	RAM_07	RAM_08
RAM_09	RAM_10	RAM_11	

**memory\_write\_disable****Purpose**

The `memory_write_disable` constraint is used to specify the top-level primary ports and values that, when simulated, will disable all write enables to memories.

**NOTE:** *The `memory_write_disable` constraint will be deprecated in a future release.*



## Product

SpyGlass DFT solution

## Syntax

The syntax of the `memory_write_disable` constraint is as follows:

```
memory_write_disable
  -name <port-name>
  -value <value>
```

## Arguments

### **-name <port-name>**

Name of the top-level primary port required to disable all write enables to memories.

You can specify a single primary port name or a space-separated list of primary port names.

### **-value <value>**

Value list for a primary port.

The value list is the sequence of one or more values (each value being 0, 1, X, Z, or a combination) that when applied to the top-level port will disable all write enables to memories.

**NOTE:** See `memory_write_pin` constraint for more details.

## Rules

The `memory_write_disable` constraint is used by the following rules:

---

### **SpyGlass DFT Solution**

---

Info_memorywritedisable	RAM_05	RAM_08
-------------------------	--------	--------

---

## memory\_write\_pin

## Purpose

The `memory_write_pin` constraint is used to specify the write-pin port name on a memory and the *inactive* value on that port.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `memorywritepin`.*

## Product

SpyGlass DFT solution

## Syntax

The syntax of the `memory_write_pin` constraint is as follows:

```
memory_write_pin
  -memname <mem-name>
  -writeport <writepin-name-list>
  -value <value>
```

## Arguments

### **-memname <mem-name>**

The memory design unit (black box) type name (specified using a `memory_type` constraint).

You can specify a single design unit name or a space-separated list of design unit names.

### **-writeport <writepin-name-list>**

Names of write pins in the specified memory module type.

You can specify a single pin name or a list of pin names.

### **-value <value>**

The inactive value (0,1, X, or Z) for this write pin *<writepin-name>*.

## Notes

- The inactive value is a single value that when applied to the write pin on this memory will disable write to memory.

- The `memory_write_pin` constraint requires use of a `memory_type` constraint with the same type specified.
- The `memory_write_pin` constraint may be used in conjunction with `memory_write_disable` and/or `test_mode` constraints.

## Rules

The `memory_write_pin` constraint is used by the following rules:

SpyGlass DFT Solution		
RAM_05	RAM_06	RAM_08

## meta\_design\_hier

### Purpose

The `meta_design_hier` constraint is used to specify the top-level design name and the hierarchical name of the design with respect to the simulation testbench to be used by the `Ac_meta01` rule.

### Product

SpyGlass CDC solution

### Syntax

The syntax of the `meta_design_hier` constraint is as follows:

```
meta_design_hier
  -name <top-design-name>
  -inst <hier-instance-name>
```

### Arguments

**-name <top-design-name>**

The name of the top-level module for a Verilog design or top-level entity name for a VHDL design.

**-inst <hier-instance-name>**

The name of the hierarchical instance of the top-level design with respect to the simulation testbench.

**Example**

Consider an example in which the name of the top-level design is FSM, which is instantiated in the testbench top, tb, as I1. In this case, the `meta_design_hier` constraint should be specified in the following manner:

```
meta_design_hier -name FSM -inst tb.I1
```

**Rules**

The `meta_design_hier` constraint is used by the following rules:

<b>SpyGlass Auto Verify Solution</b>			
Av_deadcode01	Av_staticnet01		
<b>SpyGlass CDC solution</b>			
Ac_meta01	Ac_datahold01a	Ac_cdc01	Ac_conv02

**meta\_inst****Purpose**

The `meta_inst` constraint is used to specify instances for which monitors should be generated by the `Ac_meta01` rule.

**Product**

SpyGlass CDC Solution

**Syntax**

The syntax of the `meta_inst` constraint is as follows:

```
meta_inst
```

```
-name <instance-name>  
-type <allow | ignore>
```

## Arguments

**-name <instance-name>**

Specifies the hierarchical name of an instance.

**-type <allow | ignore>**

Set the value to `allow` to generate monitors for the signals present in the specified instance hierarchy.

Set the value to `ignore` to disable monitors generation for the signals present in the specified instance hierarchy.

## Example

Specify the following constraint to generate monitors for the `top.U1.U2` instance:

```
meta_inst -name top.U1.U2 -type allow
```

## Rules

The `meta_inst` constraint is used by the following rule:

---

**SpyGlass CDC solution**

---

`Ac_meta01`

---

## meta\_module

### Purpose

The `meta_module` constraint is used to specify modules/entities for which monitors should be generated by the `Ac_meta01` rule.

## Product

SpyGlass CDC Solution

## Syntax

The syntax of the `meta_module` constraint is as follows:

```
meta_module
  -name <du-name>
  -type <allow | ignore>
```

## Arguments

**-name <du-name>**

For Verilog design, this argument specifies a module name.

For VHDL design, this argument specifies an entity name (*<entity-name>*) or an architecture name (*<entity-name>. <architecture-name>*).

**-type <allow | ignore>**

Set the value to `allow` to generate monitors for the specified module/entity.

Set the value to `ignore` to disable monitors generation for the specified module/entity.

## Example

Specify the following constraint to generate monitors for the MOD module:

```
meta_module -name MOD -allow
```

## Rules

The `meta_module` constraint is used by the following rule:

---

**SpyGlass CDC solution**

---

Ac\_meta01

---

## meta\_monitor\_options

### Purpose

The `meta_monitor_options` constraint is used to specify the attributes used during monitor generation by the `Ac_meta01` rule.

### Product

SpyGlass CDC Solution

### Syntax

The syntax of the `meta_monitor_options` constraint is as follows:

```
meta_monitor_options
  -setup_hold_time <setup-hold-time>
  -init_time <init-time>
  -error_inject_threshold <threshold>
  -phase_shift_control <cycle-count>
  -log_file <file-name>
  -print_setup_hold_violation <violation-type>
```

### Arguments

#### **-setup\_hold\_time <setup-hold-time>**

Use this argument to define the setup/hold time (in percentage).

For example, if you set this argument to 10 and the period of the `clk` clock is 90, the setup/hold time for `clk` will be 9 ns. In this case, the `Ac_meta01` rule will force metastability only if the difference between the clock edge and data change is within 9 ns.

**Allowed value:** Integer greater than 0

**Default value:** 10

**NOTE:** You can also use the ***setup\_hold\_time parameter*** to specify the setup/hold time. However, preference is given to this constraint argument, if specified.

#### **-init\_time <init-time>**

Use this argument to specify the initialization time before starting

metastability analysis.

**Allowed value:** Integer greater or equal to 0. The value should be specified in the ns range.

**Default value:** 1 (which means analysis starts after 1 ns)

**NOTE:** *You can also use the **meta\_init\_time parameter** to specify the initialization time. However, preference is given to this constraint argument, if specified.*

#### **-error\_inject\_threshold <threshold>**

Use this argument to specify the threshold value for the random value above which error is injected in monitors in case of setup/hold violations.

**Allowed value:** Any value between 0 to 1

**Default value:** 0.5

**NOTE:** *You can also use the **meta\_error\_inject\_threshold parameter** to specify the threshold limit. However, preference is given to this constraint argument, if specified.*

#### **-phase\_shift\_control <cycle-count>**

Use this argument to specify the number of clock cycles after which the phase of a clock should be shifted by one unit during metastability analysis.

**Allowed value:** Any integer greater than or equal to 0. If you specify 0, phase of the clock is never shifted.

**NOTE:** *You can also use the **phase\_shift\_control parameter** to specify the phase shift. However, preference is given to this constraint argument, if specified.*

#### **-log\_file <file-name>**

Use this argument to specify the log file where all the messages generated during simulation are saved.

**Default value:** ""

#### **-print\_setup\_hold\_violation <violation-type>**

Use this argument to specify the criteria to dump setup/hold violations on the screen output or in the file specified by *-log\_file <file-name>*.

**Allowed values:**



error	Details of setup/hold violations when an error is injected are dumped
no_error	Details of setup/hold violations when no error is injected are dumped
both	Details of all the setup/hold violations are dumped
none	Details of none of the setup/hold violations are dumped

**Default value:** both

## Example

The following example shows the usage of the `meta_monitor_options` constraint:

```
meta_monitor_options -setup_hold_time 10 -init_time 2
-error_inject_threshold 0.3 -phase_shift_control 1000
-dump_in_file -log_file "log.txt"
```

## Rules

The `meta_monitor_options` constraint is used by the following rule:

---

### SpyGlass CDC solution

---

Ac\_meta01

---

## mode\_condition

### Purpose

The `mode_condition` constraint is used to specify conditions in which a mode of the specified mode set should be active.

Refer to the [Example](#) section for more details.

See also the PESAE08 rule documentation.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax of the `mode_condition` constraint is as follows:

```
mode_condition
  -name <mode_set_name>
  -value <mode_name>
  -on_condition <condition>
```

## Arguments

**-name <mode\_set\_name>**

Specifies the name of a mode set.

**-value <mode\_name>**

Specifies the name of the mode for which conditions are being defined.

**-on\_condition <condition>**

Specifies mode conditions in which the specified mode should be active. You can specify mode condition by creating a logical expression of valid nets in a design.

## Example

After you create a mode set and modes, you need to specify conditions under which a particular mode should be active by using the `mode_condition` constraint. You can create such conditions by using logical expressions.

Consider the following specification:

```
mode_condition -name SPEED -value SLOW -on_condition "top.w1
& top.w2 | (!top.speed) "
```

The above constraint implies that the mode "SLOW" of the mode set "SPEED is active when mode condition as specified by the expression with the `on_condition` switch is **true**.

## Rules

The `mode_condition` constraint is used by the following rules:

---

### SpyGlass Power Estimation and SpyGlass Power Reduction solutions

---

All rules running in EST mode

---

## module\_bypass

### Purpose

The `module_bypass` constraint is used to specify modules such as memories that are designed with a bypass between data-in port and data-out port.

The `module_bypass` constraint is used to propagate the test signals through the instance of a module, such as an analog level shifter. This constraint is used by the testability analysis engine. It helps transfer the controllability values from the input side to the output side and observable figures in the reverse direction.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `module_bypass` constraint is as follows:

```
module_bypass
  -name <du-name>
  [ -bpin <bpin-name-list> -value <value-list> ]
  -iport <port-name-list>
  -oport <port-name-list>
  [-invert_polarity]
```

**NOTE:** *The `module_bypass` constraint supports wildcard characters.*

## Arguments

### **-name <du-name>**

The name of the design unit (black box) to be bypassed.

The design unit must be a black box, that is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are treated as bypassed.

You can specify a single design unit name or a space-separated list of design unit names.

### **-bpin <bpin-name-list>**

(Optional) List of names of the bypass pins on a design unit (black box).

### **-value <value-list>**

(Optional) The active value (0 or 1) for all the bypass pins *<bpin-name-list>*.

The specified bypass pins and values are assumed to be mapped on one-to-one basis. You need to specify the exact same number of bypass pins and values with both the `-bpin` and `-value` arguments.

### **-iport <port-name-list> / -oport <port-name-list>**

Input/Output port name list of the module being bypassed.

The port name list is a space-separated simple name list, as in the following example:

```
...  
-iport in1 in2 -oport q qbar  
...
```

Then, the specified input ports and output ports are assumed to be directly connected on one-to-one basis. The mapped ports transfer the controllability and simulation values from the input ports to the output ports and the observable values from the output ports to the input ports.

You need to specify the exact same number of ports with both the `-iport`

and `-oport` arguments. If the number of ports with two arguments is different, the extra ports are ignored because they do not have corresponding elements to map and transfer values.

**NOTE:** *The module names, specified using the `-name` argument of `module_bypass` constraint, are automatically created as black boxes for SpyGlass DFT solution analysis.*

### **-invert\_polarity**

(Optional) Specify for inverting the polarity of output signals. This applies to all in port and out port specified with `iport/oport` pair in the constraint.

## **Rules**

The `module_bypass` constraint is used by the following rules:

---

### **SpyGlass DFT Solution**

All rules

---

### **SpyGlass Connectivity Verify Solution**

All rules

---

## **module\_pin**

### **Purpose**

The `module_pin` constraint is used to specify the set of pins for which the hierarchy of all instantiations is to be generated.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `modulePin`.*

### **Product**

SpyGlass DFT solution

### **Syntax**

The syntax of the `module_pin` constraint is as follows:

```
module_pin
```

```
-name <du-name>  
[ -pin <pin-name-list> ]  
[ -allpins ]  
[ -allinputs ]  
[ -alloutputs ]  
[ -exclude_module_list ]
```

## Arguments

### **-name <du-name>**

Name of the design unit for which you are specifying pins.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs).

You can only specify a single design unit name.

### **-pin <pin-name-list>**

(Optional) Space-separated list of pin names under the specified design unit.

If you specify none of the arguments (*-pin*, *-allpins*, *-allinputs*, or *-alloutputs*), the Soc\_06 rule generates hierarchy of all instantiations for all design unit pins.

### **-allpins**

(Optional) Equivalent to specifying the *-pin* argument. Specify a space-separated list of all the pins on the specified design unit.

### **-allinputs**

(Optional) Equivalent to specifying the *-pin* argument. Specify a space-separated list of all input pins on the specified design unit.

### **-alloutputs**

(Optional) Equivalent to specifying *-pin* argument. Specify a space-separated list of all output pins on the specified design unit.

**-exclude\_module\_list**

(Optional) Space-separated list of modules to be excluded from generating the instance list.

**NOTE:** *This argument supports wildcard characters.*

**Rules**

The `module_pin` constraint is used by the following rule:

---

**SpyGlass DFT Solution**


---

Soc\_06

---

**monitor\_time****Purpose**

The `monitor_time` constraint is used to specify the design initialization time frames during simulation. The rest of the simulation time is considered as the design's functional time.

The SystemVerilog Assertions (SVA) generated for assumptions and partially proved rules during simulation are active in the design's functional time. These SVA assertions are stored in SystemVerilog files that are located in the `test_reports/clock-reset/assertions/` directory. For details, refer to `cdc_dump_assertions` parameter documentation in the *SpyGlass CDC Rules Reference Guide*.

**Product**

SpyGlass CDC solution

**Syntax**

The syntax of the `monitor_time` constraint is as follows:

```
monitor_time
  -type <time-frame-type>
  -frame <start-time> <end-time>
```

## Arguments

**-type <time-frame-type>**

Specifies the type of time frame. The valid value is `initialization`.

**-frame <start-time> <end-time>**

Specifies a space-separated list of start and end times of the time frames of the specified type.

## Examples

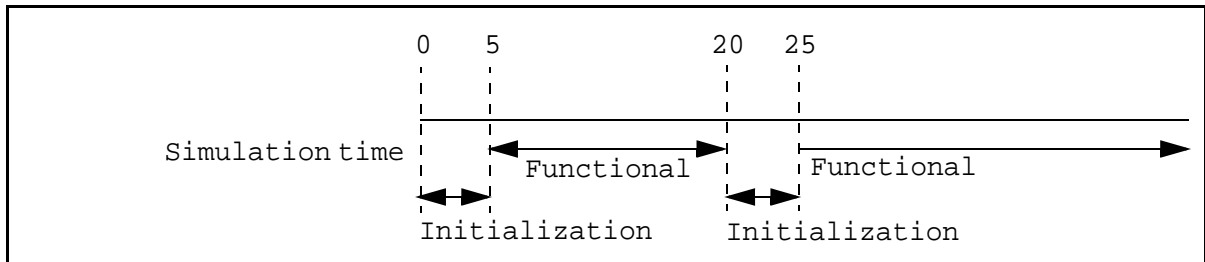
Consider the following `monitor_time` constraints:

```
monitor_time -type initialization -frame {0 5}
```

```
monitor_time -type initialization -frame {20 25}
```

The above constraints imply that the initialization time frames are from 0 to 5 seconds and then from 20 to 25 seconds.

In addition, it is also implied that the functional time frames are from 5 to 20 seconds and then from 25 seconds onwards. This is also illustrated in the following figure:



## Rules

The `monitor_time` constraint is used by the following rules:

SpyGlass CDC Solution		
Ac_datahold01a	Ac_cdc01	Ac_conv02



## multivt\_lib

### Purpose

The `multivt_lib` constraint is used to specify the VT library groups and their member libraries for the LPSVM29, LPSVM33, LPSVM33A, and LPSVM34 rules and the MTCMOS libraries for the LPSVM35 and LPSVM36 rules of the SpyGlass Power Verify solution.

### Product

SpyGlass Power Verify solution

### Syntax

#### Specifying VT Library Groups

The syntax to specify the `multivt_lib` constraint for VT Library Groups is as follows:

```
current_design <du-name>
  multivt_lib
    -type <lib-group-name>
    -names <lib-name-list>
```

#### Specifying MTCMOS Libraries

The syntax to specify the `multivt_lib` constraint for MTCMOS Libraries is as follows:

```
current_design <du-name>
  multivt_lib
    -type <keyword>
    -names <lib-name-list>
```

### Arguments

**<du-name>**

Name of the design unit under which you are specifying the VT library groups or MTCMOS libraries.

**-type <lib-group-name>**

Identifier for the VT library group.

**<keyword>**

Identifier for the MTCMOS libraries.

The supported values of *<keyword>* are:

- `mtcmOSH` (indicates an MTCMOS library with cells having supply-side sleep devices [transistors])
- `mtcomsF` (indicates an MTCMOS library with cells having ground-side sleep devices [transistors])
- `mtcmOSHf` (indicates an MTCMOS library with cells having both supply-side and ground-side sleep devices [transistors])

All other libraries are assumed to be CMOS libraries.

**-name <lib-name-list>**

Space-separated list of member library names (without the extension).

You can specify the names of gate (`.lib`) libraries (usually specified with the `read_file -type gateslib <file>` command in the project file) as well as the SpyGlass precompiled gate libraries (`.sglib`) libraries (usually specified by using the `read_file -type sourcelist <file-name>.f` command in the project file).

**Examples**

Consider the following `multivt_lib` constraint specification:

```
multivt_lib -type highvt -names CORE9GPHS1 CORE9GPHS2
multivt_lib -type lowvt -names CORE9GPLL1 CORE9GPLL2
```

The above specification indicates that there is a VT library group named `highvt` of libraries named `CORE9GPHS1` and `CORE9GPHS2` (ignoring the extension). This means that you would be specifying either of the following while running SpyGlass analysis:

```
spyglass -batch ...
  -gateslib CORE9GPHS1.lib -gateslib CORE9GPHS1.lib
  ...
```

Or

```
spyglass -batch ...
  -sglib CORE9GPHS1.sglib -sglib CORE9GPHS1.sglib
  ...
```

Similarly, there is another VT library group named `lowvt` of libraries named `CORE9GPLL1` and `CORE9GPLL2` (ignoring the extension).

## Rules

The `multivt_lib` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM29	LPSVM33	LPSVM33A	LPSVM34
LPSVM35	LPSVM36		

## network\_allowed\_cells

### Purpose

Specifies cells that can be allowed or disallowed in clock trees (or other trees).

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `network_allowed_cells` constraint is as follows:

```
current_design <du-name>
  network_allowed_cells
    -name <name-list>
    [ -type <type-list> ]
    [ -from <from-list> ]
    [ -disallow ]
```

## Arguments

### **-name <name-list>**

Specifies a space-separated list of names of the allowed/disallowed gate-level netlist cells.

### **-type <type-list>**

Specifies whether only clock, reset, or both trees should be checked.

### **-from <from-list>**

Specifies a space-separated list of names of nets from which the check should start.

You can use a combination of wildcard characters ('\*' and '?') when specifying netlist cells, clock and/or reset trees, and nets.

**NOTE:** *Set the hier\_wild\_card parameter to yes to match the expression specified in this argument with hierarchies.*

### **-disallow**

Indicates that the cells specified with the `-name` argument should be disallowed. By default, the specified cells are the only cells allowed in the network. If the `-disallow` argument is specified, these cells are the only cells NOT allowed in the network.

If you supply the `-type` argument, you must also specify the clock names, reset names, or both using the `clock` and `reset` constraints.

If you supply both the `-type` argument and the `-from` argument, the `-type` argument overrides the `-from` argument.

If you do not supply both `-type` and `-from` arguments, the complete design is checked for allowed/disallowed cells. For example, consider the following:

```
current_design top
  clock -name clk1
  network_allowed_cells -name A1234 -type clock
```

In this case, the `Clock_Reset_check01` rule reports instances of cells other than `A1234` found in the clock tree of clock signal `clk1`.

## Examples

Now consider the following example:

```
current_design top
  reset -name rst1
  network_allowed_cells -name A1234
    -type reset -disallow
```

In this case, the `Clock_Reset_check01` rule reports instances of `A1234` found in the reset tree of reset signal `rst1`.

Another example is as follows:

```
current_design top
  network_allowed_cells -name A1234
    -from w1 -disallow
```

In this case, the `Clock_Reset_check01` rule reports instances of the `A1234` found in the fan-out tree of net `w1`.

**NOTE:** *Only one `network_allowed_cells` constraint can be specified. If you want to specify constraints for multiple clocks or resets, use a single `network_allowed_cells` constraint.*

## Rules

The `network_allowed_cells` constraint is used by the following rule:

---

### SpyGlass CDC Solution

---

`Clock_Reset_check01`

---

## no\_atspeed

### Purpose

The `no_atspeed` constraint is used to exclude flip-flops from being used in at-speed testing, even if they so qualify. The `no_atspeed` constraint may

be used when there is no intention to use module or circuit for at-speed testing.

## Product

SpyGlass DFT DSM solution

## Syntax

The syntax for specifying the `no_atspeed` constraint is as follows:

```
no_atspeed
-name <du-name> | <net-name> | <hier-inst> |
-clock_control <signal-name> |
-set_control <signal-name> |
-reset_control <signal-name>
-register_suffix <suffixes>
-module_suffix <suffixes>
```

**NOTE:** *The `no_atspeed` constraint supports wildcard characters. Using wildcards, expression is expanded only within the hierarchy.*

## Arguments

### **-name <du-name>**

The name of the design unit from which flip-flops should be excluded in at-speed testing. You can specify design units that are single flip-flops or design units where one or more flip-flops are described besides other logic. Then, all flip-flops in the specified design unit are excluded from at-speed testing. The design unit name `<du-name>` can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered. You can specify a single design unit name or a space-separated list of design unit names.

### **-name <net-name>**

The name of a net that is connected to the output pin of a flip-flop. Then, the corresponding flip-flop is excluded from at-speed testing. You can specify a simple net name or a hierarchical net name. The net specified as simple net name is searched at the top-level. You can specify a single net name or a space-separated list of net names.

**<signal-name>**

Flip-flops whose control pins (clock, set, or reset) are driven by this signal are excluded in the scan. The <signal-name> argument can have the following values:

- **CLOCK**: For the clock\_control option, if the <signal-name> argument is driving CLOCK pin of the flip-flop.
- **SET**: For the set\_control option, if the <signal-name> argument is driving SET pin of the flip-flop.
- **RESET**: For the reset\_control option, if the <signal-name> argument is driving RESET pin of the flip-flop.

**-register\_suffix <suffixes>**

Space-separated list of suffixes to be specified as no\_atspeed. The -register\_suffix argument should not be used along with other arguments of the no\_atspeed constraint, that is, -name, -clock\_control, -set\_control, or -reset\_control.

If the value of the dft\_treat\_suffix\_as\_pattern 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.

If the value of the dft\_check\_path\_name\_for\_register\_suffix parameter is on, the value of the -register\_suffix field will be matched with the register name along with the path in which the register is present.

**-module\_suffix <suffixes>**

Define this field to use suffix based pattern match for all module names.

If the value of the dft\_treat\_suffix\_as\_pattern parameter is on, the value of the -module\_suffix field will be matched with the module name along with the path in which the module is present.

**Examples**

You can use the no\_atspeed constraint in the following ways:

- Specifying only the design unit names with the -name argument

By specifying a design unit name using the `-name` argument only, all instances of this design unit are not considered for at-speed testing. The following `no_atspeed` constraint indicates that all flip-flops within all instances of `modName1` will not be considered for at-speed testing:

```
no_atspeed -name modName1
```

- Specifying only the net names with the `-name` argument

By specifying a net name using the `-name` argument only, the corresponding flip-flop is not considered for at-speed testing. The following `no_atspeed` constraint indicates that the flip-flop whose output pin is connected to net `reg_123` (at the top-level) is not considered for at-speed testing:

```
no_atspeed -name reg_123
```

- Specifying only the hierarchical instance names with the `-name` argument

By specifying a hierarchical instance name using the `-name` argument only, all the flip-flops inside the given hierarchy are not considered for at-speed testing. The following `no_atspeed` constraint indicates that the flip-flop that lies inside the hierarchy `top.inst1` is not considered for at-speed testing.

```
no_atspeed -name top.inst1
```

- Specifying list of suffixes using the `-register_suffix` argument:

Consider the following example:

```
current_design top_no_scan
  force_no_scan -register_suffix ff1
```

In the above example, All flip-flops with name ending with `ff1` will be marked `force_no_scan`.

## Rules

The `no_atspeed` constraint is used by the following rules:

<b>SpyGlass DFT DSM Solution</b>		
Info_transitionCoverage	Info_noAtspeed	Atspeed_11
Info_transitionCoverage_audit		



## no\_convergence\_check

### Purpose

The `no_convergence_check` constraint is used to specify the net names that should not be checked for convergence.

If convergence is found on a gate, SpyGlass does not report a violation on that gate if you specify its output net through this constraint.

**NOTE:** *Use the `cdc_attribute` constraint instead of the `no_convergence_check` constraint. If the `no_convergence_check` constraint is used together with the `cdc_attribute` constraint, SpyGlass CDC ignores the `no_convergence_check` constraint and honors the `cdc_attribute` constraint.*

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `no_convergence_check` constraint is as follows:

```
no_convergence_check -name <sig-name-list>
```

### Arguments

**-name <sig-name-list>**

Space-separated list of hierarchical net names, hierarchical terminals, and ports that should not be checked for convergence.

**NOTE:** *Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.*

### Examples

The following example shows the usage of this constraint:

```
no_convergence_check -name top.U1.net1 top.net2
```

When you specify the above constraint, SpyGlass does not check the `top.U1.net1` and `top.net2` rules for convergence.

## Rules

The `no_convergence_check` constraint is used by the following rules:

SpyGlass CDC Solution		
Clock_sync03a	Clock_sync03b	Ac_conv01
Ac_conv02	Ac_conv03	

## no\_fault

### Purpose

The `no_fault` constraint is used to prevent faulting for a module.

As SpyGlass DFT solution has the means to estimate fault coverage (see the *Info\_coverage* rule), the `no_fault` constraint is useful to obtain more representative fault coverage numbers under special conditions. For example, when a design contains a synthesizable module that will be tested by some not yet implemented means, faults within each instance of that module will be included in the fault count total, thereby decreasing the fault coverage estimate. Use of the `no_fault` constraint removes such faults from consideration.

**NOTE:** *Prior to SpyGlass 4.4 release, the name of this constraint was `nofault`.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `no_fault` constraint is as follows:

```
no_fault -name <du-name> | <inst-name>
[- fault <hier_pin_names> ]
[- net <hier_net_names> ]
[- net_input <hier_net_names> ]
[- net_output <hier_net_names> ]
```

```

[- clock_control <hier_net_names> ]
[- set_control <hier_net_names> ]
[- reset_control <hier_net_names> ]
[- register_suffix ]
[- instance_suffix ]
[- module_suffix ]

```

**NOTE:** *The no\_fault constraint supports wildcard characters. Using wildcards, expression is expanded only within the hierarchy.*

## Arguments

The no\_fault constraint has the following arguments:

### **-name <du-name>**

Name of the design unit whose instances are to be bypassed.

The design unit name <du-name> can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are treated as no-fault design units.

You can specify a single design unit name or a space-separated list of design unit names.

### **-name <inst-name>**

Hierarchical name of the instance to be bypassed.

You can specify a single hierarchical instance name or a space-separated list of hierarchical instance names.

**NOTE:** *You can specify design unit names, hierarchical instance names, or a combination of both.*

### **-fault <hier\_pin\_names>**

(Optional) Space-separated list of hierarchical names of pins or ports.

**NOTE:** *Do not use this argument in case of RTL design because pin names will contain generated names and will fail SGDC sanity check at the RTL.*

### **-net <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the

faults in the direct fanin or fanout of the net as `no_fault`.

**-net\_input <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults in the direct fanin of the net as `no_fault`.

**-net\_output <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults in the direct fanout of the net as `no_fault`.

**-clock\_control <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the faults associated with the registers, where clock pin is topologically driven by the specified clock, as `no_fault`.

**-set\_control <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the fault associated with the registers, where set pin is topologically driven by the specified set signal, as `no_fault`.

**-reset\_control <hier\_net\_names>**

(Optional) Space-separated list of hierarchical names of nets. Mark all the fault associated with the registers, where reset pin is topologically driven by the specified reset signal, as `no_fault`.

**-register\_suffix <suffixes>**

Space-separated list of suffixes to be specified as `no_fault`. The `-register_suffix` argument should not be used along with other arguments of the `no_fault` constraint, that is, `-name`, `-clock_control`, `-set_control`, or `-reset_control`.

If the value of the `dft_treat_suffix_as_pattern` 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.

If the value of the `dft_check_path_name_for_register_suffix` parameter is `on`, the value of the `-register_suffix` field will be

matched with the register name along with the path in which the register is present.

**-instance\_suffix <suffixes>**

Define this field to use suffix based pattern match for all instance names.

**-module\_suffix <suffixes>**

Define this field to use suffix based pattern match for all module names.

If the value of the `dft_treat_suffix_as_pattern` parameter is on, the value of the `-module_suffix` field will be matched with the module name along with the path in which the module is present.

## Examples

### Specifying list of suffixes using the `-register_suffix` argument

Consider the following example:

```
R1 (register 1) name: top.u_ctrl.u2.u1.ff1_ctrl
R2 (register 2) name: top.u_ctrl.u2.u1.ff1_state
R3 (register 3) name: top.u_core.u2.u1.ff1_state_ctrl
R4 (register 4) name: top.u_ctrl_state.u2.u1.ff1_ctrl_state
```

Now, consider the following *no\_fault* descriptions:

```
no_fault -register_suffix ctrl
no_fault -register_suffix state
```

The following table lists the results when combination of values are used for the *dft\_treat\_suffix\_as\_pattern* and *dft\_check\_path\_name\_for\_register\_suffix* parameters:

**TABLE 7** Pattern Matching for the `-register_suffix` argument

Value of <code>dft_treat_suffix_as_pattern</code>	Value of <code>dft_check_path_name_for_register_suffix</code>	Value of <code>-register_suffix</code>	Matched Registers
off	off	ctrl	R1, R3
		state	R2, R4
off	on	ctrl	R1, R2, R3
		state	R2, R4
on	off	ctrl	R1, R3, R4
		state	R2, R3, R4
on	on	ctrl	R1, R2, R3, R4
		state	R2, R3, R4

### Example 3: Specifying list of suffixes using the `-instance_suffix` argument

Consider the following example:

```
R1 (register 1) name: top.u_ctrl.u2.u1.ff1_ctrl
R2 (register 2) name: top.u_ctrl.u2.u1.ff1_state
R3 (register 3) name: top.u_core.u2.u1.ff1_state_ctrl
R4 (register 4) name: top.u_ctrl_state.u2.u1.ff1_ctrl_state

I1 (instance 1) name: top.u_ctrl.u2.u1.inst1_ctrl
I2 (instance 2) name: top.u_ctrl.u2.u1.inst1_state
I3 (instance 3) name: top.u_core.u2.u1.inst1_state_ctrl
I4 (instance 4) name:
top.u_ctrl_state.u2.u1.inst1_ctrl_state
```

Now, consider the following *no\_fault* descriptions:

```
no_fault -instance_suffix ctrl
no_fault -instance_suffix state
```

The following table lists the results when combination of values are used for the *dft\_treat\_suffix\_as\_pattern* and *dft\_check\_path\_name\_for\_instance\_suffix* parameters:

**TABLE 8** Pattern Matching for the -instance argument

Value of <i>dft_treat_suffix_as_pattern</i>	Value of <i>dft_check_path_name_for_instance_suffix</i>	Value of - <i>instance_suffix</i>	Matched Registers/Instances
off	off	ctrl	R1, R3, I1, I3
		state	R2, R4, I2, I4
off	o	ctrl	R1, R2, R3, I1, I2, I3
		state	R2, R4, I2, I4
on	off	ctrl	R1, R3, R4, I1, I3, I4
		state	R2, R3, R4, I2, I3, I4
on	on	ctrl	R1, R2, R3, R4, I1, I2, I3, I4
		state	R2, R3, R4, I2, I3, I4

## Rules

The `no_fault` constraint is used by the following rules:

SpyGlass DFT Solution	
Info_coverage	Coverage_audit
SpyGlass DFT DSM Solution	
Info_transitionCoverage	Info_transitionCoverage_audit

## no\_test\_point

### Purpose

The `no_test_point` constraint is used to exclude modules or instance, which should not be considered for suggesting test points.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax of the `no_test_point` constraint is as follows:

```
no_test_point
  -name <module-name | <instance_list>
```

### Arguments

The `no_test_point` constraint has the following arguments:

**-name <module-name | <instance\_list>**

Name of the module or the list of instances to be considered for suggesting the test points.

### Rules

The `no_test_point` constraint is used by the `Info_random_resistance` rule.



## noclockcell\_start

### Purpose

Specifies start points (ports or nets) for checking.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `noclockcell_start` constraint is as follows:

```
current_design <du-name>
  noclockcell_start
    -name <port-net-name>
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the start point

#### -name <port-net-name>

The hierarchical name of the start point port/net.

You can specify multiple start point signals using multiple `noclockcell_start` constraints.

**NOTE:** *You must specify at least one `noclockcell_start` constraint for the `NoClockCell` rule to perform rule-checking.*

### Examples

The following example specifies net `clk1` of design unit `top` as the start point:

```
current_design top
  noclockcell_start -name top.clk1
```

## Rules

The `noclockcell_start` constraint is used by the following rule:

---

### SpyGlass CDC Solution

---

NoClockCell

---

## noclockcell\_stop\_instance

### Purpose

Specifies the instance where the NoClockCell rule should stop further traversal along the clock tree when the clock pin of the specified instance is hit.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `noclockcell_stop_instance` constraint is as follows:

```
current_design <du-name>
  noclockcell_stop_instance
    -name <inst-name>
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the stop point instance

#### -name <inst-name>

The hierarchical name of the stop point instance.

You can specify multiple stop point instances using multiple `noclockcell_stop_instance` constraints.

## Examples

The following example specifies instance I21 as the stop point instance under the design unit `top`:

```
current_design top
  noclockcell_stop_instance -name top.I21
```

Here, the NoClockCell rule stops further traversal along the clock tree when the clock pin of instance `top.I21` is hit.

The clock traversal automatically stops at instances of a stopped design unit (stopped using the `set_option stop <du-name>` command in the project file or the equivalent option in Atrenta Console GUI). Therefore, you need not specify such design unit instances with the `noclockcell_stop_instance` constraint.

## Rules

The `noclockcell_stop_instance` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

NoClockCell

---

## noclockcell\_stop\_module

### Purpose

Specifies the design unit where the NoClockCell rule should stop further traversal along the clock tree when the clock pin of an instance of the specified design unit is hit.

### Product

SpyGlass CDC solution

## Syntax

The syntax for specifying the `noclockcell_stop_module` constraint is as follows:

```
current_design <du-name>
  noclockcell_stop_module
    -name <du-name>
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the stop-point design unit.

### -name <du-name>

Name of the stop-point design unit.

You can specify multiple stop point design units using multiple `noclockcell_stop_module` constraints.

## Examples

The following example specifies design unit `m1` as the stop-point design unit under the design unit `top`:

```
current_design top
  noclockcell_stop_module -name m1
```

Here, the `NoClockCell` rule stops further traversal along the clock tree when the clock pin of an instance of design unit `m1` is hit.

The clock traversal automatically stops at instances of a stopped design unit (stopped using the `set_option stop <du-name>` command in the project file or the equivalent option in Atrenta Console GUI). Therefore, you need not specify such design units with the `noclockcell_stop_module` constraint.

## Rules

The `noclockcell_stop_module` constraint is used by the following

rule:

---

**SpyGlass CDC Solution**

---

NoClockCell

---

## noclockcell\_stop\_signal

### Purpose

Specifies the design points (ports, pins, or nets) where the NoClockCell rule should stop further traversal along the clock tree.

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the `noclockcell_stop_signal` constraint is as follows:

```
current_design <du-name>
  noclockcell_stop_signal
    -name <sig-name>
```

### Arguments

**<du-name>**

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) under which you are specifying the stop point.

**-name <sig-name>**

The hierarchical name of the end point port/pin/net.

You can specify multiple stop points using multiple `noclockcell_stop_signal` constraints.

## Example

The following example specifies pin `d1` of instance `I1` in design unit `top` as the stop point:

```
current_design top
  noclockcell_stop_signal -name top.I1.d1
```

Here, the `NoClockCell` rule stops further traversal along the clock tree when the pin `top.I1.d1` is hit.

The clock traversal automatically stops at the design points (ports, pins, or nets) that belong to instances of a stopped design unit (stopped using the `set_option stop <du-name>` command in the project file or the equivalent option in Atrenta Console GUI). Therefore, you need not specify such design points with the `noclockcell_stop_signal` constraint.

## Rules

The `noclockcell_stop_signal` constraint is used by the following rule:

---

**SpyGlass CDC Solution**

---

NoClockCell

---

## non\_pd\_inputcells

### Purpose

The `non_pd_inputcells` constraint is used to specify the cells that should not be present at the input stage of the power/voltage domain.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `non_pd_inputcells` constraint is as follows:

```

current_design <du-name>
  non_pd_inputcells
    -names <cell-name-list>
    [ -pd <pd-name> ]
    [ -pins <pin_list> ]

```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the input cells.

**-names <cell-name-list>**

Space-separated list of cell names. You can use wildcard characters while specifying cell names using the -names argument.

**-pd <pd-name>**

Name of the always-on domain/power domain for which the specified cells are to be checked.

If you do not specify the -pd argument, SpyGlass assumes that the specified cells are to be checked for all specified power/voltage domains.

**<pin-list>**

Name of the input pins of cells, which should not be present in at the input. When the pin list is not specified, all input pins are considered.

## Rules

The `non_pd_inputcells` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

LPSVM50

---

## num\_flops

## Purpose

Specifies the minimum number of flip-flops required in synchronizer chain for the Conventional Multi-Flop Synchronization Scheme.

## Product

SpyGlass CDC solution

## Syntax

The syntax of the `num_flops` constraint is as follows:

```
num_flops
  -to_domain <dest-clk-domain>
  -value <num>
  [ -from_domain <src-clk-domain> ]
  [ -cells <library-cell-names> ]
  [ -lib <library-names> ]
  [ -reset ]
  [ -rdc ]
```

or

```
num_flops
  -to_clk <dest-clk-name>
  -value <num>
  [ -from_clk <src-clk-name> ]
  [ -cells <library-cell-names> ]
  [ -lib <library-names> ]
  [ -reset ]
  [ -rdc ]
```

or

```
num_flops
  -to_period <dest_clk_period>
  -value <num>
  [ -cells <library-cell-names> ]
  [ -lib <library-names> ]
  [ -reset ]
```



```
[ -rdc ]
```

or

```
num_flops -default <def_val>
  [ -cells <library-cell-names> ]
  [ -lib <library-names> ]
  [ -reset ]
```

**NOTE:** Please note the following points:

- ☞ You must specify the `-to_clk` argument while specifying the `-from_clk` argument.
- ☞ You must specify the `-to_domain` argument while specifying the `-from_domain` argument.

## Arguments

The syntax of using the `num_flops` constraint in a SpyGlass Design Constraints file is as follows:

**`-from_domain <src-clk-domain>`**

The domain name of the source clock.

**`-to_domain <dest-clk-domain>`**

The domain name of the destination clock.

**`-from_clk <src-clk-name>`**

Name of the source clock.

**`-to_clk <dest-clk-name>`**

Name of the destination clock.

**`-to_period <dest-clk-period>`**

Period of the destination clock. The value is applicable to all the crossings with destination clocks that have period less than or equal to `<dest-clk-period>`.

**-value <num>**

The minimum number of flip-flops required in a synchronizer chain for the crossing specified through:

- The source clock specified by `-from_clk <src-clk-name>` or the source clock of the domain specified by `-from_domain <src-clk-domain>`.
- The destination clock specified by `-to_clk <dest-clk-name>` or the destination clock of the domain specified by `-to_domain <dest-clk-domain>`.

The synchronizer chain in this case is the structure of the *Conventional Multi-Flop Synchronization Scheme*.

**NOTE:** *The minimum value for this argument is 1. However, if you specify the `-reset` argument, the minimum value for this argument should be 2.*

For domain pairs not mentioned with this constraint, the number of flip-flops to be used for multi-flop synchronization is the value of the parameter, `num_flops`, which is 2 by default.

**-default <def\_val>**

The number of flip-flops for crossings not specified with the `num_flops` constraint.

Some of the examples of the `num_flops` constraint are as follows:

```
num_flops -from_clk clk1 -to_clk clk2 -value 3
```

The above specification specifies that for multi-flop synchronization, at least three flip-flops should be used for the crossings between the clocks, `clk1` and `clk2`.

```
num_flops -from_domain D1 -to_domain D2 -value 3
```

The above specification specifies that for multi-flop synchronization, at least three flip-flops should be used for the crossings between all the clocks in domains, `D1` and `D2`.

```
num_flops -to_period 20 -value 4
```

The above specification specifies that for all the crossings with destination clocks that have period less than or equal to 20, at least four flip-flops should be used for multi-flop synchronization.

```
num_flops -default 2
```

The above specification specifies that for all the clock crossings for which the `num_flops` constraint is not specified, at least two flip-flops should be used for multi-flop synchronization.

#### **-cells <library-cell-names>**

Space-separated list of library cell names. You can use wildcard characters while specifying library cell names.

The following example shows the usage of the `-cells` argument:

```
num_flops -from_domain D1 -to_domain D2 -cells X_CELL
-value 3
```

In this case, for crossings from D1 to D2, the `X_CELL` cell is allowed to be a part of the synchronizer chain and there should be minimum of three such cells in the chain, including the destination instance.

#### **-lib <library-names>**

Space-separated list of library names.

The following example shows the usage of the `-lib` argument:

```
num_flops -from_domain D1 -to_domain D2 -lib LIB1
-value 3
```

In this case, for crossings from D1 to D2, only the cells from the `LIB1` library are allowed to be a part of the synchronizer chain and there should be minimum three such cells in the chain, including the destination instance.

#### **-reset**

Applies the `num_flops` constraint on reset synchronizers with respect to a clock, domain, or frequency.

When you specify this argument, the minimum value specified to the `-value <num>` argument should be 2.

**NOTE:** *You cannot specify the `-from_clk`/`-from_domain` arguments along with the `-reset` argument.*

#### **-rdc**

Enables the `Ar_resetcross01` rule to apply the `num_flops` constraints on

reset domain crossings.

It is used to set a limit on the number of synchronizer flip-flops.

When you specify this argument, the minimum value specified to the *-value <num>* argument should be 2.

## Rules

The `num_flops` constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_sync01	Ac_sync02	Ac_unsync01	Ac_unsync02
Ar_resetcross_m atrix01	Ar_cross_analysi s01	Clock_sync08	Clock_sync03a
Clock_sync03b	Clock_sync08a	Clock_sync09	Ac_handshake01
Ac_handshake02	Ac_glitch03	Ac_cdc01a	Ac_cdc01b
Ac_cdc01c	Ac_cdc08	Ac_conv01	Ac_conv02
Ac_conv03	Ar_sync01	Reset_sync04	Reset_sync01
Ar_unsync01	Reset_sync03	Ar_resetcross01	Ac_conv04

## operating\_mode\_set

### Purpose

The `operating_mode_set` constraint is used to assign modes to a mode set.

For example, you can create a mode set, `SPEED`, that contains a set of two modes, `FAST` and `SLOW`.

Refer to the [Example](#) section for more details.

See also the PESAE08 rule documentation

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax of the `operating_mode_set` constraint is as follows:

```
operating_mode_set
  -name <mode_set_name>
  -values <mode_names>
```

## Arguments

**-name <mode\_set\_name>**

Specifies the name of a mode set

**-values <mode\_names>**

Specifies mode names that should be included in the mode set

## Example

A design can operate in multiple modes at a time. For example, it can operate in TX mode, RX mode, SLOW mode, or FAST mode.

You can group all related modes under one category, where each category is known as a mode set.

For example, you can group the TX and RX mode under one mode set, STATE by specifying the following constraint:

```
operating_mode_set -name STATE -values TX RX
```

The following specification shows how to create a mode set, SPEED, with SLOW and FAST modes by using the `operating_mode_set` constraint:

```
operating_mode_set -name SPEED -values SLOW FAST
```

## Rules

The `operating_mode_set` constraint is used by the following rules:

---

**SpyGlass Power Estimation and SpyGlass Power Reduction solutions**

---

All rules running in EST mode

---

## output

### Purpose

The `output` constraint is used to specify clock domain at output ports. If you want to check synchronization for paths ending on output ports, you need to provide information about the destination clocks associated with those ports. The `output` constraints are used for this purpose.

You need to specify effective source clocks or destination clocks using the `clock` keyword and port-clock pairs using the `output` keyword in a SpyGlass Design Constraints file.

### Product

SpyGlass CDC solution

### Syntax

The syntax of using the `output` constraint in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>  
  clock -name <clk-name> -domain <domain-name>  
  output -name <output-name-list>  
  -clock <dest-clk-name>
```

### Arguments

**<du-name>**

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**-name <clk-name>**

The clock signal name or a virtual clock name.

**-domain <domain-name>**

The clock domain name.

**-name <output-name-list>**

Port names that can be scalar ports, bus ports, or wildcard names (matching against all top-level ports of appropriate type).

If you specify the same output port in multiple output constraint specifications, SpyGlass considers the last output constraint specification. Consider the following example in which the same output port, out1, is specified in two different output constraint specifications:

```
output -name out1 -clock clk2
output -name out1 -clock clk4
```

In the above example, SpyGlass considers the last output constraint specification, that is, output out1 clocked by clock clk4.

**-clock <dest-clk-name>**

The name of the destination clock for a primary output port or a virtual clock specified by the *-tag <logical-clock-name>* argument of the *clock* constraint. For virtual clock specification, see [Example 2](#).

You can use the match many (\*) and match one (?) wildcard characters with the *-name* argument of the output constraints by specifying the regular expression enclosed in double quotes ("").

**NOTE:** *Do not specify clock domain names to this argument.*

## Examples

### Example 1

Consider the following output constraint specification:

```
output -name "out?" -clock clk1
```

The above specification matches all output ports whose names start with out followed by zero or one character, such as out, out1.

## Example 2

Consider the following constraints:

```
clock -name clk1 -domain d1
clock -name clk2 -domain d2 -tag c1
clock -tag c2 -domain d3
```

```
output -name out1 -clock c2
```

In the above specification, the virtual clock `c2` is specified to the `-clock` argument of the `output` constraint. However, note that you cannot specify `c1` to the `-clock` argument as `c1` is not a virtual clock.

## Rules

The `output` constraint is used by the following rules:

---

### SpyGlass CDC Solution

---

All clock synchronization rules except the `Clock_sync06` rule

---

## output\_not\_used

### Purpose

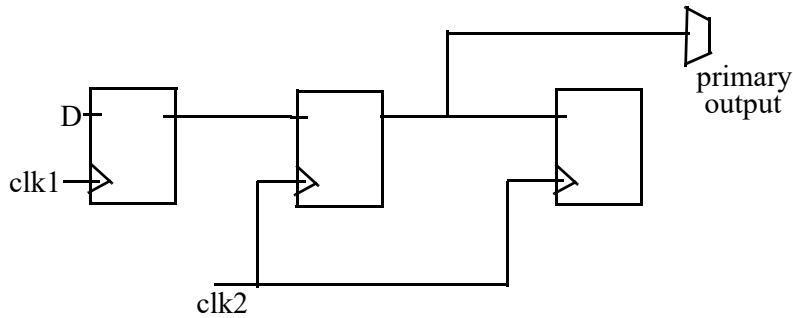
In a multi-flop synchronizer, if the synchronizing flip-flops feed more than one destination, the value seen by the destination may be subject to cycle uncertainty or even go metastable, making the synchronizer ineffective. Synchronization check will fail if synchronizing flip-flops feed multiple destinations.

It is often possible that a design has synchronizing flip-flops with multiple fan-outs where only one fan-out is active at a time. Provide case analysis that will enable only one fan-out thereby preventing *Ac\_unsync01/* *Ac\_unsync02* violations.

There can be cases in which a fan-out is feeding a primary output (ignoring buffers and inverters) and case-analysis cannot be used, as shown in the



following figure.



**FIGURE 41.** Fan-out Feeding Primary Output

In such cases, you may use the constraint, `output_not_used`.

## Product

SpyGlass CDC solution

## Syntax

Use the `output_not_used` constraint as follows to specify the name of the primary output port so that the connection is ignored while checking for synchronization:

```
current_design <du-name>
  output_not_used -name { <port-name> }
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

### -name <port-name>

The name of the primary output port.

You can use a combination of wildcard characters ('\*' and '?') when

specifying port names.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

## Rules

The `output_not_used` constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_sync01	Ac_sync02	Clock_sync08	Clock_sync03a
Clock_sync03b	Clock_sync08a	Clock_sync09	Propagate_clocks
Ac_cdc01a	Ac_cdc01b	Ac_cdc01c	Ac_cdc08
Ac_conv02	Ac_conv03	Ac_handshake01	Ac_handshake02
Ac_conv01	Ac_unsync01	Ac_unsync02	

## pg\_cell

### Purpose

The `pg_cell` constraint is used to specify the names of power/ground pins for cells present in the input netlist, which are missing from the respective PLIB/LIB/LEF libraries.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `pgcell`.

### Product

SpyGlass Power Verify solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `pg_cell` constraint is as follows:

```
current_design <du-name>
  pg_cell
    -name <lib-group-name>
    [ -powerTerms <term-name-list> ]
```

```
[ -groundTerms <term-name-list> ]
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the PG cell pins.

**-name <lib-group-name>**

Specifies an identifier for a PG cell.

You can use wildcard characters while specifying cell names by using the -name argument.

**-powerTerms <term-name-list> / -groundTerms <term-name-list>**

Space-separated list of member power/ground terminal names.

## Examples

The following example declares a PG cell named PX\_120\_LS having power terminal named VDDEXT and ground terminal named VSSEXT:

```
pg_cell -name PX_120_LS
  -powerTerms VDDEXT -groundTerms VSSEXT
```

The following example declares a PG cell named PX\_150\_LS having power terminals named VDDEXT and VDDBULK and ground terminals named VSSEXT and VSSBULK:

```
pg_cell -name PX_150_LS
  -powerTerms VDDEXT VDDBULK
  -groundTerms VSSEXT VSSBULK
```

## Rules

The `pg_cell` constraint is used by the following rules:

<b>SpyGlass Power Verify Solution</b>			
LPPLIB04	LPPLIB05	LPPLIB06	LPPLIB07
LPPLIB08	LPPLIB11	LPPLIB15	LPPLIB16

---

**SpyGlass Power Estimation and SpyGlass Power Reduction solutions**

---

All rules running in EST mode

---

## pg\_pins\_naming

### Purpose

Specifies power/ground pin names for single supply cells that need to be checked.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `pgpins_naming`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `pg_pins_naming` constraint is as follows:

```
current_design <du-name>
  pg_pins_naming
    [ -power <pwr-pin-names>]
    [ -ground <gnd-pin-names>]
    [ -biaspower <bias-pwr-pin-names>]
    [ -biasground <bias-gnd-pin-names>]
```

### Arguments

**<du-name>**

Name of the design unit under which you are specifying the power/ground pin names for single supply cells.

**-power <pwr-pin-names> & -ground <gnd-pin-names>**

Simple pin name lists or lists of regular expressions. You can use wildcard characters while specifying power/ground pin names.

**-biaspower <bias-pwr-pin-names> & -biasground <bias-gnd-pin-names>**

Simple bias pin name lists or lists of regular expressions. You can use wildcard characters while specifying power/ground pin names.

## Rules

The `pg_pins_naming` constraint is used by the following rules:

SpyGlass Power Verify Solution		
LPSVM49	LPPLIB18A	LPPLIB18B

## pin\_voltage

### Purpose

Specifies voltage/power domain for primary ports, pins of design units, or instances in the design.

**NOTE:** The *voltage\_domain* constraint allows you to specify the voltage/power domains of design units and instances. Use the `pin_voltage` constraint to specify voltage/power domains of pins.

The different ways to specify the `pin_voltage` constraint are as follows:

- Specify voltage/power domain for all pins of all instances of a specified design unit

Use the `-module` and `-default` arguments, as in the following example:

```
current_design top
  voltage_domain -name VD1 ...
  pin_voltage -voltage VD1 -module my_block -default
```

The above specification indicates that the voltage/power domain of all pins of all instances of design unit `my_block` is `VD1`.

- Specify voltage/power domain for named pins of all instances of a specified design unit

Use the `-module` and `-names` arguments, as in the following

example:

```
current_design top
  voltage_domain -name VD1 ...
  pin_voltage -voltage VD1 -module my_block -names A D
```

The above specification indicates that the voltage/power domain of pins named A and D of all instances of design unit `my_block` is VD1.

- Specify the voltage/power domain for all pins of the specified instance  
Use the `-instance` and `-default` arguments as in the following example:

```
current_design top
  voltage_domain -name VD1 ...
  pin_voltage -voltage VD1 -instance top.U1 -default
```

The above specification indicates that the voltage/power domain of all pins of instance U1 under design unit `top` is VD1.

- Specify the voltage/power domain for named pins of the specified instance  
Use the `-instance` and `-names` arguments, as in the following example:

```
current_design top
  voltage_domain -name VD1 ...
  pin_voltage -voltage VD1 -instance top.U1 -names A D
```

The above specification indicates that the voltage/power domain of pins named A and D of instance U1 under design unit `top` is VD1.

- Specify the voltage/power domain for selected pins of the bus of the specified instance  
Use the `-instance` and `-names` arguments as in the following example:

```
current_design top1
  voltage_domain -name Vtop -value 1.2 -modname top1
  pin_voltage -voltage Vtop -instance top1.mid1 \
    -names in[0:2]
```

The above specification indicates that the voltage/power domain of pins named `in[0]`, `in[1]`, and `in[2]` of instance `mid1` under design unit

top1 is Vtop.

Please note the following:

- A `pin_voltage` specification using the `-instance` argument overrides the `pin_voltage` constraint using the `-module` argument for the same instance.

For example, assume design unit M1 has two instances U1 and U2 in design unit top. If you specify the following `pin_voltage` constraints, the inferred voltage/power domain for pin D of instance U2 under design unit top is VD2:

```
...
pin_voltage -voltage VD1 -module M1 -default
...
pin_voltage -voltage VD2 -instance top.U2 -names D
...
```

Here, the inferred voltage/power domain for all pins of instance U1 under design unit top and all pins except pin D of instance U2 under design unit top is VD1.

- The inferred voltage/power domain for all pins for which a `pin_voltage` constraint is not specified is the same as voltage/power domain of their parent instance.
- All voltage/power domain checking is applicable to pins specified with the `pin_voltage` constraint just like for design units and instances specified with the [voltage\\_domain](#) constraint.
- You can also specify external voltage/power domains (created with the standalone `-external` argument of the [voltage\\_domain](#) constraint) with the `-voltage` argument of the `pin_voltage` constraint.

The `pin_voltage` constraint overrides the [voltage\\_domain](#) constraint for pins. Therefore, if you have specified the voltage/power domain for a pin with both these constraints, the `pin_voltage` constraint information is used and the `voltage_domain` constraint information is ignored.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `pinvoltage`.

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `pin_voltage` constraint is as follows:

```
current_design <du-name>
  pin_voltage
    -voltage <vpd-name>
    [ -module <du-name> -default ]
    | [ -module <du-name> -names <pin-name-list> ]
    | [ -instance <du-name> -default ]
    | [ -instance <du-name> -names <pin-name-list> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the voltage/power domain for pins.

### -voltage <vpd-name>

Name of a voltage domain or a power domain already specified using the [voltage\\_domain](#) constraint.

You can use wildcard characters while specifying module names (using the `-module` argument) and instance names (using the `-instance` argument).

## Rules

The `pin_voltage` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM04A	LPSVM04B	LPSVM04C	LPSVM04D



## pll

### Purpose

The `pll` constraint is used to specify the PLL (Phase Lock Loops) modules.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `pll` constraint is as follows:

```
pll -name <mod-name>
    -clkin <port-name>
    -clkout <port-name>
    [-reset <rst-port-name>
     -value <value-that-resets-the-pll>]
```

**NOTE:** *The `pll` constraint supports wildcard characters.*

### Arguments

The `pll` constraint has the following arguments:

**-name <mod-name>**

The PLL module name.

**NOTE:** *Any module declared as a PLL will automatically be created as a black box in SpyGlass DFT DSM solution.*

**-clkin <port-name> / -clkout <port-name>**

Input/Output port name of the `pll` module.

Consider the following example:

```
...
-clkin in1 -clkout qbar
...
```

**NOTE:** *Only one `clkin` port name is allowed.*

**-reset** <rst-port-name>

(Optional) Name of the reset pin on the design unit (black box). You can specify only a single pin name.

**-value** <value-that-resets-the-pll>

(Optional) The active value (0 or 1) for the pin <rst-port-name>.

## Rules

This pll constraint is used by the following rules:

---

### SpyGlass DFT Solution

---

All rules

---

### SpyGlass DFT DSM Solution

---

PLL\_01

PLL\_02

---

## port\_time\_delay

### Purpose

Specifies the design units to be checked by the PortTimeDelay rule.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was porttimedelay.*

### Product

SpyGlass CDC solution

### Syntax

The syntax for specifying the port\_time\_delay constraint is as follows:

```
current_design <du-name>
  port_time_delay
    -name <ptddu-name>
    [ -ignore_instances <instance-list> ]
    [ -ignore_ports <port-name-list> ]
```

```
[ -notimedelay_ports <ntdport-name-list> ]
```

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in *<entity-name>.<arch-name>* format (for VHDL designs) under which you are specifying the design unit *<ptddu-name>* to be checked.

### -ignore\_instances <instance-list>

(Optional) Allows you to specify a space-separated hierarchical name list *<instance-list>* of instances that should be ignored by the PortTimeDelay rule, if found in the path.

### -ignore\_ports <port-name-list>

(Optional) Allows you to specify a space-separated name list *<port-name-list>* of ports of the design unit *<ptddu-name>* specified with the *-name* argument that should be ignored by the PortTimeDelay rule.

**NOTE:** Set the *hier\_wild\_card* parameter to *yes* to match the expression specified in this argument with hierarchies.

### -notimedelay\_ports <ntdport-name-list>

(Optional) Allows you to specify a space-separated name list *<ntdport-name-list>* of ports of the design unit *<ptddu-name>* specified with the *-name* argument that should be checked for unexpected time delay value by the PortTimeDelay rule.

**NOTE:** Set the *hier\_wild\_card* parameter to *yes* to match the expression specified in this argument with hierarchies.

## Rules

The *port\_time\_delay* constraint is used by the following rule:

---

**SpyGlass CDC Solution**

---

PortTimeDelay

---

## power\_data

### Purpose

Specifies the details of files from which power data is to be taken.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `power_data` constraint is as follows:

```
power_data
  -format <CPF | UPF> ]
  -file <file-name-list>
  -version <version-number>
```

### Arguments

#### **-file <file-name-list>**

List of files that contain power data.

**NOTE:** For the CPF or UPF flow, SpyGlass also honors the following SGDC commands:

<a href="#">activity</a>	<a href="#">always_on_buffer</a>	<a href="#">assume_path</a>
<a href="#">cell_hookup</a>	<a href="#">cell_pin_info</a>	<a href="#">cell_tie_class</a>
<a href="#">multivt_lib</a>	<a href="#">non_pd_inputcells</a>	<a href="#">power_data</a>
<a href="#">pg_pins_naming</a>	<a href="#">power_down_sequence</a>	<a href="#">ram_instance</a>
<a href="#">set_case_analysis</a>	<a href="#">switchoff_wrapper_instance</a>	
<a href="#">special_cell</a>	<a href="#">assertion_signal</a>	<a href="#">clock</a>
<a href="#">power_down</a>	<a href="#">ram_switch</a>	

#### **-version <version-number>**

Specify this argument as 1 or 1.0 (for UPF 1.0) and 2 or 2.0 (for UPF 2.0). This would set the default UPF version.

## Rules

The `power_data` constraint is used by the following rules:

<b>SpyGlass Low Power</b>			
LPPLIB04	LPPLIB07	LPLSH01	LPLSH02
LPLSH03	LPLSH04	LPLSH05	LPSVM04A
LPSVM04B	LPSVM04C	LPSVM04D	LPSVM04E
LPSVM17	LPSVM24	LPISO01	LPISO02
LPISO03	LPISO04A	LPISO04B	LPISO04C
LPISO05	LPISO06A	LPISO06B	LPSVM08
LPSVM09	LPSVM010	LPSVM12A	LPSVM12B
LPSVM22	LPSVM26	LPSVM31	LPSVM47
LPSVM51	LPSVM60	LPPLIB10	LPSVM38
LPSVM56B	LPSVM57	LPSVM59	LPCONN01
LPCONN02	LPCONN03	LPPLIB06	LPPLIB15
LPPLIB16	LPPLIB17	LPPLIB18A	LPPLIB18B
LPPLIB19A	LPSUP03	LPPSW01	LPPSW02
LPPSW03	LPPSW04	LP_DECOMPILE_ CONSTR	UPF_lowpower02
UPF_lowpower03	UPF_lowpower04	UPF_lowpower05	UPF_lowpower08
UPF_lowpower11	UPF_lowpower12	UPF_lowpower13	UPF_lowpower14
UPF_lowpower16			

## power\_down

### Purpose

Specifies power-down conditions as used by the LPSVM28 rule.

### Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `power_down` constraint is as follows:

```
current_design <du-name>
  power_down
    [ -domain <pd-name> ]
    -signame <sig-name-list>
    -value <value-list>
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the power-down conditions.

### -domain <pd-name>

Power domain name.

When you specify a power domain name using the `-domain` argument, the LPSVM28 rule of the *SpyGlass Power Verify* solution checks whether the specified pin(s)/net(s) attain the expected value when the specified power domain is switched off. If you do not specify the `-domain` argument of the `power_down` constraint, the LPSVM28 rule checks whether the specified pin(s)/net(s) attain the expected value when all power domains are switched off.

### -signame <sig-name-list>

Space-separated list of pin/net names that are to be checked.

You can use wildcard characters (\*) while specifying the pin/net names.

### -value <value-list>

Space-separated list of expected values.

**NOTE:** For the `-value` argument of the `power_down` constraint, use only 0 or 1 to specify expected values. The constraint check reports any other value specified.

## Examples

The following specification indicates that all pins of instance `top.mod1` are to be expected to attain a value of 1 when the power domain V3 is switched off:

```
power_down -domain V3 -signame "top.mod1.*" -value 1
```

The following specification indicates that all pins of instance `top.I1` with names starting with `vd_in` are to be expected to attain a value of 0 when the power domain V3 is switched off:

```
power_down -domain V3 -signame "top.I1.vd_in*" -value 0
```

You can also specify vector nets using the `power_down` constraint. These can be specified in different ways. For example, a 4-bit input signal can be specified as follows:

```
power_down -domain V3 -signame top.I1.vd_in[0:3] -value 0
```

```
power_down -domain V3 -signame top.I1.vd_in -value 0
```

```
power_down -domain V3 -signame top.I1.vd_in[0] -value 0 -  
signame top.I1.vd_in[1] -value 1 -signame top.I1.vd_in[2] -  
value 0 -signame top.I1.vd_in[3] -value 1
```

## Rules

The `power_down` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

---

LPSVM28

---

## power\_down\_sequence

### Purpose

The `power_down_sequence` constraint is used to specify the registers that should be connected to the specified power-down signal.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `pdsequence`.



## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `power_down_sequence` constraint is as follows:

```
current_design <du-name>
  power_down_sequence
    -signalname <sig-name>
    -instnames <inst-name-list>
    [ -ignorecells <ignorecell-name-list> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the power-down signals.

### -signalname <sig-name>

Name of the power-down signal to be checked.

### -instnames <inst-name-list>

Space-separated list of register cell instance names. You can use wildcards while specifying the cell instance names.

### -ignorecells <ignorecell-name-list>

Space-separated list of cells to be ignored while traversing the fan-out of the specified power-down signal. You can use wildcard characters while specifying cells to be ignored using the `-ignorecells` argument.

## Examples

The following example specifies that signal `vdd` should be directly connected to instances `top.mid.inst1`, `top.mid.inst2` and `top.mid.inst3` only (ignoring instances of cells `isoBuf` and `isoAnd` while traversing the fan-out of the power-down signal):

```
power_down_sequence
  -signalname vdd
  -instnames top.mid.inst1 top.mid.inst2 top.mid.inst3
  -ignorecells isoBuf isoAnd
```

## Rules

The `power_down_sequence` constraint is used by the following rule:

---

### SpyGlass Power Verify Solution

---

LPSVM41

---

## power\_management\_test\_control\_cell

### Purpose

A `power_management_test_control_cell` is used to specify PMUWRs in the design.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `power_management_test_control_cell` constraint is as follows:

```
power_management_test_control_cell
  -name <module-name>
  -control_output <control_pin>
```

### Arguments

**-name <module-name>**

The name of the module to be declared as PMUWR.

**-control\_output <control\_pin>**

Output control pin of PMUWR.

## Rules

The `power_management_test_control_cell` constraint is used by the following rules:

SpyGlass DFT DSM Solution			
SP_02	SP_03	SP_04	SP_05

## power\_management\_unit

### Purpose

A `power_management_unit` is used to specify PMUs in the design.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `power_management_unit` constraint is as follows:

```
power_management_unit
  -name <module-name>
```

### Arguments

**-name <module-name>**

The name of the module to be declared as PMU. This is a mandatory field.

### Rules

The `power_management_unit` constraint is used by the following rules:

SpyGlass DFT DSM Solution		
SP_02	SP_03	SP_04

## power\_rail\_mapping

### Purpose

The `power_rail_mapping` constraint specifies a mapping between the supply rail(s) in design and power rails in the technology library for an instance of the design.

If there are multiple technology libraries, you need to specify a mapping with each library separately.

**NOTE:** *The supply rails in design can be specified by using the [supply](#) constraint.*

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `power_rail_mapping` constraint is as follows:

```
current_design <top-du-name>
  power_rail_mapping
    [ -design <inst-name> ]
    -lib <lib-name>
    -lib_rail <lib-rail-name>
    -design_rail <inst-rail-name>
    -default_lib_rail
```

### Arguments

#### <top-du-name>

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

**-design <inst-name>**

Name of the instance (under the environment *<top-du-name>*) for which you are specifying the instance-to-supply rail mapping. If you do not specify an instance name using the `-design` argument, the specified instance-to-supply rail mapping is applicable to all instances under the environment *<top-du-name>* unless overridden by another `power_rail_mapping` constraint specifying a particular instance.

**NOTE:** *The -design argument is optional.*

**-lib <lib-name>**

Name of library from which some gates are mapped to the gates in instance *<inst-name>*.

**-lib\_rail <lib-rail-name>**

Name of library supply applicable for instance *<inst-name>*.

**-default\_lib\_rail**

The `-default_lib_rail` argument is used to specify default supply rail for the libraries where `default_power_rail` attribute is not specified.

**NOTE:** *You should not use -default\_lib\_rail along with the -lib\_rail argument. This will result in a fatal error generated by the SGDC\_power\_est25 rule.*

**-design\_rail <inst-rail-name>**

Name of the supply rail (defined using the `supply` constraint) that maps to the library supply *<lib-rail-name>* for instance *<inst-name>*.

## Example

Consider a library, `lib1`, which has three power rails, `rail1`, `rail2`, and `rail3`. Also, consider a top-level design, `top`, and an instance, `inst`, instantiated in that design. Design, `top`, has two supply rails, `VDD1` and `VDD2`, specified by using the `supply` constraint.

Now, if you want to map the design rail, `VDD1`, with power rails, `rail1`

and rail2, and map design rail, VDD2, with power rail, rail3, use the `power_rail_mapping` constraint, as shown below:

```
power_rail_mapping -design top -design_rail VDD1 -lib lib1 -
lib_rail rail1
```

```
power_rail_mapping -design top -design_rail VDD1 -lib lib1 -
lib_rail rail2
```

```
power_rail_mapping -design top -design_rail VDD2 -lib lib1 -
lib_rail rail3
```

The above commands connect library rails, rail1 and rail2, to design rail, VDD1, and library rail, rail3, to design rail, VDD2. The power values corresponding to rail1 and rail2 are reported in VDD1. Similarly, power values corresponding to rail3 are reported in VDD2.

If you want to specify a separate mapping for inst, then specify the following commands:

```
power_rail_mapping -design top.inst -design_rail VDD1 lib
lib1 -lib_rail rail1
```

```
power_rail_mapping -design top.inst -design_rail VDD2 lib
lib1 -lib_rail rail3
```

**NOTE:** *If a mapping between library rail and supply rail is not defined, the power corresponding to the library rail is reported in an internally generated supply rail, OTHER\_RAILS.*

## Rule

The `power_rail_mapping` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction solutions		
PEPWR01	PEPWR02	poweraudit

## power\_state

## Purpose

Specifies the legal combinations of domain states (voltage values), that is, those combinations of domain states that can exist at the same time during operation of the design.

**NOTE:** *The voltage specified in the `power_state` constraint is applied for the respective domain. However, if you do not specify a domain in any of the defined power states, the corresponding voltage value is taken from the [voltage\\_domain](#) constraint specification.*

**NOTE:** *For the SpyGlass Power Estimation and SpyGlass Power Reduction solutions, only one power state is honored at a time, currently.*

## Product

SpyGlass Power Verify solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `power_state` constraint is as follows:

```
current_design <du-name>
  power_state
    -name <pd-condition-name>
    -domains <domain-name-list>
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the power state conditions.

**-name <pd-condition-name>**

Name of the power state condition.

**-domains <domain-name-list>**

Domain name@voltage-value. For example:

```
power_state -name LL -domains VA@0.8 VB@0.8
```

**NOTE:** *A domain that is declared as voltage domain (always-on) is always on. However, a*

*domain declared as power domain (switchable) can be either on or off. Each valid state for the full design will have some domains on and some domains off. The power\_state constraint describes one power state. You can use the constraint to specify all power domains that are on in a specific power mode. Each power domain specified by the -domains argument list is powered on, and the other power domains that are not listed are powered off.*

## Rules

The power\_state constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM08	LPSVM09	LPSVM10	LPSVM23
LPSVM47	LPSVM48	LPSVM60	
SpyGlass Power Estimate			
All rules that use voltage information			

## power\_switch

### Purpose

The power\_switch constraint is used to specify the power switches in power domains.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was powerswitch.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the power\_switch constraint is as follows:

```
current_design <du-name>
  power_switch
    -name <name>
```



```

[ -pwROUTpin <out-pin-name> ]
[ -pwrINpin <in-pin-name> ]
[ -en_INV_in <en-in-pin-name> ]
[ -en_INV_in_2 <en-in-pin2-name> ]
[ -en_INV_out <en-out-pin-name> ]
[ -en_BUF_in <en-buf-pin-name> ]
[ -enableVAL <0 | 1> ]
[ -enableVAL_2 <0 | 1> ]
[ -exclude <pin-name-list> ]

```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the power switch.

**-name <name>**

Name of the power switch. You can specify names using wildcards.

**-pwROUTpin <out-pin-name>**

Name of the power-out terminal

**-pwrINpin <in-pin-name>**

Name of the power-in terminal

**-en\_INV\_in <en-in-pin-name>**

Name of the enable input terminal

**-en\_INV\_in\_2 <en-in-pin2-name>**

Name of the enable input terminal

**NOTE:** *This argument is used in case of dual enable power switch.*

**-en\_INV\_out <en-out-pin-name>**

Name of the enable output terminal

**-en\_buf\_in <en-buf-pin-name>**

Name of the buffer enable input terminal.

If you specify the `-en_inv_out` and `-en_buf_in` arguments, the power switch is assumed as dual enable port power switch.

**-enableval <0 | 1>**

Specifies the value of the signal at enable pin. You can specify the argument as 0 (active low) or 1 (active high).

**-enableval\_2 <0 | 1>**

Specifies the value of the signal at the second enable pin. You can specify the argument as 0 (active low) or 1 (active high). This option is used in case of dual enable port power switch.

**-exclude <pin-name-list>**

Specifies a list of supply pins that are not to be checked for power and ground connections.

**NOTE:** *You can specify the same design unit as the power switch for multiple power domains.*

## Rules

The `power_switch` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM06	LPSVM45	LPPLIB17	LPPLIB15

## pr\_safe\_clocks

### Purpose

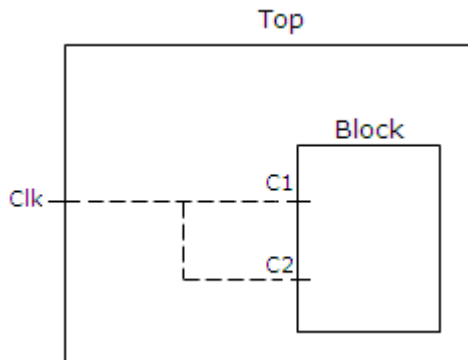
Power reduction rules recommend the RTL changes that do not introduce any clock domain issues in the design. Therefore, the rules need to ensure that every transformation is *safe* from the clock crossing perspective.

A design may contain clocks that are physically different but have the same

domain, frequency, and phase. Such clocks are termed as *safe* clocks.

The `pr_safe_clocks` constraint is used to specify a list of such clocks that can be used by SpyGlass to report power reduction recommendations.

For example, consider the following figure:



In the above figure, when you run SpyGlass for the `Block` block, SpyGlass sees two clocks, `C1` and `C2`. However, these clock nets are connected to the `Clk` clock at the chip level. In this case, you should specify the following command in the SGDC file:

```
pr_safe_clocks -name C1 C2
```

The above command specifies that `C1` and `C2` are actually same and any transformation across them is *safe*.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `pr_safe_clocks` constraint is as follows:

```
pr_safe_clocks
  -name <clk-list>
```

## Arguments

**-name** <clk-list>

List of clocks that are physically different but have the same domain, frequency, and phase

## Rules

The `pr_safe_clocks` constraint is used by the following rules:

---

**SpyGlass Power Estimation and SpyGlass Power Reduction solutions**

---

Rules of SpyGlass Power Reduce

---

## pulldown

### Purpose

The `pulldown` constraint is used to specify the pull-down design units so that various rules can take appropriate actions.

The `Scan_21` rule of the SpyGlass DFT solution uses the `pulldown` constraint to check whether a scan flip-flop exists in the fan-in of the enable pin. The `Tristate_09` rule uses the `pulldown` constraint to check whether the enable pin of the pull-down design unit gets the expected value in the capture mode. The `Topology_12` rule uses the `pulldown` constraint to check whether primary inputs are connected to a pull-down device.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `pullDown`.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `pulldown` constraint is as follows:

```
pulldown
```

```

-name <du-name>
[ -pin <pin-name> ]
[ -value <value> ]

```

**NOTE:** *The pulldown constraint supports wildcard characters.*

## Arguments

### <du-name>

The pull-down design unit (black box) name.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered.

You can specify a single design unit name or a space-separated list of design unit names.

### -pin <pin-name>

(Optional) The enable pin on the pull-down design unit (black box).

You can specify only a single pin name.

If you do not specify the enable pin, the design unit (black box) is always assumed as an enabled pull-down.

### -value <value>

(Optional) The expected value (0, 1, X, or Z) on the enable pin *<pin-name>* under the shift mode.

You need to specify the value only if you have specified the pin name.

## Rules

The pulldown constraint is used by the following rules:

SpyGlass DFT Solution			
Scan_21	Tristate_06	Topology_12	Tristate_09

---

**SpyGlass DFT DSM Solution**

---

All rules

---

## pullup

### Purpose

The `pullup` constraint is used to specify the pull-up design units so that various rules of SpyGlass DFT solution can take appropriate actions.

The `Scan_21` rule uses the `pullup` constraint to check whether a scan flip-flop exists in the fan-in of the enable pin. The `Tristate_09` uses the `pullup` constraint to check whether the enable pin of the design unit gets the expected value in the capture mode. The `Topology_12` rule uses the `pullup` constraint to check whether primary inputs are connected to a pull-up device.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `pullUp`.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `pullup` constraint is as follows:

```
pullup
  -name <du-name>
  [ -pin <pin-name> ]
  [ -value <value> ]
```

**NOTE:** *The `pullup` constraint supports wildcard characters.*

### Arguments

**<du-name>**

The pull-up design unit (black box) name.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered.

You can specify a single design unit name or a space-separated list of design unit names.

#### **-pin <pin-name>**

(Optional) The enable pin on the pull-up design unit (black box).

You can specify only a single pin name.

If you do not specify the enable pin, the design unit (black box) is always assumed as an enabled pull-up.

#### **-value <value>**

(Optional) The expected value (0, 1, X, or Z) on the enable pin *<pin-name>* under the shift mode.

You need to specify the value only if you have specified the pin name.

## Rules

The `pullup` constraint is used by the following rules:

<b>SpyGlass DFT Solution</b>			
Scan_21	Tristate_06	Topology_12	Tristate_09
<b>SpyGlass DFT DSM Solution</b>			
All rules			

## qualifier

### Purpose

Specifies a qualifier for synchronizing a clock domain crossing by the *Qualifier Synchronization* scheme.

## Product

SpyGlass CDC solution

## Syntax

The syntax of the `qualifier` constraint is as follows:

### Usage 1:

```
qualifier
  -name <qualifier-name>
  -from_clk <src_clk_list>
  -to_clk <dest_clk_list>
  -type <type> -strict
  [ -crossing ]
  [ -ignore ]
  [ -reset ]
```

### Usage 2:

```
qualifier
  -name <qualifier-name>
  -from_domain <src_domain_list>
  -to_domain <dest_domain_list>
  -type <type> -strict
  [ -crossing ]
  [ -ignore ]
  [ -reset ]
```

### Usage 3:

```
qualifier
  -name <qualifier-name>
  -ignore
```

### Usage 4:



```

qualifier
  -name <qualifier-name>
  -from_obj <src-obj-list>
  -to_obj <dest-obj-list>
  -ignore

```

**Usage 5:**

```

qualifier
  -enable <hierarchical-verilog-expression>
  -from_obj <src-obj-list>
  -to_obj <dest-obj-list>

```

**Usage 6:**

```

qualifier
  -name <qualifier-name>
  -from_obj <src-obj-list>
  -to_obj <dest-obj-list>

```

**Usage 7:**

```

qualifier
  -name <qualifier-name>
  -rdc
  [-from_obj <src-obj-list> -to_obj <dest-obj-list>]
  [-to_clk <dest-clock-list>]
  [-to_domain <dest-domain-list>]
  [-from_rst <source_reset_list> -to_rst <dest_reset_list>]

```

**Usage 8:**

```

qualifier
  [-src_qual <src_qual_name>]
  [-dest_qual <dest_qual_name>]
  [-dest_qual_depth <dest_qual_depth>]
  [-src_qual_depth <src_qual_depth>]
  -from_clk <src_clk_list>

```

```
-to_clk <des_clk_list>
```

**Usage 9:**

qualifier

```
[-src_qual <src_qual_name>]
[-dest_qual <dest_qual_name>]
-from_domain <src_domain_list>
-to_domain <des_domain_list>
```

**Usage 10:**

qualifier

```
[-src_qual <src_qual_name>]
[-dest_qual <dest_qual_name>]
-from_obj <src_obj_of_crossing>
-to_obj <des_obj_of_crossing>
```

## Arguments

**-name <qualifier-name>**

Name of the qualifier (port, net, hierarchical terminal). You can also specify the output of the destination in a control synchronized crossing to this argument.

You can also specify a pin name (of a master design unit) in the `<du-name>/<pin-name>` format.

You can also use wildcard characters ('\*' and '?') while specifying names.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression with the hierarchies. For example, the `top.*.n1` expression is matched to `top.u1.n1` and `top.u1.u2.n1`. By default, the qualifier constraint matches only `top.u1.n1`.

Setting the value of the `hier_wild_card` parameter to `yes`, run-time performance of the qualifier constraint is impacted.

**-from\_clk <src-clk-list>**

Space-separated list of clock names that drive the source of a crossing to be synchronized by the qualifier specified by *-name <qualifier-name>*.

Specify this argument with *-to\_clk <dest-clk-list>*.

See [Example 1](#).

**NOTE:** *While specifying the list of clock names, either specify individual clock names in double quotes or do not use double quotes at all. See [Example 8](#).*

**-to\_clk <dest-clk-list>**

Space-separated list of clock names that drive the destination of a crossing to be synchronized by the qualifier specified by *-name <qualifier-name>*.

Specify this argument with *-from\_clk <src-clk-list>*.

See [Example 1](#).

**NOTE:** *While specifying the list of clock names, either specify individual clock names in double quotes or do not use double quotes at all. See [Example 8](#).*

**-from\_domain <src-domain-list>**

Space-separated list of domain names that drive the source of a crossing to be synchronized by the qualifier specified by *-name <qualifier-name>*.

Specify this argument with *-to\_domain <dest-domain-list>*.

See [Example 2](#).

**-to\_domain <dest-domain-list>**

Space-separated list of domain names that drive the destination of a crossing to be synchronized by the qualifier specified by *-name <qualifier-name>*.

Specify this argument with *-from\_domain <src-domain-list>*.

See [Example 2](#).

**-from\_obj <src-obj-list>**

Space-separated list of objects (hierarchical net, pin, or port) of a source such that the crossings containing such sources are not synchronized by the qualifier specified by *-name <qualifier-name>*.

Specify this argument with *-to\_obj <dest-obj-list>* and *-ignore*.

See [Example 3](#).

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

#### **-to\_obj <dest-obj-list>**

Space-separated list of objects (hierarchical net, pin, or port) of a destination such that the crossings containing such destinations are not synchronized by the qualifier specified by `-name <qualifier-name>`.

For library cells and black boxes, specify their input pin or input net names to this argument.

Specify this argument with `-from_obj <src-obj-list>` and `-ignore`.

See [Example 3](#).

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

#### **-type <type>**

Can be `des` if the qualifier reaches to the destination of a crossing or it can be `src` if it reaches to the source of a crossing.

By default, the type is `des`.

See [Example 4](#) and [Example 5](#).

#### **-strict**

Whether to allow strict synchronization checks for qualifier synchronization scheme. Specify `-strict` option to allow these checks.

Qualifier propagation stops on arithmetic macros and memories when the `-strict` option is specified in the `qualifier` constraint.

**NOTE:** Refer to the *Qualifier Synchronization Scheme* section of the *SpyGlass CDC Rules Reference Guide* for details.

See [Example 6](#).

#### **-crossing**

Specify this argument to consider a crossing as synchronized if the specified qualifier defines a crossing output that contains only a single destination flip-flop.

**-ignore**

Marks the qualifier specified by the *-name <qualifier-name>* argument as an invalid qualifier for the specified objects, domains, or clock crossings in the design.

This argument is useful when SpyGlass infers a qualifier in the design but you do not want SpyGlass to consider it as a valid qualifier.

See [Example 1](#), [Example 2](#), [Example 3](#), and [Example 4](#).

**NOTE:** *This argument should not be used with the *-type <type>*, *-strict*, and *-crossing* argument of the qualifier constraint.*

**-enable <hierarchical-verilog-expression>**

Specifies the expression representing an enable condition of a qualifier. The expression uses Verilog hierarchical names and operators.

The names specified to this argument should be valid design objects and should have the same domain as that of *-to\_obj <dest-obj-list>* in their fan-in.

**NOTE:** *It is mandatory to specify the both the *-to\_obj <dest-obj-list>* and *-from\_obj <src-obj-list>* arguments with the *-enable* argument.*

The *-enable* argument supports hierarchical scoping. If any qualifier *-enable* is provided at block level *sgdc* and the *sgdc* is used in top-level design, the qualifier constraint is migrated to the top and therefore the signals are migrated to the top.

The following example shows the usage of this argument:

```
qualifier -enable "top.en_out" -from_obj top.src.q -to_obj
top.des.q
```

**-reset**

Specify this argument to consider a crossing as synchronized if the specified qualifier is present on a reset path.

For example, specify the following constraint to synchronize a clock domain crossing only on reset paths.

```
qualifier -name qual -from_clk clk1 -to_clk clk2 -reset
```

**-rdc**

Specify this argument to consider a *rdc* crossing as synchronized if the

specified qualifier is present on a data or control path.

For example, specify the following constraint to synchronize a reset domain crossing.

```
qualifier -name qual -from_rst r1 -to_rst r2 -rdc
```

### **-src\_qual**

Specifies name of the valid qualifier (port, net, hierarchical terminal), which when reaches the source of the crossing at the enable pin, or the select pin of the recirculation multiplexer at that source, or enable pin of the clock-gating cell driving the clock pin of the source can synchronize the data crossing.

See [Example 9](#), [Example 10](#), and [Example 11](#)

**NOTE:** *When both `src_qual` and `dest_qual` are specified, then both of them should satisfy their individual criteria for a data crossing to be considered as synchronized.*

### **-dest\_qual**

Use this argument to specify the name of the valid qualifier (port, net, hierarchical terminal), which can block the source before reaching the destination to synchronize the data crossing.

See [Example 9](#), [Example 10](#), and [Example 11](#)

**NOTE:** *The `-src_qual` and `-dest_qual` arguments cannot be specified with the following arguments of the qualifier constraint: `-name`, `-enable`, `-from_rst`, `-to_rst`, `-thru_obj`, `-type`, `-crossing`, `-ignore`, and `-rdc`.*

### **-src\_stable**

Use this argument to specify the source stability criteria. When this argument is specified, mux-based synchronization does not check for destination domain signal on other data pins of mux and synchronization analysis reports it as synchronized.

The `src_qual` and the `dest_qual` are both mandatory arguments with the `src_stable` argument. For a qualifier reaching mux select pin, use the `dest_qual` argument to specify the destination qualifier.

### **-src\_qual\_depth**

Specifies the qualifier depth i.e. the number of sequential elements allowed from qualifier declaration till source of crossings.

See [Example 9](#), [Example 10](#), and [Example 11](#)

### -dest\_qual\_depth

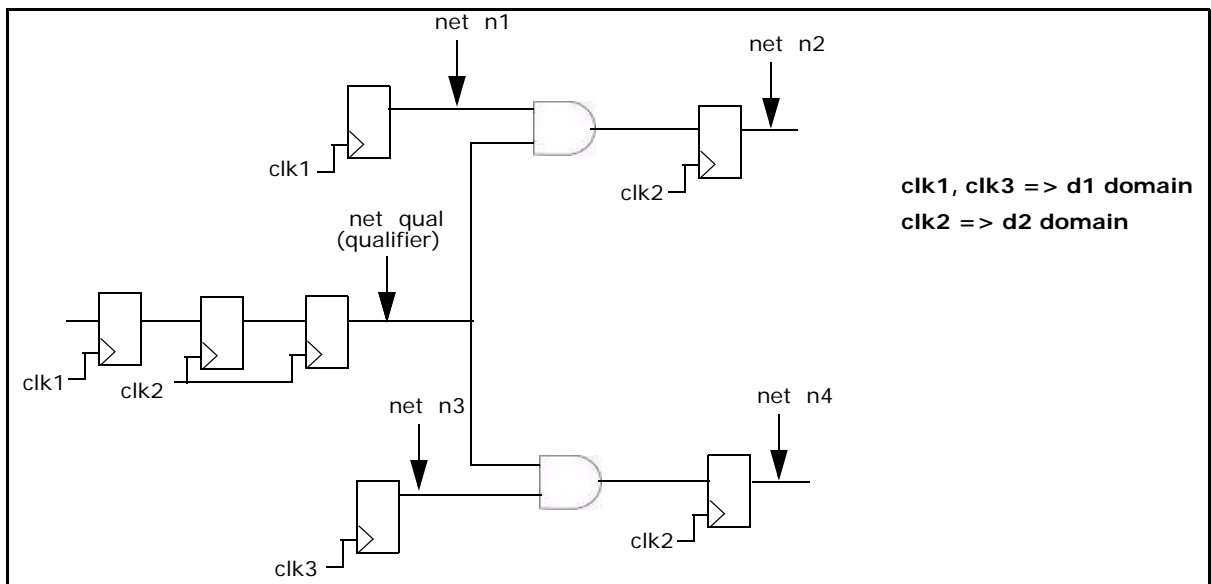
Specifies the qualifier depth i.e. the number of sequential elements allowed from qualifier declaration till destination of crossings.

See [Example 9](#), [Example 10](#), and [Example 11](#)

**NOTE:** When `cdc_qualifier_depth` is not specified, then the rule will give a sanity message that use `cdc_qualifier_depth` to modify depth and `depth = 3` has been used in current analysis.

## Examples

Consider the scenario shown in the following figure:



**FIGURE 42.** Specifying crossings synchronized by a qualifier

In the above scenario, you can specify the crossings that should or should not be synchronized by the `qual` qualifier. This is described in the examples below.

### Example 1

In the scenario shown in [Figure 42](#), to enable `qual` to synchronize the crossing between the source driven by the `clk1` clock and destination driven by the `clk2` clock, specify the following constraint:

```
qualifier -name qual -from_clk clk1 -to_clk clk2
```

To ignore this crossing such that `qual` does not synchronize this crossing, specify `-ignore` to this constraint, as shown below:

```
qualifier -name qual -from_clk clk1 -to_clk clk2 -ignore
```

### Example 2

In the scenario shown in [Figure 42](#), to enable `qual` to synchronize the crossing between the source driven by the `d1` domain clock and destination driven by the `d2` domain clock, specify the following constraint:

```
qualifier -name qual -from_domain d1 -to_domain d2
```

To ignore this crossing such that `qual` does not synchronize this crossing, specify `-ignore` to this constraint, as shown below:

```
qualifier -name qual -from_domain d1 -to_domain d2 -ignore
```

### Example 3

In the scenario shown in [Figure 42](#), if you do not want `qual` to synchronize the crossing between the source containing the `n1` net and a destination containing the `n2` net, specify the following constraint:

```
qualifier -name qual -from_obj n1 -to_obj n2 -ignore
```

### Example 4

Consider the following constraint:

```
qualifier -name qual -from_clk c1 -to_clk c2 c3 -type src
```

The above constraint specifies a qualifier for synchronization of the



crossings where the source is driven by `c1` and destination by `c2` and `c3`. The type `src` specifies that `qual` is reaching the source of the crossings.

### Example 5

Consider the following constraint:

```
qualifier -name qual -from_domain d1 d2 -to_domain d3 d4
-type des
```

The above constraint specifies a qualifier for synchronization of the crossings where the source is driven by clocks from domain `d1` and `d2` and destination by clocks of domain `d3` and `d4`. The type `des` specifies that `qual` is reaching the destination of the crossings.

### Example 6

Consider the following constraint:

```
qualifier -name qual -from_clk c1 c2 -to_clk c3 -strict
```

The above constraint specifies a qualifier for synchronization of the crossings where the source is driven by `c1` and `c2` and destination by `c3` and `qual` is reaching the destination of the crossings. In addition, the constraint specifies that strict synchronization checks should be allowed for qualifier synchronization scheme.

### Example 7

The following constraint shows the usage of the `-enable` argument with respect to the scenario shown in [Figure 42](#):

```
qualifier -enable "qual" -from_obj "n1" -to_obj "n2"
```

### Example 8

The following example shows the correct way to specify clock names in `-from_clk` and `-to_clk` where either individual clock names are specified in double quotes or no double quotes are used at all:

```
qualifier -name "xyz" -from_clk "clk1" "clk2" "clk3" -to_clk
```

```
"ck1" "ck2"
qualifier -name "xyz" -from_clk clk1 clk2 clk3 -to_clk ck1
ck2
```

The following example shows the incorrect way of specifying clock names in `-from_clk` and `-to_clk`:

```
qualifier -name "xyz" -from_clk "clk1 clk2 clk3" -to_clk
"ck1 ck2"
```

### Example 9

The following example shows the qualifier constraint specification for synchronizing a crossing from the `clk1` clock to the `clk2` clock using the source qualifier.

```
qualifier -src_qual squal -from_clk clk1 -to_clk clk2
```

This constraint specification synchronizes the crossing if the `squal` source qualifier is reaching the source of the crossing at the enable pin or the select pin of the recirculation multiplexer at that source or enable pin of the clock-gating cell driving the clock pin of the source.

In the following schematic, the `squal` source qualifier is reaching the enable pin of the source of the crossing `src1_reg`.

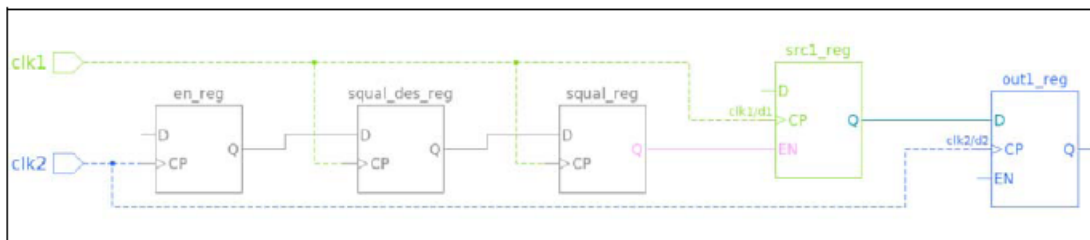


FIGURE 43.

### Example 10

The following example shows the qualifier constraint specification for synchronizing a crossing from the `clk1` clock to the `clk2` clock.

```
qualifier -src_qual squal -dest_qual dqual -from_clk clk1
```

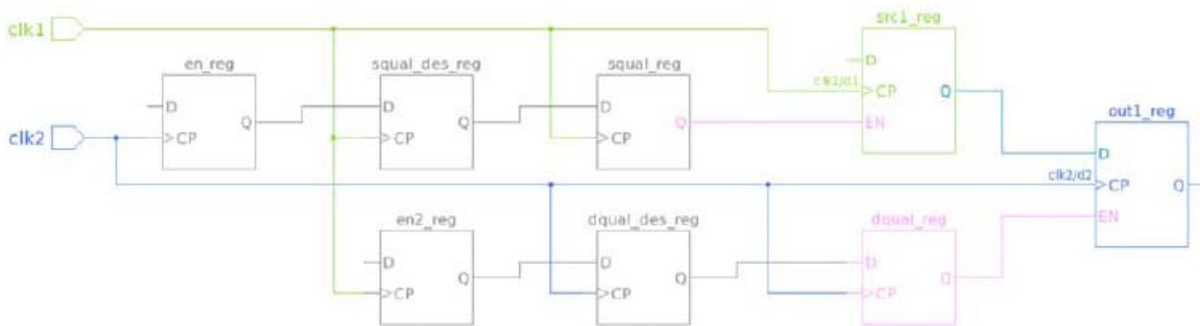
## SpyGlass Design Constraints

```
-to_clk clk2
```

This constraint specification synchronizes the crossing if:

- The `squal` source qualifier is reaching the source of that crossing at the enable pin or select pin of the recirculation multiplexer at that source or enable pin of the clock-gating cell driving the clock pin of the source.
- The `dqual` destination qualifier is blocking the source before reaching the destination.

In the following schematic, the `squal` source qualifier is reaching the enable pin of the source of the crossing `src1_reg` and the `dqual` destination qualifier is reaching the enable pin of the destination flip-flop `out1_reg`.



**FIGURE 44.**

### Example 11

The following example shows the qualifier constraint specification for synchronizing a crossing from the `clk1` clock to the `clk2` clock using the destination qualifier with permissible sequential depth of 2.

```
qualifier -dest_qual qual -from_clk clk1 -to_clk clk2
-dest_qual_depth 2
```

In the following schematic, the `qual` destination qualifier is qualifying the crossing through depth 2.

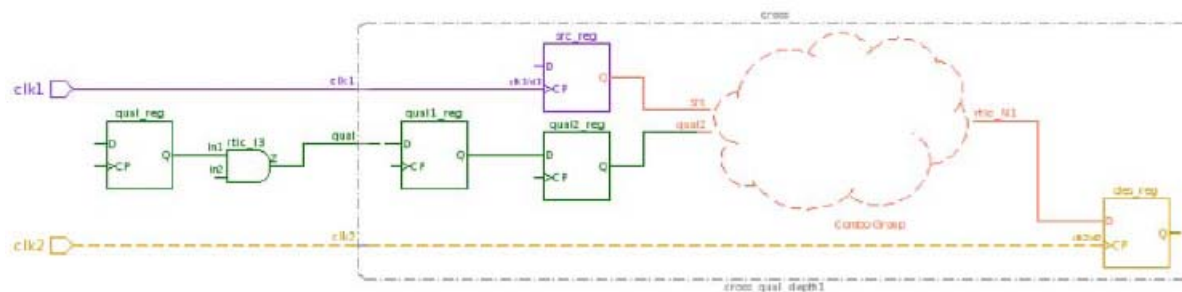


FIGURE 45.

## Rules

The `qualifier` constraint is used by the following rules:

SpyGlass CDC Solution			
Clock_sync08	Clock_sync08a	Clock_sync03a	Clock_sync03b
Clock_sync09	Ac_cdc01a	Ac_cdc01b	Ac_cdc01c
Ac_cdc08	Propagate_Clocks	Ac_conv02	Ac_sync01
Ac_sync02	Ac_unsync01	Ac_unsync02	Ac_conv04
Ar_cross_analysis01	Ar_resetcross01		

## quasi\_static

### For SpyGlass TXV solution

#### Purpose

There are some primary inputs, flip-flops, nets, or terms present in the design, which may change briefly at the start but assume a static value of 0 or 1 for the rest of the circuit operation.

You can specify the primary inputs, flip-flops, nets, or terms as static

signals using the `quasi_static` constraint to skip the verification of paths that involve such signals.

## Product

SpyGlass TXV solution

## Syntax

The syntax to specify the `quasi_static` constraint is as follows:

```
quasi_static -name <sig-name-list>
```

## Arguments

**-name <sig-name-list>**

List of hierarchical net/term names of the static signal

## Example

If A is a static signal then you need to specify the `quasi_static` constraint in the `.sgdc` file as follows:

```
quasi_static -name top.A
```

## Note

- Wildcard characters "\*" and "?" can be used with the `quasi_static` constraint.
- If the hierarchical net/term name of a static signal specified with the `-name` argument does not exist as a net/term in the current design, `SGDC_quasi_static01rule` reports a violation.

## Rules

The `quasi_static` constraint is used by the following rules:

SpyGlass TXV Solution		
FP_Pass_Verif05	FP_Skip_Verif02	MCP_Pass_Verif03
MCP_As_FP_Verif07	MCP_Skip_Verif03	

## For SpyGlass CDC solution

### Purpose

The `quasi_static` constraint is used to specify signals whose value is predominantly static. Clock domain crossings containing such static signals are reported as synchronized under the *Delay Signals Synchronization Scheme*.

You can use this constraint if any of the following conditions hold true:

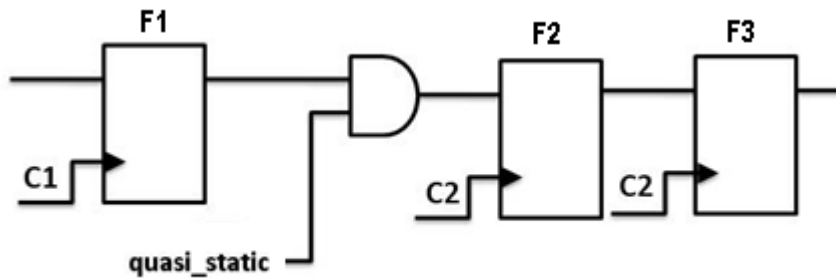
- If the value of signals is predominantly static
- If a clock on the destination flip-flop is stopped
- If a reset is active on a destination flip-flop
- If the logic in the clock domain crossing path is not sensitive to any metastability issue

Signals specified by this constraint are propagated beyond buffers, inverters, and transparent latches. In addition, a signal is propagated if all inputs of an RTL gate or all related inputs of a black box that has the `assume_path` constraint set on it are quasi-static.

If all inputs of a combinational logic present in a crossing path are quasi-static except the source path, the logic is considered as safe even when the `allow_combo_logic` parameter is set to `no`. Therefore, the logic is allowed while checking in the *Conventional Multi-Flop Synchronization Scheme*.

**NOTE:** *The `quasi_static` constraint impacts the behavior of the `Reset_check07` rule. Refer to the help on `Reset_check07` rule for details.*

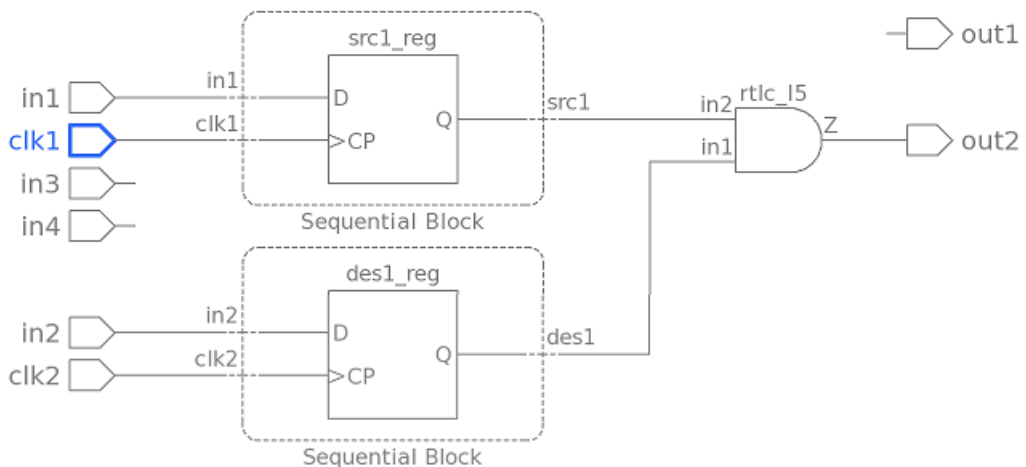
For example, consider the following figure:



(allow\_combo\_logic parameter set to no)

In the above figure, crossing between the C1 and C2 clocks is reported as synchronized by using the *Conventional Multi-Flop Synchronization Scheme*.

Quasi static signals are auto inferred when a flop, latch, or CGC is receiving a constant clock. SpyGlass CDC infers the output net of such flop, latch, or CGC as quasi\_static. For example, consider the following schematic:



In the above schematic, `clk1` is held at constant value (`set_case_analysis -name clk1 value 1`), and the `src1_reg` flop is receiving constant clock. Therefore, SpyGlass CDC infers the flop

output `src1` as quasi static.

**NOTE:** *Please note the following points:*

- Quasi-static signals are not propagated beyond flip-flops, normal latches, and pure black boxes.
- This constraint is only applicable to Clock Domain Crossing (CDC) data paths. It is NOT applicable to either clock paths or side paths/inputs to CDC data paths.
- It is recommended that you use this constraint very carefully. If not used carefully, this constraint may mask some of the real design issues.
- If a crossing contains signals other than static signals, it is recommended that you use the [cdc\\_false\\_path](#) constraint to specify false paths.
- By default, the `quasi_static` constraint does not propagate through flops/sequential elements. Use the `num_quasi_seq_elem` parameter to specify the depth of flops/sequential element up to which the `quasi_static` constraint should propagate. Set the `num_quasi_seq_elem` parameter to -1 to propagate the constraint through infinite number of flops/sequential elements.

## Product

SpyGlass CDC solution

## Syntax

The syntax to specify the `quasi_static` constraint is as follows:

```
current_design <du-name>  
    quasi_static -name <sig-name-list>
```

## Arguments

**<du-name>**

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs)



**-name <sig-name-list>**

The list of port names, hierarchical net/term names, and/or hierarchical pin names of predominantly static signals.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression with the hierarchies. For example, the `top.*.n1` expression is matched to `top.u1.n1` and `top.u1.u2.n1`. By default, the `quasi_static` constraint matches only `top.u1.n1`.

Setting the value of the `hier_wild_card` parameter to `yes` runtime performance of the `quasi_static` constraint is impacted.

**Example**

You can specify bit-selects and part-selects with the `quasi_static` constraint, as in the following example:

```
current_design top
  quasi_static -name "top.I1.NetA2B[0]"
  quasi_static -name "top.I1.NetA2B[1]"
  clock -name clkA -domain A -value rtz
  clock -name clkB -domain B -value rtz
```

You can also use regular expressions and wildcard characters while specifying signal names. For example, consider that your design contains buses, such as `netBus1[0:7]` and `netBus2[0:15]`. In this case, if you want to specify both these buses, specify the following constraint:

```
quasi_static -name "top.netBus*"
```

For details on using regular expressions and wildcard characters, refer to the *Using Regular Expressions and Wildcard Characters* topic of the *Atrenta Console User Guide*.

**Rules**

The `quasi_static` constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Clock_info05b	Reset_check07	Clock_sync03a	Clock_sync03b
Clock_sync05	Clock_sync06	Clock_sync08	Clock_sync08a

Clock_sync09	Ac_cdc01a	Ac_cdc01b	Ac_cdc01c
Ac_cdc08	Ac_handshake01	Ac_handshake02	Ac_conv01
Ac_conv02	Ac_conv03	Ac_sync01	Ac_sync02
Ac_unsync01	Ac_unsync02	Reset_sync02	Setup_quasi_static 01
Ac_coherency06	Propagate_Clocks	Ar_converge01	

## For SpyGlass Power Family

### Purpose

Specifies quasi static signals in a design.

Quasi static signals are the signals that are constant for most of the time and toggle for the remaining time. This feature of these signals can be used to find clock gating opportunities in a design.

### Product

SpyGlass Power Family

### Syntax

The syntax of the quasi\_static signal is as follows:

```
current_design <du-name>
  quasi_static -name <sig-name-list>
  [ -value <value> ]
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in *<entity-name>.<arch-name>* format (for VHDL designs)

#### -name <signal-name>

Name of the signal that is considered as quasi static.

The signal name can be the name of a port, hierarchical net/term, or hierarchical pin.

#### **-value <value>**

The static value on the quasi static signal.

If you do not specify this argument, the value is picked from the simulation file.

## Rules

The `quasi_static` constraint is used by the following rules:

SpyGlass Power Family			
PEPWR20	PEPWR21	PEPWR22	PEPWR23
PEPWR24	PEPWR25	PEPWR33	

## quasi\_static\_style

### Purpose

Specifies a criterion based on which SpyGlass infers quasi-static signals in a design.

If a signal matches the specified criteria, SpyGlass infers it as a quasi-static signal.

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `quasi_static_style` constraint is as follows:

```
current_design <du-name>
  quasi_static_style
    [ -min_seq_fanouts <fanout-count> ]
```

```
[ -min_domain_fanouts <domain-count> ]  
[ -names <pattern-list> ]  
[ -check_all_signals ]  
[ -report_full_count ]
```

## Arguments

### **-min\_seq\_fanouts <fanout-count>**

Specifies the minimum number of sequential elements a signal must drive.  
The default value is 10.

### **-min\_domain\_fanouts <domain-count>**

Specifies the minimum number of clock-domains in the fan-out cone of a signal.

The default value is 1.

### **-names <pattern-list>**

Specifies a naming pattern for signals.

If a signal matches the specified pattern, only then SpyGlass checks if that signal is a quasi-static signal based on the criteria specified by other arguments of this constraint.

For details, see [Example 1](#), [Example 2](#), and [Example 3](#).

**NOTE:** *You can use wildcard expressions while specifying patterns.*

### **-check\_all\_signals**

Specifies that all the output of sequential elements, output pins of black boxes, or input ports should be checked to see if they are quasi-static signals.

By default, SpyGlass considers only the sources of clock-domain crossings to detect quasi-static signals.

### **-report\_full\_count**

Specify this argument to report the complete sequential fan-out count and domain count of the inferred `quasi_static` signal in the spreadsheet

report.

## Examples

### Example 1

Consider the following example:

```
quasi_static_style -names "cfg*" "*_cfg" "*_stable?"
```

When you specify the above constraint, SpyGlass infers the signals that match all the following criteria:

- The signal is the source of a clock-domain crossing.
- The signal name matches any of the following criteria:
  - The name starts with the string `cfg`.
  - The name ends with `_cfg`.
  - The name is succeeded by a character at the end of `_stable`.
- The fan-out count of the signal is at least 10.

**NOTE:** *In the absence of the `-min_seq_fanouts <fanout-count>` argument, SpyGlass considers the default fan-out count as 10.*

- The domain count of the signal is at least 1.

**NOTE:** *In the absence of the `-min_domain_fanouts <domain-count>` argument, SpyGlass considers the default domain count as 1.*

### Example 2

Consider the following example:

```
quasi_static_style -names "*cfg*" -min_seq_fanouts 20  
-check_all_signals
```

When you specify the above constraint, SpyGlass infers the signals that match all the following criteria:

- The signal name contains the string `cfg`.
- The fan-out count of the signal is at least 20.
- The domain count of the signal is at least 1.

**NOTE:** *In the absence of the `-min_domain_fanouts <domain-count>` argument, SpyGlass considers the default domain count as 1.*

### Example 3

Consider the following example:

```
quasi_static_style -names "cfg" -min_seq_fanouts 20
-min_domain_fanouts 3
```

When you specify the above constraint, SpyGlass infers the signals that match all the following criteria:

- The signal is the source of a clock-domain crossing.
- The signal name is `cfg`, such as `top.SB1.cfg` and `top.cfg`.
- The fan-out count of the signal is at least 20.
- The domain count of the signal is at least 3.

### Example 4

Consider the following example:

```
quasi_static_style -min_seq_fanouts 20 -min_domain_fanouts 3
```

When you specify the above constraint, SpyGlass infers the signals that match all the following criteria:

- The fan-out count of the signal is at least 20.
- The domain count of the signal is at least 3.

## Rules

The `quasi_static_style` constraint is used by the following rules:

---

#### SpyGlass CDC Solution

---

Setup\_quasi\_static01 SGDC\_quasi\_static\_style01 SGDC\_quasi\_static\_style02

---

## ram\_instance

## Purpose

Specifies RAM switch instance-to-RAM instance connections.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `raminstance`.*

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `ram_instance` constraint is as follows:

```
current_design <du-name>
  ram_instance
    -switch_instance <swt-inst-name>
    -ram_instance <ram-inst-name>
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the connection.

**-switch\_instance <swt-inst-name>**

Name of the RAM switch instance.

**-ram\_instance <ram-inst-name>**

Name of the RAM instance.

## Rules

The `ram_instance` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

LPPLIB12

---

## ram\_switch

## Purpose

Specifies the RAM switches as checked by the LPSVM46 and LPPLIB12 rules.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was ramswitch.*

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the ram\_switch constraint is as follows:

```
current_design <du-name>
  ram_switch
    -switch_name <swt-name>
    -switch_enable_pin <swt-en-pin-name>
    [ -switch_vss_outpin <swt-vss-outpin-name> ]
    -memory_cellnames <mem-cell-name-list>
    [ -memory_vss_pin <mem-vss-pin-name> ]
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the RAM switch.

**-switch\_name <swt-name>**

Name of the RAM switch

**-switch\_enable\_pin <swt-en-pin-name>**

Name of the enable pin of the RAM switch.

**-switch\_vss\_outpin <swt-vss-outpin-name>**

Name of the VSS pin of the RAM switch.

**-memory\_cellnames <mem-cell-name-list>**

Space-separated cell name list of memory cells



---

**SpyGlass Design Constraints**

**-memory\_vss\_pin** <mem-vss-pin-name>

Name of the memory cell VSS pin.

## Rules

The `ram_switch` constraint is used by the following rules:

---

**SpyGlass Power Verify Solution**

---

LPPLIB12	LPSVM46
----------	---------

---

## rdc\_false\_path

### Purpose

**NOTE:** *This constraint is deprecated. Use the [reset\\_filter\\_path](#) constraint instead of this constraint.*

The `rdc_false_path` constraint is used to specify false paths so that reset crossings along these paths are ignored from rule checking.

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `rdc_false_path` constraint is as follows:

```
current_design <du-name>
rdc_false_path
  -from_rst <frm-rst-list>
  -to_rst <to-rst-list>
  -clock <clk-obj-list>
  -from_obj <from-obj-list>
  -to_obj <to-obj-list>
  -type <rdc | sync | deassert>
```

### Arguments

**-from\_rst <frm-rst-list>**

Space-separated list of objects (hierarchical net, pin, or port) of a source such that the reset crossings containing such sources are not reported by *Ar\_resetcross01* rule.

**-to\_rst <to-rst-list>**

Space-separated list of objects (hierarchical net, pin, or port) of a destination such that the reset crossings containing such destinations are not reported by *Ar\_resetcross01* rule.

**-clock <clk-obj-list>**

Space-separated list of source or destination clocks.

**-from\_obj <from-obj-list>**

Space-separated list of hierarchical nets, pins, or ports of the source in the reset crossing so that the reset crossings containing such sources are not reported by the *Ar\_resetcross01* rule.

**-to\_obj <to-obj-list>**

Space-separated list of hierarchical nets, pins, or ports of the destination in the reset crossing so that the reset crossings containing such destinations are not reported by the *Ar\_resetcross01* rule.

**-type <rdc | sync | deassert>**

(Optional) The default value is `rdc`. This value is for backward compatibility.

Set this argument to `deassert` to filter the *Ar\_asyncdeassert01* and *Ar\_syncdeassert01* rule violations reported for the reset paths specified by this constraint.

Set this argument to `sync` to filter the *Ar\_unsync01* and *Ar\_sync01* rule violations reported for the reset paths specified by this constraint.

**NOTE:** *If you specify the `sync` or `deassert` value to this argument then you can specify only `-from_rst` and `-clock` arguments to the `reset_filter_path` constraint.*

## Examples

### Example 1

Consider the following *Ar\_resetcross01* spreadsheet showing violations related to invalid crossings:

A	B	C	D	E	F	G
ID	SOURCE	SOURCE RESET	DEST.	DEST. RESET	CLOCKS	WAIVED
<a href="#">2C</a>	top.F1.q	top.r1	top.F2.q	top.r2	top.c	No
<a href="#">2D</a>	top.F3.q	top.r2	top.F4.q	top.r1	top.c	No
<a href="#">33</a>	top1.F7.q	top1.r1	top1.F8.q	top1.r2	top1.c	No
<a href="#">39</a>	top2.F3.q	top2.r1	top2.F4.q	top2.r2	top2.c	No
<a href="#">3A</a>	top2.F5.q	top2.r1	top2.F6.q	top2.r2	top2.c	No

**FIGURE 46.**

From the above set of violations, if you do not want to report the reset crossings between the top.r1 and top.r2 resets, specify the following constraint:

```
rdc_false_path -from_rst r1 -to_rst r2
```

After specifying the above constraint, the violations reported in the cells 2C and 33 in [Figure 46](#) are not reported.

### Example 2

To suppress the *Ar\_resecross01* violations for the reset crossings between r1 to r2, r1 to r3, and r1 to r4 resets, specify the following constraint:

```
rdc_false_path -from_rst r1 -to_rst r2 r3 r4
```

## Rules

The `rdc_false_path` constraint is used by the following rules:

SpyGlass CDC Solution	
Ar_resetcross01	Ar_resetcross_matrix01

## ref\_power\_data

### Purpose

The `ref_power_data` constraint is used to specify values for reference power numbers in the Correlation View of the Power Explorer.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `ref_power_data` constraint is as follows:

```
current_design <du-name>
ref_power_data
  -type <component-type>
  -<power-type> <value>
```

### Arguments

#### **-type <component-type>**

The type can be total, combinational, sequential, blackbox, others, iopad, memory, clock, or megacell.

#### **-<power-type> <value>**

This argument can be leakage, internal, switching, or total. The value can be specified in floating point format and not in scientific format.

### Examples

The following SGDC specification defines combinational, sequential, memory, and total power attributes and their values. These values are populated in the SpyGlass Power Reference Matrix.

```
ref_power_data -type combinational -leakage 0.001
ref_power_data -type sequential -internal 0.008
```

```
ref_power_data -type memory -total 0.0002  
ref_power_data -type total -total 0.10
```

## Rules

The `ref_power_data` constraint is used by the following rules:

---

**SpyGlass Power Estimation and SpyGlass Power Reduction Solutions**

---

PEPWR01

---

PEPWR02

---

## reference\_toplevel\_isolation\_signal

### Purpose

The `reference_toplevel_isolation_signal` constraint is used to specify reference top-level isolation signal at SoC level.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `reference_toplevel_isolation_signal` constraint is as follows:

```
reference_toplevel_isolation_signal  
-name <isolation-control-signal-name-at-soc>  
-supply <supply-name>
```

### Arguments

**-name <isolation-control-signal-name-at-soc>**

Use this argument to specify the name of the correct isolation enable signal at the SoC level.

**-supply <supply-name>**

Use this argument to specify the correct supply of the driver of isolation data pin. This is matched against the actual supply of the driver of the isolation cell data pin.

### Examples

The following SGDC specification defines the `top.isol` signal as a reference top-level isolation signal for supply `VDD1`.

```
reference_toplevel_isolation_signal -name top.isol -supply  
VDD1
```

## Rules

The `reference_toplevel_isolation_signal` constraint is used by the following rules:

---

SpyGlass Power Verify Solutions	
LPSVM22	LPISO01

---

## repeater

### Purpose

Specifies the name of repeater modules or library cells inserted between sequential elements to meet timing requirements of a design.

### Product

SpyGlass CDC solutions

### Syntax

The syntax to specify the `repeater` constraint is as follows:

```
repeater -names <object-names>
```

### Arguments

**-names <object-names>**

Space-separated list of repeaters.

**NOTE:** Set the `hier_wild_card` parameter to `yes` to match the expression specified in this argument with hierarchies.

### Examples

#### Example 1

The following constraint specifies two repeater modules, MOD1 and MOD2:



```
repeater -names MOD1 MOD2
```

### Example 2

The following example uses wildcard expression to indicate that all instances starting with the string Rep should be considered as repeaters:

```
repeater -names "Rep*"
```

## Rules

The `repeater` constraint is used by the following rules:

---

**SpyGlass CDC Solution**

---

Ac\_repeater01

---

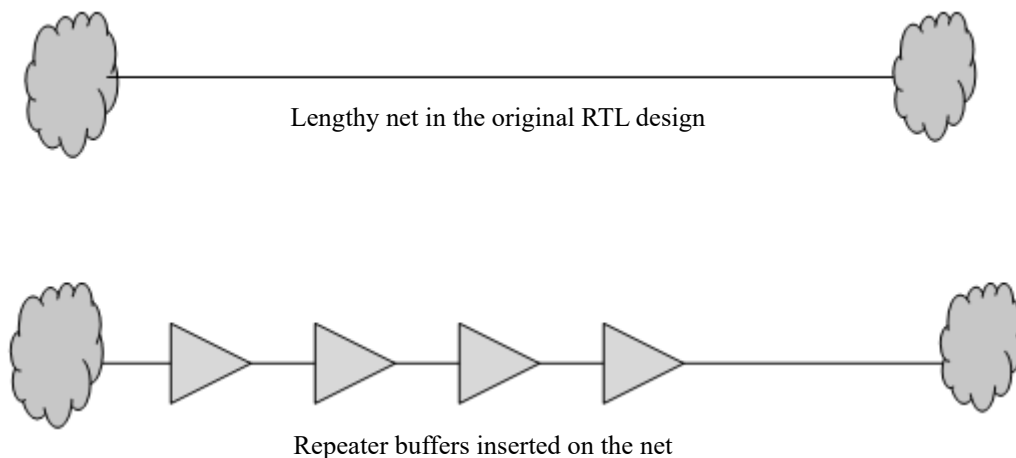
## repeater\_buffer

### Purpose

A design may contain some very long nets that may or may not have a very high fan-out. After placement of such designs, repeater buffers can be inserted on such lengthy nets to ensure that the high capacitance of these nets can be switched properly.

The `repeater_buffer` constraint is used to specify information of such repeater buffers that can be inserted on a lengthy net.

Repeater buffers are inserted in series on a net, as shown in the following figure:



**FIGURE 47.** Repeater Buffers Inserted

Please note the following points:

- This constraint is considered independent of the value of the `pe_infer_clock_net_bufs` and `pe_infer_high_fanout_bufs` parameters.
- When the `pe_infer_clock_net_bufs` and/or `pe_infer_high_fanout_bufs` parameter is set to `yes`, only the `repeater_buffer` constraint information is used for the nets for which it is provided.  
For the nets for which the `repeater_buffer` constraint information is not provided, the default virtual buffer algorithm is used.
- This constraint honors the `mix_vt_constraint` and `syn_set_dont_use` commands.
- Power of such lengthy nets is reported in the same hierarchy because it happens for nets with virtual buffers.
- Capacitance of additional nets is calculated based on wire-load. So even if the capacitance of the original net is set using SPEF or SDC (`set_load`), `repeater_buffer` constraints will be applied.

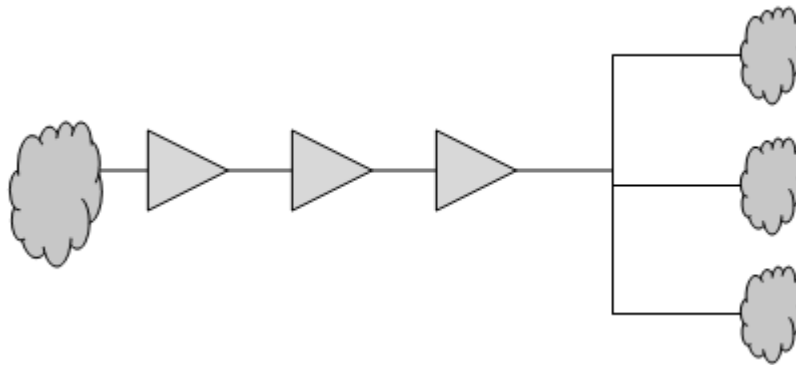
Note that this is a different behavior with respect to virtual buffers.

- Power of nets with virtual buffers is proportionately distributed in the hierarchies where the fan-out elements of that net are present. This is because the assumption there is that a buffer tree is getting created. Therefore, a majority of buffers are in the side where the fan-out elements are present. However, in case of a `repeater_buffer` structure, power of repeater buffers will be assigned and the nets in the hierarchy which completely encloses that net. Such nets will not honor the `pe_report_virtual_power_at_driving_instance` parameter.
- Consider a net that has a fan-out of more than one, as shown in the following figure:



**FIGURE 48.** Net with Multiple Fan-outs

Now consider that you set three repeater buffers on the above net. In this case, SpyGlass will assume the following structure:



**FIGURE 49.** Net with Three Repeater Buffers

Therefore, in this case also, the additional elements are N buffers and N nets with a single fan-out.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `repeater_buffer` constraint is as follows:

```
repeater_buffer
  -name <net-name-list>
  -cell <cell-name>
  [ -lib <library-name> ]
  -count <count>
```

## Arguments

### **-name <net-name-list>**

Specifies a space-separated list of nets in a design for which repeater buffers should be considered.

**-cell <cell-name>**

Specifies a cell name that should be repeated on a net.

**-lib <library-name>**

Specifies the library from which the cell should be picked.

**-count <count>**

Specifies the number of buffers that should be considered.

**Examples**

The following command specifies two buffers of type BUF1 that should be used as repeater buffers, each on NET1 and NET2:

```
repeater_buffer -name "NET1 NET2" -cell BUF1 -count 2
```

You can specify the same information separately for each net, as shown in the following example:

```
repeater_buffer -name NET2 -cell BUF1 -count 2  
repeater_buffer -name NET1 -cell BUF1 -count 2
```

**Rules**

The `repeater_buffer` constraint is used by the following rule:

---

**SpyGlass Power Estimation and SpyGlass Power Reduction solutions**PEPW02

---

## require\_constraint\_message\_tag

### Purpose

The *require\_constraint\_message\_tag* checks whether *constraint\_message\_tag\_expression* is used for the design nodes or not. It reports violation, if specific combination of tags are absent or present in the design for a particular node.

### Syntax

The syntax for the *require\_constraint\_message\_tag* constraint is as follows:

```
require_constraint_message_tag
  [-name <nodename>]
  [-except <except_nodename>]
  [-except_type <exceptDo-nodename>]
  [-type <DO_nodename>]
  [-constraint_message_tag_expression
  <constraint_message_tag_expression>]
```

### Arguments

**-name <nodename>**

Specifies the name of the top-module port, or any internal net or terminal or leaf instance for which the specified tag expression must be satisfied. A module name specified expands to list of all its instantiations (full hierarchical name). When at least one of the pins of the instance gets required expression, a PASS status is generated. However, if none of the pins of the instance gets required expression, a FAIL status is generated.

For more information, see [Example 3](#).

**-except <except\_nodename>**

Specifies the name of the top-module port, any internal net, terminal, or leaf instance name, which needs to be excluded from rule checking.

**-except\_type <exceptDo-nodename>**

Specifies the name of macro for which the specified tag expression must be satisfied.

**-type <DO\_nodename>**

Specifies the name of the macro, which needs to be excluded from rule checking.

**-constraint\_message\_tag\_expression <constraint\_message\_tag\_expression>**

Signifies the message tag expression specified using logical '||' and '&&' operator, their combinations, and :PASS and :FAIL values of tags. You can also use braces ('(',')') when specifying message tag expression.

**Examples****Example 1**

```
require_value -name "top.l_1.out" -value 0
-constraint_message_tag LATCH_VALUE_CHECK
require_path -from_type LATCH_OUT -to_type FLIP_FLOP_RESET
-constraint_message_tag LATCH_CHECK
require_constraint_message_tag -type LATCH
-constraint_message_tag_expression "LATCH_CHECK:PASS &&
LATCH_VALUE_CHECK:PASS"
```

In the above example, the `require_constraint_message_tag` looks for absence of any violation on `LATCH_CHECK` and `LATCH_VALUE_CHECK` constraint\_message\_tags. It will report info message, if the specified expression is met otherwise gives error.

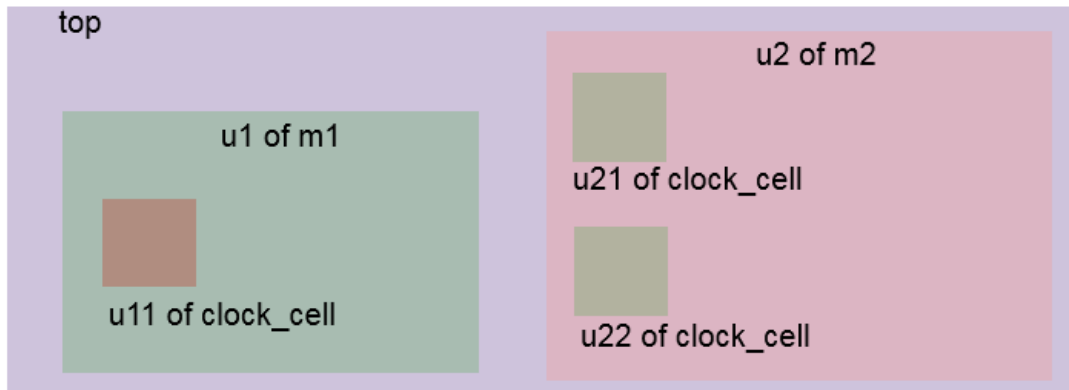
**Example 2**

```
require_constraint_message_tag -type FLIP_FLOP
-constraint_message_tag_expression "( CHECK_1:PASS ||
CHECK_2:FAIL) && CHECK_3:PASS"
```

In the above example, the `require_constraint_message_tag` looks for the absence of violation on `CHECK_1` and `CHECK_3` and presence of violation `CHECK_2`.

### Example 3

Consider the following top instantiation:



**FIGURE 50.** Top Instantiation

Now, consider the following SGDC command:

```
require_constraint_message_tag -name clock_cell <other options>
```

The above SGDC command implies:

```
require_constraint_message_tag -name top.u1.u11 top.u2.u21  
top.u2.u22 <other options>
```



## require\_path

### Purpose

The `require_path` constraint checks for the existence of the path between the specified nodes. However, the destination node may have paths existing to other nodes.

To check for the existence of the path originating from the source node that does not have any other destination node, except for the specified one, use the [require\\_strict\\_path](#) constraint.

**NOTE:** *Buffers and inverters are single-input, single-output devices. Therefore, there is no interference from external logic while tracing a net connectivity.*

### Product

SpyGlass DFT solution

### Syntax

The syntax for `require_path` constraint is as follows:

```
require_path
[ -tag <condname> | -use_shift |
  -use_capture | -use_captureATspeed ]
[ -from <frompinlist> ]
[ -from_one_of <fromoneof_pinlist> ]
[ -except_from <exceptfrom_pinlist> ]
[ -from_type <from_DO_pinlist> ]
[ -from_one_of_type <fromoneof_DO_pinlist> ]
[ -except_from_type <exceptfrom_DO_pinlist> ]
[ -exact_sequential_depth <sequential_depth> ]
[ -to <topinlist> ]
[ -to_one_of <tooneof_pinlist> ]
[ -except_to <exceptto_pinlist> ]
[ -to_type <to_DO_pinlist> ]
[ -to_one_of_type <tooneof_DO_pinlist> ]
[ -except_to_type <exceptto_DO_pinlist> ]
[ -invert ] [ -noinvert ]
[ -parallel ]
```

```

[ -path_type <buffered | sensitized | sensitizable | direct
| topological>]
[-sequential_depth <value>]
[-constraint_message_tag <value>]
[-min_to_paths <value>]
[-max_to_paths <value>]
[-report_failure_as_info]
[-filter_in_cmt_from <constraint_message_tag_expression>]
[-filter_in_cmt_to <constraint_message_tag_expression>]
[ -named_association]
[ -positional_association]
[ -instance_association]
[-instance_filter_in_cmt_from
  <constraint_message_tag_expression>]
[-instance_filter_in_cmt_to
  <constraint_message_tag_expression>]
[ -filter_in_from <include_from_pinlist> ]
[ -filter_in_to <include_to_pinlist> ]
[ -filter_in_type_from <include_from_DO_pinlist> ]
[ -filter_in_type_to <include_to_DO_pinlist> ]
[ -ignorecase ]

```

## Arguments

### **-tag <condname>**

(Optional) Specifies a condition name that is previously defined by using the *define\_tag* constraint.

This condition describes stimulation of circuit into a desired state. Please note only one condition name can be defined in a *require\_path* specification. However, simulation for a given condition name simulates all pin-value specification simultaneously, whether defined directly by pin-value specification, or merge of other *define\_tag* specifications. The built-in power-ground connections are also simulated in this process automatically.

If the *-tag* argument is specified, an un-blocked path must exist when the defined condition is simulated. If the *-tag* argument is not specified, the

`require_path` constraint defines a connectivity check that requires a net connection to exist from pin(s) specified with the `-from` argument to pin(s) specified with the `-to` argument. A net connection means a net connection across hierarchical boundaries between the specified start point and end-point(s) and may only contain buffers or inverters.

**`-use_shift` | `-use_capture` | `-use_captureATspeed`**

For any of these modifiers, `require_path` simulates test mode of that particular mode.

If `-use_shift`, `-use_capture`, or `-use_captureATspeed` argument is specified, the constraint simulates all, `shift`, `capture`, or `captureATspeed` test\_mode constraints, respectively.

**NOTE:** *If more than one of the `-tag`, `-use_shift`, `-use_capture`, or `-use_captureATspeed` arguments is specified, an error condition occurs. You should specify only one or none of these modifiers with `require_path` constraint.*

**`-from <frompinlist>`, `-to <topinlist>`**

These are the start-point and end-point nodes, respectively, in a circuit (as controlled by effective `current_design` specification) for which a path is searched after the circuit has been simulated by LE into the desired state (with the `-tag` argument specified) or a net connection is to be checked (without the `-tag` argument specified).

Either of these can be a list of top-module port names, internal net names, or terminal names. It is effectively read as a concise description of connectivity check from the each node in the `<frompinlist>` list to each node in the `<topinlist>` list.

**`-from_one_of <fromoneof_pinlist>`**

(Optional) Specifies that no error message is reported, if there is at least one success case among the specified nodes.

**`-except_from <exceptfrom_pinlist>`**

(Optional) Excludes the specified objects from nodes.

**-from\_type <from\_DO\_pinlist>**

(Optional) Same as <frompinlist> but it takes only macros as inputs.

**-from\_one\_of\_type <fromoneof\_DO\_pinlist>**

(Optional) Same as <fromoneof\_pinlist> but it takes only macros as inputs

**-except\_from\_type <exceptfrom\_DO\_pinlist>**

(Optional) Same as <exceptfrom\_pinlist> but it takes only macros as inputs.

**-exact\_sequential\_depth <sequential\_depth>**

(Optional) Defines the exact sequential depth. This argument takes an integer as an input.

Note that you can not use this argument with the -sequential\_depth argument.

**-to\_one\_of <tooneof\_pinlist>**

(Optional) Specifies that no error message is reported, if there is at least one success case among the specified nodes.

**-except\_to <exceptto\_pinlist>**

(Optional) Excludes the specified objects from nodes.

**-to\_type <to\_DO\_pinlist>**

(Optional) Same as <topinlist> but it takes only macros as inputs.

**-to\_one\_of\_type <tooneof\_DO\_pinlist>**

(Optional) Same as <tooneof\_pinlist> but it takes only macros as inputs.

**-except\_to\_type <exceptto\_DO\_pinlist>**

(Optional) Same as <tofrom\_pinlist> but it takes only macros as inputs.

**-invert | -noinvert**

(Optional) These additional qualifiers mean that check is performed to check 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**

(Optional) Specifies that paths should be searched from a node in the *<frompinlist>* list to a node at the same relative position in the *<topinlist>* list. Thus, the paths should be 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, you need to specify the exact same number of nodes in both the *<frompinlist>* list and the *<topinlist>* list. Otherwise, the constraint is not processed.

If the *-parallel* argument is specified, the *require\_path* constraint assumes that paths should be searched from a node specified with the *-from* argument to a node specified with the *-to* argument at the same relative position.

**-path\_type**

The *-path\_type* argument accepts only the following predefined list of values: *buffered*, *sensitized*, *sensitizable*, *direct*, and *topological*. The default value of this qualifier is *sensitizable*.

For more information, see [Example 2](#) and [Example 3](#).

**-sequential\_depth <value>**

Specifies the number of sequential elements between end points specified on a success path. This means that the *require\_path* check will go through the specified number of sequential elements. You can specify an integer value as an input to this argument.

**-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

**-min\_to\_paths <value>**

Specifies minimum number of expected successful paths. You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**-max\_to\_paths <value>**

Specifies maximum number of expected successful paths. Following are the rules for using this parameter:

- If you are using both the `-min_to_paths` and `-max_to_paths` arguments, then the value of the `-max_to_paths` argument should be greater than the `-min_to_paths` argument.
- You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**-report\_failure\_as\_info/-report\_failures\_as\_info**

Reports all the failures as info severity message.

**NOTE:** *The `require_path` constraint supports wildcard expressions. The supported meta-characters are `*` (star) and `?` (question mark) where `*` matches any number of characters and `?` matches only one character. The wildcard support is applicable for non-escaped names only. If the meta-characters appear inside an escaped name, they are treated as literals. For example, in the expression `"top.\mid*\bottom"`, `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 wildcard, any sub-portion of this path that contains the wildcard 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 wildcard string. For example, the expression `"top.mid*.bottom"` will expand to `"top.mid1.bottom"`, and not to `"top.mid2.bottom"`.*

**NOTE:** *The expression on which a wildcard is used should always be enclosed within double quotes. For example, `"top.mid*.bottom"`.*

**NOTE:** *The wildcard support is applicable for design objects only. For non-design objects, the support is not applicable.*

**-named\_association**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same name, from the expanded from-to-set. Individual checks are then performed on each such sub-group.

**-positional\_association**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same position, from the expanded from-to-set. Individual checks are then performed on each such sub-group.

**-instance\_association**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same instance, from the expanded from-to-set. Individual checks are then performed on each such sub-group. This argument is useful while looking for a self-loop type structure.

**-filter\_in\_cmt\_from <constraint\_message\_tag\_expression>, -filter\_in\_cmt\_to <constraint\_message\_tag\_expression>**

(Optional) Filters the starting-point and end-point when the `constraint_message_tag_expression` holds TRUE for the specified node.

**NOTE:** *You can not use `-filter_in_cmt_from` argument with the `-instance_filter_in_cmt_from` argument and `-filter_in_cmt_to` argument with the `-instance_filter_in_cmt_to` argument.*

**-instance\_filter\_in\_cmt\_from <constraint\_message\_tag\_expression>, instance\_filter\_in\_cmt\_to <constraint\_message\_tag\_expression>**

(Optional) Filters the starting-point and end-point when the `constraint_message_tag_expression` holds TRUE for the associated instance of the specified node.

**NOTE:** *You can not use `-instance_filter_in_cmt_from` argument with the `-filter_in_cmt_from` argument and `-instance_filter_in_cmt_to` argument with the `-filter_in_cmt_to` argument.*

**-filter\_in\_from <include\_from\_pinlist>, -filter\_in\_to <include\_to\_pinlist>**

(Optional) Same as the `-from` and `-to` arguments but defines design nodes that are to be included.

**-filter\_in\_type\_from <include\_from\_DO\_pinlist>, -filter\_in\_type\_to <include\_from\_DO\_pinlist>**

(Optional) Same as the `-from_type` and `-to_type` arguments but defines design nodes that are to be included.

**-ignorecase**

(Optional) to ignore the case for <nodename> specified as the -from, -to, -except\_from, -except\_to, -filter\_in\_name\_from, and -filter\_in\_name\_to arguments.

**NOTE:** *It applies on all fields which take design-node name as input.*

**Supported Macros**

SpyGlass DFT solution supports the following macros:

<b>Macros Representing Collection Of Pins/Ports</b>		
FLIP_FLOP_DATA	FLIP_FLOP_OUT	FLIP_FLIP_CLOCK
FLIP_FLOP_SET	FLIP_FLOP_RESET	FLIP_FLOP_ENABLE
SCAN_FLIP_FLOP_DATA	SCAN_FLIP_FLOP_OUT	SCAN_FLIP_FLOP_CLOCK
SCAN_FLIP_FLOP_SET	SCAN_FLIP_FLOP_RESET	SCAN_FLIP_FLOP_ENABLE
LATCH_DATA	LATCH_OUT	LATCH_SET
LATCH_RESET	LATCH_ENABLE	MUX_SELECT
CGC_CLOCK_IN	CGC_CLOCK_OUT	CGC_SYSTEM_ENABLE
CGC_TEST_ENABLE	BLACK_BOX	BLACK_BOX_OUTPUT
BLACK_BOX_INPUT	INPUT_PORTS	OUTPUT_PORTS
INOUT_PORTS	ALL_PORTS	TIED_0
TIED_1	TIED_SGDC	TIED_0_SGDC
TIED_1_SGDC	MEMORY_ADDRESS	MEMORY_CLOCK
MEMORY_DATA	MEMORY_ENABLE	MEMORY_OUT
MUX_DATA	MUX_OUTPUT	UNDRIVEN_NET
UNDRIVEN_PIN	X_SOURCES	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	
<b>Macros Representing Collection of Instances</b>		



FLIP_FLOP	SCAN_FLIP_FLOP	LATCH
MUX	LATCH	ICG
CGC	BLCAK_BOX	MEMORY

**NOTE:** *TIED\_\* macros are used to filter out already selected set using the `-except_type`, `-except_from_type`, or `-except_to_type` arguments.*

For more information on using macros in SGDC, see the **Using Macros in SGDC** section in the *SpyGlass DFT Rules Reference Guide*.

## Examples

### Example 1

Some examples of the `require_path` constraint are as follows:

```
require_path -tag s1 -from top.SEF
             -to top.U1.U3.inst_add.D
require_path -tag s100mode -from top.U10.inst_mdd.Q
             -to top.Q1
require_path -from top.EN
             -to top.U1.U2.DIF top.U1.U4.DIF
```

Some examples of the `require_path` constraint with the `-parallel` argument are as follows:

```
require_path -tag s11 -from top.SEF top.SFF
             -to top.U1.D, top.U2.D -parallel
```

Here, the paths will be searched from the `top.SEF` node to the `top.U1.D` node and from the `top.SFF` node to the `top.U2.D` node.

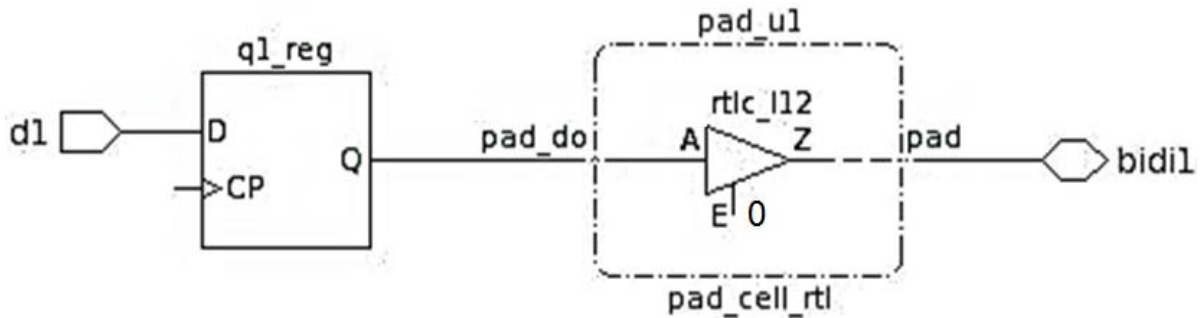
```
require_path -tag s101 -from top.SEF[2:0] top.SFF
             -to top.FGT[3:0] - parallel
```

The above specifications imply that the following paths should be searched:

- From the `top.SEF [2]` node to the `top.FGT [3]` node
- From the `top.SEF [1]` node to the `top.FGT [2]` node
- From the `top.SEF [0]` node to the `top.FGT [1]` node
- From the `top.SFF` node (exists as a scalar node) to the `top.FGT [0]` node

**Example 2**

Consider the following schematic:



**FIGURE 51.**

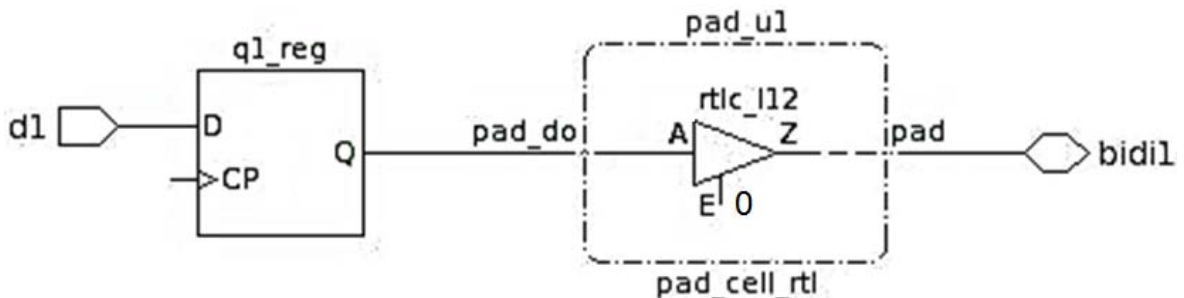
Now, consider the following constraint description:

```
require_path -from d1 -to bidir1 -path_type topological -
sequential_depth 1
```

For the above example, a topological path is traced through the flip-flop if the sequential depth greater than 0 is specified, even if the tristate buffer is disabled.

**Example 3**

Consider the following schematic:



**FIGURE 52.**

Now, consider the following constraint description:

```
require_path -from clk1 -to clk_out1 -path_type topological
```

For the above example, a topological path is traced through the flip-flop if the sequential depth greater than 0 is specified, even if the tristate buffer is disabled.

#### Example 4

Consider the following constraint specification

```
require_path -from_type FLIP_FLOP_OUTPUT -to sig1
-filter_in_cmt_from "X1:PASS"
```

In the above example, the *require\_path* constraint checks for path to sig1 from flip-flop outputs, which have passed the check for constraint\_message\_tag, X1.

#### Example 5

Consider the following constraint specification:

```
require_path -from sig1 -to_type FLIP_FLOP_RESET -
filter_in_cmt_to "X1:PASS"
```

In the above example, the *require\_path* constraint checks for path from sig1 to flip-flop reset pins, which have passed the check for constraint\_message\_tag, X1.

#### Example 6

Consider the following constraint specification:

```
require_path -from sig1 -to sig2 -filter_in_cmt_from
"X1:PASS" -filter_in_cmt_to "X2:PASS"
```

In the above example, the *require\_path* constraint checks for path from sig1 to sig, where sig1 has passed the check for constraint\_message\_tag, X1 and sig2 has passed the check for constraint\_message\_tag, X2.

#### Example 7

Consider the following constraint specification:

```
require_path -from_type FLIP_FLOP_OUTPUT -to sig1
-instance_filter_in_cmt_from "X1:PASS"
```

In the above example, the *require\_path* constraint checks for path to sig1

from flip-flop outputs, whose instances have passed the check for `constraint_message_tag, X1`.

### Example 8

Consider the following constraint specification:

```
require_path -from sig1 -to_type FLIP_FLOP_RESET
-instance_filter_in_cmt_to "X1:PASS"
```

In the above example, the `require_path` constraint checks for path from `sig1` to flip-flop reset pins, whose instances have passed the check for `constraint_message_tag, X1`.

### Example 9

Consider the following constraint specification:

```
require_path -from sig1 -to sig2 -
instance_filter_in_cmt_from "X1:PASS" -
instance_filter_in_cmt_to "X2:PASS"
```

In the above example, the `require_path` constraint checks for path from `sig1` to `sig`, where `sig1`'s instances has passed the check for `constraint_message_tag, X1` and `sig2`'s instances have passed the check for `constraint_message_tag, X2`.

### Example 10

Consider the following constraint specifications:

```
require_path -filter_in_from "in*" -from_type INPUT_PORTS
-filter_in_to "out*" -to_type OUTPUT_PORTS -ignorecase
```

```
require_path -filter_in_from "in*" -from_type INPUT_PORTS
-to "out*" -filter_in_type_to OUTPUT_PORTS -ignorecase
```

```
require_path -from "in*" -filter_in_type_from INPUT_PORTS
-filter_in_to "out*" -to_type OUTPUT_PORTS -ignorecase
```

```
require_path -from "in*" -filter_in_type_from INPUT_PORTS
-to "out*" -filter_in_type_to OUTPUT_PORTS -ignorecase
```

In above cases, each `require_path` constraint checks for the path from

input ports matching with "in\*" (case-insensitive) to output ports matching with "out\*" (case-insensitive).

### Example 11

Consider the following constraint specification:

```
require_path -from FLIP_FLOP_OUT to FLIP_FLOP_DATA -
instance_association
```

In the above example, the require paths are split into N numbers, that is, number equal to the flip-flop count of the design. Each require path checks for a path between flip-flop-Q-pin and its own D-pin (self-loop). In absence of '-instance\_association' modifier, check would imply path from a flip-flop-Q to any flip-flop-D-pin.

## Rules

The `require_path` constraint is used by the following rules:

---

### SpyGlass DFT Solution

Conn\_02

---

### SpyGlass Connectivity Verify Solution

Soc\_02

---

## require\_pulse

### Purpose

The `require_pulse` constraint defines 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.

### Product

SpyGlass DFT solution

## Syntax

```
require_pulse
  -tag <condition_name>
  [-name <pin/net hier name>]
  [ -count <num-of-pulses> ]
  [ -except <except_pin_hier_name> ]
  [ -except_type <except_DO> ]
  [ -after <observe_after_bit_number> ]
  [ -before <observe_before_bit_number> ]
  [ -high_width <number_of_bits_during_high_phase> ]
  [ -low_width <number_of_bits_during_low_phase> ]
  [ -type <DO_objlist>]
  [-constraint_message_tag <value>]
```

## Arguments

### **-tag <condition-name>**

(Mandatory) Tag name of the [define\\_tag](#) constraint under which simulation is to be done.

### **-name <pin/net hier name>**

(Optional) Pin or net hier name, where pulse pattern is to be checked.

### **-count <num-of-pulses>**

(Optional) Total number of pulses required. The default value of this argument is 1.

### **-after <observe\_after\_bit\_number>**

(Optional) Pulse pattern checking starts after this bit number. The default value of this argument is 0.

### **-before <observe\_before\_bit\_number>**

(Optional) Pulse pattern checking ends before this bit number. The default value of this argument is the value of the last simulation cycle.

**-high\_width <number\_of\_bits\_during\_high\_phase>**

(Optional) Number of bits during high phase. The default value of this argument is 1.

**-low\_width <number\_of\_bits\_durin\_low\_phase>**

(Optional) Number of bits during low phase. The default value of this argument is 1.

**-type <DO\_objlist>**

(Optional) Specifies a collection of design objects, which require a pulse. This argument takes macros as input.

**-except <except\_pin\_hier\_name>**

same as -name but defines design nodes which are not to be used as name.

**-except\_type <except\_DO>**

same as <except\_pin\_hier\_name> but it takes only macros as inputs.

**-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

**Supported Macros**

To view the list of macros supported by the `require_pulse` constraint, see [Supported Macros](#).

**Rules**

The `require_pulse` constraint is used by the following rules:

---

**SpyGlass DFT Solution**

---

Conn\_15

---

**SpyGlass DFT DSM Solution**

---

Atspeed\_21      Info\_Atspeed\_21

---

## require\_stable\_value

### Purpose

The `require_stable_value` constraint specifies nodes whose value is expected to be stable.

### Product

SpyGlass DFT solution

### Syntax

The syntax for the `require_stable_value` constraint is as follows:

```
require_stable_value
  [-name <nodename> ]
  [-value <value>]
  [ -except <except_nodename> ]
  [ -type <DO_nodename> ]
  [ -except_type <exceptDO_nodename> ]
  [ -tag <condname> | -use_shift | -use_capture |
    -use_captureATspeed ]
  [-constraint_message_tag <value>]
  [-report_failure_as_info]
```

### Arguments

#### **-name <nodename>**

(Optional) The name can be a top-module port, or any internal net name, or terminal name. More than one pin name can be specified, and it is effectively read as a concise description of as many individual value checks.

**NOTE:** Specify either *-name* or *-type* argument.

#### **-value <value>**

(Optional) You can specify the following values: 0, 1 and 0\_or\_1.

SpyGlass DFT first checks whether the signal is stable. Then, the current value of the signal is compared with the value specified using this



argument. Accordingly, the Conn\_14 rule displays the violation messages.

**-except <except\_nodename>**

(Optional) Same as <nodename> but defines design nodes that are not to be used as name.

**-type <DO\_nodename>**

Same as <nodename> but it takes only macros as inputs.

**NOTE:** *Specify either -name or -type argument.*

To view the list of macros supported by the `require_stable_value` constraint, see [Supported Macros](#).

**-except\_type <exceptDO\_nodename>**

Same as <DO\_nodename> but it takes only macros as inputs.

**-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

**-tag <condname>**

(Optional) A condition previously defined by using the [define\\_tag](#) constraint. It describes a stimulation condition.

Note that only one condition name can be defined in a `require_stable_value` specification. However, simulation for a given condition name simulates all pin-value specifications simultaneously. The built-in power-ground connections are also simulated in this process.

**-use\_shift | -use\_capture | -use\_captureATspeed**

For any of these modifiers, `require_stable_value` simulates test mode of that particular mode.

If `-use_shift`, `-use_capture`, or `-use_captureATspeed` argument is specified, the constraint simulates all, shift, capture, or captureATspeed test\_mode constraints, respectively.

**NOTE:** *If more than one of the -tag, -use\_shift, -use\_capture, or -use\_captureATspeed arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with require\_stable\_value constraint.*

**-report\_failure\_as\_info/-report\_failures\_as\_info**

Reports all the failures as info severity message.

## Examples

Currently Unavailable

## Rules

The `require_stable_value` constraint is used by the following rule:

---

**SpyGlass Connectivity Verify Solution**

---

Soc\_14

---

**SpyGlass DFT Solution**

---

Conn\_14

---

## require\_strict\_path

### Purpose

The `require_strict_path` constraint is used to check for the existence of path between the specified nodes. However, no other path, in the fanout cone of the specified from-node' should exist.

The following illegal end-points are not user-controlled in the `require_strict_path` constraint:

- Primary output ports
- Flip-flop
- Black-box
- Non-transparent latch
- Any logic when the `path_type = direct`
- Any gate which is neither a buffer nor an inverter when `path_type = buffered`

The design analysis is stopped when one of the following conditions hold true:

- A valid end-point is found in the design
- The path is blocked

**NOTE:** Use combination of the `require_path` and `illegal_path` constraints if you want to specify both (required and illegal) as the type of end-points.

To check for the existence of the path between the specified nodes, which may have paths to other nodes, use the [require\\_path](#) constraint.

**NOTE:** Buffers and inverters are single-input, single-output devices. Therefore, there is no interference from external logic while tracing a net connectivity.

### Product

SpyGlass DFT solution

### Syntax

The syntax for the `require_strict_path` constraint is as follows:

```
require_strict_path
```

```

[-from_one_of_type <fromoneof_DO_pinlist> ]
[-to_one_of_type <tooneof_DO_pinlist> ]
[-except_from_type <exceptfrom_DO_pinlist> ]
[-except_to_type <exceptto_DO_pinlist> ]
[-from_type <from_DO_pinlist> ]
[-to_type <to_DO_pinlist>]
[-from <from_pinlist> ]
[-except_from <exceptfrom_pinlist>
[-from_one_of <fromoneof_pinlist> ]
[-to <to_pinlist> ]
[-except_to <exceptto_pinlist> ]
[ -to_one_of <tooneof_pinlist> ]
[ -tag <cond-name> | -use_shift | -use_capture |
-use_captureATspeed ]
[ -path_type <buffered | sensitized | sensitizable |
direct | topological>]
[ -sequential_depth <value>]
[-constraint_message_tag <value>]
[-min_to_paths <value>]
[-max_to_paths <value>]
[-report_failure_as_info]
[ -filter_in_from <include_from_pinlist> ]
[ -filter_in_to <include_to_pinlist> ]
[ -filter_in_type_from <include_from_DO_pinlist> ]
[ -filter_in_type_to <include_to_DO_pinlist> ]
[ -filter_in_cmt_from <from_cmt_expression> ]
[ -filter_in_cmt_to <to_cmt_expression> ]
[ -instance_filter_in_cmt_from <from_cmt_expression> ]
[ -instance_filter_in_cmt_to <to_cmt_expression> ]
[ -named_association]
[ -positional_association]
[ -instance_association]
[ -ignorecase ]

```

## Arguments

**-from\_one\_of\_type** <fromoneof\_DO\_pinlist>

(Optional) Same as <fromoneof\_pinlist> but it takes only macros as inputs.

**-to\_one\_of\_type** <tooneof\_DO\_pinlist>

(Optional) Same as <tooneof\_pinlist> but it takes only macros as inputs.

**-except\_from\_type** <exceptfrom\_DO\_pinlist>

(Optional) Same as <exceptfrom\_pinlist> but it takes only macros as inputs.

**-except\_to\_type** <exceptto\_DO\_pinlist>

(Optional) Same as <tofrom\_pinlist> but it takes only macros as inputs.

**-from\_type** <from\_DO\_pinlist>

(Optional) Same as <frompinlist> but it takes only macros as inputs.

**-to\_type** <to\_DO\_pinlist>

(Optional) Same as <topinlist> but it takes only macros as inputs.

**-except\_from** <exceptfrom\_pinlist>

(Optional) Excludes the specified objects from nodes.

**-from\_one\_of** <fromoneof\_pinlist>

(Optional) Specifies that no error message is reported, if there is at least one success case among the specified nodes.

**-except\_to** <exceptto\_pinlist>

(Optional) Excludes the specified objects from nodes.

**-to\_one\_of <tooneof\_pinlist>**

(Optional) Specifies that no error message is reported, if there is at least one success case among the specified nodes.

**-from <from-pinlist>, -to <to-pinlist>**

These are the start-point and end-point nodes, respectively, in a circuit (as controlled by effective `current_design` specification) for which a path is searched after the circuit has been simulated by LE into the desired state (with the `-tag` argument specified) or a net connection is to be checked (without the `-tag` argument specified).

Either of these can be a list of top-module port names, internal net names, or terminal names. It is effectively read as a concise description of connectivity check from each node in the `<from-pinlist>` list to each node in the `<to-pinlist>` list.

**-tag <cond-name>**

(Optional) Specifies a condition name that is previously defined by using the `define_tag` constraint.

This condition describes stimulation of circuit into a desired state. Please note only one condition name can be defined in a `require_strict_path` specification. However, simulation for a given condition name simulates all pin-value specification simultaneously, whether defined directly by pin-value specification, or merge of other `define_tag` specifications. The built-in power-ground connections are also simulated in this process automatically.

If the `-tag` argument is specified, an un-blocked path must exist when the defined condition is simulated. If the `-tag` argument is not specified, the `require_strict_path` constraint defines a connectivity check that requires a net connection to exist from pin(s) specified with the `-from` argument to pin(s) specified with the `-to` argument. A net connection means a net connection across hierarchical boundaries between the specified start point and end-point(s) and may only contain buffers or inverters.

**-use\_shift | -use\_capture | -use\_captureATspeed**

(Optional) For any of these modifiers, the `require_strict_path` constraint simulates the test mode of that particular mode.

If the `-use_shift`, `-use_capture`, or `-use_captureATspeed` argument is specified, the constraint simulates the shift, capture, or captureATspeed mode, respectively.

**NOTE:** *Specify only one of the `-tag`, `-use_shift`, `-use_capture`, or `-use_captureATspeed` fields with the `require_strict_path` constraint.*

**-path\_type**

The `-path_type` argument accepts only the following predefined list of values: `buffered`, `sensitized`, `sensitizable`, `direct`, and `topological`. The default value of this qualifier is `sensitizable`.

**-sequential\_depth <value>**

Specifies the number of sequential elements between end points specified on a success path. This means that the `require_path` check will go through the specified number of sequential elements. You can specify an integer value as an input to this argument.

**-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

**-min\_to\_paths <value>**

Specifies minimum number of expected successful paths. You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

**-max\_to\_paths <value>**

Specifies maximum number of expected successful paths. Following are the rules for using this parameter:

- If you are using both the `-min_to_paths` and `-max_to_paths` arguments, then the value of the `-max_to_paths` argument should be greater than the `-min_to_paths` argument.
- You can not specify this argument with `-from_one_of` and `-from_one_of_type` arguments.

### Supported Macros

To view the list of macros supported by the `require_strict_path` constraint, see [Supported Macros](#).

#### `-report_failure_as_info/-report_failures_as_info`

Reports all the failures as info severity message.

#### `-filter_in_from <include_from_pinlist>, -filter_in_to <include_to_pinlist>`

(Optional) Same as the `-from` and `-to` arguments but defines design nodes that are to be included.

#### `-filter_in_type_from <include_from_DO_pinlist>, -filter_in_type_to <include_from_DO_pinlist>`

(Optional) Same as the `-from_type` and `-to_type` arguments but defines design nodes that are to be included.

#### `-filter_in_cmt_from <include_from_cmt_expression>, -filter_in_cmt_to <include_to_cmt_expression>`

(Optional) Same as the `-from` and `-to` arguments but it considers `constraint_message_tag` (cmt) expressions as input and includes design nodes for which cmt expression holds true, that is, on that specific node.

#### `-instance_filter_in_cmt_from <include_from_cmt_expression>, -instance_filter_in_cmt_to <include_to_cmt_expression>`

(Optional) Same as the `-filter_in_cmt_from` and `-filter_in_cmt_to` arguments but it includes design nodes for which cmt expression holds true on the associated instance.

The cmt expression holds true on associated instance if it holds true on any



pin of the that instance including the design node.

**NOTE:** *You cannot use the `-instance_filter_in_cmt_from` argument with the `-filter_in_cmt_from` argument and the `-instance_filter_in_cmt_to` argument with the `-filter_in_cmt_to` argument.*

#### **-named\_association**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same name, from the expanded from-to-set. Individual checks are then performed on each such sub-group.

#### **-positional\_association**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same position, from the expanded from-to-set. Individual checks are then performed on each such sub-group.

#### **-instance\_association**

(Optional) Use this argument to create multiple groups of from-to nodes, based on the same instance, from the expanded from-to-set. Individual checks are then performed on each such sub-group. This argument is useful while looking for a self-loop type structure.

#### **-ignorecase**

(Optional) Ignores the case for the nodename specified using the `-from`, `-to`, `-except_from`, `-except_to`, `-filter_in_name_from`, and the `-filter_in_name_to` arguments.

This is applicable for all the arguments, which consider the design-node name as input.

## **Examples**

Some examples of the `require_strict_path` constraint are as follows:

```
require_strict_path -tag s1
  -from top.SEF -to top.U1.U3.inst_add.D
require_strict_path -tag s100mode
  -from top.U10.inst_mdd.Q -to top.Q1
```

```
require_strict_path
  -from top.EN -to top.U1.U2.DIF top.U1.U4.DIF
```

## Rules

The `require_strict_path` constraint is used by the following rule:

---

**SpyGlass DFT Solution**

---

Conn\_08

---

**SpyGlass Connectivity Verify Solution**

---

Soc\_08

---

## require\_structure

### Purpose

The `require_structure` constraint is used to define a structure check for all paths from source pins to destination pin.

The source pins are the hierarchical pins, which are specified using the `-from` and `-from_one_of` arguments. You can specify multiple values for the source pin.

The destination pins are the hierarchical pins, which are specified using the `-to` argument. The destination pin can assume only one value.

The structure type is controlled by the `-structure` argument, which may take one of the following values: `and`, `or`, or `xor`.

The type of check is controlled by the `-type` argument, which may take one of the following values: `Structural` or `Simulation`

### Product

SpyGlass DFT solution

### Syntax

```
require_structure
```

```

-[from <src_node_names>]
[-from_one_of <src_node_names>
-to <destn_node_name>
-structure <and | or | xor>
[ -type <STRUCTURAL | SIMULATION> ]
[ -cycle <number> ]
[-constraint_message_tag <value>]

```

## Arguments

### **-from <src\_node\_names>**

Name of hierarchical source pin. You can specify multiple values for the source pin.

### **-from\_one\_of <src\_node\_names>**

Name of hierarchical source pin. You can specify multiple values for the source pin. However, rule checks whether at least one of the values specified using this argument is present in the fan-in.

### **-to <destn\_node\_name>**

Name of hierarchical destination pin.

### **-structure <and | or | xor>**

Specifies the structure, which may take one of the following values: and, or, xor.

### **-type <STRUCTURAL | SIMULATION>**

Specifies the type of check as Structural or Simulation. Default type is Structural.

### **-cycle <number>**

Specifies number of simulation cycles

### **-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message

generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

## Examples

Consider the following example:

```
require_structure -from top.pin1 top.pin2 top.pin3 -to  
top.pinX -structure and
```

## Rules

The `require_structure` constraint is used by the following rules:

---

**SpyGlass DFT Solution**

---

Conn\_07

---

**SpyGlass Connectivity Verify Solution**

---

Soc\_07

---

## require\_value

### Purpose

The `require_value` constraint defines 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.

The `require_value` constraint has two optional fields: `-allowInversion` and `matchNBits`.

When `require_value` is used twice for the same node, SpyGlass checks twice on the net per the specified value.

**NOTE:** *The `require_value` constraint supports scoped statements and wildcards for both the module name and the pin/net name.*

### Product

SpyGlass DFT solution

### Syntax

The syntax for `require_value` constraint is as follows:

```
require_value
  [-name <nodename> ]
  [ -except <except_nodename> ]
  [ -type <DO_nodename> ]
  [ -except_type <exceptDO_nodename> ]
  [ -value <value> ]
  [ -value_type <type> ]
  [ -tag <condname> | -use_shift |
    -use_capture | -use_captureATspeed ]
  [ -allowInversion ]
  [ -matchNBits <num> ]
  [-constraint_message_tag <value>]
  -[report_failure_as_info]
  [-filter_in_cmt <constraint_message_tag_expression>]
  [-instance_filter_in_cmt
  <constraint_message_tag_expression>]
  [ -filter_in_name <include_nodename> ]
```

```
[ -filter_in_type <include_DO_nodename> ]  
[ -ignorecase ]
```

## Arguments

### **-name <nodename>**

(Optional) The name can be a top-module port, or any internal net name, or terminal name. More than one pin name can be specified, and it is effectively read as a concise description of as many individual value checks.

**NOTE:** *Specify either -name or -type argument.*

### **-except <except\_nodename>**

(Optional) Same as *<nodename>* but defines design nodes that are not to be used as name.

### **-type <DO\_nodename>**

Same as *<nodename>* but it takes only macros as inputs.

**NOTE:** *Specify either -name or -type argument.*

To view the list of macros supported by the `require_value` constraint, see [Supported Macros](#).

### **-except\_type <exceptDO\_nodename>**

Same as *<DO\_nodename>* but it takes only macros as inputs.

### **-value <value>**

The value is a logic value string of 0, 1, X, Z, 1\_or\_0, and 0\_or\_1. A single-bit value means check at end of complete simulation. The X value is treated as do-not-compare. A multi-bit value means check on cycle-by-cycle simulation basis. For specification of a vector value, SGDC multi-bit specification format (same as used for the `test_mode` constraint) should be used.

You can specify repeat sequences for the `require_value` constraint.

For fields that require repeat sequence, you can specify the values as

<I\*S>. Here, S is any string that does not contain the <, >, and \* characters. However, S can contain another <I\*S> expression. I is an integer that is always interpreted as a decimal value. The expression <I\*S> means that the sequence S will be repeated I number of times.

#### **value\_type <type>**

Specify one the following values:

- **active:** Implies 1 for non-inverting pin and 0 for inverting clock-pin.
- **inactive:** Implies 0 for non-inverting pin and 1 for inverting clock-pin.

**NOTE:** *Specify either -value or -value\_type argument. Otherwise, the require\_value constraint is ignored for analysis.*

#### **List of Macros Supported by the require\_value and illegal\_value Constraints**

You can use only the following set of macros along with the -value\_type argument of the require\_value and illegal\_value constraints:

- FLIP\_FLOP\_RESET
- SCAN\_FLIP\_FLOP\_RESET
- LATCH\_RESET
- FLIP\_FLOP\_SET
- SCAN\_FLIP\_FLOP\_SET
- LATCH\_SET
- FLIP\_FLOP\_ENABLE
- SCAN\_FLIP\_FLOP\_ENABLE
- LATCH\_ENABLE
- FLIP\_FLOP\_CLOCK
- SCAN\_FLIP\_FLOP\_CLOCK

#### **-constraint\_message\_tag <value>**

Specifies a string value that gets prefixed in the violation message generated by the respective rule for the said constraint.

**NOTE:** *This argument accepts only alpha-numeric characters and underscore.*

**-tag <condname>**

(Optional) A condition previously defined by using the *define\_tag* constraint. It describes a stimulation condition.

Note that only one condition name can be defined in a *require\_value* specification. However, simulation for a given condition name simulates all pin-value specifications simultaneously. The built-in power-ground connections are also simulated in this process.

**-use\_shift | -use\_capture | -use\_captureATspeed**

For any of these modifiers, *require\_value* simulates test mode of that particular mode.

If *-use\_shift*, *-use\_capture*, or *-use\_captureATspeed* argument is specified, the constraint simulates all, shift, capture, or captureATspeed test\_mode constraints, respectively.

**NOTE:** *If more than one of the -tag, -use\_shift, -use\_capture, or -use\_captureATspeed arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with require\_value constraint.*

**-allowInversion**

(Optional) Indicates that the inverted simulated pattern is also considered as valid pattern.

**-matchNBits <num>**

(Optional) Specifies that only the *<num>* number of least significant bits are to be considered. If *<num>* is greater than *<value>* (specified with *-value* argument), the latter is padded with *X* to match the former's width.

**-report\_failure\_as\_info/-report\_failures\_as\_info**

Reports all the failures as info severity message.

**-filter\_in\_cmt <constraint\_message\_tag\_expression>**

(Optional) Filters the specified node when the *constraint\_message\_tag\_expression* holds TRUE on the node itself.

**NOTE:** *Note that you can not use this argument with the -instance\_filter\_in\_cmt*



*argument.*

**-instance\_filter\_in\_cmt <constraint\_message\_tag\_expression**

Filters the specified node when the constraint\_message\_tag\_expression holds TRUE for the associated instance of the node.

**NOTE:** *You can not use this argument with the -filter\_in\_cmt argument.*

**-filter\_in\_name <include\_nodename>**

(Optional) Same as the -name argument but defines design nodes that are to be included.

**-filter\_in\_type <include\_DO\_nodename>**

(Optional) Same as the -type argument but defines design nodes that are to be included.

**-ignorecase**

(Optional) Ignores the case for the nodename specified using the -name, -except\_to, and -filter\_in\_name arguments.

**NOTE:** *It is applicable to all the arguments, which take design-node name as an input.*

## Examples

Consider the following examples:

### Example 1

```
require_value -name abc -value "<5*10>"
```

The above example will be expanded as follows:

```
require_value -name abc -value 1010101010
```

### Example 2

```
require_value -name abc -value "11<5*10>010"
```

The above example will be expanded as follows:

```
require_value -name abc -value 111010101010010
```

### Example 3

```
require_value -name abc -value "<50*11<5*10>>010"
```

The above example will be expanded as follows:

```
require_value -name abc -value 111010101010...(repeated 50 times
followed by 010)
```

You can also set a variable using the command `setvar` to obtain the above result as follows:

```
setvar x 11<5*10>
require_value -name abc -value "<5*${x}>010"
```

The above example will be expanded as follows:

```
require_value -name abc -value 111010101010...(repeated 50 times
followed by 010)
```

**NOTE:** *Tagging for nesting is not allowed. For example, the following `require_value` statements are not allowed:*

```
require_value -name sub_seq -value <5*01>
require_value -name main_seq -value <100*sub_seq>
```

*However, you can achieve the same result by using the `setvar` command.*

#### Example 4

```
require_value -tag s1 -name top.U1.U2.SEF
-value 1010 -allowInversion -matchNBits 2
```

#### Example 5

Consider the following example:

```
-require_value -name p1 -value 0X110_or_110
```

The above constraint specification means:

- first bit must not be 0
- second bit is don't care
- third and fourth bits must not be 1
- fifth should be neither 0 nor 1
- sixth should not be 1
- seventh bit should not be 0

#### Example 6

Consider the following sample input values:

```
require_value -name vec[3:0] -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "1010"
require_value -name vec[1] -value "0000"
require_value -name vec[2] -value "0000"
require_value -name vec[3] -value "0000"
```

### Example 7

Consider the following sample input values:

```
require_value -name vec[3:0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "0"
require_value -name vec[1] -value "1"
require_value -name vec[2] -value "0"
require_value -name vec[3] -value "1"
```

### Example 8

Consider the following sample input values:

```
require_value -name vec[3:0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "1"
require_value -name vec[1] -value "0"
require_value -name vec[2] -value "0"
require_value -name vec[3] -value "0"
```

### Example 9

Consider the following sample input values:

```
require_value -name vec -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "1010"
```

```
require_value -name vec[1] -value "0000"  
require_value -name vec[2] -value "0000"  
require_value -name vec[3] -value "0000"
```

### Example 10

Consider the following sample input values:

```
require_value -name vec -value { b 1010 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "0"  
require_value -name vec[1] -value "1"  
require_value -name vec[2] -value "0"  
require_value -name vec[3] -value "1"
```

### Example 11

Consider the following sample input values:

```
require_value -name vec -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "1"  
require_value -name vec[1] -value "0"  
require_value -name vec[2] -value "0"  
require_value -name vec[3] -value "0"
```

### Example 12

Consider the following sample input values:

```
require_value -name vec[0] -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "1010"
```

### Example 13

Consider the following sample input values:

```
require_value -name vec[0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "0"
```

#### **Example 14**

Consider the following sample input values:

```
require_value -name vec[0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "1"
```

#### **Example 15**

Consider the following sample input values:

```
require_value -name sclr -value { b 1 0 1 0 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
require_value -name sclr -value "1010"
```

#### **Example 16**

Consider the following sample input values:

```
require_value -name sclr -value { b 1010 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
require_value -name sclr -value "0"
```

#### **Example 17**

Consider the following sample input values:

```
require_value -name sclr -value { b 1 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
require_value -name sclr -value "1"
```

**Example 18**

Consider the following sample input values:

```
require_value -name vec -value { h 6 }
```

where `vec` is the 3:0 vector net

The above input is expanded as shown below:

```
require_value -name vec[0] -value "0"
require_value -name vec[1] -value "1"
require_value -name vec[2] -value "1"
require_value -name vec[3] -value "0"
```

**Example 19**

Consider the following constraint specification:

```
require_value -type FLIP_FLOP_OUTPUT -value 0_or_1
-filter_in_cmt "X1:PASS && X2:FAIL"
```

In the above example, the *require\_value* constraint will check for value 0/1 on flip-flop outputs which have passed the check for the *constraint\_message\_tag* X1 and failed the check for *constraint\_message\_tag* X2.

**Example 20**

Consider the following constraint specification:

```
require_value -type FLIP_FLOP_OUTPUT -value 0_or_1
-instance_filter_in_cmt "X1:PASS && X2:FAIL"
```

In the above example, the *require\_value* constraint will check for value 0/1 on flip-flop outputs whose instance have passed the check for *constraint\_message\_tag* X1 and failed the check for *constraint\_message\_tag* X2.

**Example 21**

Consider the following constraint specifications:

```
require_value -filter_in_name "out*" -type OUTPUT_PORTS -
ignorecase -value 0
```

```
require_path -name "out*" -filter_in_type OUTPUT_PORTS -
ignorecase -value 0
```

In above cases, each `require_value` constraint will check for value 0 on output ports matching with "out\*" (case-insensitive).

## Rules

The `require_value` constraint is used by the following rule:

---

**SpyGlass Connectivity Verify Solution**

---

Soc\_01                  Soc\_01\_Info

---

**SpyGlass DFT Solution**

---

Conn\_01

---

## reset

### Purpose

Rules that check for resets require you to specify the names of reset signals using the `reset` keyword in a SpyGlass Design Constraints file.

### Product

SpyGlass Auto Verify solution, SpyGlass CDC solution, SpyGlass DFT DSM solution, SpyGlass TXV solution, and SpyGlass ERC product

### Syntax

The syntax of using the `reset` keyword in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>
  reset -name <rst-name>
    [ -async | -sync ]
    [ -value <0 | 1> ]
    [ -soft ]
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

#### -name <rst-name>

The reset port/pin name.

The pin can be a top-level port/pin/net as well as an internal pin.

You can specify a single port/pin/net name or a space-separated list of port/pin/net names.

For top-level port/pin/nets, `<rst-name>` can be the port/pin/net's full hierarchical name or its simple name. For internal pins, `<rst-name>` must be the pin's full hierarchical name.



**-async, -sync**

(Optional) Specifies the reset type as asynchronous or synchronous.

If neither the `-async` argument nor the `-sync` argument is specified, the reset is assumed to be an asynchronous reset.

Any combinational gate in the data transfer path between flip-flops at the clock domain crossing or the data paths around the synchronizing flip-flops is allowed if one of the inputs to the gate is a signal declared as synchronous reset using the `reset` constraint with the `-sync` argument.

All advanced clock rules and the `Reset_Check03` and `Reset_Check04` rules also work with synchronous resets and therefore require you to specify these signals using the `reset` constraint with the `-sync` argument.

**-value <0 | 1>**

(Optional) Specifies the reset value as 1 or 0.

If the value is specified as 0, the reset is active low. Otherwise, the reset is active high.

If the `-value` option is not specified, the SpyGlass TXV solution assumes the reset to be active high for auto initialization.

**-soft**

**NOTE:** *This option is not applicable for the SpyGlass DFT DSM solution.*

(Optional) Specifies if the reset is a soft reset.

If this option is specified in the `reset` constraint, the reset signal is marked as a soft reset; otherwise it is marked as a hard reset.

Active state of the reset is used during initialization irrespective of hard or soft reset. However, if reset is defined as hard, it will be deactivated during functional analysis while soft reset will be used as other signals during functional analysis.

**NOTE:** *The -soft argument is only applicable to SpyGlass CDC formal rules. This argument has no impact on SpyGlass CDC structural rules. If a reset is mentioned by using the -soft argument, that reset is used for initialization and then ignored for functional analysis.*

**For SpyGlass TXV Solution**

The SpyGlass TXV solution honors the SGDC `reset` constraint as follows:

- If you do not provide the `-soft` option, the SpyGlass TXV solution deactivates the corresponding reset signal during verification by placing “~ (active-value)” in it. You can specify the `-soft` option to stop the deactivation so that the signal is considered during verification.
- The `-sync/-async` options do not have any impact in the verification. The SpyGlass TXV solution deactivates all reset signals unless the `-soft` option is specified.

## Rules

The `reset` constraint is used by the following rules:

<b>SpyGlass Auto Verify Solution</b>			
All rules			
<b>SpyGlass CDC solution</b>			
Clock_Reset_check01	Clock_Reset_Info01	Propagate_resets	Reset_check03
Reset_check04	Reset_check06	Reset_check10	Reset_check11
Reset_sync02	Reset_sync03	Reset_sync04	Reset_info02
Ac_cdc01a	Ac_cdc01b	Ac_cdc01c	Ac_cdc08
Ac_fifo01	Ac_handshake01	Ac_handshake02	Ac_conv01
Ac_conv02	Ac_conv03	Ar_resetcross_matrix01	Reset_sync01
Setup_quasi_static01	Ac_unsync01	Ac_unsync02	Ac_conv04
Ac_conv05	Ar_resetcross01		
<b>SpyGlass DFT DSM Solution</b>			
PLL_02			
<b>SpyGlass ERC Solution</b>			
resetPinConnectedToResetNet			
<b>SpyGlass TXV Solution</b>			
All rules			

## reset -async

## Purpose

**NOTE:** `reset -async` is an alias for the `reset` constraint used by the SpyGlass DFT solution.

The `reset -async` constraint is used to specify asynchronous set or reset pins in test mode as used by the `Async_10` rule of the SpyGlass DFT solution.

## Product

SpyGlass DFT solution

## Syntax

```
reset -async
      -name <port-name>
```

## Arguments

**-name <port-name>**

The asynchronous port name.

## Rules

The `reset -async` constraint is used by the following rules:

---

### SpyGlass DFT Solution

---

<code>Async_10</code>	<code>Info_blackboxDriver</code>
-----------------------	----------------------------------

---

## reset\_filter\_path

### Purpose

The `reset_filter_path` constraint specifies reset paths so that the reset crossings on these paths are ignored from SpyGlass analysis.

For example, the `Ar_resetcross01` and `Ar_sync_group` rules will not report violations for the destination reset and destination clock specified by this constraint.

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `reset_filter_path` constraint is as follows:

```
current_design <du-name>
reset_filter_path
  [-from_rst <frm-rst-list>]
  [ -to_rst <to-rst-list> ]
  [ -to_clock <destination-clock-obj-list> ]
  [ -from_clock <source-clock-obj-list> ]
  [ -clock <destination-clock-obj-list> ]
  [ -from_obj <from-obj-list> ]
  [ -to_obj <to-obj-list> ]
  [ -type <rdc | sync | deassert | reset_sync02> ]
  [ -no_des_rst ]
```

### Arguments

**-from\_rst <frm-rst-list>**

Space-separated list of objects (hierarchical net, pin, or port) of a source such that the reset crossings containing such sources are not considered for SpyGlass analysis.

**-to\_rst <to-rst-list>**

Space-separated list of objects (hierarchical net, pin, or port) of a destination such that the reset crossings containing such destinations are not reported.

**-from\_clock <source-clk-list>**

Space-separated list of source clocks.

**-to\_clock <dest-clk-list>**

Space-separated list of destination clocks.

**-no\_des\_rst**

Optional argument to waive sequential instances that are involved in RDC but do not have any resets.

**-from\_obj <from-obj-list>**

Space-separated list of hierarchical nets, pins, or ports of the source in the reset crossing so that the reset crossings containing such sources are not reported.

**-to\_obj <to-obj-list>**

Space-separated list of hierarchical nets, pins, or ports of the destination in the reset crossing so that the reset crossings containing such destinations are not reported.

**-clock <destination-clock-obj-list>**

(Optional) Space-separated list of destination clocks. Note that the `-clock` argument behaves the same way as the `-to_clock` argument.

**-type <rdc | sync | deassert | reset\_sync02>**

(Optional) The default value is `rdc`. This value is for backward compatibility. This option is applicable only for the `Ar_resetcross01` rule.

Set this argument to `deassert` to filter the `Ar_asyncdeassert01` and `Ar_syncdeassert01` rule violations reported for the reset paths specified by this constraint.

Set this argument to `sync` to filter the `Ar_unsync01` and `Ar_sync01` rule violations reported for the reset paths specified by this constraint.

Set this argument to `reset_sync02` to filter the `Reset_sync02` rule violations reported for the reset paths specified by this constraint.

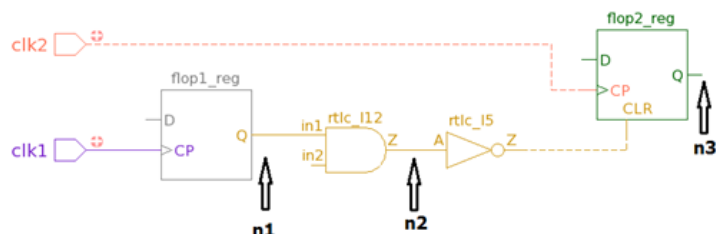
**NOTE:** If you specify the **`sync`** or **`deassert`** value to this argument then you can specify only **`-from_rst`** and **`-clock`** arguments to the `reset_filter_path` constraint.

The `reset_filter_path` constraint supports the following net names in the `-from_rst` argument of the `reset_filter_path` constraint:

- Nets connected to source flop output
- Nets connected to source port
- Reset nets reported in the violation message of `Reset_sync02`

The `reset_filter_path` constraint also supports net connected to destination flop output in the `-to_rst` argument.

For example, consider the schematic shown in [Figure 53](#).



**FIGURE 53.**

In this case, you can specify the `n1` or the `n2` source resets in the `-from_rst` argument and `n3` in the `-to_rst` argument of the `reset_filter_path` constraint to not consider the reset crossings containing such reset nets during SpyGlass analysis.

If any of the nets specified in the `-from_rst`/`-to_rst` do not have the reset constraint specified, set the `allow_unconstrained_reset_in_rfp` parameter to `yes` to consider the unconstrained reset net.

## Handling Multiple Clocks or Resets

For example, if a destination flip-flop receives the R1 and R2 resets and the C1 and C2 clocks, specify the following constraints to filter the *Ar\_unsync01*, *Ar\_sync01*, *Ar\_asyncdeassert01*, and *Ar\_syncdeassert01* rule violations:

```
reset_filter_path -from_rst R1 R2 -clock C1 C2 -type sync
reset_filter_path -from_rst R1 R2 -clock C1 C2 -type deassert
```

## Examples

### Example 1

Consider the following *Ar\_asyncdeassert01* spreadsheet showing violations related to invalid crossings:

A ID	B SOURCE	C SOURCE RESET	D DEST.	E DEST. RESET	F CLOCKS	G WAIVED
<a href="#">2C</a>	top.F1.q	top.r1	top.F2.q	top.r2	top.c	No
<a href="#">2D</a>	top.F3.q	top.r2	top.F4.q	top.r1	top.c	No
<a href="#">33</a>	top1.F7.q	top1.r1	top1.F8.q	top1.r2	top1.c	No
<a href="#">39</a>	top2.F3.q	top2.r1	top2.F4.q	top2.r2	top2.c	No
<a href="#">3A</a>	top2.F5.q	top2.r1	top2.F6.q	top2.r2	top2.c	No

FIGURE 54.

From the above set of violations, if you want to suppress reporting *Ar\_asyncdeassert01* and *Ar\_syncdeassert01* violations from the `top.r1` reset to `top.c` clock, specify the following constraint:

```
reset_filter_path -from_rst r1 -clock c -type deassert
```

After specifying the above constraint, the violations reported in the cells 2C and 33 in [Figure 54](#) are not reported.

## Rules

The `reset_filter_path` constraint is used by the following rules:

---

**SpyGlass CDC Solution and RDC**

---

Ar_unsync01	Ar_sync01	Ar_asyncdeassert01
Ar_sync_group	Ar_syncdeassert01	Ar_resetcross01
RFPSetup	Ar_resetcross_matrix01	

---



## reset\_pin

### Purpose

The `reset_pin` constraint is used to specify black box pins that should be assumed to be reset pins.

Then, the `Async_07` rule of the *SpyGlass DFT* solution that uses asynchronous source analysis treats a pin specified with the `reset_pin` constraints just like it does the reset pins on flip-flops. The source of such a pin must be test mode controlled just as any other asynchronous source.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `reset_pin` constraint is as follows:

```
reset_pin
  -name <du-name>.<port-name>
  [ -value <value> ]
  [ -synchronous ]
  [ -asynchronous ]
```

### Arguments

#### <du-name>

The name of the design unit (black box) for which you are specifying the reset pin.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are processed.

You can specify a single design unit name or a space-separated list of design unit names.

**<port-name>**

Name of the asynchronous reset port on the design unit (black box).

You can specify only a single port name.

**-value <value>**

The active value (0 or 1) for this asynchronous reset port *<port-name>*.

**-synchronous**

Implies that the reset pins are synchronous.

**-asynchronous**

Implies that the reset pins are asynchronous.

**Rules**

The `reset_pin` constraint is used by the following rules:

---

SpyGlass DFT Solution	
Async_07	Async_07Lssd

---

## reset\_synchronizer

**Purpose**

The `reset_synchronizer` constraint is used to specify a reset synchronizer signal along with its asserted reset value.

During SpyGlass analysis, the tool uses this signal as a synchronizing point as if a multi-flop reset synchronizer is present with its initial state value.

**Product**

SpyGlass CDC solution

## Syntax

The syntax of the `reset_synchronizer` constraint is as follows:

```
reset_synchronizer
  -name <synchronizer-name>
  -reset <source-reset>
  -clock <synchronizer-clock> | <tag-name>
  -value <synchronizer-reset-active-value>
  -ignore
```

## Arguments

### **-name <synchronizer-name>**

Specifies the name of the synchronizer output.

You can specify a hierarchical net name, pin name, or top port name to this argument.

**NOTE:** *You can use a combination of wildcard characters, such as \* and ? while specifying the name of a synchronizer output.*

### **-reset <source-reset>**

Specifies the name of the source reset for which `<synchronizer-name>` is acting as a synchronizer.

You can specify a hierarchical net name, pin name, or top port name to this argument.

**NOTE:** *You can use a combination of wildcard characters, such as \* and ? while specifying the name of a reset source.*

### **-clock <synchronizer-clock>**

Specifies the clock of the synchronizer.

You can specify a hierarchical net name, pin name, or top port name to this argument.

**NOTE:** *You can use a combination of wildcard characters, such as \* and ? while specifying the clock of a synchronizer.*

**-clock <tag-name>**

Specifies the tag name specified by the `-tag` argument of the clock constraint.

**-value <synchronizer-reset-active-value>**

Specifies 0 or 1 to indicate the active assertion value of the reset synchronizer provided by the user by using this constraint.

SpyGlass uses this value for de-assertion verification purposes. This value does not have any impact on the `Ar_sync01` and `Ar_unsync01` rules.

**-ignore**

Specifies that all the synchronizers from the object specified in the `-name` argument be ignored for synchronization of reset crossings.

**NOTE:** *If you use the `-ignore` argument along with the `-name` argument, the other arguments of the `reset_synchronizer` constraint are not required.*

**Rules**

The `reset_synchronizer` constraint is used by the following rules:

SpyGlass CDC Solution		
Ar_sync01	Ar_unsync01	Ar_syncdeassert01
Ar_asyncdeassert01	Reset_sync02	Reset_check07
Reset_check10		

**retention\_cell****Purpose**

Specifies the retention latch cells as used by the LPSVM33, LPSVM33A, LPSVM34, LPSVM56, LPSVM57, LPSVM58, LPSVM59 rules or the SRPG cells as used by the LPSVM38 and LPPLIB10 rules of the SpyGlass Power Verify solution.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `retencell`.*

## Product

SpyGlass Power Verify solution

## Syntax

### Defining Retention Latch Cells

The syntax to specify the `retention_cell` constraint for LPSVM33, LPSVM33A, LPSVM34, and LPSVM58 rules of the *SpyGlass Power Verify* solution is as follows:

```
current_design <du-name>
  retention_cell
    -name <cell-name-list>
    [ -save <save-pin-name> ]
    [ -restore <restore-pin-name> ]
    [ -clk <clk-pin-name> ]
    [ -clkval <0 | 1> ]
    [ -qTerm <q-pin-name> ]
    [-memory]
```

### Defining SRPG Cells

The syntax to specify the `retention_cell` constraint for the LPSVM38 and LPPLIB10 rules of the *SpyGlass Power Verify* solution is as follows:

```
current_design <du-name>
  retention_cell
    -name <cell-name-list>
    [ -domains <domain-name-list> ]
    [ -vddpin <vdd-pin-name> ]
    [ -vddcpin <vddc-pin-name> ]
```

### Defining Retention Cells with Control Pins

The syntax to specify the `retention_cell` constraint for LPSVM56, LPSVM57 and LPSVM59 rules of the *SpyGlass Power Verify* solution is as follows:

```
current_design <du-name>
  retention_cell
    -name <cell-name-list>
```

```
[ -sleep <sl-pin-name> ]
[ -save <sa-pin-name> ]
[ -restore <rst-pin-name> ]
[ -sleepval <0 | 1> ]
[ -saveval <0 | 1> ]
[ -restoreval <0 | 1> ]
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying retention latch cells, SRPG cells, or retention cells with control pins.

**-name <cell-name-list>**

Space-separated name list of the retention latch cells or SRPG Cells or retention cells with control pins.

**-save <save-pin-name> | -restore <restore-pin-name> | -clk <clk-pin-name> | -qTerm <q-pin-name>**

Names of the save pin, restore pin, clock pin, and the output pin of the retention latch cell, respectively.

The LPSVM33 rule of the *SpyGlass Power Verify* solution uses the output pin name specified using the `-qTerm` argument. If you do not specify the `-qTerm` argument, the LPSVM33 rule assumes the output pin name to be Q.

The LPSVM33A rule of the *SpyGlass Power Verify* solution uses only the retention cell names.

The LPSVM34 rule of the *SpyGlass Power Verify* solution uses the save, restore, and clock pin names specified using the `-save`, `-restore`, and `-clk` arguments, respectively. You can specify any combination of these arguments. If you do not specify any of these arguments, the LPSVM34 rule does not perform any checks.

**-clk <clk-pin-name> | -clkval <0 | 1>**

The LPSVM58 rule of the *SpyGlass Power Verify* solution uses the `-clk`

argument defining the clock pin of the retention cell and the `-clkval` argument defining the active value of that clock.

### **-memory**

The LPSVM57 rule of the *SpyGlass Power Verify* solution uses the `-memory` argument to define the retention memories. Other retention cell violations are not reported when this Boolean variable is given with the `retention_cell` constraint.

### **-domains <domain-name-list>**

Space-separated voltage domain name list.

### **-vddpin <vdd-pin-name> | -vddcpin <vddc-pin-name>**

Names of the Vdd pin (the pin to be connected to the OFF supply) and the Vddc pin (the pin to be connected to the always-on supply) of the SRPG cell, respectively. These values are used by the LPPLIB10 rule of the *SpyGlass Power Verify* solution.

### **-sleep <sl\_pin-name> | -save <sa-pin-name> | -restore <rst-pin-name>**

Name of the sleep pin, save pin, and restore pin.

### **-sleepval <0|1> | -saveval <0|1> | -restoreval <0|1>**

The `-sleepval`, `-saveval`, and `-restoreval` arguments specify the active value of the respective control pins.

The LPSVM59 rule of the *SpyGlass Power Verify* solution uses the `-sleep` argument defining the sleep pin of the retention cell and the `-sleepval` argument defining the active value of that sleep pin.

The LPSVM59 rule of the *SpyGlass Power Verify* solution also uses `-save` argument defining the save pin of the retention cell and the `-saveval` argument defining the active value of that save pin. The rule also uses `-restore` argument defining the restore pin of the retention cell and the `-restoreval` argument defining the active value of that restore pin.

## **Rules**

The `retention_cell` constraint is used by the following rules:

SpyGlass Power Verify Solution			
LPSVM33	LPSVM33A	LPSVM34	LPSVM56
LPSVM57	LPSVM58	LPSVM59	LPSVM38
LPPLIB10			

## retention\_instance

### Purpose

Specifies the hierarchy information of retention cell instances.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was retain\_instance.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `retention_instance` constraint is as follows:

```
current_design <du-name>
  retention_instance
    -name <inst-name-list>
```

### Arguments

#### <du-name>

Name of the design unit under which you are specifying the retention cell instances.

#### -name <inst-name-list>

Space-separated hierarchical instance name list (name of any instance or the top-level design unit) indicating the regions to be checked.

**NOTE:** *Wildcard support is provided for -name argument of the*



`retention_instance` *constraint*.

## Rules

The `retention_instance` constraint is used by the following rules:

---

### SpyGlass Power Verify Solution

---

LPSVM38	LPSVM58
---------	---------

---

## rme\_config

### Purpose

The `rme_config` constraint is used to configure the output of RME.

### Product

SpyGlass DFT solution

### Syntax

```
rme_config
  -module_prefix <module-prefix>
  | -module_suffix <module-suffix>
  | -instance_prefix <instance-prefix>
  | -instance_suffix <instance-suffix>
  | -port_prefix <port-prefix>
  | -wire_prefix <wire-prefix>
  | -file_prefix <file-prefix>
  | -file_suffix <file-suffix>
  | -tm_port <tm-port-name>
  | -tclk_port <tclk-port-name>
  | -logic_at_top
  | -insert_disable_logic
  | -port_type <port_type>
  | -wire_naming_style <wire_naming_style>
  | -delay_set_reset_type <delay_set_reset_type>
  | -delay_clock_type <delay_clock_type>
```

```
| -verilog_delay_text <verilog_delay_string>  
| -vhdl_delay_text <vhdl_delay_string>  
| -insert_undef_in_clone_module <undef_value>  
| -obs_tp_module_name <module_name>  
| -cnt_tp_module_name <module_name>  
| -cnt0_tp_module_name <module_name>  
| -cnt1_tp_module_name <module_name>
```

## Arguments

### **-module\_prefix <module-prefix>**

Specifies the prefix of modified design modules and entities.

For example, if you want to specify the prefix of modified design modules as `atrenta_modified_`, specify the `rme_config` constraint as follows:

```
rme_config -module_prefix atrenta_modified_
```

The default value is "".

**NOTE:** *The -module\_prefix argument of the rme\_config constraint is not applicable for the design top module.*

### **-module\_suffix <module-suffix>**

Specifies the suffix of modified design modules and entities.

The default value is "".

**NOTE:** *The -module\_suffix argument of the rme\_config constraint is not applicable for the design top module.*

### **-instance\_prefix <instance-prefix>**

Specifies the prefix of modified instances.

The default value is "".

### **-instance\_suffix <instance-suffix>**

Specifies the suffix of modified instances.

The default value is "".

**-port\_prefix <port-prefix>**

Specifies the prefix of new ports that are added to design modules and instances.

For input ports, the default value is `atrenta_iport_`. For output ports, the default value is `atrenta_oport_`.

**-wire\_prefix <wire-prefix>**

Specifies the prefix of new wires that are added to design modules.

The default value is `atrenta_wire`.

**-file\_prefix <file-prefix>**

Specifies the prefix of modified and newly generated files.

For modified files, the default value is `atrenta_modified_`. For newly generated files, the default value is `atrenta_generated_`.

The user-specified prefix is applied only if the following conditions are satisfied:

- The RTL file contains description for single design module only.
- The name of RTL file (without extension) is the same as that of the design module.

**-file\_suffix <file-suffix>**

Specifies the suffix of modified and newly generated files.

The default value is `""`.

**-tm\_port <tm-port-name>**

Test mode port name. This argument is specific to the SpyGlass DFT solution.

**-tclk\_port <tclk-port-name>**

Test clock port name. This argument is specific to the SpyGlass DFT solution.

**-logic\_at\_top**

Specifies that all glue logic should be created in the top module. This

argument is specific to the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

#### **-insert\_disable\_logic**

Specifies that disable logic should be inserted. This argument is specific to the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

#### **-port\_type <port\_type>**

Specifies the data type of port in V2K and SystemVerilog modules. Possible values are `wire` and `reg_wire`.

#### **-wire\_naming\_style <wire\_naming\_style>**

Specifies the naming style of wires inserted by the AutoFix feature.

The possible values are as follows:

<b>Value</b>	<b>Naming Style of Inserted Wires</b>
<code>prefix_only</code>	<code>&lt;prefix&gt;_&lt;count&gt;</code>
<code>prefix_and_pin</code>	<code>&lt;prefix&gt;_&lt;driver_instance_pin&gt;_&lt;count&gt;</code>

#### **-delay\_set\_reset\_type <delay\_set\_reset\_type>**

Specifies whether the flip-flop inserted by the AutoFix feature is a set flip-flop or a reset flip-flop. In addition, it also specifies the polarity of the set/reset pin of the flip-flop.

The possible values are as follows:

<b>Value</b>	<b>Type of the flip-flop</b>
<code>active_high_set</code>	An active high set flip-flop
<code>active_low_set</code>	An active low set flip-flop
<code>active_high_reset</code>	An active high reset flip-flop
<code>active_low_reset</code>	An active low reset flip-flop
<code>active_high_set_reset</code>	An active high set/reset flip-flop
<code>active_low_set_reset</code>	An active low set/reset flip-flop

**-delay\_clock\_type <delay\_clock\_type>**

Specifies the polarity of the clock of the flip-flop inserted by the AutoFix feature.

The possible values are as follows:

Value	Type of the clock
active_high	An active high clock
active_low	An active low clock

**-verilog\_delay\_text <verilog\_delay\_string>**

Specifies the delay string to be inserted in a flip-flop definition in Verilog. The string is inserted in the following format:

```
q <= <verilog_delay_string> d;
```

Where *q* and *d* are the output pin and the input pin, respectively, of the flip-flop.

For example, if you want to add a numeric delay in a flip-flop definition in Verilog, specify the delay string as follows:

```
-verilog_delay_text = "# 10"
```

In this case, the modified RTL of the flip-flop will be as follows:

```
q <= # 10 d;
```

As another example, if you want to add a tick define delay in a flip-flop definition in Verilog, specify the delay string as follows:

```
-verilog_delay_text = "# `MY_DELAY"
```

In this case, the modified RTL of the flip-flop will be as follows:

```
q <= # `MY_DELAY d;
```

Please note that you have to specify a proper declaration of `MY_DELAY` in this case.

**-vhdl\_delay\_text <vhdl\_delay\_string>**

Specifies the delay string to be inserted in a flip-flop definition in VHDL. In addition, the `after` keyword is also inserted in the flip-flop definition.

The delay string and the `after` keyword are inserted in the following format:

```
q <= d after <vhdl_delay_string>;
```

Where, `q` and `d` are the output pin and the input pin, respectively, of the flip-flop.

For example, if you want to add a numeric delay in a flip-flop definition in VHDL, specify the delay string as follows:

```
-vhdl_delay_text = "10ns"
```

In this case, the modified RTL of the flip-flop will be as follows:

```
q <= d after 10ns;
```

As another example, if you want to add a tick define delay in a flip-flop definition in VHDL, specify the delay string as follows:

```
-vhdl_delay_text = "MY_DELAY"
```

In this case, the modified RTL of the flip-flop will be as follows:

```
q <= d after MY_DELAY;
```

Please note that you have to specify a proper declaration of `MY_DELAY` in this case.

#### **-insert\_undef\_in\_clone\_module <undef\_value>**

Specify to add the ``undef` of macros in the Atrenta uniquified blocks. The possible values are 0 & 1.

If `undef_value` is 1, then ``undef` macros are inserted in Atrenta uniquified blocks. Default `undef_value` is 0.

#### **-obs\_tp\_module\_name <module\_name>**

Specifies the module name of the observe testpoint.

#### **-cnt\_tp\_module\_name <module\_name>**

Specifies the module name of the control testpoint.

#### **-cnt0\_tp\_module\_name <module\_name>**

Specifies the module name of the control-0 testpoint.

**-cnt1\_tp\_module\_name** <module\_name>

Specifies the module name of the control-1 testpoint.

## Rules

The `rme_config` constraint is used by the following rules:

SpyGlass DFT Solution			
Async_07	Clock_11	Clock_11_capture	Latch_08
TA_06	Topology_01		

## set\_slew

### Purpose

The `set_slew` constraint is used to specify slew value of all the nets of a clock domain of type data or clock. Apart from the `set_slew` constraint, the slew value can be specified in various ways. The following list shows the order from the highest priority to the lowest priority:

1. The `set_annotated_transition` SDC constraint
2. The `set_slew` SGDC constraint
3. The value from SpyGlass Physical in the SpyGlass Physical - Power Estimate flow
4. The `pe_auto_infer_slew` parameter
5. The `pe_slew` parameter

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax of the `set_slew` constraint is as follows:

```
set_slew
  -value <val>
```

```
[-clock <clock-net-name>]
[-type <data | clock>]
[-rise]
[-fall]
```

**NOTE:** *If both -rise and -fall arguments are defined, the value is considered as both the value of the rise slew and the fall slew.*

## Arguments

**-value <val>**

Specifies positive float value with the unit of time. By default, the unit is nanoseconds.

**-clock <clock-net-name>**

Specifies the clock net name. List is supported.

**-type <data | clock>**

Specifies the type of net.

**-rise**

Specifies the rise slew value.

**-fall**

Specifies the fall slew value.

## Examples

### Example 1

In the following SGDC specification, the value v1 is applicable for all the nets of the design.

```
set_slew -value v1
```



**Example 2**

You can overwrite the value for clock or data nets by specifying the type argument. In the following SGDC specification, the value `v2` is applicable to all the data nets of the design.

```
set_slew -value v2 -type data
```

**Example 3**

In the following SGDC specification, the value `v3` is applicable for all the clock nets of the clock domain `c1`

```
set_slew -value v3 -type clock -clock c1
```

**Example 4**

In the following SGDC specification, value `v1` is applied to nets of type clock and value `v2` is applied to nets of type of data. The value `v3` is not applied to any net and is ignored.

```
set_slew -value v1 -type clock -clock clk1
```

```
set_slew -value v2 -type data -clock clk1
```

```
set_slew -value v3 -clock clk1
```

**Example 5**

In the following SGDC specification, the first `set_slew` SGDC specification is taken. Therefore, value `v1` is applied. All other SGDC constraint specifications of the same constraint are ignored.

```
set_slew -type data -value v1
```

```
set_slew -type data -value v2
```

**Example 6**

For nets in a merged clock domain, the value specified for the fastest clock domain is used. Suppose, clock net `n1` is a clock net in a merged clock domain of clocks `c1` and `c2`. In addition, the clock `c1` is faster than clock `c2`. In the following specification, the value `v1` is taken.

```
set_slew -clock c1 -value v1
```

```
set_slew -clock c2 -type clock -value v2
```

## Rules

The `set_slew` constraint is used by the following rules:

---

### SpyGlass Power Estimation and SpyGlass Power Reduction Solutions

---

All rules

---

## force\_scan

### Purpose

The `force_scan` constraint is used to declare flip-flops as scannable even if they do not so qualify.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `force_scan` constraint is as follows:

```
force_scan
  -name <du-name> | <reg-name> |
  -clock_control <signal-name> |
  -set_control <signal-name> |
  -reset_control <signal-name>
  -register_suffix <suffixes>
  -module_suffix <suffixes>
  [-latch]
  [-flip_flop]
```

**NOTE:** *The `force_scan` constraint supports wildcard characters. Using wildcards, expression is expanded only within the hierarchy.*

### Arguments

**-name <du-name>**

Specifies the name of the design unit from which scan is included.

You can specify design units that are single flip-flops or design units where one or more flip-flops are described besides other logic. Then, all flip-flops in the specified design unit are included in scan.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered.

You can specify a single design unit name or a space-separated list of design unit names.

#### **-name <reg-name>**

The name of a net that is connected to the output pin of a flip-flop.

Then, the corresponding flip-flop is included in scan.

You can specify a simple net name or a hierarchical net name. The net specified as simple net name is searched at the top-level.

You can specify a single net name or a space-separated list of net names.

**NOTE:** *You can specify design unit names, net names, or a combination of both.*

#### **<signal-name>**

Flip-flops whose control pins (clock, set, or reset) are driven by this signal are included in the scan.

**NOTE:** *The traversal involved in this is structural only.*

The *<signal-name>* argument can have the following values:

- **CLOCK:** For the *clock\_control* option, if the *<signal-name>* argument is driving CLOCK pin of the flip-flop.
- **SET:** For the *set\_control* option, if the *<signal-name>* argument is driving SET pin of the flip-flop.
- **RESET:** For the *reset\_control* option, if the *<signal-name>* argument is driving RESET pin of the flip-flop.

#### **-register\_suffix <suffixes>**

Space-separated list of suffixes to be specified as scannable. The *-register\_suffix* argument should not be used along with other arguments of the *force\_scan* constraint, that is, *-name*, *-clock\_control*, *-set\_control*, or *-reset\_control*.

If the value of the `dft_treat_suffix_as_pattern` 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.

If the value of the `dft_check_path_name_for_register_suffix` parameter is `on`, the value of the `-register_suffix` field will be matched with the register name along with the path in which the register is present.

#### **-module\_suffix <suffixes>**

Define this field to use suffix based pattern match for all module names.

If the value of the `dft_treat_suffix_as_pattern` parameter is `on`, the value of the `-module_suffix` field will be matched with the module name along with the path in which the module is present.

#### **-latch**

Marks only latches as scannable.

#### **-flip\_flop**

Marks only flip-flops as scannable.

**NOTE:** *If you do not specify either `-latch` or `-flip-flop` options, then, both latches and flip-flops are marked as scannable.*

## **Examples**

You can use the `force_scan` constraint in the following ways:

### **Specifying only the design unit names with the `-name` argument**

By specifying a design unit name using the `-name` argument only, all instances of this design unit are considered scannable. The following `force_scan` constraint indicates that all flip-flops within all instances of `modName1` will be considered scannable:

```
force_scan -name modName1
```

### **Specifying only the register names with the `-name` argument**

By specifying a net name using the `-name` argument only, the corresponding flip-flop will be considered scannable. The following

`force_scan` constraint indicates that the flip-flop whose output pin is connected to net `reg_123` (at the top-level) will be considered scannable:

```
force_scan -name reg_123
```

### Specifying list of suffixes using the `-register_suffix` argument

Consider the following example:

```
current_design top_scan
  force_scan -register_suffix ff1
```

In the above example, All flip-flops with name ending with `ff1` will be marked as scan.

## Rules

The `force_scan` constraint is used by the following rules:

---

### SpyGlass DFT Solution

---

All rules

---

## force\_stable\_value

### Purpose

The `force_stable_value` constraint specifies nodes that will have a stable value during the scan shift and capture modes.

### Product

SpyGlass DFT solution

### Syntax

The syntax for the `force_stable_value` constraint is as follows:

```
force_stable_value
  [-name <nodename> ]
  [ -except <except_nodename> ]
  [ -type <DO_nodename> ]
  [ -except_type <exceptDO_nodename> ]
```

```
[ -value <value> ]
[ -tag <condname> | -use_shift | -use_capture |
  -use_captureATspeed ]
```

## Arguments

### **-name <nodename>**

(Optional) The name can be a top-module port, or any internal net name, or terminal name. More than one pin name can be specified, and it is effectively read as a concise description of as many individual value checks.

**NOTE:** *Specify either -name or -type argument.*

### **-except <except\_nodename>**

(Optional) Same as *<nodename>* but defines design nodes that are not to be used as name.

### **-type <DO\_nodename>**

Same as *<nodename>* but it takes only macros as inputs.

**NOTE:** *Specify either -name or -type argument.*

To view the list of macros supported by the `force_stable_value` constraint, see [Supported Macros](#).

### **-except\_type <exceptDO\_nodename>**

Same as *<DO\_nodename>* but it takes only macros as inputs.

### **-value <value>**

The value is a logic value string of 0, 1, X, Z, 1\_or\_0, and 0\_or\_1. A single-bit value means check at end of complete simulation. The X value is treated as do-not-compare. A multi-bit value means check on cycle-by-cycle simulation basis. For specification of a vector value, SGDC multi-bit specification format (same as used for the `test_mode` constraint) should be used.

You can specify repeat sequences for the `force_stable_value` constraint.

For fields that require repeat sequence, you can specify the values as `<I*S>`. Here, S is any string that does not contain the `<`, `>`, and `*` characters. However, S can contain another `<I*S>` expression. I is an integer that is always interpreted as a decimal value. The expression `<I*S>` means that the sequence S will be repeated I number of times.

### **-tag <condname>**

(Optional) A condition previously defined by using the [define\\_tag](#) constraint. It describes a stimulation condition.

Note that only one condition name can be defined in a `force_stable_value` specification. However, simulation for a given condition name simulates all pin-value specifications simultaneously. The built-in power-ground connections are also simulated in this process.

### **-use\_shift | -use\_capture | -use\_captureATspeed**

For any of these modifiers, `force_stable_value` simulates test mode of that particular mode.

If `-use_shift`, `-use_capture`, or `-use_captureATspeed` argument is specified, the constraint simulates all, shift, capture, or captureATspeed test\_mode constraints, respectively.

**NOTE:** *If more than one of the -tag, -use\_shift, -use\_capture, or -use\_captureATspeed arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with require\_stable\_value constraint.*

## **Examples**

### **Rules**

The `force_stable_value` constraint is used by the following rule:

---

**SpyGlass Connectivity Verify Solution**

---

Soc\_14

---

## **force\_unstable\_value**

## Purpose

The `force_unstable_value` constraint specifies nodes that will have an unstable value during the scan shift and capture modes.

## Product

SpyGlass DFT solution

## Syntax

The syntax for the `force_unstable_value` constraint is as follows:

```
force_unstable_value
  [-name <nodename> ]
  [ -except <except_nodename> ]
  [ -type <DO_nodename> ]
  [ -except_type <exceptDO_nodename> ]
  [ -value <value> ]
  [ -tag <condname> | -use_shift | -use_capture |
    -use_captureATspeed ]
```

## Arguments

### **-name <nodename>**

(Optional) The name can be a top-module port, or any internal net name, or terminal name. More than one pin name can be specified, and it is effectively read as a concise description of as many individual value checks.

**NOTE:** *Specify either -name or -type argument.*

### **-except <except\_nodename>**

(Optional) Same as `<nodename>` but defines design nodes that are not to be used as name.

### **-type <DO\_nodename>**

Same as `<nodename>` but it takes only macros as inputs.

**NOTE:** *Specify either -name or -type argument.*



To view the list of macros supported by the `force_unstable_value` constraint, see [Supported Macros](#).

**-except\_type <exceptDO\_nodename>**

Same as `<DO_nodename>` but it takes only macros as inputs.

**-value <value>**

The value is a logic value string of 0, 1, X, Z, 1\_or\_0, and 0\_or\_1. A single-bit value means check at end of complete simulation. The X value is treated as do-not-compare. A multi-bit value means check on cycle-by-cycle simulation basis. For specification of a vector value, SGDC multi-bit specification format (same as used for the `test_mode` constraint) should be used.

You can specify repeat sequences for the `force_unstable_value` constraint.

For fields that require repeat sequence, you can specify the values as `<I*S>`. Here, S is any string that does not contain the `<`, `>`, and `*` characters. However, S can contain another `<I*S>` expression. I is an integer that is always interpreted as a decimal value. The expression `<I*S>` means that the sequence S will be repeated I number of times.

**-tag <condname>**

(Optional) A condition previously defined by using the [define\\_tag](#) constraint. It describes a stimulation condition.

Note that only one condition name can be defined in a `force_unstable_value` specification. However, simulation for a given condition name simulates all pin-value specifications simultaneously. The built-in power-ground connections are also simulated in this process.

**-use\_shift | -use\_capture | -use\_captureATspeed**

For any of these modifiers, `force_unstable_value` simulates test mode of that particular mode.

If `-use_shift`, `-use_capture`, or `-use_captureATspeed` argument is specified, the constraint simulates all, `shift`, `capture`, or `captureATspeed` test\_mode constraints, respectively.

**NOTE:** *If more than one of the -tag, -use\_shift, -use\_capture, or*

*-use\_captureATspeed arguments is specified, an error condition occurs. You should specify exactly one of these modifiers with require\_stable\_value constraint.*

## Examples

Currently Unavailable

## Rules

The `force_unstable_value` constraint is used by the following rule:

---

**SpyGlass Connectivity Verify Solution**

---

Soc\_14

---

## scan\_cell

### Purpose

The `scan_cell` constraint is used to define scan cell design units and their data pins as used by the Scan\_18 rule.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `scan_cell` constraint is as follows:

```
scan_cell
  -module <du-name>
  -datapin <dpin-name>
```

### Arguments

**-module <du-name>**

The name of the scan cell design unit.

The design unit name *<du-name>* can be specified as module name (for

Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered.

You can only specify a single design unit name.

**-datapin** <dpin-name>

Name of the data pin of the scan cell design unit.

## Rules

The `scan_cell` constraint is used by the following rule:

---

### SpyGlass DFT Solution

---

Scan\_18

---

## scan\_chain

### Purpose

The `scan_chain` constraint is used to specify the scan chains for the rules mentioned in the [Rules](#) section.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `scanchain`.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `scan_chain` constraint is as follows:

```
scan_chain
  [ -module <name> ]
  -scanin <pin-name>
  -scanout <pin-name>
  [ -scanin_clock_pin <pin-name> ]
  [ -scanout_clock_pin <pin-name> ]
  [ -scanin_clock_pin_phase_inverted ]
```

```
[ -scanout_clock_pin_phase_inverted ]
```

## Arguments

**-scanin <pin-name> / -scanout <pin-name>**

Complete hierarchical name of starting/ending port/pin.

The starting/ending points can be primary ports as well as internal pins.

**NOTE:** *The -scanin and -scanout arguments support wildcard expressions.*

For primary ports, you can also specify the simple port name as in the following example:

```
current_design top
  scan_chain -scanin in15 -scanout out23 ...
```

**-module <name>**

(Optional) Name of the black box module.

Specifying this option implies that the software should assume a valid scan chain from its specified scan-in through the scan-out port.

**-scanin\_clock\_pin <pin-name>**

(Optional) Specifies the scanin clock pin name.

**-scanout\_clock\_pin <pin-name>**

(Optional) Specifies the scanout clock pin name.

**-scanin\_clock\_pin\_phase\_inverted**

(Optional) Specifies whether the scan-in clock pin phase is inverted or not.

**-scanout\_clock\_pin\_phase\_inverted**

(Optional) Specifies whether the scan-out clock pin phase is inverted or not.

**NOTE:** *The SpyGlass DFT product honors the -scanin\_clock\_pin, -scanout\_clock\_pin, -scanin\_clock\_pin\_phase\_inverted, and -scanout\_clock\_pin\_phase\_inverted arguments only when you have specified a black-box module using the -module argument.*

## Rules

The `scan_chain` constraint is used by the following rules:

<b>SpyGlass DFT Solution</b>			
Info_schain	Info_stilFile	Diagnose_ScanChain	Scan_22
Scan_24	Scan_25	Scan_26	Scan_29
Scan_32	Scan_33	Scan_34	Scan_35
Scan_36	Scan_38	Scan_39	Scan_40
Scan_41	Topology_15	dftSGDCSTX_069	
<b>SpyGlass DFT DSM Solution</b>			
SP_01	SP_05		
<b>SpyGlass Built-in Solution</b>			
SGDC_schain	SGDC_schain0	SGDC_schain03	
01	2		

## Examples

The following examples show the usage of the `scan_chain` constraint.

### Example 1

Consider the following example:

```
module test (input [1:0] si, output [1:0] so);
scan_chain -scanin si -scanout so
```

The above example will be expanded as follows:

```
scan_chain -scanin si[1] -scanout so[1]
scan_chain -scanin si[0] -scanout so[0]
```

### Example 2

Consider the following example:

```
module test (input [19:0] scanin_bus, output [63:0]
scanout_bus);
scan_chain -scanin_bus[15:0] -scanout_bus[31:16]
```

The above example will be expanded as follows:

```
scan_chain -scanin scanin_bus[15] -scanout scanout_bus[31]
scan_chain -scanin scanin_bus[14] -scanout scanout_bus[30]
scan_chain -scanin scanin_bus[13] -scanout scanout_bus[29]
...
scan_chain -scanin scanin_bus[1] -scanout scanout_bus[17]
scan_chain -scanin scanin_bus[0] -scanout scanout_bus[16]
```

### Example 3

Consider the following example:

```
scan_chain -module bbox_scan_chain -scanin si1 -scanout so1
-scanin_clock_pin si1clk -scanout_clock_pin so1clk_inv
-scanin_clock_pin_phase_inverted false
-scanout_clock_pin_phase_inverted true
```

In the above example, the black box module, `bbox_scan_chain`, has a scan chain from its scanin pin, `si1`, to its scanout pin, `so1`. In this case, the clock pin of scan in flop is driven by pin, `si1clk`, with no phase inversion. Whereas, the scanout flop clock pin is driven by pin, `so1clk_inv`, with inversion.

## scan\_enable\_source

### Purpose

This constraint is used to define the source signal for scan chain enables. Such sources are then used by various Scan Enable rules.

If this constraint is not used, potential nodes are determined as candidates for use in a `scan_enable_source` constraint. In that case, Scan Enable rules will use these potential nodes as the basis for checking rule compliance.

### Product

SpyGlass DFT DSM solution

### Syntax

The syntax of the `scan_enable_source` constraint is as follows:

```
scan_enable_source
  -name <SE_name>
  [ -active_value <0 | 1> ]
  [ -mode <combinational | sequential> ]
  [ -clock <clock-name> ]
```

### Arguments

**-name <SE\_name>**

Specifies the scan enable name

**-active\_value <0 | 1>**

(Optional) Check that when this `scan_enable` source has `-active_value` (and any specified define tags are also simulated), the scan enable pins on flip-flops fed by this source are enabled for scan.

**-mode <combinational | sequential>**

(Optional) Specifies how to treat the associated source node, that is, whether to consider it as combinational or sequential.

**-clock** <clock-name>

(Optional) Specifies the name of the clock.

## Rules

The `scan_enable_source` constraint is used by the following rules:

---

### SpyGlass DFT DSM Solution

---

All Scan Enable rules

---

## scan\_ratio

### Purpose

The `scan_ratio` constraint is used to determine what fraction of the flip-flops should be scanned or scannable. You can calculate `scan_ratio` using the following formula:

```
scan_ratio = (scannable flip-flops + forced/inferred scan  
flip-flops) / (total number of flip-flops - forced/  
inferred_no_scan flip-flops - synthesis_redundant flip-  
flops)
```

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `scanratio`.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `scan_ratio` constraint is as follows:

```
scan_ratio -value <value>
```



## Arguments

**-value <value>**

A scan ratio as a decimal value less than 1 such as 0.95.

## Rules

The `scan_ratio` constraint is used by the following rule:

---

**SpyGlass DFT Solution**

---

Scan\_11

---

## scan\_type

### Purpose

The `scan_type` constraint is used to specify the SpyGlass DFT type (LSSD or MUXSCAN).

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `scantype`.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `scan_type` constraint is as follows:

```
scan_type
  [ -lssd | -clock | -muxscan | -allscan ]
```

### Arguments

The `scan_type` constraint has the following arguments:

**-lssd**

(Optional) SpyGlass DFT solution scannability type.

The `-lssd` argument specifies that scannability depends only on sets and resets being inactive in the shift mode. Scannability no longer depends on test clock controllability since, in LSSD, a separate test clock that is independent of all system clocks is inserted during scan replacement and synthesis.

**-clock**

The `-clock` argument specifies that scannability depends only on the test clock controllability.

**-muxscan**

The `-muxscan` argument (default) specifies that scannability depends on the test clock controllability and set/reset controllability.

**NOTE:** *If both `-lssd` argument and `-clock` argument are specified, it is assumed that the `-muxscan` argument has been specified.*

**-allscan**

The `-allscan` argument specifies that all flip-flops except those declared as no scan flip-flops (using the `force_no_scan` constraint) are to be considered as scannable.

**Rules**

The `scan_type` constraint is used by the following rules:

<b>SpyGlass DFT Solution</b>			
Async_07Lssd	Async_09	Bist_04	clock_23
Info_uncontrollable	Info_unobservable	Info_undetecCause	Info_coverage
Info_untestable	Info_unused	Info_noscanFlopsTextReport	Info_scanchain
Coverage_audit	Latch_15	Scan_08	Scan_11
Scan_16	Scan_17	Scan_18	Scan_19

---

**SpyGlass Design Constraints**

Scan_20	Scan_21	Scan_22	Scan_24
Scan_25	Scan_26	TA_06	

## scan\_wrap

### Purpose

The `scan_wrap` constraint is used to specify black box design units or instances that will be designed with scan wrappers.

Use of the `scan_wrap` constraint causes the input pins of the designated black box design unit or instance to be treated as observable, output pins to be treated as controllable and inout pins to be treated as both controllable and observable.

You can use the [dftSetAllIBBScanwrapped](#) parameter to make all the black boxes scan-wrapped.

You can use [force\\_ta](#) constraint to override the controllability and observability of individual pins of the scan wrapped module. Please refer to the [Specifying force\\_ta constraint on the individual pins of the scan wrapped module](#) section in the Examples section for more information.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `scanwrap`.

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The syntax of the `scan_wrap` constraint is as follows:

```
scan_wrap
  -name <du-name> | <inst-name>
  [ -enable <en-pin-list> ]
  [ -envalue <en-value-list> ]
```

**NOTE:** The `scan_wrap` constraint supports wildcard characters.

### Arguments

**-name <du-name>**

The scanwrap design unit (black box) name.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name `<du-name>` can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are considered.

You can specify a single design unit name or a space-separated list of design unit names.

#### **-name <inst-name>**

The scanwrap design unit (black box) hierarchical instance name.

The master design unit of the specified instance must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

You can specify a single instance name or a space-separated list of instance names.

#### **-enable <en-pin-list>**

List of enable pins of design unit. These pins should be activated for scan wrap to be enabled.

#### **-envalue <en-value-list>**

The enable value of the enable pins. It is the list of 0 or 1 values. These values specify the signals that activate the corresponding enable pins.

## Examples

You can use the `scan_wrap` constraint in the following ways:

### **Specifying only the design unit names with the -name argument**

By specifying a black box design unit name using the `-name` argument only, all instances of this design unit are considered as scan-wrapped. The following `scan_wrap` constraint indicates that all instances of `modName1` will be considered scan-wrapped:

```
scan_wrap -name modName1
```

### **Specifying only the instance names with the -name argument**

By specifying a hierarchical instance name using the `-name` argument,

only the corresponding instance will be considered scan-wrapped. The following `scan_wrap` constraint indicates that the instance `BB_I123` (at the top-level) will be considered scan-wrapped:

```
scan_wrap -name BB_I123
```

### Specifying enable pins of the scan wrapped design unit

Consider the following command:

```
scan_wrap -name bb_inst1 -enable en1 en2 -envalue 0 1
```

The above command describes that enable pins, `en1` and `en2`, should have values 0 and 1 respectively for scan wrap to be enabled for `bb_inst1`.

### Specifying force\_ta constraint on the individual pins of the scan wrapped module

Consider the following constraint description file, `test.sgdc`:

```
current_design top
  test_mode -name en -value 1

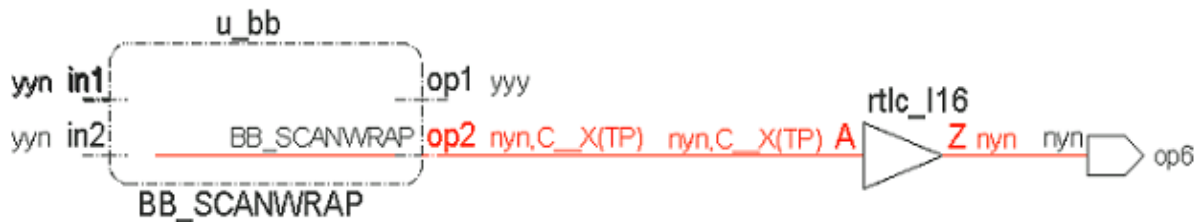
  scan_wrap -name bbox_en      -enable en -envalue 1
  scan_wrap -name non_bbox
  scan_wrap -name bbox

  force_ta -name "bbox::op2" -control nyn
  force_ta -name "bbox::in2" -observe n
```

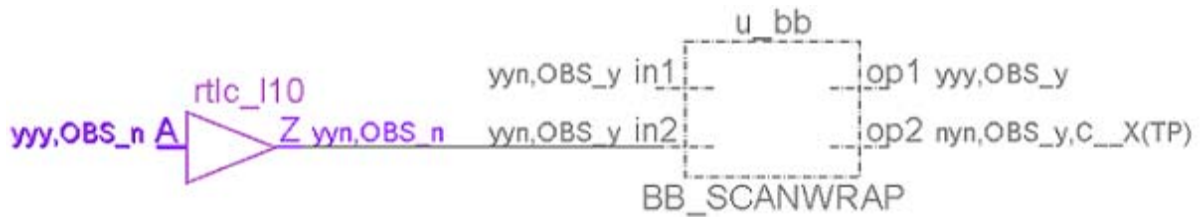
In the above example, `scan_wrap` constraint is applied on the module, `bbox`. Then, `force_ta` constraint is applied on black box instances, `op2` and `in2`. Here, `force_ta` constraint takes precedence over the `scan_wrap` constraint. Therefore, `op2` and `in2` are uncontrollable and unobservable, respectively.

[Figure 55](#) and [Figure 56](#) illustrate the uncontrollable and unobservable conditions after specifying the `force_ta` constraint.

## SpyGlass Design Constraints



**FIGURE 55.** force\_ta: Uncontrollable condition



**FIGURE 56.** force\_ta: Unobservable condition

## Rules

The scan\_wrap constraint is used by the following rules:

<b>SpyGlass DFT Solution</b>			
Async_02_shift	Async_02_capture	Bist_04	Info_uncontrollable
Info_unobservable	Info_undetectedCause	Info_coverage	Info_untestable
Info_unused	Coverage_audit	RAM_01	RAM_02
RAM_03	RAM_04	RAM_05	RAM_06
RAM_07	RAM_08	RAM_09	RAM_10
Scan_17	TA_06		
<b>SpyGlass DFT DSM Solution</b>			
All rules			

## sdc\_data

### Purpose

The `sdc_data` constraint is used to specify the SDC file.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `sdcschema`.*

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions, SpyGlass Constraints solution, SpyGlass DFT DSM solution, SpyGlass CDC solution

## For the SpyGlass Power Estimation and SpyGlass Power Reduction Solutions

### Syntax

The syntax to specify the `sdc_data` constraint is as follows:

```
current_design <du-name>  
  sdc_data -file <file-list>
```

### Arguments

**<du-name>**

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**-file <file-list>**

Space-separated list of SDC files or a single SDC file.

### Rules

The `sdc_data` constraint is used by the following rules:



SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## For SpyGlass Constraints solution

### Syntax

The syntax to specify the `sdc_data` constraint is as follows:

```
current_design <du-name>
  sdc_data -file <file-list>
  [ -level <level> ]
  [ -mode <mode> ]
  [ -corner best | worst ]
```

### Arguments

#### <du-name>

The top-level module name (for Verilog designs) or the top-level entity name (for VHDL designs) or a synthesis partition name specified using the `block` keyword.

#### -file <file-list>

Specifies a list of associated SDC files. You can also specify compressed SDC files in the gzip format with this argument.

If in the SGDC file, multiple SDC files are specified in different SDC schemas with identical values of `mode/corner` options or if no value is specified for these options, SpyGlass reports a violation message. Therefore, the correct way to specify multiple SDC files is based on the following conditions:

- If the SDC files are dependent, specify all the files with a single SDC-schema as shown in the following example:

```
sdc_data -file bar1.sdc bar2.sdc
```

- If the SDC files are independent, specify each file in a different schema with its respective mode as shown in the following example:

```
sdc_data -file bar1.sdc -mode test -corner best
sdc_data -file bar2.sdc -mode func -corner typical
```

If you are specifying a list of SDC files with the `-file` argument, ensure that the SDC files are specified in the correct order so that the definition of a variable is available before it is used.

All of the following arguments are attached to these SDC files to qualify the scope of these SDC files.

If rules that require SDC are run and no `sdc_data` constraint is specified in SGDC, the following fatal message is reported.

```
SGDC_sdc_data02: SDC_data is required to run some of the
selected rule(s).
```

**NOTE:** *The `-type` argument that was used to specify SDC files is replaced by the `-file` argument. While the `-type` argument is still available for backward compatibility, its functionality is expected to change in future.*

### **-level <level>**

(Optional) Specifies the scope of the SDC File type by design phase. You can specify either `rtl`, `prelayout`, `postlayout`, or `all`.

Use this argument to specify the design phase to which a particular SDC file is applicable. In the SpyGlass Constraint solution, most rules work in a specific, pre-defined design phase. Pre-defined design phase is configurable by specifying goals. Therefore, you do not need to specify the level argument.

The rules that work across design phases are: `Const_Struct03` and `Clk_Uncert04`. Therefore, these rules leverage the design phase specified in the `level` argument. The following example illustrates the use of the `level` argument.

```
#Constraints.sgdc
current_design top
sdc_data -file top_function_pre.sdc -level prelayout -mode
function
```

## SpyGlass Design Constraints

```
sd_data -file top_function_post.sdc -level postlayout -mode
function
```

```
sd_data -file top_scan_all.sdc -level all -mode scan
```

```
#top_function_pre.sdc
```

```
set_case_analysis 1 [get_ports "MODE1"]
```

```
set_case_analysis 1 [get_ports "MODE2"]
```

```
#top_function_post.sdc
```

```
set_case_analysis 1 [get_ports "MODE1"]
```

```
set_case_analysis 0 [get_ports "MODE2"]
```

```
#top_scan_all.sdc
```

```
set_case_analysis 1 [get_ports "MODE1"]
```

```
set_case_analysis 1 [get_ports "MODE2"]
```

After you run the Const\_Struct03 rule, violation messages are reported because:

- the `set_case_analysis` settings for the function mode, which is specified in `top_function_pre.sdc`, and scan mode, which is specified in `top_scan_all.sdc`, are identical.
- the `set_case_analysis` settings for the same function mode in different phases, which is `top_function_pre.sdc` file for the Pre-layout phase and `top_function_post.sdc` file for the Post-layout phase, are not the same.

#### **-mode <mode>**

(Optional) Specify the mode name to indicate a functional mode of operation, such as test mode, functional mode, or bypass mode, for the design. The rules in the SpyGlass Constraints solution are independently evaluated for each mode of operation and coverage is generated for each specified mode.

For example, you can specify the functional mode or the scan shift mode. As the mode name is user-defined, it does not follow any pattern; its only purpose is to differentiate one mode from another. If omitted, the current schema applies to all modes.

In addition, you can specify a seed SDC file for the `SDC_GenerateIncr` rule by specifying `seed` as the mode name.

When using constraints management of the SpyGlass Constraints solution, the mode values have specific meanings. For example, mode could be `flatTop` or `block` for the `Equiv_SDC_block` rule.

### **-corner [ best | worst ]**

(Optional) Identifies the analysis corner for the `Const_Struct08`, `Const_Struct09`, `Inp_Trans05`, and `Inp_Trans06` rules.

For all other rules, the associated files will be assumed valid for both best and worst corners, where applicable.

### **block -name <name-list>**

(Optional) Specifies sub-blocks that are partitions from synthesis if the current design is actually a top-level design.

Each name in `<name-list>` should name a module (for Verilog) or entity (for VHDL) appearing in the design that should be treated as a top-level partition. More than one `block` constraint may appear, or all blocks may be specified in one constraint.

**NOTE:** *The top design is also considered as a block and all block-level rules are run on it unless explicitly mentioned otherwise.*

### **blocksize [ -min <num> ] [ -max <num> ]**

(Optional) Specify the minimum and maximum allowed block sizes in terms of gate count.

You must specify at least one of the `-min` and `-max` arguments with valid integer values.

By default, the minimum block size is 5000 gates and maximum block size is 100000 gates.

## Rules

The `sdc_data` constraint is used by the following rules.

---

### SpyGlass Constraints Solution

---

All rules except the ParamSanityCheck01a, ParamSanityCheck01b, Const\_Struct02, SDC\_GenerateIncr, and Block02 rules

---

## For SpyGlass DFT DSM solution

### Syntax

The syntax to specify the `sdc_data` constraint is as follows:

```
current_design <du-name>
  sdc_data -file <sdc-file-name>
```

### Arguments

**<du-name>**

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**-file <sdc-file-name>**

Name of the SDC file.

### Rules

The `sdc_data` constraint is used by the following rule:

---

### SpyGlass DFT DSM Solution

---

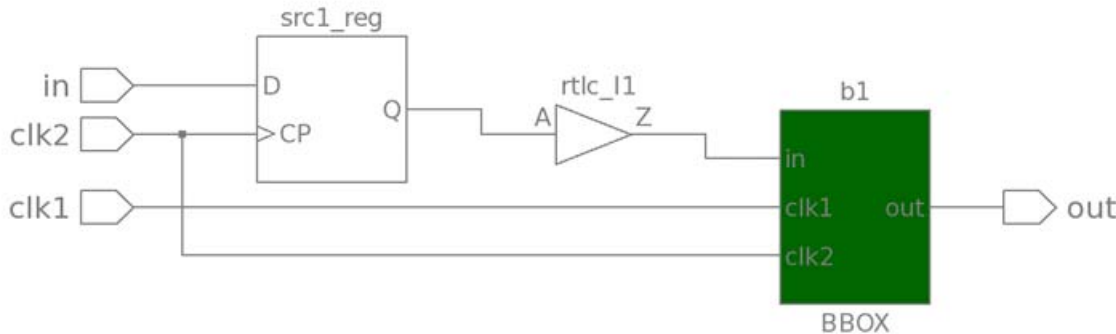
Atspeed\_05

---

## validation\_filter\_path

## Purpose

The `validation_filter_path` constraint is used to filter data domain violations reported during block validation. For example, consider the following schematic:



**FIGURE 57.**

In addition, consider the following SGDC constraints at the top level and block level respectively:

### Sgdc constraint at top level

```
clock -name clk1 -domain d1
clock -name clk2 -domain d2
validation_filter_path -from_obj "test.src1" -to_clock
"clk1"
```

### Sgdc constraint at block level

```
clock -name clk1 -domain d1
clock -name clk2 -domain d2
abstract_port -ports in -clock clk1
```

In the above scenario, data domain violations between `src1_reg` and `b1.in` are filtered because the `validation_filter_path` constraint has been specified.

## Product

SpyGlass CDC solution

## For SpyGlass CDC Solution

### Syntax

The syntax to specify the `validation_filter_path` constraint is as follows:

```
validation_filter_path -from_obj <src-object-name>
-to_clock <clock-of-destination-port>
```

### Arguments

**-from\_obj** <src-object-name>

Name of the source object that is causing the data domain violation.

**-to\_clock** <clock-of-destination-port>

Name of the block's clock port associated with the destination block port that is causing the data domain violation.

### Rules

The `validation_filter_path` constraint is used by the following rules:

---

**SpyGlass CDC Solution**

---

Ac\_abstract\_validation01

---

## select\_wireload\_model

### Purpose

The `select_wireload_model` constraint is used to specify the wire-load model for the design.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `selectwireloadmodel`.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `select_wireload_model` constraint is as follows:

```
select_wireload_model
  [ -instname <inst-name-list> ]
  -wireload <wl-name> | -wireloadtable <whtable-name>
  -net_type < clock | clock_leaf | clock_non_leaf |
  hard_macro | std_logic >
  -size <float>
  -start_size <float>
  -end_size <float>
  -tvf <string>
```

## Arguments

**-wireload <wl-name>**

Name of the `wire_load` structure in the associated technology library.

**-wireloadtable <whtable-name>**

Name of the `wire_load_table` structure in the associated technology library.

**-instname <inst-name-list>**

The instance names must be hierarchical names with respect to the `current_design <du-name>`. This requirement is checked by the `SGDC_power_est17` rule. If you do not specify instance names using the optional `-instname` argument, the specified wire-load model condition is applicable to all instances under the environment `<du-name>` unless



overridden by another `select_wireload_model` constraint specifying particular instance(s).

You can specify multiple `select_wireload_model` constraints.

If you do not specify any `select_wireload_model` constraint, the `default_wire_load_selection` library attribute is used for inferring the wire-load models. If both are not specified or available, the `default_wire_load` library attribute is used. This requirement is checked by the `SGDC_power_est10` rule.

The actual wire-loads used by these rules are reported in the `pe_wireload` Report.

**NOTE:** *The `-instname` argument cannot be specified while specifying the wire-load model for the top-level design unit. Until now, you needed to specify the name of the top-level design unit with the `-instname` argument. Therefore, you must modify your existing `select_wireload_model` constraints for the top-level design units accordingly. Otherwise, SpyGlass will exit with a fatal error (`SGDC_power_est17`).*

**-net\_type** < `clock` | `clock_leaf` | `clock_non_leaf` | `hard_macro` | `std_logic` >

Defines the type of net:

- **clock:** Clock nets
- **clock\_leaf:** Clock nets, which are directly connected to the clock pin of latches and registers, are considered as `clock_leaf` nets.
- **clock\_non\_leaf:** Clock nets, which are directly connected to buffers, inverters, or ICGCs, are considered as `clock_non_leaf` nets.
- **hard\_macro:** Nets are the input of cells with no functionality (memories, blackboxes etc.)
- **std\_logic:** Combinational and sequential nets

For `clock_leaf` and `clock_non_leaf` types of nets, additional capacitance tables are generated when the `power/power_calibration` goal is run on the reference netlist.

**-size** <float>

Specifies the relative size of a cell.

The smaller the specified value, the smaller is the size.

The specified value must be greater than 0. In addition, float values that are less than 1 are supported. For any value greater than 1, specify an integer. If you specify an unsupported value, a violation message is reported.

After the SpyGlass Power Estimate product run, a `pe_cell_sizing_info.csv` file is created. Open this file to review the sizing interpretation of the libraries.

#### **-start\_size <float>**

Defines a specific start size or a size range.

The specified value must be greater than 0. In addition, float values that are less than 1 are supported. For any value greater than 1, specify an integer. If you specify an unsupported value, a violation message is reported.

#### **-end\_size <float>**

Defines a specific start size or a size range.

The specified value must be greater than 0. In addition, float values that are less than 1 are supported. For any value greater than 1, specify an integer. If you specify an unsupported value, a violation message is reported.

#### **-tvlg <string>**

Defines the string which corresponds to vt of the design.

## **Rules**

The `select_wireload_model` constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction Solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## seq\_atpg

### Purpose

The `seq_atpg` constraint is used when sequential ATPG tools are planned. Sequential ATPG will not require all flip-flops to be scanned. Instead, the state of non-scan flip-flops is controlled by properly controlling their next state function as well as their clocks. Repeated actions of this type, which will results in multi vector tests, allow various states to be loaded into the non-scan flip-flops.

The default condition for SpyGlass DFT solution is combinational ATPG. In that case, the state of non-scan flip-flops is controllable only if their set and reset pins, if any, are controllable. When the `seq_atpg` constraint is set to sequential mode, the state of non-scan flip-flops is controllable only if their data pins are controllable.

The `seq_atpg` constraint causes controllability of non-scannable designs to accurately model the power of an ideal sequential ATPG and, therefore, provide a more realistic estimate of fault coverage. The `seq_atpg` constraint may be applied whether or not scan is intended.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `seq_atpg` constraint is as follows:

```
seq_atpg -value 1
```

### Arguments

#### `-value`

The `seq_atpg` constraint is specified with value 1 to enable the feature.

### Rules

The `seq_atpg` constraint is used by the following rules:

---

**SpyGlass DFT Solution**

---

Info_undetectedCause	Info_coverage
----------------------	---------------

---

## set

### Purpose

The `set` constraint is used to specify the names of set signals.

### Product

SpyGlass ERC Product

### Syntax

The syntax of using the `set` keyword in a SpyGlass Design Constraints file is as follows:

```
current_design <du-name>
  set -name <set-name>
```

### Arguments

#### <du-name>

The module name (for Verilog designs) or the design unit name in *<entity-name>.<arch-name>* format (for VHDL designs).

#### -name <set-name>

The set port/pin name.

The pin can be a top-level port/pin as well as an internal pin.

You can specify a single port/pin name or a space-separated list of port/pin names.

For top-level port/pins, *<set-name>* can be the port/pin's full hierarchical name or its simple name. For internal pins, *<set-name>* must be the pin's full hierarchical name.

## Rules

The `set` constraint is used by the following rule:

<b>SpyGlass ERC Solution</b>
<code>setPinConnectedToSetNet</code>

## set\_case\_analysis

### Purpose

The `set_case_analysis` constraint specifies the case analysis conditions.

**NOTE:** For *SpyGlass DFT* and *SpyGlass DFT DSM* products, the `set_case_analysis` constraint is treated as the `test_mode` -functional constraint.

### Product

SpyGlass Auto Verify solution, SpyGlass Power Verify solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions, SpyGlass ERC product, SpyGlass CDC solution, SpyGlass latch product, SpyGlass OpenMore product, and SpyGlass STARC product., SpyGlass DFT solution, SpyGlass DFT DSM solution

### Syntax

The `set_case_analysis` constraint uses the following syntax:

```
current_design <du-name>
set_case_analysis
  -name {<name>}
  -value <value>
```

## Arguments

### <du-name>

The module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs) in the design

### -name <name>

A primary port name or hierarchical name of a pin/net. The pin can be a primary pin or an internal pin.

You can also specify a space-separated list of primary port names or hierarchical pin/net names.

The following example shows that the `set_case_analysis` constraint has been defined on internal net `top.U1.U13.tm1` and `top.U3.tm3` with value 0:

```
current_design top
  set_case_analysis
    -name top.U1.U13.tm1 -value 0
```

For primary ports, you can also specify the simple port name as in the following example:

```
current_design top
  set_case_analysis -name in15 -value 1
```

**NOTE:** *The value set on the signal specified with the `set_case_analysis` keyword is automatically propagated through the design. You do not need to set values for other signals in the path.*

**NOTE:** *The `set_case_analysis` constraints support wildcard characters. The supported meta-characters are `*` (star) and `?` (question mark), where `*` matches any number of characters and `?` matches only one character. The wildcard support is applicable for non-escaped names only. If the meta-characters appear inside an escaped name, they are treated as literals. For example, in the expression `"top.\mid* .bottom"`, `\mid*` is considered as a literal and does not expand to `mid1`, `mid2`, and so on. In addition, if you specify a hierarchical path by using a wildcard, any sub-portion of this path that contains the wildcard 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 wildcard*

string. For example, the expression "top.mid\*.bottom" will expand to top.mid1.bottom", and not to "top.mid1.u1.bottom".

**NOTE:** The expression on which a wildcard is used should always be enclosed within double quotes. For example, "top.mid\*.bottom".

**NOTE:** The wildcard support is applicable for design objects only. For non-design objects, the support is not applicable.

### **-value <value>**

Value list for a pin. The value list is the sequence of one or more values (each value being 0, 1, X, Z, or a combination).

You can also specify repeat sequences for the value as <I\*S>. Here, S is any string that does not contain the <, >, and \* characters. However, S can contain another <I\*S> expression. I is an integer that is always interpreted as a decimal value. The expression <I\*S> means that the sequence S will be repeated I number of times.

Consider the following examples:

#### **Example1:**

```
set_case_analysis -name abc -value "<5*10>"
```

The above example will be expanded as follows:

```
set_case_analysis -name abc -value 1010101010
```

#### **Example2:**

```
set_case_analysis -name abc -value "11<5*10>010"
```

The above example will be expanded as follows:

```
set_case_analysis -name abc -value 111010101010010
```

#### **Example3:**

```
set_case_analysis -name abc -value "<50*11<5*10>>010"
```

The above example will be expanded as follows:

```
set_case_analysis -name abc -value 111010101010...(repeated
50 times followed by 010)
```

You can also set a variable using the command `setvar` to obtain the above result as follows:

```
setvar x 11<5*10>
set_case_analysis -name abc -value "<50*${x}>010"
```

The above example will be expanded as follows:

```
set_case_analysis -name abc -value 111010101010...(repeated
50 times followed by 010)
```

**NOTE:** *If the `set_case_analysis` keyword is not specified or is incorrectly specified, the feature is not enabled.*

To specify the values for a vector signal, you can specify the value as a binary/decimal/hexadecimal number enclosed in curly brackets as in the following example:

```
set_case_analysis -name a[7:0] -value {h 0F}
```

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

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

Values are assigned depending on the position of 0th bit in the Bus declaration as shown in [Figure 58](#). If the 0th Bit is the Left Significant Bit (LSB), the assignment happens from left to right. If the 0th Bit is the Right Significant Bit (RSB), the assignment happens from right to left.

– Bus: in1[3:0]	SGDC: -name in1 -value {b 1011}	Expansion: Bits[ 3,2,1,0] = 1011
– Bus: in2[7:0]	SGDC -name in2 -value {b 1011}	Expansion: Bits[7,6,5,4,3,2,1,0] = 0000_1011
– Bus: in3[0:7]	SGDC -name in3 -value {b 1011_0101}	Expansion: Bits[7,6,5,4,3,2,1,0] = 1010_1101

**FIGURE 58.**

If the SGDC sequence is less than the bits on the bus, the remaining bits get 0 as shown in [Figure 59](#).



## SpyGlass Design Constraints

– Bus: in2[7:0]    SGDC –name in2 –value {b 1011}                    Expansion: Bits[7,6,5,4,3,2,1,0] = 0000\_1011

– Bus: in2[0:7]    SGDC –name in2 –value {b 1011}                    Expansion: Bits[7,6,5,4,3,2,1,0] = 0000\_1101

**FIGURE 59.**

If the SGDC sequence is more than the bits on the bus, the extra SGDC sequence is ignored as shown in [Figure 60](#).

– Bus: in4[0:7]    SGDC –name in4 –value {b 1110\_1101\_01}            Expansion: Bits[7,6,5,4,3,2,1,0] = 1011\_0111

– Bus: in5[7:0]    SGDC –name in5 –value {b 1110\_1101\_01}            Expansion: Bits[7,6,5,4,3,2,1,0] = 1011\_0101

**FIGURE 60.**

You can also specify the `set_case_analysis` constraint using `.sdc` file as follows:

```
set_case_analysis 0|1 [get_ports <name>]
```

While specifying the `set_case_analysis` constraint using the `.sdc` file, specify the following command in the SGDC file:

```
sdc_data -file <sdc-file-name>
```

### For SpyGlass ERC Product and SpyGlass CDC solution

For SpyGlass ERC product and SpyGlass CDC solution, if any combinational gate is found between a starting point and destination point, it is assumed that the signal would reach the destination depending on the other inputs to these gates. If the signal is found to be blocked due to the scalar values specified for other inputs at these gates, by using the `set_case_analysis` constraint, you can stop the further processing of that particular path.

**NOTE:** For the SpyGlass CDC solution, the information supplied by the `set_case_analysis` constraint is also printed in the Section A: Case Analysis Settings section of the SpyGlass CDC-Summary Report.

## Examples

### For SpyGlass CDC solution

The following examples show usage of `set_case_analysis` constraint.

Consider the following example:

```
module test(clk1, clk2, testclock, tm, data, out);
    input clk1, clk2, testclock, tm, data;
    output out;

    reg out, temp;

    wire clock1, clock2;

    assign clock1 = tm ? testclock : clk1;
    assign clock2 = tm ? testclock : clk2;

    always @(posedge clock1)
        temp <= data;

    always @(posedge clock2)
        out <= temp;
endmodule
```

Normally, SpyGlass will mark the clock crossing between flip-flops `temp` and `out` if the clock names `clk1` and `clk2` are specified using the `clock` keyword in a design constraints file.

However, SpyGlass will not mark the clock crossing between flip-flops `temp` and `out` if you also supply a design constraint file as follows:

```
current_design test
    set_case_analysis -name tm -value 1
```

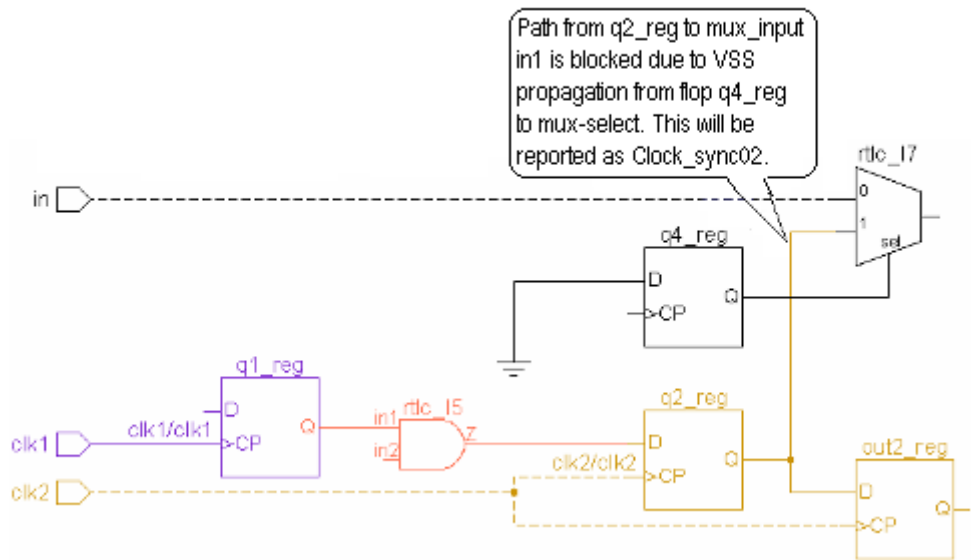
If you supply `set_case_analysis` constraints, the information is also printed in the Section A: Case Analysis Settings section of the SpyGlass CDC Summary Report.

Set-case analysis and power-ground can be propagated through flip-flops without specifying clock events when the `set_option enable_const_prop_thru_seq yes` command is specified in the

## SpyGlass Design Constraints

project file. With this option, flip-flops will be considered as feed through buffers during case analysis and constant propagation, regardless of the clock, set, and reset feeding the flip-flops. The same process is applied to latches and other sequential elements.

This is illustrated in the following figure:



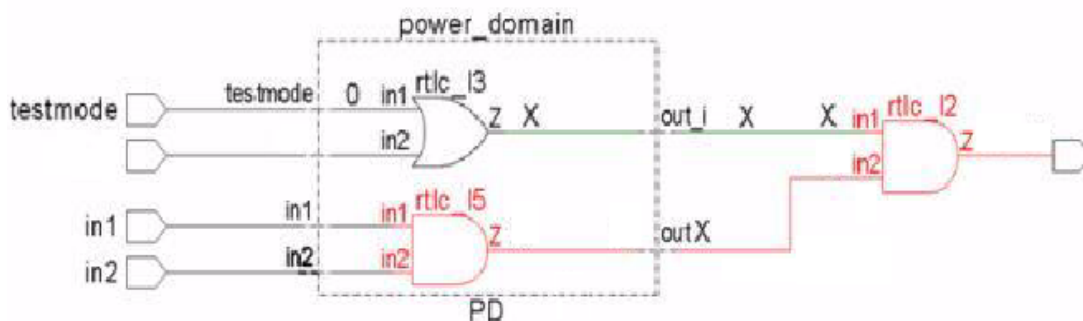
**FIGURE 61.** Flip-Flops as Feed-Through Buffers

### For SpyGlass Power Verify solution

**NOTE:** For the SpyGlass Power Verify solution, the value for `set_case_analysis` will not be propagated, if the combination logic lies in power domain only for rules based on simulation (use `LE` for simulation).

### Examples

As shown in the following figure, the value for `test_mode` is specified as 0 by `set_case_analysis` constraint and it goes to the logic (OR gate) `rtlc_I3` is in power domain, the value of `set_case_analysis` is not propagated beyond OR gate and the value at output net will be X.



**FIGURE 62.** The test\_mode constraint used by the set\_case\_analysis constraint

## Examples For SpyGlass DFT Solution

### Example 1

Consider the following sample input values:

```
set_case_analysis -name vec[3:0] -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below

```
set_case_analysis -name vec[0] -value "1010"
set_case_analysis -name vec[1] -value "0000"
set_case_analysis -name vec[2] -value "0000"
set_case_analysis -name vec[3] -value "0000"
```

### Example 2

Consider the following sample input values:

```
set_case_analysis -name vec[3:0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "0"
set_case_analysis -name vec[1] -value "1"
set_case_analysis -name vec[2] -value "0"
set_case_analysis -name vec[3] -value "1"
```

**Example 3**

Consider the following sample input values:

```
set_case_analysis -name vec[3:0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "1"  
set_case_analysis -name vec[1] -value "0"  
set_case_analysis -name vec[2] -value "0"  
set_case_analysis -name vec[3] -value "0"
```

**Example 4**

Consider the following sample input values:

```
set_case_analysis -name vec -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "1010"  
set_case_analysis -name vec[1] -value "0000"  
set_case_analysis -name vec[2] -value "0000"  
set_case_analysis -name vec[3] -value "0000"
```

**Example 5**

Consider the following sample input values:

```
set_case_analysis -name vec -value { b 1010 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "0"  
set_case_analysis -name vec[1] -value "1"  
set_case_analysis -name vec[2] -value "0"  
set_case_analysis -name vec[3] -value "1"
```

**Example 6**

Consider the following sample input values:

```
set_case_analysis -name vec -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "1"  
set_case_analysis -name vec[1] -value "0"  
set_case_analysis -name vec[2] -value "0"  
set_case_analysis -name vec[3] -value "0"
```

### Example 7

Consider the following sample input values:

```
set_case_analysis -name vec[0] -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "1010"
```

### Example 8

Consider the following sample input values:

```
set_case_analysis -name vec[0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "0"
```

### Example 9

Consider the following sample input values:

```
set_case_analysis -name vec[0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "1"
```

### Example 10

Consider the following sample input values:

```
set_case_analysis -name sclr -value { b 1 0 1 0 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
set_case_analysis -name sclr -value "1010"
```

**Example 11**

Consider the following sample input values:

```
set_case_analysis -name sclr -value { b 1010 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
set_case_analysis -name sclr -value "0"
```

**Example 12**

Consider the following sample input values:

```
set_case_analysis -name sclr -value { b 1 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
set_case_analysis -name sclr -value "1"
```

**Example 13**

Consider the following sample input values:

```
set_case_analysis -name vec -value { h 6 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
set_case_analysis -name vec[0] -value "0"
```

```
set_case_analysis -name vec[1] -value "1"
```

```
set_case_analysis -name vec[2] -value "1"
```

```
set_case_analysis -name vec[3] -value "0"
```

**Rules**

The `set_case_analysis` constraint is used by the following rules:

**SpyGlass Auto Verify Solution**

All rules

**SpyGlass CDC Solution**

All rules except the DeltaDelay01, NoClockCell, and PortTimeDelay rules.

**SpyGlass DFT Product**

All rules			
<b>SpyGlass DFT DSM Product</b>			
All rules			
<b>SpyGlass ERC Product</b>			
FloatingInputs	FlopClockConstant	FlopClockUndriven	FlopDataConstant
FlopDataUndriven	FlopSRConst	FlopEConst	FlopSR
LatchEnableUndriven	LatchEnableConstant	LatchDataUndriven	LatchDataConstant
MuxSelConst	DisabledAnd	DisabledOr	TristateConst
FlopClockX	FlopDataX	FlopSREX	LatchEnableX
LatchDataX	checkNetDriver	checkIOPinConnectedToNet	noCombinatorialFeedback
clockPinsConnectedToClkNets	resetPinConnectedToResetNet	setPinConnectedToSetNet	
<b>SpyGlass latch Product</b>			
LatchFeedback			
<b>SpyGlass OpenMore Product</b>			
CombLoop			
<b>SpyGlass STARC Product</b>			
STARC05-1.2.1.3			
<b>SpyGlass Power Estimation and SpyGlass Power Reduction Solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	PESTR05	PESTR06
PESTR07	PESTR08	PESTR09	PESTR10
PESTR11	PESTR12	PESTR03	PESTR13
poweraudit			
<b>SpyGlass Power Verify Solution</b>			
LPSVM08	LPSVM09	LPSVM10	LPSVM28
LPSVM31	LPSVM47	LPSVM51	LPSVM52
LPSVM53	LPSVM56	LPSVM57	LPSVM59
LPPLIB17	LPSVM12	LPSVM15	LPSVM60



**NOTE:** *For the SpyGlass Power Estimation and SpyGlass Power Reduction solutions, the `set_case_analysis` constraint set on a net, overrides the simulation file (VCD/FSDB) and activity constraint information. In addition, if you want the `set_case_analysis` constraint to propagate through the sequential cells, set the `set_option enable_const_prop_thru_seq yes` command in the project file.*

## enable\_seq\_propagation

### Purpose

The `enable_seq_propagation` constraint provides flexibility to allow constant propagation through specific modules. In constant propagation, flip-flops are considered as feed through buffers during simulation regardless of the clock, set, and reset feeding the flip-flops. The same process is applied to latches and other sequential elements.

Use the `enable_seq_propagation` constraint to specify modules in which constants need to be propagated through sequential circuits during simulation.

### Product

SpyGlass CDC

### Syntax

The syntax to specify the `enable_seq_propagation` constraint is as follows:

```
enable_seq_propagation
  -module <module-name>
```

### Arguments

**-module <module-name>**

Specifies the module name.

### Examples

To allow sequential propagation within all instances of the `mod1` module, set the following constraint:

```
enable_seq_propagation
  -module mod1
```

## set\_cell\_allocation

### Purpose

The `set_cell_allocation` constraint is used to achieve a distribution across multiple sizes for the same cell types in the different clock domains in the design.

The sizes are relative, that is, size 1 refers to smallest size of a particular type and 2 the second higher size. In addition, each size has an associated percentage.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `set_cell_allocation` constraint is as follows:

```
set_cell_allocation
  -type <combinational | sequential>
  -size <float> | -alphabet_size <char>
  -group <grp-name> | -cellname <cell-name-pattern> |
  -libname <lib-name>
  -area
  -percentage <float>
  [ -clock_period <float> ]
  [ -clock <string> ]
  [ -instname <string> ]
```

**NOTE:** The `-clock_period`, `-clock`, and `-instname` arguments are mutually exclusive.

### Arguments

**-type <combinational | sequential>**

Specifies the type of cell.

Possible values are `combinational` and `sequential`.

**-size <float>**

Specifies the relative size of a cell.

The smaller the specified value, the smaller is the size.

The specified value must be greater than 0. In addition, float values that are less than 1 are supported. For any value greater than 1, specify an integer. If you specify an unsupported value, a violation message is reported.

After the SpyGlass Power Estimate product is run, a `pe_cell_sizing_info.csv` file is created. Open this file to review the sizing interpretation of the libraries.

See [Example 1](#), [Example 2](#), and [Example 3](#) for more information.

**NOTE:** *You cannot specify this argument with the `-alphabet_size` argument.*

**-alphabet\_size <char>**

Specifies the relative size of a cell in character size format, such as L or M.

If you specify an unsupported value, a violation message is reported.

After the SpyGlass Power Estimate product is run, a `pe_cell_sizing_info.csv` file is created. Open this file to review the sizing interpretation of the libraries.

See [Example 3](#) for more information.

**NOTE:** *You cannot specify this argument with the `-size` argument.*

**-group <grp-size>**

Specifies the name of the threshold voltage group (name specified with the `default_threshold_voltage_group/threshold_voltage_group` attribute in the library). This argument supports lists. See [Example 4](#) for more information.

**-cellname <cell-name-pattern>**

Specifies a pattern to match cell names of a certain group of cells. This argument supports lists.

**-libname <lib-name>**

Specifies the name of the library. This argument supports lists.

**-area**

Specifies that the size of the cell will be calculated based on the area of the library cell. By default, the `max_cap` attribute of the library cell is used to calculate the size of the cell.

**-percentage <float>**

Specifies the percentage of cell size specified in the `-size` argument.

The specified percentage can be between 0 and 100 or 100. For example, you can specify the percentage as 0.1, 20, and 100. However, please note that the percentage cannot be 0.

**NOTE:** *The sum of the values specified for a type of cell (sequential / combinational) in different `set_cell_allocation` constraints for different clocks periods and clock names must be equal to 100.*

**-clock\_period <float>**

(Optional) Specifies the period of the clock for which the constraints is to be applied.

**NOTE:** *The `-clock_period`, `-clock`, and `-instname` arguments are mutually exclusive.*

**-clock <string>**

(Optional) Specifies the clock name for which the constraint is to be applied. The type of clock nets supported are:

- Port clocks
- Clocks driven by black boxes
- Clocks specified using the `clock` SGDC constraint

**NOTE:** *The `-clock_period`, `-clock`, and `-instname` arguments are mutually exclusive.*

**-instname <string>**

(Optional) Specifies the hierarchical instance name for which the constraint is to be applied. See [Example 2](#) for more information.

**NOTE:** *The `-clock_period`, `-clock`, and `-instname` arguments are mutually exclusive.*

## Examples

### Example 1

Consider that you have a set of sequential cells of four different sizes in your SGLIB file. Also, consider the following:

- 80% of the cells in clock domain CLK1 should use cells of size 2 and 20% of the cells in clock domain CLK1 should use size 4
- 75% of the cells in clock domain CLK2 should use cells of size 4 and 25% of the cells in clock domain CLK2 should use size 8

To specify the above in SpyGlass, specify the following set of constraints:

```
set_cell_allocation -type sequential -size 2  
-percentage 80 -clock CLK1
```

```
set_cell_allocation -type sequential -size 4  
-percentage 20 -clock CLK1
```

```
set_cell_allocation -type sequential -size 4  
-percentage 75 -clock CLK2
```

```
set_cell_allocation -type sequential -size 8  
-percentage 25 -clock CLK2
```

Please note that in this case, cells of size 1 and 3 will not be present in the design.

### Example 2

You can specify the cell distribution as per the hierarchical instance. For example, suppose there is an instance called **top.u1**. In this instance, consider the following:

- 40% of sequential cells should use cells of size 2
- 60% of sequential cells should use cells of size 4

To specify the above in SpyGlass, specify the following set of constraints:

```
current_design top
```

```
set_cell_allocation -instname top.u1 -type sequential -size 2
-percentage 40

set_cell_allocation -instname top.u1 -type sequential -size 4
-percentage 60
```

...

### Example 3

You can specify the cell distribution as per the hierarchical instance. For example, suppose there is an instance called **top.u1**. In this instance, consider the following:

- 40% of sequential cells should use cells of size L
- 60% of sequential cells should use cells of size 4

To specify the above in SpyGlass, specify the following set of constraints:

```
current_design top

set_cell_allocation -instname top.u1 -type sequential -
alphabet_size L -percentage 40

set_cell_allocation -instname top.u1 -type sequential -size 4
-percentage 60
```

...

### Example 4

Consider that you have the following distribution of the cells in the input Netlist with respect to their size and threshold voltage group in the liberty file:

- 40% of the sequential cell area in clock domain CLK1 is occupied by cells of size 2 and group "group1"
- 40% of the sequential cell area in clock domain CLK1 is occupied by cells of size 2 and group "group2"
- 20% of the sequential cell area in clock domain CLK1 is occupied by cells of size 4 and group "group1"
- 40% of the sequential cell area in clock domain CLK2 is occupied by cells of size 4 and group "group1"
- 40% of the sequential cell area in clock domain CLK2 is occupied by cells of size 4 and group "group2"

- 20% of the sequential cell area in clock domain CLK2 is occupied by cells of size 8 and group "group1"

The *PECELLDIST* rule extracts the following set of constraints on the above Netlist:

```
set_cell_allocation -type sequential -size 2 -group "group1"
-percentage 40 -clock CLK1
set_cell_allocation -type sequential -size 2 -group "group2"
-percentage 40 -clock CLK1
set_cell_allocation -type sequential -size 4 -group "group1"
-percentage 20 -clock CLK1
set_cell_allocation -type sequential -size 4 -group "group1"
-percentage 40 -clock CLK2
set_cell_allocation -type sequential -size 4 -group "group2"
-percentage 40 -clock CLK2
set_cell_allocation -type sequential -size 8 -group "group1"
-percentage 20 -clock CLK2
```

## Rules

The `set_cell_allocation` constraint is used by the following rules:

---

<b>SpyGlass Power Estimation and SpyGlass Power Reduction Solutions</b>		
PECELLDIST	PECHECK37	PECHECK40

---



## set\_cell\_name\_pattern

### Purpose

The `set_cell_name_pattern` constraint is used to identify the sizing information by using the naming convention of cells.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `set_cell_name_pattern` constraint is as follows:

```
set_cell_name_pattern
  -name <pattern>
  [-expand]
```

### Arguments

#### **-name <pattern>**

Specifies the predefined values that form the naming convention of cells.

The pattern takes the following syntax `%sx%d`, where

- `%s` stands for the string part of the name that does not convey the size
- `x` in this case is a constant string that exists in each cell name
- `%d` stands for the drive of the cell; it is not restricted to a number and can be some predefined characters, such as:
  - L: Low strength
  - M: Medium strength
  - V: High strength

See [Example 1](#), [Example 2](#), and [Example 3](#).

Alternatively, this pattern can take the following syntax `%sx[%d/<alphabets>]`, where

- %s stands for the string part of the name that does not convey the size
- x in this case is a constant string that exists in each cell name
- [%d/<alphanets>] represents the drive of the cell. Brackets are used to denote size of the cell. Therefore, either %d or brackets can be inside a cell name pattern.

The Brackets are introduced to match some specific type of cells with the drive strength given inside the brackets. This extends to match the drive not only to L,M, or V but any valid size in alphabets such as P, Q etc. (if the library supports such type of cells). For example, if you specify %sx [LPQ] , it matches only those cells which have drive strength L, P or Q.

Only alphabetic characters and %d is allowed inside brackets. The following are examples of wrong usage:

- ❑ %sX[12]
- ❑ %sX[\_0]

See [Example 4](#), [Example 5](#), and [Example 6](#).

### **-expand**

(Optional) Enables you to specify all possible cell name patterns (multiple trigger constraints) in a single expression in the `set_cell_name_pattern` constraint.

Use the **-expand** argument of the `set_cell_name_pattern` constraint to specify multiple trigger constraints in a single expression. SpyGlass expands the specified expression to all possible patterns during correlation.

For example, earlier if you needed to specify the following 13 constraints:

```
vt_mix_percentage -cellname "XY12_LH_*"
vt_mix_percentage -cellname "XY12_LHD_*"
vt_mix_percentage -cellname "XY12_LHS_*"
vt_mix_percentage -cellname "XY12_LHS1_*"
vt_mix_percentage -cellname "XY12_LL_*"
vt_mix_percentage -cellname "XY12_LLD_*"
vt_mix_percentage -cellname "XY12_LLS_*
```

```
vt_mix_percentage -cellname "XY12_LLS1_*"
vt_mix_percentage -cellname "XY12_LS_*"
vt_mix_percentage -cellname "XY12_LSD_*"
vt_mix_percentage -cellname "XY12_LSP_*"
vt_mix_percentage -cellname "XY12_LSS_*"
vt_mix_percentage -cellname "XY12_LSS1_*
```

Now, you can specify these constraints using only a single expression, as follows:

```
set_cell_name_pattern -name "XY12_L{H,L,S}{*,D,S}{*,1}_" -
expand
```

See [Example 7](#).

## Examples

### Example 1

In the following `set_cell_name_pattern` constraint specification, the cell named **dti\_hvt\_28m\_and2x2** is represented.

```
set_cell_name_pattern -name "%sx%d"
```

In this example, the

- %s corresponds to **dti\_hvt\_28m\_and2**
- x corresponds to **x**
- %d corresponds to the drive, which is **2**

### Example 2

In the following `set_cell_name_pattern` constraint specification, the cell named **dti\_hvt\_28m\_and2xp5** is represented. In this cell, the size is 0.5.

```
set_cell_name_pattern -name "%sx%d"
```

In this example:

- %s corresponds to **dti\_hvt\_28m\_and2**
- x corresponds to **x**

- %d corresponds to the drive, which is **p5**. The **p** signifies a decimal value.

However, for the following `set_cell_name_pattern` constraint specification, the **p** is considered as a character. Therefore, the size is 5.

```
set_cell_name_pattern -name "%sxp%d"
```

### Example 3

In the following `set_cell_name_pattern` constraint specification, the cell named **abc\_x2\_def\_x4** is represented. In this cell, the drive is 4.

```
set_cell_name_pattern -name "%sx%d"
```

In this example, the

- %s corresponds to **abc\_x2\_def\_**
- x corresponds to **x**
- %d corresponds to the drive, which is **4**.

In this example, if the drive was represented by 2 in the cell name **abc\_x2\_def\_x4**, the following `set_cell_name_pattern` constraint specification can be used:

```
set_cell_name_pattern -name "%sx%d%s"
```

### Example 4

In the following `set_cell_name_pattern` constraint specification, the **ABC2FGXP** (drive P) and **ABCDE2333LMXQ** (drive Q) cells are represented.

```
set_cell_name_pattern %sX[PQ]
```

In this example, the

- %s corresponds to **ABC2FG** and **ABCDE2333LM**
- x corresponds to **X**
- [PQ] corresponds to cells with drive strengths P or Q matching the pattern %sXP or %sXQ.

### Example 5

In the following `set_cell_name_pattern` constraint specification, all the cells with drive strengths L or M and with patterns as `%sL%s` or `%sM%s` are matched.

```
set_cell_name_pattern %s[LM]%s
```

Therefore, the following cells are matched:

- **ABC2LPQR**: This cell is matched with cell strength as L and not 2.
- **ABCTT2MW**: This cell is matched with cell strength as M and not 2.

However, a cell named **ABCPQR2L** is not matched because the cell might have L as drive strength, but it does not match the pattern.

### Example 6

In the following `set_cell_name_pattern` constraint specification, all cells that have the `%sX%d` or `%sXT` patterns are matched:

```
set_cell_name_pattern %sX[%dT]
```

%d and brackets cannot be used in one cell name pattern. However, you can use %d inside brackets. For example, `%sX[PQR%d]` means to match `%sXP`, `%sXQ`, `%sXR`, or `%sX%d`.

### Example 7

Consider the following expression:

```
set_cell_name_pattern -name
"AB{C1,D1}*{H,*}{LVT,HVT,MVT}X%d*" -expand
```

For the above example, SpyGlass expands the `"AB{C1,D1}*{H,*}{LVT,HVT,MVT}X%d*" expression to all the possible patterns, as follows:`

```
ABC1*HLVTX*
ABC1*HHVTX*
ABC1*HMVTX*
ABC1*LVTX*
```

ABC1 \*HVTX\*  
ABC1 \*MVTX\*  
ABD1 \*HLVTX\*  
ABD1 \*HHVTX\*  
ABD1 \*HMVTX\*  
ABD1 \*LVTX\*  
ABD1 \*HVTX\*  
ABD1 \*MVTX\*

**NOTE:** *Since, a %d is present in the expression, the sizing constraint is also generated internally. Hence, the AB{C1,D1} \*{H,\*}{LVT,HVT,MVT}X%d\* expression is converted into "AB%sX%d%s."*

## Rules

The set\_cell\_name\_pattern constraint is used by the following rules:

---

**SpyGlass Power Estimation and SpyGlass Power Reduction Solutions**

---

PECELLDIST

PECHECK40

SGDC\_power\_est67

---

## set\_clock\_gating\_type

### Purpose

The `set_clock_gating_type` constraint is used to specify the name of the cell that should be used as the clock-gating cell.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `set_clock_gating_type` constraint is as follows:

```
current_design <du-name>
  set_clock_gating_type
    [ -libname <lib-name> ]
    -cellname <cell-name>
    [ -control_location <control-locations>
    [ -control_function <control-function> ]
    [ -posedge ]
    [ -negedge ]
```

### Arguments

#### <du-name>

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

#### -libname <lib-name>

(Optional) Name of the library in which the ICGC is located.

#### -cellname <cell-name>

Name of the cell to be used as the clock-gating cell.

**NOTE:** *You can also specify cell names using regular expressions.*

**-control\_location <control\_location>**

(Optional) Specifies the control point on the clock gate cell. This field can have the following enumeration values:

none | before | after

If you specify the value as *before* / *after*, a clock gate cell having an OR gate with the original register enable and a test signal is picked up. These values (*before/after*) signify the position of the OR gate with respect to the register.

**-control\_function <control\_function>**

(Optional) Specifies the control signal on the clock gate cells. It can have the following enumeration values:

test\_mode | scan\_enable

**-posedge**

(Optional) Specifies the polarity of the clock-edge logic. This field means that a clock-gate cell having positive edge clock logic should be picked-up.

**-negedge**

(Optional) Specifies the polarity of the clock-edge logic. This field means that a clock-gate cell having negative edge clock logic should be picked-up.

**NOTE:** *If you do not specify either of the -posedge or -negedge options, the best ICG cell present in the library is chosen. Further, if you specify either of the two options along with an unknown/invalid combination for the other two values, the best ICG cell with posedge or negedge polarity is selected, respectively, along with an appropriate warning.*

**NOTE:** *The clock-gate name fields (cellname and libname) and clock-gate characteristic fields (control\_location, control\_function, posedge, negedge) should not be specified together.*

Consider the following example:

```
set_clock_gating_type -control_location before -posedge -  
control_function scan_enable
```

Here, a clock-gating cell having positive edge logic at clock and an OR gate before the register with *scan\_enable* signal and enable signal of register



is picked up.

SpyGlass selects the best ICGC cell based on the following table:

<b>Clock-edge polarity</b>	<b>Control signal</b>	<b>Control point</b>	<b>ICGC type</b>
posedge	-	-	latch_posedge
negedge	-	-	latch_negedge
posedge	scan_enable	before	latch_posedge_precontrol
posedge	scan_enable	after	latch_posedge_postcontrol
posedge	test_mode	before	latch_posedge_precontrol_obs
posedge	test_mode	after	latch_posedge_postcontrol_obs
negedge	scan_enable	before	latch_negedge_precontrol
negedge	scan_enable	after	latch_negedge_postcontrol
negedge	test_mode	before	latch_negedge_precontrol_obs
negedge	test_mode	after	latch_negedge_postcontrol_obs

## Rules

The `set_clock_gating_type` constraint is used by the following rules:

<b>SpyGlass Power Estimation and SpyGlass Power Reduction Solutions</b>			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR14	poweraudit		

## set\_fully\_decoded\_bus

### Purpose

The `set_fully_decoded_bus` constraint is used 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.

### Product

SpyGlass DFT DSM Solution

### Syntax

The syntax of `set_fully_decoded_bus` is as follows:

```
set_fully_decoded_bus
  -name <net_name_list>
```

### Arguments

**-name <node\_name>**

List of fully decoded three state buses.

### Rules

The `set_fully_decoded_bus` constraint is used by the *Info\_random\_resistance* rule of the SpyGlass DFT DSM solution.

### Examples

The following example shows the usage of the `set_fully_decoded_bus` constraint:

```
set_fully_decoded_bus -name top.net
```

## set\_mega\_cell

### Purpose

The `set_mega_cell` constraint is used to defined megacells in a design.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

```
set_mega_cell  
  -name <cell-names>
```

### Arguments

**-name <cell-names>**

The names of the cells that you want SpyGlass to consider as megacells. You can only specify black boxes, grey boxes, or library cells that have no functionality as megacells.

To specify a range of cells as megacells, use wildcards.

### Examples

#### Example 1: Specifying Multiple Megacells

The following constraint specification defines three cells as megacells:

```
set_mega_cell -name MPX41PND MPX42PND MPX43PND
```

#### Example 2: Specifying Multiple Megacells by Using Wildcards.

The following constraint specification defines all cells that begin with the prefix MPX as megacells:

```
set_mega_cell -name MPX*
```

## set\_power\_info

### Purpose

The `set_power_info` constraint is used to specify user attributes on the pins of library cell.

**NOTE:** Wildcard support is provided for the `set_power_info` constraint. Supported meta-characters are `*` (star) and `?` (question mark), where `*` matches any number of characters and `?` matches only one character.

### Product

SpyGlass Power Verify

### Syntax

The syntax to specify the `set_power_info` constraint is as follows:

```
current_design <du-name>
  set_power_info
    -cell <cell-name>
    -pin <pin-name>
    -attribute <attribute-name>
    -value < true | false>
```

### Arguments

**current\_design <du-name>**

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

**cell <cell-name>**

Name of the library cell.

**pin <pin-name>**

Name of the pin of the library cell.

**attribute** <attribute-name>

Name of the user specified attribute that needs to be applied on the pin of the library cell.

**value** <true | false>

Value of the user specified attribute applied on the pin of the library cell.

**Examples**

In this example, the `has_pass_gate` attribute is applied to the macro cell at `in1` pin.

```
set_power_info -cell macro -pin in1 -attribute has_pass_gate
-value true
```

**Rules**

The `set_power_info` constraint is used by the following rule:

---

**SpyGlass Power Verify**


---

LPCONN06

---

**stil\_data****Purpose**

The `stil_data` constraint is used to specify the STIL file.

**Product**

SpyGlass DFT

**Syntax**

The syntax to specify the `stil_data` constraint is as follows:

```
stil_data
-file <file_name>
[-mode <mode_name>]
```

## Arguments

**-file** <file\_name>

Specifies the path to a stil file.

**-mode** <mode\_name>

Specifies the mode that is described in STIL file. If not specified, global mode without name is selected.

## Examples

Consider the following example:

```
stil_data -file test.stil -mode Internal_scan
```

## sg\_multicycle\_path

### Purpose

The `sg_multicycle_path` constraint enables you to specify multi-cycle paths, which you want to exclude from the at-speed testing.

**NOTE:** *You can convert the `set_multicycle_path` SDC command to `sg_multicycle_path` SGDC command using the SDC to SGDC conversion. For details, refer to the *Atrenta Console Reference Guide*.*

### Product

SpyGlass DFT DSM

### Syntax

The syntax to specify the `sg_multicycle_path` constraint is as follows:

```
sg_multicycle_path  
[-from <from_list>]
```

```

[-to <to_list>]
[-through <through_list>]
[-path_multiplier <integer_value>]

```

## Arguments

### **-from <from\_list>**

Specifies a list of objects that act as the start point for the multi-cycle path. A valid start point is a clock, a primary input or inout port, a sequential cell, a clock pin of a sequential cell. If a clock is specified, all registers and primary inputs related to the clock are considered as start points.

### **-to <to\_list>**

Specifies a list of objects that act as endpoint for the multi-cycle path. A valid endpoint is a primary output or inout port, a sequential cell, a data pin of a sequential cell.

### **-through <through\_list>**

Specifies a list of pins or nets that you want to disable.

**NOTE:** *The DFT DSM policy does not support the -through option.*

### **-path\_multiplier <integer\_value>**

Specifies the number of cycles that a path must have for setup or hold, relative to the start point or end point clock, before data is required at the endpoint. For example, specifying a path\_multiplier of 2 for setup implies a 2-cycle data path.

## Examples

Consider the following example:

```
sg_multicycle_path -from FF1 -to FF2 -path_multiplier 2
```

Here, path from FF1 to FF2 is a 2-cycle path.

## syn\_set\_dont\_use

### Purpose

The `syn_set_dont_use` constraint is used to exclude selected cells from technology mapping and timing optimization.

**NOTE:** *Wildcard support is provided for the `syn_set_dont_use` constraint. Supported meta-characters are \* (star) and ? (question mark), where \* matches any number of characters and ? matches only one character.*

The SpyGlass Power Estimation and SpyGlass Power Reduction solutions allow overrides for the `syn_set_dont_use` constraint. However, the override statements are taken collectively and no order is followed while processing them.

Let us see the implication on the `syn_set_dont_use` constraint using the following example:

```
syn_set_dont_use -libname TSMC -cellname "*"
syn_set_dont_use -libname TSMC -cellname "*VT*" -exclude 1
syn_set_dont_use -libname TSMC -cellname "*HVT*" -exclude 1
```

Here, the first command tells SpyGlass to ignore all cells (specified with an \*). The second command excludes \*VT\* cells from the ignored set. Third one excludes \*HVT\* cells (that are already included in the second statement). SpyGlass processes all these statements collectively and uses only \*VT\* cells for technology mapping and timing optimization.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `syn_set_dont_use` constraint is as follows:

```
current_design <du-name>
  syn_set_dont_use
    [-libname <lib-name>]
    -cellname <cell-names-list>
```



```
-exclude <0|1|yes|no>
```

## Arguments

### <du-name>

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

### -libname <lib-name>

(Optional) Name of the library in which the cells are located.

### -cellname <cell-names-list>

A space-separated list of cells that are to be excluded.

**NOTE:** *You can also specify cell names using regular expressions.*

### -exclude <0|1|yes|no>

(Optional) In some cases, where a cell in the technology library has `dont_use` attribute set as `true`, and you want to use that cell for mapping, set the `-exclude` argument to `1`. The specified cell gets included as a candidate for mapping, when this argument is set to `1`.

You should use the `-exclude` argument for including cells like integrating clock-gating cells that have the `dont_use` attribute set as `true` in the technology library for mapping.

## Rules

The `syn_set_dont_use` constraint is used by the following rules:

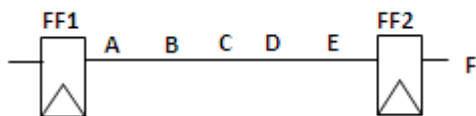
SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## ignore\_nets

### Purpose

The `ignore_nets` constraint is used to exclude nets from the expression of a new/strengthened enable.

For example, consider the following figure:



In the above figure, the FF1 flip-flop is connected to the A net, which is connected to the B net. The B net is connected to the C net, which is then connected to the D net. The D net is connected to the E net, which is connected to the FF2 flip-flop. The FF2 flip-flop is connected to the F net.

Now consider that the enable expression in this case is  $C \ \& \ X$ , where X is a net in the design (not shown in the above figure).

Consider that you have specified the following `ignore_nets` constraint:

```
ignore_nets -net C
```

In this case, the nets to be excluded from the enable expression are A, B, C, D, and E. Therefore, the enable expression in this case will be X.

The search for the net to be excluded is done forward and backward. However, the navigation from one net to another net is limited to going through buffers and inverters. For example, consider the following expression:

$$C = A \ \& \ B$$

For the above expression, if the constraint is set on the B net, the C net will not be excluded from the expression.

**NOTE:** *If the specified net is not available after synthesis, the net is not removed from the enable expression.*

The `ignore_nets` constraint is applied on a net directly when the net name is specified. However, the constraint can also be indirectly applied on

the net by specifying a buffer name or an inverter name.

**NOTE:** *Wildcard support is provided for the `ignore_nets` constraint. Supported meta-characters are `*` (star) and `?` (question mark), where `*` matches any number of characters and `?` matches only one character.*

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `ignore_nets` constraint is as follows:

```
current_design <du-name>
  ignore_nets
    -net <net-names-list>
```

## Arguments

### <du-name>

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

### -net <net-names-list>

A space-separated list of nets that are to be excluded from the expression of a new/strengthened enable.

**NOTE:** *You can also specify net names using regular expressions.*

## Rules

The `ignore_nets` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR20	PEPWR21	PEPWR22	PEPWR23
PEPWR24	PEPWR25	PEPWR28	PESTR20

---

PESTR21	PESTR22	PESTR23	PESTR24
PESTR25	PESTR28		

---

## ser\_data

### Purpose

The `ser_data` constraint is used to specify register properties with respect to soft errors occurrence and detection.

### Product

SpyGlass DFT

### Syntax

The syntax to specify the `ser_data` constraint is as follows:

```
ser_data
  -name <module/instance name>
  -lreg <data>
  [-lsm <data>]
  [-dc <data>]
  [-dcl <data>]
```

### Arguments

**name** <module/instance name>

Name of the module or instance.

**-lreg** <data>

**-lsm** <data>

```
-dc <data>
```

```
-dc1 <data>
```

## Example

## Rules

The `ser_data` constraint is used by the following rules:

---

**SpyGlass DFT**

---

Info\_soft\_error\_propagation

---

## safety\_related

### Purpose

The `safety_related` constraint specifies safety related modules/instances/registers and output ports.

By default, all outputs are safety related. However, explicitly specified registers and outputs safety overrides the default behavior.

### Product

SpyGlass DFT

### Syntax

The syntax to specify the `safety_related` constraint is as follows:

```
safety_related
  -name <module/ instance name>
  -output <port_name>
```

## Arguments

**name** <module/instance name>

Name of the module or instance.

**-output** <port\_name>

Name of the safety related output ports. To specify all the output ports, use "\*".

## Example

```
safety_related -output "*"
```

The above example considers all output ports as safety related.

## Rules

The `safety_related` constraint is used by the following rules:

<b>SpyGlass DFT</b>
Info_soft_error_propagation

## non\_safety\_related

### Purpose

The `non_safety_related` constraint specifies non-safety related modules/instances/registers and output ports.

By default, all registers are non-safety related. However, explicitly specified registers and outputs non-safety overrides the default behavior.

### Product

SpyGlass DFT

### Syntax

The syntax to specify the `non_safety_related` constraint is as follows:

```
non_safety_related
  -name <module/ instance name>
  -output <port_name>
```

## Arguments

**name** <module/instance name>

Name of the module or instance.

**-output** <port\_name>

Name of the non-safety related output ports. To specify all the output ports, use "\*".

## Example

```
non_safety_related -output "*"
```

The above example considers all output ports as non-safety related.

## Rules

The `non_safety_related` constraint is used by the following rules:

---

**SpyGlass DFT**

---

Info\_soft\_error\_propagation

---

## set\_lib\_timing\_mode

### Purpose

The `set_lib_timing_mode` constraint is used to select the active timing mode of cell groups.

### Product

SpyGlass Core

### Syntax

The syntax to specify the `set_lib_timing_mode` constraint is as follows:

```
set_lib_timing_mode
  -modes <timing-mode-names>
  -instances <instances-list>
```

### Arguments

**-modes <timing-mode-names>**

List of the names of timing modes defined in a liberty file.

**-instances <instance-list>**

List of instances of liberty cells.

### Example

Consider the following cell definition in a liberty file:

```
// Define mode definition
mode_definition (Mode_def1) {
  mode_value( mode1) { /* empty when-sdf_cond */ }
  mode_value( mode2) { /* empty when-sdf_cond */ }
}

pin(Q) {
  direction : output;
```



```

timing() {
  related_pin : "clk1";
  mode(Mode_def1, model);
}
timing() {

```

Now consider U1 as the instance of the library cell. In this case, if you define the following constraint, SpyGlass enables the timing arc only for clk1 and therefore, only clk1 will be propagated for U1:

```
set_lib_timing_mode -modes model -instances U1
```

## Rules

The `set_lib_timing_mode` constraint is used by the following rules:

---

### SpyGlass Core

---

SGDC\_set\_lib\_timing\_mode01

SGDC\_set\_lib\_timing\_mode02

---

## set\_lib\_name

### Purpose

The `set_lib_name` constraint is used to specify the cell name defined in the corresponding liberty file.

### Product

SpyGlass Power Verify

### Syntax

The syntax to specify the `set_lib_name` constraint is as follows:

```
set_lib_name -cell <cell-name>
```

## Arguments

`-cell <cell-name>`

Name of the cell defined in the liberty file.

## Example

Consider the following command:

```
...  
set_lib_name -cell cellA -cell cellB  
...
```

If this constraint is not specified in the SGDC file, the UPF\_lowpower26 rule does not run and reports following message:

```
set_lib_name is not specified in SGDC file, it is mandatory for  
the rule to run"
```

## Rules

The `set_lib_name` constraint is used by the following rule:

---

**SpyGlass Power Verify**

---

UPF\_lowpower26

---

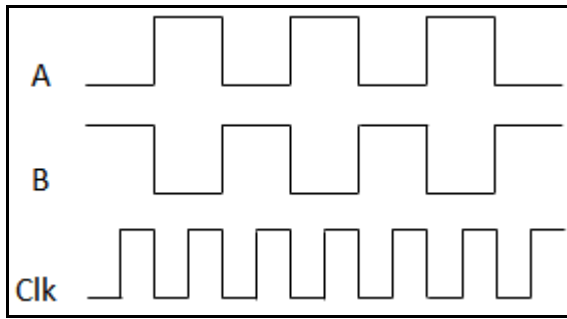
## set\_monitor\_cell

### Purpose

The `set_monitor_cell` constraint is specified to estimate the power of special cells, such as mega cells, which consume huge power.

By default, the SpyGlass Power Estimate solution estimates the power of a cell based on the activity and probability values. However, when the cell is specified using this constraint, all the instances of the cell are calculated in a high accuracy mode. In this mode, all the when conditions of the internal and leakage power of the cell are calculated by considering individual signals and the temporal behavior of the cell.

Consider that A and B are pins of an AND cell and their signals are as follows:



Now consider that the activity and probability for the A and B pins are as follows:

Pin	Activity	Probability
A	1	0.5
B	1	0.5

The following table shows the activity and probability of the AND condition when the calculations are based on the activity and probability of the A and B pins:

AND Condition	Activity	Probability
A&B	1	0.25

Now, if you apply the `set_monitor_cell` constraint on these pins, the activity and probability of the AND condition will be as follows:

AND Condition	Activity	Probability
A&B	0	0

The activity and probability values in the above table are more accurate as compared to the values based on the activity and probability of the A and B

pins.

**NOTE:** *The run time and memory consumption increases if the `set_monitor_cell` constraint is applied on all the cells in a design.*

**NOTE:** *This constraint is applied by default for all memories and hard macros. Set the value of the `pe_enable_monitor_for_mem` parameter to 0 to disable automatic power estimation by monitors on memories and hard macros.*

**NOTE:** *If you are using the `set_monitor_cell` constraint along with the `pe_num_clock_cycles_avg_power` parameter in the `PECHECK34` rule, the value of the `pe_num_clock_cycles_avg_power` parameter should be reasonably high. It is recommended that you set the value of the parameter to a value higher than 50.*

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `set_monitor_cell` constraint is as follows:

```
set_monitor_cell
  -name <cell-name>
```

## Arguments

**-name <cell-name>**

Name of the library cell for which power accuracy calculation is to be used. You can also use wildcard characters while specifying cell names.

## Examples

Consider the following example:

```
set_monitor_cell -name "RSHL*"
```

The above example implies that all the cell names that match with the wildcard expansion will be evaluated in the high accuracy mode.

## Rules

The `set_monitor_cell` constraint is used by the following rules:

---

**SpyGlass Power Estimation and SpyGlass  
Power Reduction Solutions**


---

PEPWR02	PECHECK34
---------	-----------

---

## set\_pin

### Purpose

The `set_pin` constraint is used to specify black box pins that should be assumed to be set/reset pins.

Then, the `Async_07` rule of the SpyGlass DFT solution treats the pin specified with the `set_pin` constraints as if it is the set/reset pin on a flip-flop. The source of such a pin must be a test mode controlled to the inactive state.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `set_pin` constraint is as follows:

```
set_pin
  -name <du-name>.<port-name>
  [ -value <value> ]
  [ -synchronous ]
  [ -asynchronous ]
```

### Arguments

#### <du-name>

The name of the design unit (black box) for which you are specifying the set pin.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name *<du-name>* can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are processed.

You can specify a single design unit name or a space-separated list of design unit names.

**<port-name>**

Name of the asynchronous set port on the design unit (black box).

You can specify only a single port name.

**-value <value>**

The active value (0 or 1) for this asynchronous set port *<port-name>*.

**-synchronous**

Implies that the set pins are synchronous.

**-asynchronous**

Implies that the set pins are asynchronous.

**Rules**

The `set_pin` constraint is used by the following rules:

---

SpyGlass DFT Solution	
Async_07	Async_07Lssd

---

## set\_power\_scaling

**Purpose**

The `set_power_scaling` constraint is used to scale the power estimation values of the design.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax of the `set_power_scaling` constraint is as follows:

```
set_power_scaling
  -type <design-element-type>
  [ -leakage <leakage-power-factor> ]
  [ -internal <internal-power-factor> ]
  [ -switching <switching-power-factor> ]
```

## Arguments

### **-type <design-element-type>**

The type of design element for which you are specifying the power scaling factor.

The possible values are as follows:

Type of Design Element	Value Specified in <design-element-type>
Black box	blackbox
Clock	clock
Combinational element	combinational
IO pad	iopad
Memory	memory
Sequential element	sequential
Megacell	megacell

**NOTE:** *The values specified in the -type argument of the `set_power_scaling` constraint are case-insensitive.*

### **-leakage <leakage-power-factor>**

Power scaling factor for leakage power of the design element (in Watts).

Default value is 1.0, which specifies that no scaling is required.

**-internal <internal-power-factor>**

Power scaling factor for internal power of the design element (in Watts).

Default value is 1.0, which specifies that no scaling is required.

**-switching <switching-power-factor>**

Power scaling factor for switching power of the design element (in Watts).

Default value is 1.0, which specifies that no scaling is required.

**NOTE:** *You must specify at least one of the -leakage, -internal, or -switching arguments of the set\_power\_scaling constraint.*

## Examples

### Example 1

Consider the following example:

```
set_power_scaling -type combinational -leakage 0.95
```

The above example specifies that the leakage power of the combinational elements in the design should be scaled by 0.95 (95%).

### Example 2

Consider the following SGDC file snippet:

```
set_power_scaling -type combinational
  -leakage 0.5 -internal 1.5 -switching 2.0
set_power_scaling -type sequential
  -leakage 0.5 -internal 1.5 -switching 2.0
set_power_scaling -type iopad
  -leakage 0.5 -internal 1.5 -switching 2.0
set_power_scaling -type memory
  -leakage 0.5 -internal 1.5 -switching 2.0
set_power_scaling -type blackbox
  -leakage 0.5 -internal 1.5 -switching 2.0
set_power_scaling -type clock
  -leakage 0.5 -internal 1.5 -switching 2.0
set_power_scaling -type megacell
```



```
-leakage 0.5 -internal 1.5 -switching 2.0
```

In this case, the pe\_summary report will contain the following section for the power scaling values:

```
## Power scaling factors used to scale the power consumed
## by each logical component of the design. This information
## is specified using 'set_power_scaling' constraint
##
## Power Scaling Factor:
## -----
##
##           Leakage      Internal      Switching
## Combinational Power : 0.50          1.50          2.00
## Sequential Power    : 0.50          1.50          2.00
## Black Box Power     : 0.50          1.50          2.00
## Memory Power        : 0.50          1.50          2.00
## IO PAD Power        : 0.50          1.50          2.00
## Clock Power         : 0.50          1.50          2.00
## Mega Cell Power     : 0.50          1.50          2.00
```

Power Summary:

```
-----
           Leakage      Internal      Switching      Total
Total Power      : 67.8nW      55.1uW      35.6uW      90.8uW
Combinational Power : 19.8nW      3.25uW      6.82uW      10.1uW
Sequential Power   : 46.6nW      46.9uW      6.13uW      53.1uW
Black Box Power    : 0W          0W          0W          0W
Memory Power       : 0W          0W          0W          0W
IO PAD Power       : 0W          0W          0W          0W
Clock Power        : 1.40nW      4.94uW      22.7uW      27.6uW
Mega Cell Power    : 2.26pW      0W          0W          2.26pW
```

The above section of the pe\_summary report has the following tables:

- **Power Scaling Factor:** Shows the power scaling factors as comments
- **Power Summary:** Shows the scaled power values

For details on this report, refer to *pe\_summary Report* in the *SpyGlass Power Estimation and SpyGlass Power Reduction Rules Reference Guide*.

## Rules

The `set_power_scaling` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR20	PEPWR21	PEPWR22	PEPWR23
PEPWR24	PEPWR25	PEPWR28	SGDC_power_est 62

## set\_supply\_node

### Purpose

The `set_supply_node` constraint is used to specify PG pin of a library cell that needs to be considered as a valid driver during the execution of the *UPF\_lowpower09* rule.

### Product

SpyGlass Power Verify solution

### Syntax

The syntax of the `set_supply_node` constraint is as follows:

```
set_supply_node -cell <cell-name> -pin <pin-name>
```

### Arguments

**-cell <cell-name>**

(Mandatory) Specify the name of the cell to be considered as supply.

**-pin <pin-name>**

(Mandatory) Specify the PG pin of the cell to be considered as a supply port.

## Examples

Consider the following example:

```
set_supply_node -cell C28SOI_BBGEN -pin vddcore
```

In the above example, the pin `vddcore` of cell `C28SOI_BBGEN` is being considered as a valid driver.

## Rules

The `set_supply_node` constraint is used by the following rules:

---

### SpyGlass Power Verify Solution

---

UPF\_lowpower09

---

## sg\_clock\_group

### Purpose

The `sg_clock_group` constraint defines asynchronous relationship between clocks.

**NOTE:** *This constraint works only if the `sta_based_clock_relationship` parameter is set to either `only_scg` or `scg_functional`. For details on this parameter, refer to the *SpyGlass CDC Rules Reference Guide*.*

### Syntax

The syntax of the `sg_clock_group` constraint is as follows:

```
current_design <du-name>
  sg_clock_group
    -group1 <clk-tag-list1>
    -group2 <clk-tag-list2>
```

### Arguments

The `sg_clock_group` constraint has the following arguments:

**-group1 <clk-tag-list1>**

The clock tag list for which asynchronous relationship is needed.

**-group2 <clk-tag-list2>**

The clock tag list for which asynchronous relationship is needed.

**Example**

Consider the following example:

```
clock -name top.clk1 -period 10 -tag T1
clock -name top.clk2 -period 12 -tag T2
sg_clock_group -group1 { T1 } -group2 { T2 }
```

In the above example, the clk1 and clk2 clocks are made asynchronous to each other.

**Rules**

The sg\_clock\_group constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Ac_sync_group rules	Ac_glitch03	Clock_sync05	Clock_sync06

**sgdc****Purpose**

The sgdc constraint is used to specify a block-level SGDC file to be imported or specify blocks for which block-level SGDC file is to be generated.

## For SpyGlass CDC Solution

### Syntax

The syntax of the `sgdc` constraint is as follows:

#### Usage 1

```
sgdc -import <block-name> <SGDC-file>  
      -instance_list <instance-list>  
      -bbox_instance_list <instance-list>
```

#### Usage 2

```
sgdc -export <blocks>
```

### Arguments

The `sgdc` constraint has the following arguments:

**-import <block-name>**

Specifies the block name.

**<SGDC-file>**

Specifies an SGDC file that needs to be imported for the specified block, <block-name>.

It is recommended that you specify an absolute path of the file.

**-instance\_list <instance-list>**

Specifies the list of instances (hierarchical name) that should be bound to the abstract model specified in the abstracted SGDC file (<SGDC-file>).

**-bbox\_instance\_list <instance-list>**

Specifies the list of instances (hierarchical name) that should be treated as black boxes.

**-export <blocks>**

Specifies a space-separated list of blocks for which a block-level SGDC file

should be generated.

## Rules

The `sgdc` constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_blksgdc01	SGDC_clock_validation01	SGDC_clock_validation02	SGDC_clock_domain_validation01
SGDC_clock_domain_validation02	SGDC_set_case_analysis_validation01	SGDC_set_case_analysis_validation02	SGDC_reset_validation01
SGDC_reset_validation02	SGDC_reset_validation03	SGDC_reset_validation04	SGDC_virtualclock_validation01
SGDC_input_validation01	SGDC_input_validation02	SGDC_num_flops_validation01	SGDC_num_flops_validation02
SGDC_output_validation01	SGDC_output_validation02	SGDC_abstract_port_validation01	SGDC_abstract_port_validation02
SGDC_abstract_port_validation03	SGDC_abstract_port_validation04	SGDC_qualifier_validation01	SGDC_qualifier_validation02
SGDC_cdc_false_path_validation01	SGDC__validation01	SGDC__validation02	

## Examples

### Example 1

Consider the following `sgdc` constraint specification:

```
sgdc -import B1 "/u/user1/B1.sgdc"
```

The above specification implies that the block-level SGDC file, `/u/user1/B1.sgdc`, is to be imported for the B1 block.

Suppose the B1 block has three instances: `i1`, `i2`, and `i3`. In this scenario, consider that you want to:

- Bind `i1` with an abstract model.
- Treat `i2` as a black box.

- Use the complete synthesized view for `i3`.

To accomplish the above requirements, specify the following command:

```
sgdc -import B1 "/u/user1/B1.sgd" -instance_list i1  
-bbox_instance_list i2
```

## Example 2

Consider the following `sgdc` constraint specification:

```
sgdc -export B1 B2 B3
```

The above specification implies that a block-level SGDC file would be generated for the B1, B2, and B3 blocks.

## For All Products

### Purpose

The `sgdc` constraint is used in hierarchical methodology flow in which an abstracted view of an IP is used during SoC-level verification. For details, refer to the *Using the Hierarchical Methodology* chapter of the *Atrenta Console User Guide*.

Use this constraint to specify an IP and the path where the abstracted view of that IP is saved.

### Syntax

The syntax of the `sgdc` constraint is as follows:

```
sgdc -import <IP-name> <abstracted-view-path>  
-instance_list <instance-list>  
-bbox_instance_list <instance-list>
```

### Arguments

The `sgdc` constraint has the following arguments:

**-import <IP-name>**

Specifies the IP name.

**<abstracted-view-path>**

Specifies the path at which the abstracted view of the specified IP is saved.

**-instance\_list <instance-list>**

Specifies the list of instances that should be bound to the abstract model specified in the abstracted SGDC file (<SGDC-file>).

**-bbox\_instance\_list <instance-list>**

Specifies the list of instances that should be treated as black boxes.

## shadow\_ratio

### Purpose

The `shadow_ratio` constraint is used to specify the fraction of the total logic that is allowed to be shadowed by memories.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `shadowratio`.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `shadow_ratio` constraint is as follows:

```
shadow_ratio -value <value>
```

### Arguments

The `shadow_ratio` constraint has the following arguments:

**-value <value>**

The shadow ratio as a decimal value less than 1 such as 0.95.

### Rules

The `shadow_ratio` constraint is used by the following rule:



---

**SpyGlass DFT Solution**

---

RAM\_01

---

## show\_power\_calc\_details

### Purpose

The `show_power_calc_details` constraint is used to specify instances for which power calculation debug data is required. Wildcard and list is supported for this constraint.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

```
show_power_calc_details
    -instname <leaf-level-instances>
```

### Arguments

**instname** <leaf-level-instances>

The instance names for which you want to generate power calculation debug data.

To specify a range of instances, use wildcards.

### Examples

#### Example 1: Specifying Multiple Instances

The following constraint specification defines two instances:

```
show_power_calc_details -instname top.A.i1 top.A.i2
```

#### Example 2: Specifying Multiple Instances by Using Wildcards.

The following constraint specification:

```
show_power_calc_details -instname top.A.*
```

Here, debug data is generated for all the leaf-level instances under the top.A hierarchy.

## signal\_in\_domain

### Purpose

By default, the schemes Synchronized Enable Synchronization Scheme, Recirculation MUX Synchronization Scheme, and MUX-Select Sync (Without Recirculation) Synchronization Scheme, fail if a black box is hit while traversing back from a flip-flop enable pin or a MUX select pin, respectively.

The `signal_in_domain` constraint is used in case a black box instance where more than one clock is reaching is encountered while marking clock domains. By default, all the input/output pins of such black boxes are considered in the domains of all the clocks reaching to the black box as it is not possible to ascertain the exact clock domain of black box input/output pins.

**NOTE:** Use this constraint if you want to associate or specify a clock on an input or output port, whereas, use the [assume\\_path](#) constraint if you want SpyGlass rules to consider combinational path from an input to output port of a black box.

### Product

SpyGlass CDC solution

### Syntax

Use the `signal_in_domain` constraint to specify that the black box input/output pins are in the same clock domain as the specified black box clock input pin or to specify that the black box output pin is synchronized with respect to the specified black box clock input pin:

```
current_design <du-name>  
  signal_in_domain  
    -name <bb-name>  
    -domain <pin-name>  
    -signal <sig-list>
```

-synchronized

## Arguments

### <du-name>

The module name (for Verilog designs) or the design unit name in *<entity-name>.<arch-name>* format (for VHDL designs).

### -name <bb-name>

The name of the black box module. You can use wildcard characters while specifying the black box module name.

### -domain <pin-name>

The name of the black box clock input or output pin.  
You can also specify virtual clocks to this argument.

### -signal <sig-list>

A list of black box input/output pin names. You can use wildcard characters while specifying pin names.

**NOTE:** *You must specify only relative path of black box input/output pins. This argument does not accept a complete hierarchical path.*

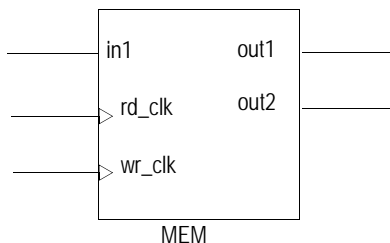
### -synchronized

The *-synchronized* argument indicates that black box output pins in the signal list *<sig-list>* are synchronized in the domain of a clock specified through the *<pin-name>* argument. This information can be useful where the specified black box output is used as a synchronized enable controlling a data crossing. In such a case, the data crossing is considered synchronized.

**NOTE:** *It is recommended to use [abstract\\_port](#) to specify the clocks and related information for black box output ports.*

## Examples

Consider the following example:



**FIGURE 63.** Signal\_in\_domain Constraint Example

The output pins out1 and out2 are in rd\_clk and wr\_clk domains, respectively. The input pin in1 is in the wr\_clk domain. This is specified using the signal\_in\_domain constraint as follows:

```
signal_in_domain -name MEM -domain rd_clk -signal out1
signal_in_domain -name MEM -domain wr_clk -signal out2
signal_in_domain -name MEM -domain wr_clk -signal in1
```

In the example given above, if the output pin out2 is considered to be synchronized in domain wr\_clk, this is specified using the signal\_in\_domain constraint as follows:

```
signal_in_domain -name MEM -domain wr_clk -signal out2 -
synchronized
```

**NOTE:** Multiple signal\_in\_domain should not be assigned on the same output pin of a black box.

**NOTE:** The assume\_path and signal\_in\_domain constraints should not propagate different domains on the same output pin of a black box.

## Rules

The signal\_in\_domain constraint is used by the following rules:

SpyGlass CDC Solution			
Ac_unsync01	Ac_unsync02	Clock_sync08	Clock_sync03a
Clock_sync03b	Ac_handshake01	Clock_sync08a	Clock_sync09
Ac_handshake02	Ac_cdc01a	Ac_cdc01b	Ac_cdc01c

Ac_cdc08	Propagate_clocks	Ac_conv01	Ac_conv02
Ac_conv03	Ac_sync02	Ac_sync01	

## signal\_type

The `signal_type` constraint specifies if a signal is a *Control Signal* or *Data Signal*.

For information on using this constraint, refer to its *Syntax* and *Arguments*.

## Control Signal

A control signal is a signal that is:

- The output of a clock-domain crossing, the source of a clock-domain crossing, or an input port acting as the source for a clock-domain crossing.
- Considered as synchronized by any of the following synchronization schemes (also known as control synchronization schemes):
  - Conventional multi-flop synchronization scheme
  - Synchronizing cell synchronization scheme
  - Qualifier synchronization scheme by using the *qualifier-crossing* constraint
  - Recirculation MUX synchronization scheme

This means that even if a control signal has a valid qualifier present in its input cone but it is not synchronized by any of the control synchronization schemes, it will be reported as unsynchronized by *Ac\_unsync01/ Ac\_unsync02* rules.

For example, consider the scenario shown in the following figure:

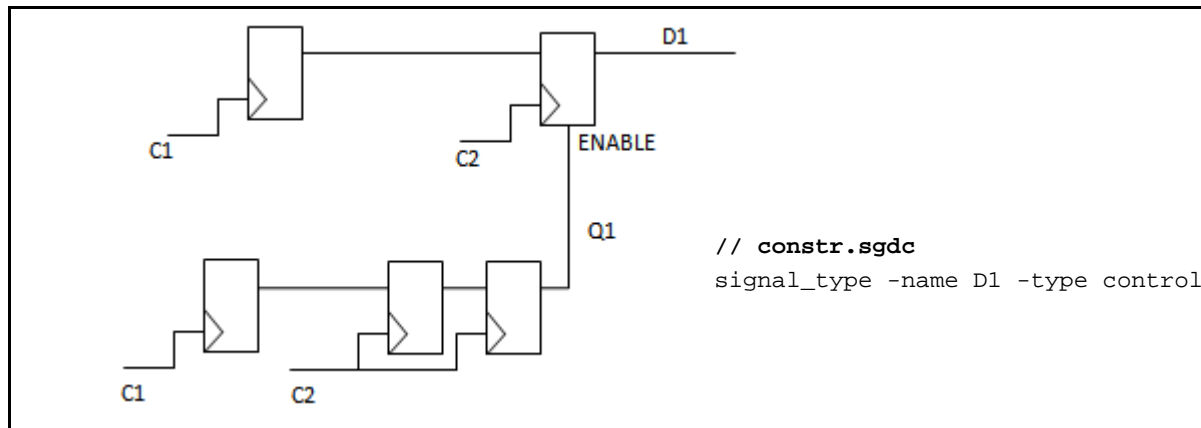


FIGURE 64.

In the above figure, although the Q1 qualifier exists on the enable pin, the control signal D1 is not synchronized by any of the control synchronization schemes. Therefore, SpyGlass reports the D1 signal as unsynchronized.

### The *Ac\_glitch03* Rule Behavior for Control Signals

The *Ac\_glitch03* rule performs glitch checks on the unsynchronized crossings having a virtual domain on the output port when both the following conditions hold true:

- The output port is specified as `-type control` in the `signal_type` constraint.
- The `glitch_on_vck_port` value is specified to the `cdc_reduce_pessimism` parameter.

For details on the *Ac\_glitch03* rule behavior based on the above conditions, refer to the documentation of the *Ac\_glitch03* rule.

### Considering the Output of Convergence as a Valid Qualifier

When a qualifier merges with a source, the output of convergence is not considered as a valid qualifier to qualify other sources if the `allowed_merged_qualifier` parameter is set to `no`. However, if the source is a control signal, the convergence output may still act as a qualifier.

By default, the `allowed_merged_qualifier` parameter is set to `yes`, and such convergence outputs are considered as valid qualifiers.

## Data Signal

A data signal is a signal that is:

- The output of a clock-domain crossing, the source of a clock-domain crossing, or an input port acting as the source for a clock-domain crossing.
- Considered as synchronized by a data synchronization scheme, which is a scheme other than the following control synchronization schemes:
  - Conventional multi-flop synchronization scheme
  - Synchronizing cell synchronization scheme
  - Qualifier synchronization scheme by using the *qualifier*-crossing constraint

This means that even if a data signal is synchronized by a multi-flop or a synchronizer cell (specified by the *sync\_cell* constraint), it will be reported as unsynchronized by *Ac\_unsync01/Ac\_unsync02* rules.

For example, consider the scenario shown in the following figure:

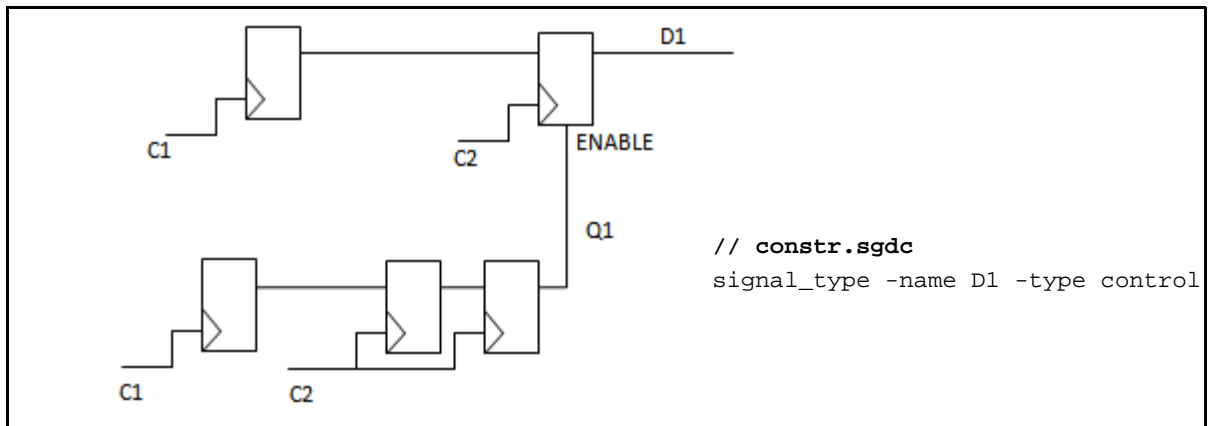


FIGURE 65.

In the above figure, the data signal D1 has a multi-flop structure in its

path, but it is not synchronized by any of the data synchronization schemes. Therefore, SpyGlass reports the D1 signal as unsynchronized.

## Product

SpyGlass CDC solution

## Syntax

The syntax of the `signal_type` constraint is as follows:

```
signal_type
  -name <signal-name>
  -type <control | data>
```

## Arguments

**-name <signal-name>**

Name of a signal that can be any of the following:

- Output of a clock-domain crossing
- Source of a clock-domain crossing
- Input port acting as the source for a clock-domain crossing

You can use wildcard characters while specifying a signal name.

**-type <control | data>**

Signal type as `data` or `control`.

## Rules

The `signal_type` constraint is used by the following rules:

<b>SpyGlass CDC Product</b>			
Ac_sync01	Ac_sync02	Ac_unsync01	Ac_unsync02
Ac_glitch03			

## simulation\_data



The `simulation_data` command is used to specify the initial state sequence for the design.

You can specify the initial state sequence in a Tcl file or have the product read it from a VCD file.

**NOTE:** *The `simulation_data` command is not required for combinational analysis.*

## Product

SpyGlass TXV, SpyGlass CDC, and SpyGlass Auto Verify

## Syntax

The syntax of the `simulation_data` constraint is as follow:

```
current_design <du-name>
  simulation_data
    -type <tcl | vcd>
    -file <file-name>
    [ -mode <mode-name> ]
    [ -time <value> ]
    [ -scopename <block-name> ]
    [ -modulename <module-name> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the initial state sequence.

### -type <tcl | vcd>

The `-type` argument specifies the input file type as `tcl` for Tcl file or `vcd` for VCD file.

### -file <file-name>

The `-file` argument specifies the Tcl file or the VCD file.

**-mode <mode-name>**

The `-mode` argument is optional and is used to specify the applicable mode (one of the mode values specified with the `-mode` argument of the `sdc_data` constraint.) If you do not specify the `-mode` argument, the SpyGlass assumes that the specified initial state sequence is applicable for all modes.

**-time <value>**

The `-time` argument is optional and is used to read the initial state by using a timestamp.

**-scopename <block-name>**

The `-scopename` argument is used if the given VCD file is for a full chip but you want to perform verification for a sub-block module. The hierarchical path of the block module's instantiation (in VCD) must be specified to the `simulation_data` constraint.

For example, consider a VCD file with a top module `top` and a sub module `block2` that lies within another block `block1`. Therefore, while performing the verification of `block2`, the following must be defined to initialize the `block2` module from the VCD file generated for the full chip design:

```
current_design block2
simulation_data -type vcd -scopename
"top.block1_inst.block2_inst" -file chip.vcd
```

**-modulename <module-name>**

The `-modulename` argument is used to specify the hierarchical name of the module in the design corresponding to the module instance in the VCD file specified in the `-scopename` option.

For example, if `top.block1_inst.block2_inst` in the VCD hierarchy corresponds to `block1.block2` in the design, specify the `simulation_data` constraint as:

```
current_design block2
simulation_data
```

```
-type vcd
-scopeName "top.block1_inst.block2_inst"
-moduleName "block1.block2" -file chip.vcd
```

**NOTE:** For sequential analysis, it is recommended that you either provide sufficient reset and set\_case\_analysis constraints so that the SpyGlass TXV solution automatically reads the initial state sequence or specify the simulation\_data constraint. If the number of sequential elements initialized is high, the quality of the result produced by the SpyGlass TXV solution is good.

## Examples

Consider the example below:

```
simulation_data -type vcd -moduleName A -scopeName B -file
design.vcd
```

In the above example, A is the name of the top module and B is the sub block module name. Here, the top module name A will replace the sub block module name B and initial state sequence is read for the sub block module.

An example of the Tcl File is as follows:

```
set sig1 0xffffffff
incr a
for {set i 1} {$i<=4} {incr i} {
    force xyz [incr a 4]
    simulate 2 -clk clk1
}
force sig2 [incr a]
simulate 2
```

## Rules

The initState constraint is used by the following rules:

---

**SpyGlass TXV Solution**

All rules

---

**SpyGlass CDC Solution**

---

`Ac_initstate01`

---

**SpyGlass Auto Verify Solution**

---

`Av_initstate01`

---

## special\_cell

### Purpose

The `special_cell` constraint is used to specify special cells as checked by the LPSVM37 and LPPLIB13 rules.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `specialcell`.*

### Product

SpyGlass Power Verify solution

### Syntax

The syntax to specify the `special_cell` constraint is as follows:

```
current_design <du-name>
  special_cell
    -name <cell-name-list>
    [ -regions <inst-name-list> ]
    [ -supplies <supply-name-list> ]
```

### Arguments

**<du-name>**

Name of the design unit under which you are specifying the special cells.

**-name <cell-name-list>**

Space-separated list of special cell names. You can use wildcard characters while specifying cells using the `-name` argument.

**-regions <inst-name-list>**

Space-separated hierarchical instance name list (name of any instance or the top-level design unit) indicating the regions to be checked as used by the LPSVM37 rule of the *SpyGlass Power Verify* solution.

**-supplies <supply-name-list>**

Space-separated supply name list as used by the LPPLIB13 rule.

**Rules**

The `special_cell` constraint is used by the following rules:

<b>SpyGlass Power Verify Solution</b>	
LPSVM37	LPPLIB13

**special\_module****Purpose**

The `special_module` command is used to define property and constraint modules. All OVL modules are supposed to be specified as special modules. If not, these modules are treated as regular design modules.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `smodule`.*

**Product**

SpyGlass Auto Verify solution

**Examples**

The following command in a SpyGlass Design Constraints file will treat all modules whose name starts with "assert\_" as assertion or constraint modules.

```
special_module -type ASSERTION -name "assert_*" -regex
```

In the presence of OVL modules, this constraint is a requirement to run

SpyGlass Auto Verify solution (otherwise the OVL checks are not performed). It is included as a constraint in the goal for SpyGlass Auto Verify solution. Therefore, you need not specify this, unless an OVL assertion is written which does not start with the keyword "assert". For example, if you define new OVL assertions, all starting with the prefix "user1...", the following command should be given in the SpyGlass Design Constraints file:

```
special_module -type ASSERTION -name "user1_*" -regexp
```

## Rules

The `special_module` constraint is used by the following rule:

---

**SpyGlass Auto Verify Solution**

---

Av\_ovl01

---

## spef\_data

### Purpose

The `spef_data` constraint specifies the SPEF (Standard Parasitic Extraction Format) file.

For post-route ("signoff") power estimation, accuracy improves if the actual routed parasitics are used to get the net capacitances instead of wire-load models. The actual routed parasitics are stored in the SPEF file, which is an accepted industry standard for storing extracted parasitics.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `spef_data` constraint is as follows:

```
current_design <du-name>  
  spef_data -file <spef-file-name>  
    [ -modname <du-name> ]
```

```
[ -instname <inst-name> ]
[ -spef_topname <top-name> ]
```

**NOTE:** *If the capacitance of a net is specified using both `spef` file and `set_load` command, the `set_load` is given preference.*

## Arguments

### <du-name>

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

### -file <spef-file-name>

Name of the SPEF file.

**NOTE:** *You can also specify compressed SPEF file that has been generated by using the `gzip` utility.*

### -modname <du-name>

(Optional) Name of the design unit (specified as module name or the entity name) for which SPEF file is specified.

### -instname <inst-name>

(Optional) Name of the instance for which SPEF file is specified.

In case neither `modname` nor `instname` is given, the SPEF file will be applied for the design unit specified using `DESIGN` in the SPEF file.

### -spef\_topname <top-name>

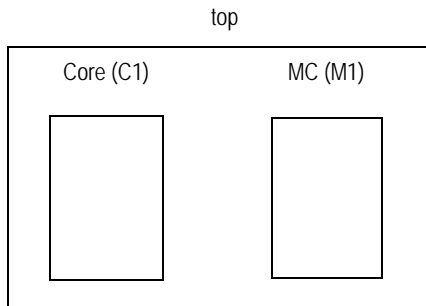
(Optional) Path of the hierarchy in the SPEF file corresponding to the design.

This argument is useful if you want to use SPEF information for a specific instantiated module from a top-level SPEF file. In this case, the `spef_topname` argument is used to specify the instantiated name of the module in the SPEF file.

## Examples

### Example 1 - Specifying modname and instname

An example of `spef_data` constraint is given below:



**FIGURE 66.** `spef_data` Constraint Example

Consider a top design module top consisting of two blocks - Core (instance C1) and MC (instance M1). There are three SPEF files - top.SPEF (for top), Core.SPEF (for instance top.C1), and MC.SPEF (for instance top.M1).

The SPEF files can be specified as follows:

```
current_design top
    spef_data -file top.SPEF -modname top
    spef_data -file Core.SPEF -instname top.C1
    spef_data -file MC.SPEF -instname top.M1
```

**NOTE:** While calculating the capacitance value for any net from the spef file, the coupling capacitance value can be included or excluded using `pe_include_coupling_capacitance` switch.

### Example 2 - Specifying spef\_topname

Suppose a design has an instance U0 of module MC\_RF. This module is inside the TOP module. For the design, the associated SPEF file is TOP.SPEF.

From the TOP.SPEF file, you can use the SPEF data only for the instance U0



by specifying the following SGDC constraints:

```
current_design MC_RF
```

```
spef_data -file TOP.SPEF -spef_topname U0
```

**NOTE:** *Hierarchical format is supported for specifying spef\_topname argument. The hierarchy separator used in the argument must match the separator defined in the SPEF file.*

## Rules

The spef\_data constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## supply

### Product

SpyGlass Power Verify solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### For SpyGlass Power Verify solution

#### Purpose

The supply constraint is used to specify the supply and ground port names for all the LPPLIB rules.

#### Syntax

The syntax to specify the supply constraint is as follows:

```
current_design <du-name>
  supply
    -name <name>
```

```
-value <fvalue>  
[ -net <net-name> ]  
[ -on <sig-name-list> ]  
[ -on_2 <sig-name-list> ]  
[ -parent <supply-name> ]  
[ -alwayson 0 | 1 ]  
[ -onval 0 | 1 ]  
[ -onval_2 0 | 1 ]  
[ -isosig <sig-name-list> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the power and ground ports

### -name <name>

Simple name of the power/ground port (post-synthesis pin)

### -value <fvalue>

Voltage value at the specified port.

### -net <net-name>

(Optional) Name of the supply nets that are inside a block. For example, the name `ua.VDDX` will be used to refer to supply net inside the block `top.ua`.

### -on <sig-name>

(Optional) Space-separated list of `-on` signals. For each power switch instance, the enable is connected to the `-on` signal. If an expression involving several terms is given, an error will be reported. Also, there is an interaction between the `-on` switch and the `-alwayson` switch. If the `-on` argument is specified, `-alwayson` should not be specified as '1'. If `-alwayson` is 1 and `-on` is also specified, an error will be reported.

**-on\_2 <sig2-name>**

(Optional) Space-separated list of `-on_2` signals. In case of dual enable power switch, for each power switch instance, the second enable is connected to the `-on_2` signal.

**-parent <supply-name>**

(Optional) Name of the parent supply net. The input and output supply net for power switches have a parent-child relationship. That is, the supply net outside the domain is the parent, and the supply net inside the domain is the child. To derive the parent-child relationship the `-parent` switch is specified. When the option is specified, for any supply `X` that has power switches, the parent supply should be connected to the input power pin of the power switch, and the supply `X` should be connected to the output power pin of the power switch.

**-alwayson <0 | 1>**

(Optional) The type of supply. Can be set to 0 (switched off) or 1 (always on).

If `-alwayson` (and also `-on` and `-parent` arguments) is not specified with the `supply` constraint, supply is considered an always-on supply (`-alwayson` is set to 1). If `-on` and `-parent` arguments are specified, supply is considered a switched off supply (`-alwayson` is set to 0).

**-onval <0 | 1>**

(Optional) Space-separated list of values for the `-on` signals. Can be set to 0 (active low) or 1 (active high).

**-onval\_2 <0 | 1>**

(Optional) Space-separated list of values for the `-on_2` signals. Can be set to 0 (active low) or 1 (active high).

**-isosig <sig-name-list>**

(Optional) Name of the isolation signal(s) for the supply being defined as a space-separated list.

## Rules

The supply constraint is used by the following rules:

---

### SpyGlass Power Verify Solution

---

All LPPLIB rules

---

## For the SpyGlass Power Estimation and SpyGlass Power Reduction Solutions

### Purpose

The supply constraint is used to specify the supply rails information.

**NOTE:** *Use the supply constraint for multiple voltage domain designs only. For single voltage domain designs, the supply rails information is inferred from the associated technology library.*

### Syntax

The syntax to specify the supply constraint is as follows:

```
current_design <du-name>
  supply
    -name <supply-name>
    -value <fvalue>
    [ -on <on-expr> ]
    [ -isnetdriver ]
```

### Arguments

#### <du-name>

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

#### -name <supply-name>

Name of the supply rail.

**-value <fvalue>**

The voltage value for the supply rail.

Use the `-on` argument to specify the condition for supply rails that may be off under certain conditions. `<on-expr>` is the signal-based Boolean expression that indicates the supply rail ON condition.

**-isnetdriver**

The `-isnetdriver` argument specifies whether the supply rail is a net driver. You can use this argument to specify the net drivers.

**Examples**

The following value indicates that supply rail S1 has a voltage value of 1.2:

```
supply -name S1 -value 1.2
```

The following example specifies a supply rail named A1 that has a supply voltage of 1.2 and is ON only when both `in1` and `in2` signals are high:

```
supply -name A1 -value 1.2 -on top.in1 & top.in2
```

The following example specifies a supply rail named G123 that has a supply voltage of 1.2 and is a net driver:

```
supply -name G123 -value 1.2 -isnetdriver
```

**Rules**

The `supply` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions		
PEPWR01	PEPWR02	poweraudit

**switchoff\_wrapper\_instance****Purpose**

The `switchoff_wrapper_instance` constraint is used to specify the

instance hierarchy.

## Product

SpyGlass Power Verify solution

## Syntax

The syntax to specify the `switchoff_wrapper_instance` constraint is as follows:

```
current_design <du-name>  
  switchoff_wrapper_instance  
    -name <hier-list>
```

## Arguments

**<du-name>**

Name of the design unit under which you are specifying the instance hierarchies.

**-name <hier-list>**

Space-separated list of hierarchical references.

## Examples

The following example specifies `top.SOW1` and `top.SOW2` as the instance hierarchies:

```
switchoff_wrapper_instance -name top.SOW1 top.SOW2
```

## Rules

The `switchoff_wrapper_instance` constraint is used by the following rule:

---

**SpyGlass Power Verify Solution**

LPSVM55

---

## sync\_cell

### Purpose

The `sync_cell` constraint is used to specify synchronizer cells that should be considered valid for control crossings that contain specified frequencies, source/destination clocks, or domains.

**NOTE:** *If you use the `synchronize_cells` or `synchronize_data_cells` parameter along with this constraint, the constraint is used for the specified clocks and frequencies and the parameter will be applicable for the rest of the crossings.*

**NOTE:** *If the `synchronize_cells` parameter is defined and the `sync_cell` constraint is specified with the `-from_clk/to_clk/from_domain/to_domain` arguments, then the `sync_cell` constraint is honored for the specific clock/domain pair. For other clock/domain pairs, the `synchronize_cells` parameter is honored. However if the `sync_cell -name` constraint is used without any argument, then it is honored by default and the parameter is ignored.*

**NOTE:** *This constraint is not applicable for data crossings. It is applicable for control crossings.*

### Product

SpyGlass CDC solution

### Syntax

The syntax to specify the `sync_cell` constraint is as follows:

#### Usage 1:

```
sync_cell
  -name <synchronizer-modules>
  [ -to_clk <dest-clk>
    [ -from_clk <src-clk> ]
  ]
  [ -user_tag <user-tag-string> ]
  [ -rdc ]
  [ -reset ]
```

**Usage 2:**

```
sync_cell
  -name <synchronizer-modules>
  [-to_domain <dest-domain
    [ -from_domain <src-domain> ]
  ]
  [ -user_tag <user-tag-string> ]
  [ -rdc ]
  [ -reset ]
```

**Usage 3:**

```
sync_cell
  -name <synchronizer-modules>
  [ -to_period <dest-period>
    [-from_period <src-period>]
  ]
  [ -user_tag <user-tag-string> ]
  [ -rdc ]
  [ -reset ]
```

**Usage 4:**

```
sync_cell
  -name <synchronizer-modules>
  [ -user_tag <user-tag-string> ]
  [ -rdc ]
  [ -reset ]
```

**NOTE:** Please note the following points:

- You can specify any one of the following combination of arguments:

- from\_clk and -to\_clk
- from\_domain and -to\_domain
- from\_period and -to\_period

You cannot mix the above combinations.



- You must specify the `-to_clk` argument while specifying the `-from_clk` argument.
- You must specify the `-to_domain` argument while specifying the `-from_domain` argument.
- You must specify the `-to_period` argument while specifying the `-from_period` argument.

## Arguments

### `-name <synchronizer-modules>`

Space-separated list of synchronizer cells/modules. You can use wildcard characters while specifying such modules.

Details of such modules are discussed in the following points:

- Such modules can either be user-defined modules containing at least one sequential element or it can be a sequential library cell.

**NOTE:** *Such modules cannot be black boxes. For black boxes, use the [qualifier constraint](#) along with the `synchronize_cells` parameter.*

- You can specify multiple synchronizer cells for the same clock, domain, or period, as shown in the following example:

```
sync_cell -name SYNC1 SYNC2 SYNC3 -to_period 10
```

- If clock names are specified in some `sync_cell` constraints and in other constraints, domain or period is specified for those clocks, all the `sync_cell` constraints will be considered.

Consider the following example:

```
clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d2 -period 20
sync_cell -name SYNC1 -from_clk top.clk1 -to_clk top.clk2
sync_cell -name SYNC2 -from_domain d1 -to_domain d2
sync_cell -name SYNC3 -to_period 20
```

In this case, all the SYNC1, SYNC2, and SYNC3 synchronizer cells will be considered valid for crossings from `clk1` (`d1`) to `clk2` (`d2`).

- For the same synchronizer cell, if multiple `sync_cell` constraints are specified for different clocks, domains, and clock periods, the synchronizer cell is considered valid for all the specified clocks, domains, and clock periods.

This is explained in the following examples:

```
// SYNC1 cell is valid for crossings from
// CLK1 to CLK2 and crossings with
// destination CLK3
sync_cell -name SYNC1 -from_clk CLK1 -to_clk CLK2
sync_cell -name SYNC1 -to_clk CLK3

// SYNC2 cell is valid for crossings with destination
// clock period 20 and 40
sync_cell -name SYNC2 -to_period 20
sync_cell -name SYNC2 -to_period 40

// SYNC3 cell is valid for crossings from domain
// D1 -> D2 and also for crossings from D3 to D4
sync_cell -name SYNC3 -from_domain D1 -to_domain D2
sync_cell -name SYNC3 -from_domain D3 -to_domain D4
```

**-from\_clk <src-clk>**

Name of source clock.

You can use wildcard characters while specifying a source clock.

**-to\_clk <dest-clk>**

Name of destination clock.

You can use wildcard characters while specifying a destination clock.

**-from\_domain <src-domain>**

Name of source domain.

**-to\_domain <dest-domain>**

Name of destination domain.

**-from\_period <src-period>**

Source clock period value.

**-to\_period <dest-period>**

Destination clock period value.

**-user\_tag <user-tag-string>**

A string that is appended to the control-crossings-related violation messages of rules using this constraint.

Use this string to filter messages as per your requirement. For example, you add a custom filter in a spreadsheet to show or hide messages containing the specified string.

**-rdc**

Enables the *Ar\_resetcross01* apply the *sync\_cell* constraints on reset domain crossings.

**-reset**

Specifies that a clock domain crossing on reset path be considered as synchronized.

## Examples

### Example 1

Consider the following constraints:

```
sync_cell -name SYNC1 -from_period 50 -to_period 10
sync_cell -name SYNC2 -to_period 20
```

In this case, *SYNC1* is the valid synchronizer module/cell for crossings with destination clock period 10 and source clock period 50. However, for crossings with destination clock period 20, *SYNC2* is the valid synchronizer.

## Rules

The `sync_cell` constraint is used by the following rules:

SpyGlass CDC Solution			
Clock_sync08	Clock_sync03a	Clock_sync03b	Ac_handshake01
Ac_handshake02	Ar_resetcross_matrix01	Ar_cross_analysis01	Clock_sync08a
Clock_sync09	Ac_cdc01a	Ac_cdc01b	Ac_cdc01c
Ac_cdc08	Propagate_Clocks	Ac_conv01	Ac_conv02
Ac_conv03	Ac_glitch02	Ac_sync01	Ac_sync02
Ac_unsync01	Ac_unsync02	Ac_glitch03	Ar_resetcross01
Ac_conv04			

## sync\_reset\_style

### Purpose

The `sync_reset_style` constraint is used by SpyGlass CDC solution during detection and validation of a synchronous reset in a design.

### Product

SpyGlass CDC solution

### Syntax

The following is the syntax of the `sync_reset_style` constraint:

```
sync_reset_style
[ -max_load <max-sequential-elements> ]
[ -min_load <min-sequential-elements> ]
[ -combo <yes | no | buf_inv> ]
[ -active <low | high | both> ]
[ -pragma <yes | no | mixed> ]
[ -first_if <yes | no> ]
```

## Arguments

### **-max\_load <max-sequential-elements>**

Specifies the maximum load allowed on a synchronous reset.

This load is specified in terms of the total number of the following types of terminals driven by a synchronous reset:

sequential element terminals	tristate terminals
mux instance control terminals	black box terminals

The default value of this argument is -1, which means that any load is allowed on a synchronous reset. You can specify the load ranging from 0 to any positive integer value.

### **-min\_load <min-sequential-elements>**

Specifies the minimum load allowed on a synchronous reset.

The default value of this argument is 0, which means that no load is allowed on a synchronous reset. You can specify the load ranging from 0 to any positive integer value.

### **-combo <yes | no | buf\_inv>**

Specifies if combo logic is allowed between the reset source and data pin of a flip-flop.

You can specify any of the following values for this argument:

Value	Description
yes	(Default) Specifies that all combinational logic is allowed
no	Specifies that no combo logic, except SpyGlass-generated buffer, is allowed in the reset path
buf_inv	Specifies that any number of buffers, inverters, or combinational logic equivalent to buffer

### **-active <low | high | both>**

Specifies the polarity of the usage path of a synchronous reset signal.

You can specify any of the following values for this argument:

Value	Description
low	Specifies that the reset is active-low
high	Specifies that the reset is active high
both	(Default) Specifies that the reset can be used as active-high or active-low

**NOTE:** *SpyGlass ignores the `-active` argument when flip-flops are created by using a library cell instance.*



**-pragma** <yes | no | mixed>

Specifies if the `synopsys_set_reset` pragma should be present in the RTL.

You can specify any of the following values for this argument:

Value	Description
yes	Specifies that pragma should be present in all usage of RTL
no	(Default) Specifies that pragma may or may not be present in the synchronous usage of RTL. SpyGlass CDC solution does not check the presence of pragma in such cases.
mixed	Specifies that pragma should be present in at least one usage of synchronous reset at RTL

**NOTE:** *Please note the following points:*

-  *VHDL does not have any pragmas, therefore, all flip-flops created using VHDL RTL are considered with presence of pragmas to avoid reporting for pragma mismatch in VHDL code.*
-  *SpyGlass CDC solution this argument in case of netlist design.*

**-first\_if** <yes | no>

Specifies the modeling of synchronous reset usage in the first `if` condition of a synchronous block.

If you set this argument to yes, synchronous reset should be used in the first `if` condition.

By default, this argument is set to `no`, which means that no rule-checking is done. In this case, synchronous reset may or may not be used in the first `if` block in the RTL design.

## Rules

The `sync_reset_style` constraint is used by the following rules:

SpyGlass CDC Solution		
Reset_info01	Ar_syncrstactive01	Ar_syncrstcombo01
Ar_syncrstload01	Ar_syncrstload02	Ar_syncrstpragma01
Ar_syncrstrtl01		

## test\_mode

### Purpose

The `test_mode` constraint specifies the set of conditions, both pins and values, that when simulated, will force the circuit in test mode.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `testmode`.*

### Product

SpyGlass DFT solution, SpyGlass DFT DSM solution, SpyGlass STARC product, SpyGlass STARC02 product, SpyGlass STARC05 product, and SpyGlass STARCad-21 product

## For SpyGlass DFT solution and SpyGlass DFT DSM solution

In SpyGlass DFT solution, the `test_mode` constraint when specified forces the circuit into a state for scan shifting or for capture. In mux-scan, for example, during scan shift, all internally generated clocks should be bypassed with a test clock and all flip-flop sets and resets should be held inactive. In capture mode, all system latches should be made transparent.

Generally, there are cases in which the same pin must have different values to differentiate scan shift from capture. To support such cases, the

syntax of the `test_mode` constraint enables you to provide the following types of values:

- Values that are the same in shift and capture (no optional syntax required)
- Values that complement from shift to capture (use the `-invertInCapture` argument)
- Values that are non-X in shift and X in capture (use the `-scanshift` argument)
- Values that are X in shift and non-X in capture (use the `-capture` argument)

When `test_mode`, `force_ta`, or `test_point` constraints are specified on the same node, following is the priority among different constraints:

- `Test_mode`
- User-specified specific *`force_ta`* / *`test_point`*
- Effect of *`dft_treat_primary_inputs_as_x_source`* and *`dft_treat_primary_outputs_as_unobservable`* parameters

For example, if `test_mode 1` and `test_point control` are applied on the same node then the `test_mode` constraint will be considered.

Also, if the `test_mode`, `force_ta`, or `test_point` constraints are found in the fanout of each other, following is the priority among different constraints:

- The constraint in the fanout gets the priority
- Fanin effect is blocked by the specified / resolved constraint on the node

For example, consider that `test_mode 1` is applied on the input of buffer and `test_point control` is applied on the output of the same buffer. In this case, input will have simulation value 1 and `nyn` controllability but output will have `yyn` controllability and no simulation value.

## Syntax

The syntax of the `test_mode` constraint is as follows:

```
test_mode
  [-name <name>]
  -value <value>
```



```

[ -scanshift ]
[ -capture ]
[ -functional ]
[ -captureATspeed ]
[ -invertInCapture ]
[ -initialize_for_bist ]
[ -memory_force ]
[ -type <design_object_type> ]
[ -except <except_node_name> ]
[ -except_type <except_design_object_type> ]
[ -filter_in_name <include_node_name> ]
[ -filter_in_type <include_design_object_type> ]
[ -ignorecase ]
[ -power_ground ]

```

**NOTE:** *The test\_mode constraint supports wildcard characters.*

## Arguments

**-name <name>**

Complete hierarchical name of a test mode port/pin.

The pin can be a primary pin as well as an internal pin.

You can specify a single port/pin's full hierarchical name or a space-separated list of full hierarchical port/pin names.

This is an optional field and you can specify multiple names in a single constraint without being split.

**NOTE:** *Specify at least one of -name or -type arguments.*

The following example shows that the test\_mode constraint has been defined on internal net top.U1.U13.tm1 and top.U3.tm3:

```

current_design top
  test_mode -name top.U1.U13.tm1, top.U3.tm3 ...

```

For primary ports, you can also specify the simple port name as in the following example:

```

current_design top

```

```
test_mode -name in15 ...
```

**-value <value>**

Value list for a test mode pin.

The value list is the sequence of one or more of the following values:

- 0
- 1
- X
- Z
- Combination
- 'p'(010) / 'P'(010)
- 'n'(101) / 'N'(101)

The 'p'(010) denotes a rising edge or a positive edge. The value 'n'(101) denotes a falling edge or a negative edge. Here a bit corresponds to a pulse. Therefore, if you specify, '1' or '0' or 'X' or 'Z', they will be repeated three times in the command.

See the [Usage of Different Clock Edges](#) section, to understand the usage of different clock edges.

Applying these values to the test mode pin causes the circuit to enter the test mode.

You can specify repeat sequences for the `test_mode` constraint.

For fields that require repeat sequence, you can specify the values as `<I*S>`. Here, `S` is any string that does not contain the `<`, `>`, and `*` characters. However, `S` can contain another `<I*S>` expression. `I` is an integer that is always interpreted as a decimal value. The expression `<I*S>` means that the sequence `S` will be repeated `I` number of times.

**-scanshift**

(Optional) Indicates that the test mode pin is only required during the scan shifting operations. Such a pin may be used for ATPG purposes when not scanning.

**NOTE:** *You can specify both the `-scanshift` argument and the `-capture` argument together.*

**NOTE:** Use of the `-scanshift` argument will generally increase the estimated fault coverage since both 0 and 1 values are allowed on such pins, whereas without this argument only the specified value is allowed.

### **-capture**

(Optional) Indicates that the test mode pin is only required during the capture operations.

**NOTE:** You can specify both the `-scanshift` argument and the `-capture` argument together.

### **-functional**

(Optional) Indicates that the test mode signal is only required during the functional mode.

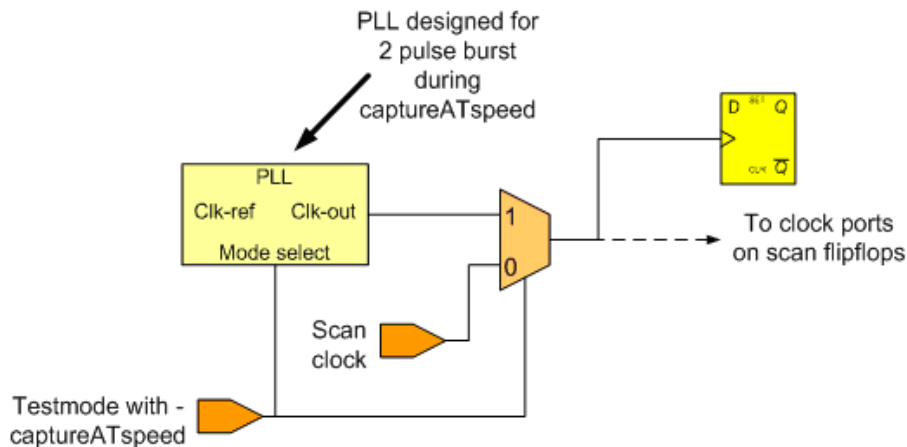
**NOTE:** The `set_case_analysis` constraint is treated as the `test_mode -functional` constraint.

### **-captureATspeed**

**NOTE:** This option is not applicable for the SpyGlass DFT solution.

(Optional) You should use this option to define test mode signals exclusively for the at-speed rules.

This option helps you to apply the test mode condition exclusively for at-speed capture cycle that can be different from the capture used by stuck-at testing or from scan shifting. Consider the following figure:



**FIGURE 67.** PLL burst during captureATspeed

### **-invertInCapture**

(Optional) Indicates that the value specified with the `-value` argument should be simulated in the shift mode as is whereas the inverse of this value (complement of the value for single-bit values or last-bit complement of pattern values) should be simulated in the capture mode.

### **-initialize\_for\_bist**

Defines a way to drive all registers to a known state.

You can drive registers to a known state asynchronously, sequentially, or in some combination of both.

If there is a global reset signal, an `initialize_for_bist` argument specifies that signal and the value that makes all asynchronous sets or resets ACTIVE.

### **-memory\_force**

Lists the primary ports and values that, when simulated, will force all memories used in a circuit to have non-x values on their outputs.

**-type <design\_object\_type>**

(Optional) Same as -name argument but it takes only macros as input.

**NOTE:** *Both static-type and dynamic-type macros are supported.*

To view the list of macros supported by test\_mode constraint, see Supported Macros.

**-except <except\_name>**

(Optional) Same as the the -name argument, but defines design nodes that are to be excluded.

**-except\_type <except\_design\_object\_type>**

(Optional) Same as the -design argument, but defines design nodes that are to be excluded.

**-filter\_in\_name <include\_name>**

(Optional) Same as the -name argument, but defines design nodes that are to be included.

**-filter\_in\_type <include\_design\_object\_type>**

(Optional) Same as the -type argument, but defines design nodes that are to be included.

**-ignorecase**

(Optional) To ignore the case for a test mode name specified using the -name, -except, or -filter\_in\_name arguments.

**NOTE:** *It applies on all fields which take design\_node\_name as an input.*

**-power\_ground**

(Optional) To specify a constant value to a design node in the power ground mode.

**NOTE:** *The test\_mode constraint considers the specified nets or other design nodes as VCC or VSS.*

## Notes

1. More than one `test_mode` constraint may be necessary for your design. If more than one `test_mode` constraint is used, they will all be simulated in parallel for determining scannability but only the `test_mode` constraints without the `-scanshift` argument will have fixed values when determining test coverage.
2. The values supplied through the `test_mode` constraint are 0, 1, Z, and X (note case-sensitivity for Z and X). Since the simulator built into SpyGlass DFT solution is designed for sequential circuits, the value lists may contain sequences of values. For example, the following constraint directive will cause three rising clock edges on a signal `tclk` and involve six simulation cycles:

```
test_mode -name tclk -value 010101
```

3. Special attention is necessary for a signal that is used both as a test mode signal (that is, used in a `test_mode` constraint) and as a test clock signal (used in a `clock` constraint with `-testclock` argument). In such a case, the last value listed in the `test_mode` constraint must be an X so that the signal is free to be pulsed as a test clock.
4. The last value in all `test_mode` value lists is used by controllability analysis and is preserved for all rules that depend on the `test_mode` constraint. For example, the following constraint will cause the signal `tclk` to be uncontrollable to a logic zero since the last value is 1:  

```
test_mode -name tclk -value 010101
```
5. Test mode signals that are required to maintain the test mode state must not have a value list that ends in X because the X-value, just as 0s and 1s, will be simulated and be the final value on this pin. This will often cause the propagation of X values into the circuit and override other simulation results established by prior values.

## Examples

### General Usage

Consider the following examples:

#### *Example 1*

```
test_mode -name abc -value "<5*10>"
```

The above example will be expanded as follows:

```
test_mode -name abc -value 1010101010
```

### **Example 2**

```
test_mode -name abc -value "11<5*10>010"
```

The above example will be expanded as follows:

```
test_mode -name abc -value 111010101010010
```

### **Example 3**

```
test_mode -name abc -value "<50*11<5*10>>010"
```

The above example will be expanded as follows:

```
test_mode -name abc -value 111010101010...(repeated 50 times followed by 010)
```

You can also set a variable using the command `setvar` to obtain the above result as follows:

```
setvar x 11<5*10>
```

```
test_mode -name abc -value "<50*${x}>010"
```

The above example will be expanded as follows:

```
test_mode -name abc -value 111010101010...(repeated 50 times followed by 010)
```

### **Example 4**

Tagging is not allowed during nesting. For example, the following `test_mode` statements are not allowed:

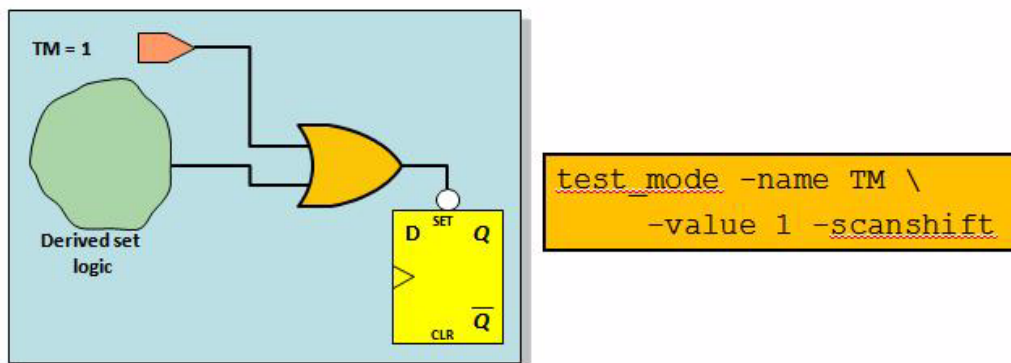
```
test_mode -name sub_seq -value <5*01>
test_mode -name main_seq -value <100*sub_seq>
```

However, you can achieve the same result by using the `setvar` command.

**NOTE:** For information and examples on specifying values for vector signals, refer to "Specifying Values for Vector Signals section" in the *SpyGlass DFT Rules Reference Guide*.

### **Example 5**

Consider the following figure:



In the above example, the value, 1, is applied on the SET pin of the flip-flop ensuring that this pin is inactive during shift mode, that is, when the scan data is shifted in the scan chain. However, during capture phase, the combo logic applies the value, 1, on the same pin, if appropriate inputs are fed to it.

### Example 6

Consider the following sample input values:

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

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "1010"
test_mode -name vec[1] -value "0000"
test_mode -name vec[2] -value "0000"
test_mode -name vec[3] -value "0000"
```

### Example 7

Consider the following sample input values:

```
test_mode -name vec[3:0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "0"
test_mode -name vec[1] -value "1"
test_mode -name vec[2] -value "0"
```



```
test_mode -name vec[3] -value "1"
```

### Example 8

Consider the following sample input values:

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

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "1"
test_mode -name vec[1] -value "0"
test_mode -name vec[2] -value "0"
test_mode -name vec[3] -value "0"
```

### Example 9

Consider the following sample input values:

```
test_mode -name vec -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "1010"
test_mode -name vec[1] -value "0000"
test_mode -name vec[2] -value "0000"
test_mode -name vec[3] -value "0000"
```

### Example 10

Consider the following sample input values:

```
test_mode -name vec -value { b 1010 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "0"
test_mode -name vec[1] -value "1"
test_mode -name vec[2] -value "0"
test_mode -name vec[3] -value "1"
```

### Example 11

Consider the following sample input values:

```
test_mode -name vec -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "1"  
test_mode -name vec[1] -value "0"  
test_mode -name vec[2] -value "0"  
test_mode -name vec[3] -value "0"
```

### Example 12

Consider the following sample input values:

```
test_mode -name vec[0] -value { b 1 0 1 0 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "1010"
```

### Example 13

Consider the following sample input values:

```
test_mode -name vec[0] -value {b 1010}
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "0"
```

### Example 14

Consider the following sample input values:

```
test_mode -name vec[0] -value { b 1 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "1"
```

### Example 15

Consider the following sample input values:

```
test_mode -name sclr -value { b 1 0 1 0 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
test_mode -name sclr -value "1010"
```

### Example 16

Consider the following sample input values:

```
test_mode -name sclr -value { b 1010 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
test_mode -name sclr -value "0"
```

### Example 17

Consider the following sample input values:

```
test_mode -name sclr -value { b 1 }
```

where sclr is the scalar net

The above input is expanded as shown below:

```
test_mode -name sclr -value "1"
```

### Example 18

Consider the following sample input values:

```
test_mode -name vec -value { h 6 }
```

where vec is the 3:0 vector net

The above input is expanded as shown below:

```
test_mode -name vec[0] -value "0"
```

```
test_mode -name vec[1] -value "1"
```

```
test_mode -name vec[2] -value "1"
```

```
test_mode -name vec[3] -value "0"
```

### Example 19

Consider the following sample input values:

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

In above example, the value 1, is applied on both tm[0] and pins.

## Usage of Different Clock Edges

### *Usage of Positive Edge*

Consider the following sample input values:

```
test_mode -name tm1 -value 1 0 0 p
test_mode -name tm2 -value 0 0 1 1
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1
test_mode -name tm5 -value 0 1 1 0
```

The above input is expanded as shown below:

```
test_mode -name tm1 -value 1 0 0 010
test_mode -name tm2 -value 0 0 1 111
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1
test_mode -name tm5 -value 0 1 1 000
```

### ***Usage Of Negative Edge***

Consider the following sample input values:

```
test_mode -name tm1 -value 1 0 0 N
test_mode -name tm2 -value 0 0 1 1
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1
test_mode -name tm5 -value 0 1 1 0
```

The above input is expanded as shown below:

```
test_mode -name tm1 -value 1 0 0 101
test_mode -name tm2 -value 0 0 1 111
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1 000
```

### ***Usage Of Positive Edge In Between***

Consider the following sample input values:

```
test_mode -name tm1 -value 1 0 0 p 0 1
test_mode -name tm2 -value 0 0 1 1 0 1
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1
test_mode -name tm5 -value 0 1 1 0 1 1
```

The above input is expanded as shown below:

```
test_mode -name tm1 -value 1 0 0 010 0 1
```

```
test_mode -name tm2 -value 0 0 1 111 0 1
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1
test_mode -name tm5 -value 0 1 1 000 1 1
```

### ***Usage Of Multiple Edges***

#### **Example 1**

Consider the following sample input values:

```
test_mode -name tm1 -value 1 0 0 p 0 1 n
test_mode -name tm2 -value 0 0 1 1 0 1 0 0 1
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1 1 1 0 0 0 0
test_mode -name tm5 -value 0 1 1 0 1 1 0 1 0
```

The above input is expanded as shown below:

```
test_mode -name tm1 -value 1 0 0 010 0 1 101
test_mode -name tm2 -value 0 0 1 111 0 1 000 0 1
test_mode -name tm3 -value 1 0
test_mode -name tm4 -value 0 0 1 111 1 0 000 0 0
test_mode -name tm5 -value 0 1 1 000 1 1 000 1 0
```

#### **Example 2**

Consider the following sample input values:

```
test_mode -name clk1 -value P P P P 0 N N N
test_mode -name din1 -value 0 0 1 1 0 1 0 0
test_mode -name din2 -value 1 0 0 1
test_mode -name din3 -value 0 0 1 1 1 0 0 0
```

The above input is expanded as shown below:

```
test_mode -name clk1 -value 010 010 010 010 0 101 101 101
test_mode -name din1 -value 000 000 111 111 0 111 000 000
test_mode -name din2 -value 111 000 000 111
test_mode -name din3 -value 000 000 111 111 1 000 000 000
```

### ***Usage of pulse in bit-wise assignment***

Consider the following sample input values:

```
test_mode -name tm[3:0] -value {b 0np1}
```

The above input, is expanded bit-wise as shown below:

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

The above input is expanded bit-wise with pulse expanded:

```
test_mode -name tm[3] -value 000
test_mode -name tm[2] -value 101
test_mode -name tm[1] -value 010
test_mode -name tm[0] -value 111
```

### Usage of UDM in test\_mode

Consider the following constraint specification:

```
define_macro -macro my_macro -name "*powergood*" -
filter_in_type INPUT_PORTS -ignorecase
test_mode -type my_macro -value 1
```

In the above example, a value 1 is defined (as test\_mode) on all input ports for which name matches with "\*powergood\*" (case-insensitive)

### Usage of -ignorecase in test\_mode

Consider the following constraint specification:

```
test_mode -name "mid::*out*" "mid::*in*" -except "mid::*in*"
-ignorecase -value 0
```

In the above example, a value 0 is defined (as test\_mode) on all pins matching "mid::\*in\*" (case-insensitive)

### OR of -name and -type

Consider the following constraint specification:

```
test_mode -name "*powergood*" -type INPUT_PORTS -ignorecase
-value 0
```

In the above example, a value 0 is defined (as test\_mode) on all input ports and objects matching with `"*powergood*"` (case-insensitive). This signifies an OR operation.

### AND (through filter\_in) of -name and -type)

Consider the following constraint specification:

```
test_mode -filter_in_name "*powergood*" -type INPUT_PORTS -
ignorecase -value 0
```

```
test_mode -name "*powergood*" -filter_in_type INPUT_PORTS -
ignorecase -value 0
```

For both of the above cases, test\_mode value 0 is defined on all input ports matching with `"*powergood*"` (case-insensitive). This signifies an AND operation.

### Exclusion based on type

Consider the following constraint specification:

```
test_mode -name "*powergood*" -except_type "INPUT_PORTS"
"OUTPUT_PORTS" -ignorecase -value 0
```

In the above example, a value 0 is defined (as test\_mode) on all internal design objects matching with `"*powergood*"` (case-insensitive).

## Rules

The test\_mode constraint is used by the following rules:

---

#### SpyGlass DFT Solution

All rules

---

#### SpyGlass DFT DSM Solution

All rules

---

## For SpyGlass STARC Product, SpyGlass STARC02 product, SpyGlass STARC05 product, and SpyGlass STARCad-21 product

### Purpose

In these products, the `test_mode` constraint causes certain rules of the products to work in test mode where all scan flip-flops are controlled by test clocks and all asynchronous set and reset pins are inactive

### Syntax

The syntax of the `test_mode` constraint is as follows:

```
current_design <du-name>
  test_mode
    -name <signame>
    -value <value>
```

### Arguments

#### <du-name>

Name of the design unit under which you are specifying the test mode pin.

#### -name <signame>

Hierarchical name of the test mode pin.

#### -value <value>

A logic pattern which, when applied on the test mode pin, will put the circuit in the test mode.

### Rules

The `test_mode` constraint is used by the following rules:

<b>SpyGlass STARC Product</b>		
STARC-3.3.2.2a	STARC-3.3.2.2.b	STARC-3.3.2.3
<b>SpyGlass STARC02 Product</b>		
STARC02-3.3.1.1	STARC02-3.3.1.4a	STARC02-3.3.2.3



<b>SpyGlass STARC05 Product</b>	
STARC05-3.3.1.4a	STARC05-3.3.2.3
<b>SpyGlass STARCad-21 Product</b>	
starcad_21_Prereq	

## test\_point

### Purpose

The `test_point` constraint specifies where a test point should be added in a design without the necessity of changing the source RTL.

The `test_point` constraint is used as seeding information for testability analysis. This seeding then influences coverage calculations just as if it was directly controllable, observable or both — depending on the value of the `-type` argument. The benefit is that when the TA\_01 or TA\_02 rules make recommendations, the source RTL description does not have to be modified.

When `test_mode`, `force_ta`, or `test_point` constraints are specified on the same node, following is the priority among different constraints:

- `Test_mode`
- User-specified specific *force\_ta* / *test\_point*
- Effect of *dft\_treat\_primary\_inputs\_as\_x\_source* and *dft\_treat\_primary\_outputs\_as\_unobservable* parameters

For example, if `test_mode 1` and `test_point control` are applied on the same node then the `test_mode` constraint will be considered.

Also, if the `test_mode`, `force_ta`, or `test_point` constraints are found in the fanout of each other, following is the priority among different constraints:

- The constraint in the fanout gets the priority
- Fanin effect is blocked by the specified / resolved constraint on the node

For example, consider that `test_mode 1` is applied on the input of `buffer` and `test_point control` is applied on the output of the same `buffer`.

In this case, input will have simulation value 1 and nyn controllability but output will have yyn controllability and no simulation value.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was testpoint.*

## Product

SpyGlass DFT solution, SpyGlass DFT DSM solution

## Syntax

The syntax of the `test_point` constraint is as follows:

```
test_point
  -name <tp-name>
  -type <control | observe | full>
```

**NOTE:** *The test\_point constraint supports wildcard characters.*

## Arguments

### **-name <tp-name>**

Complete hierarchical name of a test point net, port, or pin.

The pin can be a primary pin as well as an internal pin.

You can specify a single port/pin/net's full hierarchical name or a space-separated list of full hierarchical port/pin/net's names.

For primary ports, you can also specify the simple port name as in the following example:

```
current_design top
  test_point -name in15 ...
```

### **-type <control | observe | full>**

Type of test point.

A value of `control` specifies that the test point is directly controllable, `observe` specifies that the test point is directly observable, and `full` specifies that the test point is directly controllable and observable.

## Example

Consider the following example:

```
test_point -name "AND::b" -type control
```

The above command will apply `test_point` to all instantiations of the AND gate, and is equivalent to the following commands:

```
testpoint -name "test.inst2.b" -type control
testpoint -name "test.inst1.b" -type control
```

## Rules

The `test_point` constraint is used by the following rules:

---

### SpyGlass DFT Solution

Info_testclock	Info_uncontrollable	Info_undetectedCause	Info_coverage
Coverage_audit	TA_06		

---

### SpyGlass DFT DSM Solution

All rules
-----------

---

## tie\_x

### Purpose

The `tie_x` constraint specifies the X-generator design unit (black box) names. Then, all instances of these design units in the design are assumed as X-generator instances. The `tie_x` constraint support wildcards.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was TIE\_X.*

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `tie_x` constraint is as follows:

```
tie_x
  -name <bb-name>
  -xpin <xpin-name-list>
```

**NOTE:** The `tie_x` constraint supports wildcard characters.

## Arguments

The `tie_x` constraint has the following arguments:

### **-name <bb-name>**

The X-generator design unit (black box) name.

The design unit must be a black box. That is, its definition must not exist in the design or in the specified libraries, if any.

The design unit name `<du-name>` can be specified as module name (for Verilog designs) or as entity name (for VHDL designs). For VHDL designs, all architectures of the specified entity are treated as X-generator design units.

You can specify a single design unit name or a space-separated list of design unit names.

### **-xpin <xpin-name-list>**

A space-separated name list of X-generator design unit (black box) output pins that are X-generator pins.

## Notes

More than one `tie_x` constraint may be necessary. If more than one `tie_x` constraint is used, they all will be processed in parallel.

## Rules

The `tie_x` constraint is used by the following rule:

---

**SpyGlass DFT Solution**

---

**BIST\_05**

---

## tristate\_cell

### Purpose

The `tristate_cell` constraint is used to define the tristate cells.

### Product

SpyGlass DFT solution

### Syntax

The syntax of the `tristate_cell` constraint is as follows:

```
tristate_cell
  -name <mod-name>
  -zpin <inout-port>
  -enpin <input-port>
  [-envalue <tri-buf-val>]
  -dpin <input-data-pin>
  -oppin <output-port>
```

**NOTE:** *The `tristate_cell` constraint supports wildcard characters.*

### Arguments

The `tristate_cell` constraint has the following arguments:

**-name <mod-name>**

Tristate buffer module name.

Names may refer to the modules where pins are generating X.

**-zpin <inout-port>**

Inout port driving the tristate buffer

**-enpin <input-port>**

Input port driving the tristate buffer

**-envalue <tri-buf-val>**

Value to make tristate cell act as a buffer. This value can be set as 1 or 0.

**-dpin <input-data-pin>**

Input data pin

**-oppin <output-port>**

Output port

## Rules

The `tristate_cell` constraint is used by the following rule:

---

### SpyGlass DFT Solution

---

Tristate\_16

---

## ungroup\_cells

### Purpose

Use this constraint to ungroup instances in the current module.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `ungroup_cells` constraint is as follows:

```
ungroup_cells
  [-inst <inst-list>]
  [ -all ]
  [ -flatten ]
  [ -small <num> ]
  [ -level <depth> ]
  [ -mod <mod-list> ]
```

```
[ -mod_all <mod-all-list> ]
```

## Arguments

### **-inst <inst-list>**

Specifies a list of instances (in the current module) that needs to be ungrouped.

You can specify the current module by using the `current_design` constraint.

For Verilog, the value provided by this argument is case-sensitive. For VHDL, this value is not case-sensitive.

### **-all**

Specifies that all instances in the current module should be ungrouped.

### **-flatten**

Specifies that the specified instances should be ungrouped recursively until leaf-level cells. By default, the behavior is single level flattening.

**NOTE:** *The -all argument is automatically considered if you specify the -flatten argument.*

### **-small <num>**

Specifies a number of leaf cells so that all instances in the current module that have less than the specified number of leaf cells should be ungrouped.

If you specify the `-small <num>` argument, all instances under the current design are considered. That is, the `-inst` argument cannot be specified in this case.

**NOTE:** *Please note the following points:*

- ☞ *The -all argument is automatically considered if you specify the -small argument.*
- ☞ *If you specify the -small and -all arguments together, SpyGlass reports a fatal violation.*

**-level <depth>**

Specifies a depth so that all instances at a depth greater or equal to the specified depth should be dissolved.

**NOTE:** Please note the following points:

- The `-all` argument is automatically considered if you specify the `-depth` argument.
- If you specify the `-level` and `-all` arguments together, SpyGlass reports a fatal violation.

**-mode <mod-list>**

Specifies a list of modules so that all instances of these modules within the current module are dissolved.

You can specify a current module by using the `current_design` command.

By default, only immediate instances of the specified modules are dissolved. To dissolve instances hierarchically until leaf level, specify the `-flatten` argument of this constraint along with the `-mod` argument.

**-mod\_all <mod-all-list>**

Specifies a list of modules so that all instances of these modules across the whole design are dissolved.

By default, only immediate instances of the specified modules are dissolved. To dissolve instances hierarchically until leaf-level, specify the `-flatten` argument of this constraint along with the `-mod_all` argument.

**NOTE:** If you specify the `-mod_all` argument along with the `-inst`, `-mod`, or `-all` argument of this constraint, SpyGlass reports a fatal violation.

**Examples**

The following examples show the usage of the `ungroup_cells` constraint:

- The following command is used to ungroup the `mid1` and `mid2` instances in the `top` module.

```
current_design top
```



```
ungroup_cells -inst mid1 mid2
```

- The following command is used to ungroup the `mid1` and `mid2` instances in the `top` module recursively till leaf-level:

```
current_design top
ungroup_cells -inst mid1 mid2 -flatten
```

- The following command is used to ungroup all instances of the `M1` and `M2` modules in the `top` module:

```
current_design top
ungroup_cells -mod M1, M2
```

- The following command is used to ungroup all instances in the `mid` module recursively till leaf-level:

```
current_design mid
ungroup_cells -all -flatten
```

- The following command is used to ungroup all instances in all hierarchies with respect to `top` that have less than `N` leaf cells:

```
current_design top
ungroup_cells -small N
```

## Rules

The `ungroup_cells` constraint is used by the following rules:

---

### **SpyGlass Power Estimation and SpyGlass Power Reduction Solutions**

---

All rules running in EST mode

---

## use\_library\_group

## Purpose

Helps in specifying the instance and library group mapping, that is, specifying the library group from which the cell definition of the instance needs to be picked.

## Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

## Syntax

The syntax to specify the `use_library_group` constraint is as follows:

```
current_design <du-name>
  use_library_group
    -name <lib-grp-name>
    -instname <inst-name-list>
```

## Arguments

### <du-name>

Module name (for Verilog designs) or design unit name in <entity-name>.<arch-name> format (for VHDL designs).

### -name <lib-grp-name>

Name of the library group(s) to which the cell definition of the specified instance belongs.

**NOTE:** *You can also use the -name argument as -names to specify a single library or a group of libraries.*

### -instname <inst-name-list>

Hierarchical path and name of an instance or a top-level design unit name to which library groups should be bound.

Consider the following example, suppose the top design unit name, `top`, has two instances, `g1` and `g2`, with the same library cell definition but the cell definitions are in different library groups (say, `G1` and `G2`) at different PVT values.

Now, if you want to pick the cell definition for instance `g1` from library group `G1` and the cell definition for instance `g2` from library group `G2`, use the `use_library_group` constraint as follows:

```
use_library_group -name G1 -instname top.g1
use_library_group -name G2 -instname top.g2
```

**NOTE:** Refer to the SGDC command [define\\_library\\_group](#) to group all libraries that are characterized at same PVT together.

## Rules

The `use_library_group` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## voltage\_domain

### Product

SpyGlass DFT DSM solution, SpyGlass Power Verify solution, SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### For SpyGlass DFT DSM Solution

#### Purpose

The `voltage_domain` constraint is used to specify the voltage/power domains in the design and its information is used by `SP_01` rule of the SpyGlass DFT DSM solution.

**NOTE:** Prior to SpyGlass 4.3.0 release, the name of this constraint was `voltagedomain`.

## Syntax

The syntax to specify the `voltage_domain` constraint is as follows:

```
current_design <du-name>
  voltage_domain
    [ -modname <du-name> ]
    [ -instname <inst-name-list> ]
```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the voltage/power domains. This is the environment for the voltage/power domains.

### -modname <du-name>

(Optional) Name of top-level design unit, specified as module name or the entity name.

Use the `-modname` argument, as in the following examples:

```
current_design top
  voltage_domain -modname top
```

The above example applies to Verilog module named `top`.

```
current_design top.rtl
  voltage_domain -modname top
```

The above example applies to VHDL entity named `top` with an architecture named `rtl`. If only the entity name is specified with `current_design` specification, the last compiled architecture is used in case of entities with multiple architectures.

### -instname <inst-name-list>

(Optional) Space-separated list of instance names.

The instance name must be a hierarchical instance name with respect to the top-level design unit specified as the environment.

You can use wildcard characters while specifying instance names.

**NOTE:** *The voltage domain information is applicable hierarchically.*

## Rules

The `voltage_domain` constraint is used by the following rule:

---

### SpyGlass DFT DSM Solution

---

SP\_01

---

## For SpyGlass Power Verify solution

### Purpose

The `voltage_domain` constraint is used to specify the voltage/power domains in the design and its information is used by voltage and power domain rules.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `voltagedomain`.*

### Syntax

The syntax to specify the `voltage_domain` constraint is as follows:

```
current_design <du-name>
  voltage_domain
    -name <name>
    -value <fvalue-list>
  [ -instname <inst-name-list> ]
  [ -portname <port-name-list> ]
  | -modname <du-name>
  [ -portname <port-name-list>]
  | -portname <port-name-list> -external
  [ -netname <net-name-list>]
  | -netname <net-name-list> -external
  | -external
  [ -isosig <iso-sig-list> -isoval <iso-value-list>
  | -inisosig <iniso-sig-list>
```

```

    -inisoval <iniso-value-list>
  | -noisosig ]
[ -generate_iso_logic ]
[ -outputs <ss-condition-name> ]
[ -supplyname <port-name-list> ]
[ -clkdomain <clk-name-list> ]
[ -enableports <en-port-name-with-limit-list> ]
[ -inputs <pd-condition-name> ]
[ -isolatedinputs <pin-name-list> ]
[ -isolatedoutputs <pin-name-list> ]
[ -isolatedoutputs <pin-name-list> ]
[ -biaspowernet <bias-pwr-net> ]
[ -biasgroundnet <bias-gnd-net> ]
  [ -sleepnet <net-name> ]
  [ -sleepval 0 | 1 ]
  [ -stopclock <net-name> ]
  [ -stopclockval 0 | 1 ]
  [ -savenet <net-name> ]
  [ -saveval 0 | 1 ]
  [ -restorenet <net-name> ]
  [ -restoreval 0 | 1 ]

```

## Arguments

### <du-name>

Name of the design unit under which you are specifying the voltage/power domains. This is the environment for the voltage/power domains.

### -name <name>

Name of the voltage/power domain.

### -value <fvalue-list>

Voltage values (floating-point values) of the voltage/power domain.

For voltage domains, you should specify only one positive non-zero value.

For power domains, you should specify two values (space-separated) that indicate the ON and OFF voltage values of the power domain. The first

value should be a positive non-zero value and the second value should be zero.

**-instname <inst-name-list>**

(Optional) Space-separated list of instances belonging to the voltage/power domain.

The instance name must be a hierarchical instance name with respect to the top-level design unit specified as the environment.

You can use wildcard characters while specifying instance names.

**NOTE:** *The voltage domain information is applicable hierarchically.*

**NOTE:** *If you do not use the -instname argument, you must specify the -modname argument or the -portname argument with the -external argument, or the -external argument.*

**-modname <du-name>**

(Optional) Name of top-level design unit (specified as module name or the entity name) belonging to the voltage/power domain. Then, all instances of the design unit are assumed to be in the voltage/power domain being specified.

Use the -modname argument as in the following examples:

```
current_design top
  voltage_domain -name vd1 -value 1.2 -modname top
```

The above example applies to Verilog module named top.

```
current_design top.rtl
  voltage_domain -name vd1 -value 1.2 -modname top
```

The above example applies to VHDL entity named top with an architecture named rtl. If only the entity name is specified with current\_design specification, the last compiled architecture is used in case of entities with multiple architectures.

**NOTE:** *You must specify the voltage domain information for top-level design unit using -modname argument.*

**NOTE:** *The voltage domain information is applicable hierarchically.*

**NOTE:** *If you do not use the -modname argument, you must specify the -instname*

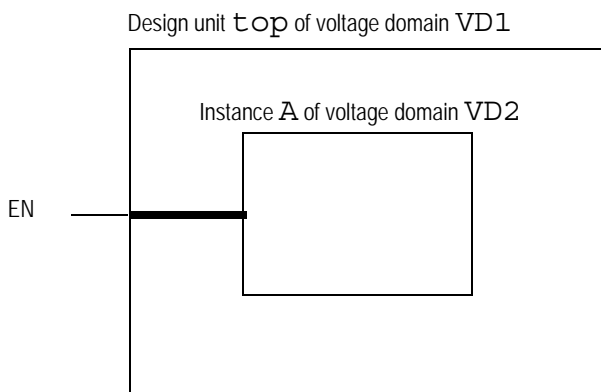
*argument, the -portname argument with the -external argument, or the -external argument.*

**-portname <port-name-list>**

(Optional) Space-separated list of ports of other design units/instances that belong to a different voltage/power domain but are directly connected to an instance of the specified voltage/power domain.

You can use wildcard characters while specifying port names where ports can be top-level or of a user-defined module.

Consider the following example:



**FIGURE 68.** Port of a Design Unit Belonging to a Different Voltage Domain

Here, the port EN of top-level design unit top of voltage domain VD1 is directly connected to the instance top.A of voltage domain VD2. If you do not specify the port EN to be in voltage domain VD2, the connection becomes a voltage domain crossing from VD1 to VD2 and you would need to insert a level shifter in the path.

The above situation is specified as follows:

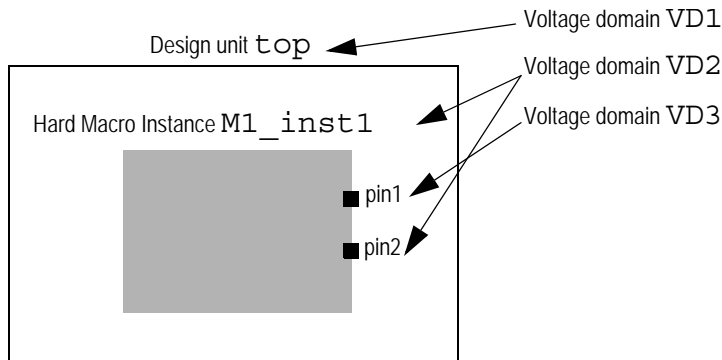
```
voltage_domain -name VD1 -value 1.2 -modname top
voltage_domain -name VD2 -value 1.5
  -instname top.A -portname EN
```

You can also specify the individual pins of hard macro instances (black box instances) to be in a different voltage/power domain than the voltage/



power domain of the hard macro instance.

Consider the following example where pin `pin1` is of a different voltage/power domain than its parent hard macro instance (black box instance) `M1_inst1`:



**FIGURE 69.** Children Pin of a Different Voltage Domain From its Parent Instance

The above situation can be specified as follows:

```
voltage_domain -name VD1 -value 1.2 -modname top
voltage_domain -name VD2 -value 1.5
  -instname top.M1_inst1
voltage_domain -name VD3 -value 1.7
  -portname top.M1_inst1.pin1
```

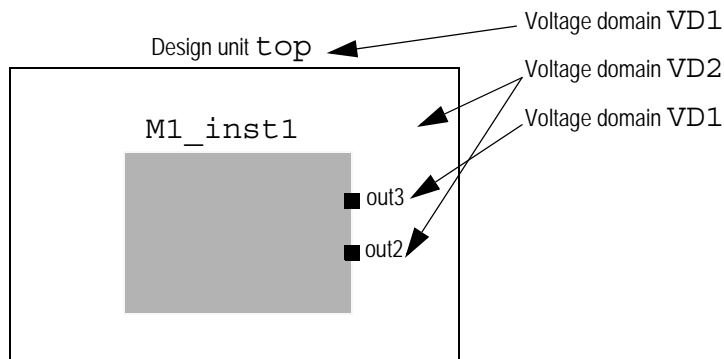
**NOTE:** You need to specify the full hierarchical name of the pin having a different voltage/power domain.

**NOTE:** Only the ports of top-level design units and pins of leaf level instances can be specified with the `-portname` argument.

**-netname** <net-name-list>

(Optional) Space-separated list of nets of other design units that belong to a different voltage/power domain but are directly driven by an instance of the specified voltage/power domain.

Consider the following example where all the output nets of the design unit `top.M1_inst1` are in domain `VD2` working at 1.5 volts except net `out3` which is being driven by 1.2 volts.



**FIGURE 70.** All Output Nets Belong to Same Voltage Domain

The above situation can be specified as follows:

```
voltage_domain -name VD1 -value 1.2 -modname top      -
netname top.M1_inst1.out3
voltage_domain -name VD2 -value 1.5
  -instname top.M1_inst1
```

You can use wildcard characters while specifying net names.

The constraint also supports specification of vector nets. For example, a 4 bit net `top.M1_inst1.out3` can be specified as either:

```
top.M1_inst1.out3, top.M1_inst1.out3[0:3]
```

or

```
top.M1_inst1.out3[0] top.M1_inst1.out3[1]
top.M1_inst1.out3[2] top.M1_inst1.out3[3]
```

You need to specify the full hierarchical name of the net having a different voltage/power domain. The field is used by LPSVM04 rule of the *SpyGlass Power Verify* solution for checking correct level shifters.

### **-external**

(Optional) Specifies that the voltage domain specified for a port is external to the design.

You can use the `-external` argument with the `-portname` argument

(to indicate that the voltage/power domain is applicable to the port and is external to the design) or as a stand-alone argument (to indicate that the voltage/power domain is external to the design).

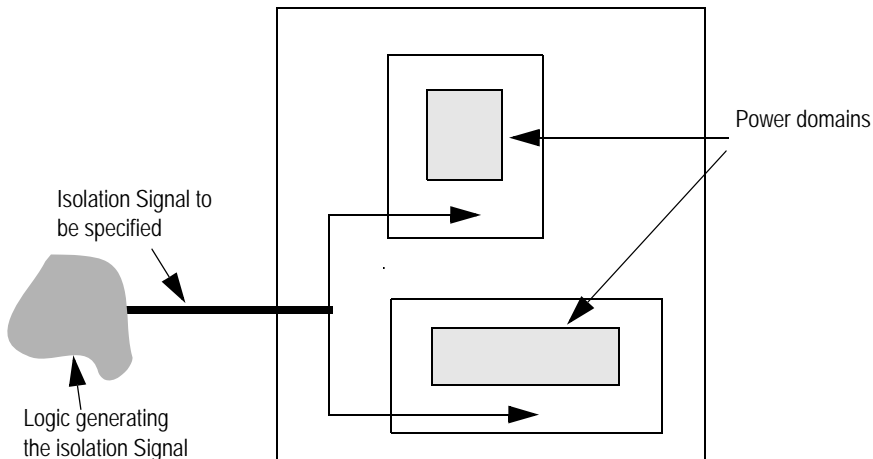
**NOTE:** You can specify the external voltage/power domains (using the stand-alone -external argument of the voltage\_domain constraint). Then, the specified domain name can be used by the pin\_voltage constraint.

**-isosig** <iso-sig-list>

(Optional) Name of the isolation signal(s) for the power domain being defined as a space-separated list.

**NOTE:** You must specify an isolation signal for a power domain for the related checks to be performed. If you are not specifying -isosig argument, it is mandatory to specify either -inisosig with the input isolation signal(s) for the power domain or -noisosig indicating that the power domain does not have an isolation signal.

If you are specifying an internal signal, please ensure that you specify the signal name at the point of creation, as in the following example:



**FIGURE 71.** Signal Name At The Point Of Creation

**-isoval <iso-value-list>**

(Optional) Assert value(s) of the isolation signal(s) being specified with the `-isosig` argument as a space-separated list.

The assert value is the value that the isolation signal attains under the power-down conditions. In normal mode, the isolation signal gets the inverse of the assert value.

**NOTE:** *The number of isolation values specified with the `-isoval` argument must match the number of isolation signals specified with the `-isosig` argument. Otherwise, SpyGlass reports a FATAL message.*

**-inisosig <iniso-sig-list>**

(Optional) Name of the input isolation signal(s) for the power domain being defined as a space-separated list.

**NOTE:** *The object(s) specified with the `-inisosig` argument should be a top-level port, net, or instance terminal in the design unit specified as current design of the `voltage_domain constraint`*

**-inisoval <iniso-value-list>**

(Optional) Assert value(s) of the input isolation signal(s) being specified with the `-inisosig` argument as a space-separated list.

The valid values for `-inisoval` argument are only 0 or 1.

**NOTE:** *The number of isolation values specified with the `-inisoval` argument must match the number of input isolation signals specified with the `-inisosig` argument. Otherwise, SpyGlass reports a FATAL message.*

**-noisosig**

(Optional) Indicates that the power domain does not have an isolation signal.

When you specify the `-noisosig` argument, the power domain checking rules do not perform isolation signal-related checking.

**NOTE:** *It is mandatory to specify either of `-isosig`, `-inisosig`, or `-noisosig` arguments.*

**-supplyname <port-name-list>**

(Optional) Specifies a space-separated simple name list of power/ground rails (ports) in the design.

These ports are associated with the supply constraint.

**-generate\_iso\_logic -outputs <ss-condition-name>**

(Optional) The `-generate_iso_logic` argument enables generation of missing isolation logic by the LPSVM23 rule of the *SpyGlass Power Verify* solution and the `-output` argument specifies the steady state conditions (any valid string) as specified using the `domain_outputs` constraint.

In addition, the LPSVM09 rule of the *SpyGlass Power Verify* solution uses the `-output` argument along with the `domain_outputs` constraint.

Consider the following example that defines the power domain V3 that does not have isolation logic:

```
voltage_domain
  -name V3 -value 1.2 0
  -instname top.u6
  -isosig isosig6
  -isoval 1
  -generate_iso_logic
  -outputs PD_V3_OUT
```

Here, the `-generate_iso_logic` argument enables generation of missing isolation logic and the `-outputs` argument defines the steady state condition as `PD_V3_OUT`. For defining `PD_V3_OUT`, use `domain_outputs` constraint as follows:

```
domain_outputs -name PD_V3_OUT -value top.d_o 0 top.ack_o 0
top.err_o 0 -default 1
```

**-clkdomain <clk-name-list>**

(Optional) Specifies the valid clock domains of the voltage domain for clocks described using the `clock` constraint.

**-enableports <en-port-name-with-limit-list>**

(Optional) Specifies a space-separated list of enable ports of a power domain along with the individual power switch enable connection limit as checked by the LPSVM45 rule of the *SpyGlass Power Verify* solution.

Consider the following constraint specification specifying power domain PD1:

```
voltage_domain -name PD1 -value 1.2 0
  -instname "TOP.lower1" -isosig top.iso_sig
  -isoval 1 -enableports 'EN1 1 EN2 100 EN3[2] 100'
```

The `-enableports` argument of the `voltage_domain` constraint specifies the ports of power domain through which enable signals of power switches must be connected. The enable port of the power domain and the maximum number of power switch enables allowed to be connected to it are specified as a space-separated pair. For example, `EN1 1` indicates that only one power switch enable can be connected to port `EN1`.

Specify `"-1"` as the limit to indicate that any number of power switch enables can be connected. For example, `EN12 "-1"`.

**-inputs <pd-condition-name>**

(Optional) Specifies the power domain input expected value condition name that is specified with the `domain_inputs` constraint.

**-isolatedinputs <pin-name-list>**

(Optional) Specifies the power domain input name list (hierarchical pin name list) when these power domain inputs are to be assumed as already isolated and, therefore, no isolation-related checking should be performed on these inputs.

You can also use wildcard characters and bus notation while specifying the pin names for this argument.

**-isolatedoutputs <pin-name-list>**

(Optional) Specifies the power domain output name list (hierarchical pin name list) when these power domain outputs are to be assumed as already isolated and, therefore, no isolation-related checking should be performed on these outputs.

You can also use wildcard characters and bus notation while specifying the pin names for this argument.

**-biaspowernet <bias-pwr-net>**

(Optional) Specifies the name of bias power net associated with the domain.

**-biasgroundnet <bias-gnd-net>**

(Optional) Specifies the name of bias ground net associated with the domain.

**-sleepnet <net-name>**

(Optional) Specifies a sleep net of a power domain as used by the LPSVM56, LPSVM57, LPSVM58, and LPSVM59 rules of the *SpyGlass Power Verify* solution.

**NOTE:** *The -sleepnet argument should be specified along with the -sleepval argument. For example,*

```
voltage_domain -instname P1 -sleepnet SL1 -sleepval 1
```

**-sleepval 0 | 1**

(Optional) Specifies the active value of a sleep net of a power domain (0 or 1) as used by the LPSVM58 and LPSVM59 rules of the *SpyGlass Power Verify* solution.

**NOTE:** *The -sleepval argument should be specified along with the -sleepnet argument.*

**-stopclock <net-name>**

(Optional) Specifies the stop clock net of a power domain.

**-stopclockval 0 | 1**

(Optional) Specifies the active value of a stop clock net of a power domain (0 or 1).

**-savenet <net-name>**

(Optional) Specifies the save net of a power domain as used by the

LPSVM57 and LPSVM59 rules of the *SpyGlass Power Verify* solution.

**NOTE:** *The -savenet argument should be specified along with the -saveval argument. For example:*

```
vol tage_domain -instance P1 -savenet SA1 -saveval 1
```

**-saveval 0 | 1**

(Optional) Specifies the active value of the save net of the power domain (0 or 1) as used by the LPSVM59 rule of the *SpyGlass Power Verify* solution.

**NOTE:** *The -saveval argument should be specified along with the -savenet argument.*

**-restorenet <net-name>**

(Optional) Specifies the restore net of a power domain as used by the LPSVM57 and LPSVM59 rules of the *SpyGlass Power Verify* solution.

**NOTE:** *The -restorenet argument should be specified along with the -restoreval argument. For example:*

```
vol tage_domain -instance P1 -restorenet RS1 -restoreval  
1
```

**-restoreval 0 | 1**

(Optional) Specifies the active value of the restore net of a power domain (0 or 1) as used by the LPSVM59 rule of the *SpyGlass Power Verify* solution.

**NOTE:** *The -restoreval argument should be specified along with the -restorenet argument.*

## Examples

Consider the following example:

```
module top(in1, in2, iso_sig, out1, out2);  
  input in1, in2, iso_sig;  
  output out1, out2;  
  
  wire C1, D1;
```



```

    lower lower1(in1, C1);
    lower \lower2 (in2, D1);
    assign out1 = C1 & iso_sig;
    assign out2 = D1 & iso_sig;
endmodule

```

```

module lower(in, out);
    input in;
    output out;

    assign out = in;
endmodule

```

All constraint specifications are with respect to the top-level module `top`. Therefore, the environment is defined as follows:

```
current_design top
```

Now, the top-level module `top` is in the voltage domain named say `VD1`. This situation is specified as follows:

```
current_design top
    voltage_domain -name VD1 -value 1.2 -modname top

```

Next, assume that the instance `lower1` is in a different voltage domain. This situation is specified as follows:

```
current_design top
...
    voltage_domain -name VD2 -value 1.5
                    -instname top.lower1

```

Since port `in1` of module `top` (which is in voltage domain `VD1`) is directly connected to instance `lower1` (which is in voltage domain `VD2`), you can also specify the port using the `-portname` argument to avoid crossing message for a voltage domain, as shown in the following example:

```
current_design top
...
    voltage_domain -name VD2 -value 1.5
                    -instname top.lower1
                    -portname in1

```

Next, assume that the instance `\lower2` is in the power domain PD1 having signal `iso_sig` as the isolation signal with assert value 1. This situation is described as follows:

```
current_design top
...
  voltage_domain -name PD1 -value 1.2 0
    -instname "top.\lower2 "
    -isosig top.iso_sig -isoval 1
```

**NOTE:** When you use flattened netlists as inputs instead of RTL files, you can specify all flattened instance names in a scope by using the asterisk (\*) wildcard character instead of listing them individually by using the `-instname` argument. For example, when the scope `top.\U1` contains three flattened instances, `top.\U1/U11`, `top.\U1/U12`, and `top.\U1/U13`, specify them as `"top.\U1*"`.

## For SpyGlass Power Estimation and SpyGlass Power Reduction Solutions

### Purpose

The `voltage_domain` constraint is used to specify the design unit present in different `voltage_domain` as used by the rules in the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

### Syntax

The syntax to specify the `voltage_domain` constraint is as follows:

```
current_design <du-name>
  voltage_domain
    -name <name>
    -value <value>
    -instname <inst-name-list>
      | -modname <du-name>
      | -portname <port-name-list>
      | -netname <net-name-list>
```

## Arguments

### <du-name>

Module name (for Verilog designs) or design unit name in <entity-name>. <arch-name> format (for VHDL designs).

### -name <name>

Name of the voltage domain.

### -value <value>

Voltage value (floating-point values) of the specified voltage domain.

**NOTE:** *The first voltage value in the `voltage_domain` constraint should be non-zero and the second should be zero (if defined). Multiple non-zero values are not allowed.*

### -instname <inst-name-list>

(Optional) Space-separated list of instances belonging to the specified voltage domain.

The instance name must be a hierarchical instance name with respect to the top-level design unit specified as the environment.

**NOTE:** *The voltage domain information is applicable hierarchically.*

**NOTE:** *You can also specify the instance names using regular expressions.*

### -modname <du-name-list>

(Optional) Space-separated list of top-level design units (specified as module name or the entity name) belonging to the specified voltage domain. Then, all instances of these design units are assumed in the specified voltage domain.

Use the -modname argument, as in the following examples:

```
current_design top
  voltage_domain -name vd1 -value 1.2 -modname top
```

The above example applies to Verilog module named top.

```
current_design top.rtl
  voltage_domain -name vd1 -value 1.2 -modname top
```

The above example applies to VHDL entity named `top` with an architecture named `rtl`. If only the entity name is specified with `current_design` specification, the last compiled architecture is used in case of entities with multiple architectures.

**NOTE:** *The voltage domain information is applicable hierarchically.*

**-portname <port-name-list>**

(Optional) Space-separated name list of primary ports belonging to the specified voltage domain.

**NOTE:** *You can also specify the port names using regular expressions.*

**-netname <net-name-list>**

(Optional) Space-separated name list of nets belonging to the specified voltage domain.

**NOTE:** *You can also specify the net names using regular expressions.*

## Notes

You must specify at least one of the `-instname`, `-modname`, `-netname`, or `-portname` arguments. This requirement is checked by the `SGDC_power_est18` rule.

## Examples

Consider the following example:

```
module top(in1, in2, iso_sig, out1, out2);
  input in1, in2, iso_sig;
  output out1, out2;

  wire C1, D1;

  lower lower1(in1, C1);
  lower \lower2 (in2, D1);
  assign out1 = C1 & iso_sig;
  assign out2 = D1 & iso_sig;
endmodule
```

## SpyGlass Design Constraints

```

module lower(in, out);
    input in;
    output out;

    assign out = in;
endmodule

```

All constraint specifications are with respect to the top-level module `top`. Therefore, the environment is defined as follows:

```
current_design top
```

Now, the top-level module `top` is in the voltage domain named say `VD1`. This situation is specified as follows:

```
current_design top
    voltage_domain -name VD1 -value 1.2 -modname top

```

Next, assume that the instance `lower1` is in a different voltage domain. This situation is specified as follows:

```
current_design top
...
    voltage_domain -name VD2 -value 1.5
                    -instname top.lower1

```

**NOTE:** When you use flattened netlists as input instead of RTL files, you can specify all flattened instance names in a scope by using the asterisk (\*) wildcard character instead of listing them individually by using the `-instname` argument. For example, when the scope `top.\U1` contains three flattened instances, `top.\U1/U11`, `top.\U1/U12`, and `top.\U1/U13`, specify them as `"top.\U1*"`.

## Rules

The `voltage_domain` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## vt\_mix\_percentage

### Purpose

The `vt_mix_percentage` constraint is used to specify the percentage number of cells of a threshold voltage group in a library or a complete library to be processed by the PEPWR01 and PEPWR02 rules.

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `vt_mix_percentage` constraint is as follows:

```
current_design <top-du-name>
vt_mix_percentage
  -group <grp-name> | -cellname <cell-name-pattern> |
  -libname <lib-name>
  [ -weight <value> ]
  [ -instname <inst-name> ]
  [ -clock <clock-name> ]
  [ -clock_period <clock-period> ]
```

### Arguments

#### <top-du-name>

Module name (for Verilog designs) or design unit name in `<entity-name>.<arch-name>` format (for VHDL designs).

#### -group <grp-name>

Name of the threshold voltage group (name specified with the `default_threshold_voltage_group/`  
`threshold_voltage_group` attribute in the library).

This argument supports lists.

**-cellname <cell-name-pattern>**

Specifies a pattern to match cell names of a certain group of cells.

This argument supports lists.

**-libname <lib-name>**

Name of the library.

This argument supports lists.

**-weight <value>**

The percentage value (a floating-point number between 0 and 100).

Do not specify this argument if the *PEVTDIST* rule is run.

**-instname <inst-name>**

Name of the instance for which the percentage value is applicable.

**-clock <clock-name>**

Name of the clock for which the percentage values are applicable.

**-clock\_period <clock\_period>**

Clock period (in nanoseconds) for which the percentage values are applicable.

**NOTE:** *Please note the following:*

- 📖 *To specify values for a top-level design unit, do not use the -instname argument. However, to specify values for a specific instance hierarchy, use the -instname argument.*
- 📖 *You must at least define the percentage value for the top-level design unit. Specifying values for specific instance hierarchies is optional.*
- 📖 *For any specific run, you can use only one of the following arguments: -group, -libname, or -cellname. You cannot use a combination of two or more of these arguments to specify the VT mix information in the same run.*
- 📖 *You must use the -group argument to specify values when the associated library has a default\_threshold\_voltage\_group/ threshold\_voltage\_group attribute set.*

- *The sum of values specified for the top-level design unit or an instance hierarchy in different `vt_mix_percentage` constraints must be equal to 100.*
- *Do not specify any two arguments together out of the `-clock`, `-clock_period`, and `-instname` arguments.*
- *You can specify only one argument when using `-clock`, `-clock_period`, and `-instname` arguments. You cannot use these three arguments together.*

The following example specifies percentage values for the top-level design unit `top` (first two `vt_mix_percentage` constraints) and for instance hierarchy `top.inst1` (Next two `vt_mix_percentage` constraints):

```
current_design topther
  vt_mix_percentage -group tvgl -weight 60
  vt_mix_percentage -group tvg2 -weight 40
  vt_mix_percentage -group tvgl
    -instname top.inst1 -weight 30

  vt_mix_percentage -group tvg2
    -instname top.inst2 -weight 70
```

## Examples

- Consider a scenario in which you have a set of libraries with cells of two threshold voltage groups, `DEFAULT1` and `DEFAULT2`. If you want to tell SpyGlass that 75% of the design should use cells from the `DEFAULT1` group and 25% from the `DEFAULT2` group, specify the following commands:

```
vt_mix_percentage -group DEFAULT1 -weight 75
vt_mix_percentage -group DEFAULT2 -weight 25
```

- Consider a scenario in which you have two libraries, `typical1NOTVG` and `typical2NOTVG`. Both the libraries have cells of a specific threshold voltage, but the `threshold_voltage_group` attribute is not specified in the library files. In this case, you can directly specify the library names, as shown below:

```
vt_mix_percentage -libname typical2NOTVG -weight 25
vt_mix_percentage -libname typical1NOTVG -weight 75
```



- You can specify the percentage usage of cells from different groups for specific hierarchical instances as well. This is shown in the following example:

```
select_wireload_model -wireload tsmc13_wl20
vt_mix_percentage -libname typical2NOTVG -weight 60
vt_mix_percentage -libname typical1NOTVG -weight 40
```

```
vt_mix_percentage -libname typical1 -instname
top.inst1 -weight 75
vt_mix_percentage -libname typical2 -instname
top.inst1 -weight 25
```

```
vt_mix_percentage -libname typical1 -instname
top.inst2 -weight 25
vt_mix_percentage -libname typical2 -instname
top.inst2 -weight 75
```

In the above example, for the top-level design, `top`, the ratio of the usage of cells from the `typical2NOTVG` and `typical1NOTVG` libraries is 60:40.

In addition, for the `top.inst1` module within the top-level design, the ratio of the usage of cells from the `typical1` and `typical2` libraries is 75:25. Similarly, for the `top.inst2` module, the ratio of the usage of cells from the `typical1` and `typical2` libraries is 25:75.

- You can also specify the percentage usage directly based on the cell names, as shown in the following example:

```
vt_mix_percentage -cellname "VT2*" -weight 75
vt_mix_percentage -cellname "VT1*" -weight 25
```

In the above example, the percentage usage of all the cell names starting with the string, `VT2`, will be 75. Similarly, the percentage usage of all the cell names starting with the string, `VT1`, will be 25.

- You can specify the percentage usage of cells from different groups for specific clocks. This is shown in the following example:

```
select_wireload_model -wireload tsmc13_wl20
vt_mix_percentage -libname typical2NOTVG -weight 60 -clock
top.clk1
```

```
vt_mix_percentage -libname typical1NOTVG -weight 40 -clock
top.clk1
```

```
vt_mix_percentage -libname typical2NOTVG -weight 30 -clock
top.clk2
```

```
vt_mix_percentage -libname typical1NOTVG -weight 70 -clock
top.clk2
```

- You can specify the percentage usage of cells from different groups for specific clock periods. This is shown in the following example:

```
select_wireload_model -wireload tsmc13_wl20
```

```
vt_mix_percentage -libname typical2NOTVG -weight 60
-clock_period 10
```

```
vt_mix_percentage -libname typical1NOTVG -weight 40
-clock_period 10
```

```
vt_mix_percentage -libname typical2NOTVG -weight 30
-clock_period 20
```

```
vt_mix_percentage -libname typical1NOTVG -weight 70
-clock_period 20
```

- The following constraints specifications show how to specify multiple library names:

```
vt_mix_percentage -libname typical1 typical2 -weight 60
```

## Rules

The `vt_mix_percentage` constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

## watchpoint

## Purpose

The `watchpoint` command can be used to specify watch points in a design.

In case of a functional rule-violation reported by SpyGlass Auto Verify solution or SpyGlass CDC solution, the simulation trace produced contains only the registers and primary inputs involved in the failure. The `watchpoint` constraint can be used to force SpyGlass Auto Verify solution and SpyGlass CDC solution to generate the waveform for an internal signal.

## Product

SpyGlass Auto Verify solution, SpyGlass CDC solution

## Syntax

The syntax of the `watchpoint` constraint is as follows:

```
watchpoint
  -name <pin-name-list>
```

## Arguments

**-name <pin-name-list>**

Space-separated list of hierarchical pin names.

## Examples

The following constraint defines signals `sig3` in the design unit `top` as a watch point:

```
watchpoint -name top.sig3
```

## Rules

The `watchpoint` constraint is used by the following rules:

<b>SpyGlass CDC Solution</b>			
Ac_cdc01a	Ac_cdc01b	Ac_cdc01c	Ac_cdc08
Ac_fifo01	Ac_handshake0 1	Ac_handshake0 2	Clock_sync03a
Ac_conv02			
<b>SpyGlass Auto Verify Solution</b>			
All rules			

## wireload\_selection

### Purpose

Specifies the default wire-load selection table to be used for power estimation by the rules in the SpyGlass Power Estimation and SpyGlass Power Reduction solutions. Normally, the `default_wire_load_selection` library attribute defines the default wire-load selection table to be used. You can overwrite this default value or specify the value if not available in the library using the `wireload_selection` constraint.

**NOTE:** *Prior to SpyGlass 4.3.0 release, the name of this constraint was `wireloadselection`.*

### Product

SpyGlass Power Estimation and SpyGlass Power Reduction solutions

### Syntax

The syntax to specify the `wireload_selection` constraint is as follows:

```
current_design <top-du-name>
  wireload_selection
    -name <wlst-name>
    [ -libname <lib-name> ]
```

## Arguments

### <top-du-name>

Module name (for Verilog designs) or design unit name in <entity-name>. <arch-name> format (for VHDL designs).

### -name <wlst-name>

Name of the wire-load selection table of the library <lib-name>.

### -libname <lib-name>

(Optional) Library name.

Specify a library name only when a wire-load selection table with the same name is available in more than one library.

**NOTE:** Please note the following points:

- ☐ If you do not use the -name argument, the default wire-load selection table name will be taken from the library specified with the -libname argument.
- ☐ If you do not use the -libname argument, the library that contains the specified name of the wire-load selection table will be taken.
- ☐ Though both the -name and -libname arguments are optional, you must specify at least one argument with the wireload\_selection constraint. If you do not specify any one argument, the above rules report a syntax error.

The actual wire-loads used by these rules are reported in the pe\_wireload Report in the SpyGlass Power Estimation and SpyGlass Power Reduction solutions.

## Rules

The wireload\_selection constraint is used by the following rules:

SpyGlass Power Estimation and SpyGlass Power Reduction Solutions			
PEPWR01	PEPWR02	PEPWR03	PEPWR05
PEPWR13	PEPWR14	poweraudit	

