SpyGlass® CDC
Rules Reference Guide

Version N-2017.12-SP2, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

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 http://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Third-Party Links

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

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

Report an Error

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

Contents

Preface

About This Book

The SpyGlass® CDC Rules Reference guide describes the SpyGlass rules that report useful clock and reset information.

Contents of This Book

The SpyGlass CDC Rules Reference guide has the following sections:

Chapter	Describes...
Introduction to SpyGlass CDC	Provides an introduction to the SpyGlass CDC solution.
Performing SpyGlass CDC Analysis	Provides an overview on performing SpyGlass CDC analysis.
Working With the Ac_sync_group Rules	Provides details on working with the Ac_sync_group rules.
Performing Functional Analysis in SpyGlass CDC	Provides details on performing functional analysis in SpyGlass CDC.
Handling generated_clock Constructs on Library Pins	Provides details on working with the generated_clock constructs.
Using the Clock Setup Window	Provides details on working with the Clock Setup window.
Clock Domain Crossing Synchronization Schemes	Provides details on various clock domain synchronization schemes in SpyGlass CDC.
Tcl Commands in SpyGlass CDC	Provides information on the SpyGlass CDC Tcl commands.
Reports and Other Files in SpyGlass CDC	Provides details on SpyGlass CDC reports.
Parameters in SpyGlass CDC	Provides details on SpyGlass CDC parameters.
Rules in SpyGlass CDC	Provides details on SpyGlass CDC rules.
Terminologies in SpyGlass CDC	Provides the meaning of various terminologies used in SpyGlass CDC.
Internal Rules in SpyGlass CDC	Provides information on the SpyGlass CDC rules that are automatically run on certain conditions.
Appendix: SGDC Constraints	Provides the list of SGDC constraints used in SpyGlass CDC.

Typographical Conventions

This document uses the following typographical conventions:

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

The following table describes the syntax used in this document:

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

Introduction to SpyGlass CDC

SpyGlass® CDC solution is used to detect issues related with Clock Domain Crossings (CDC) in a design. It ensures that proper synchronization is added in the circuit to avoid such issues.

For example, it detects the following problems associated with clock-domain crossings:

- Issues related to metastability
- Issues related to complex synchronizers
- Issues related to reset synchronization
- Issues related with the implementation of clocks, resets, and crossings
- Data hold in fast-to-slow crossings
- Data correlation and race conditions

SpyGlass CDC solution provides the above information in the form of rule messages, reports, and related files.

NOTE: *The SpyGlass CDC solution has an associated methodology document. It is recommended to read that document to learn the recommended approach to use SpyGlass CDC solution.*

Performing SpyGlass CDC Analysis

SpyGlass CDC analysis enables you to identify CDC issues in your design.

During SpyGlass CDC analysis, you run SpyGlass CDC goals in different stages. In each stage, fix the reported violations and move to the next stage. Using this step-wise approach enables you to reach a handful of CDC issues that need consideration.

If you do not follow this step-wise approach, you may see large number of CDC violations, majority of which appear due to incorrect setup or not fixing violations of the previous stages.

Before performing SpyGlass CDC analysis, check the *Prerequisites for Performing SpyGlass CDC Analysis*.

SpyGlass CDC analysis is divided in the following stages:

1. *Creating SpyGlass CDC Setup*
2. *Fixing Clock and Reset Integrity Problems*
3. *Performing CDC Verification*
4. *Debugging CDC Issues*

Prerequisites for Performing SpyGlass CDC Analysis

The prerequisites for performing SpyGlass CDC analysis are as follows:

- Run the design-read process successfully.
- Ensure that the design contains minimum number of unintended black boxes.
- Provide the corresponding technology libraries (.lib) for instantiated technology library cells in the design.
- Specify information about the clocks in the design, and also resets (if possible).

  It is recommended to gather this information from design specifications, IPs, or chip leads before starting SpyGlass CDC run.

  SpyGlass CDC provides assistance in automatically detecting clocks and resets if you do not have access to this information. For details, see *Generating Clocks and Resets for a Design*.

NOTE: *You can run the SpyGlass CDC solution with the* `basepolicy so` *license feature that is used for all base products. However, you cannot use this feature* for *CDC Verification Rules, CDC Verification Rules, and Delta Delay Rules. For these rules and schemes, you need the* `Advanced_CDC` *and* `adv_checker` *license features.*

Also note that if the Advanced_CDC license is not checked out in save mode and a CDC rule that requires the Advanced CDC license is run in the restore run, SpyGlass runs in force save mode under the use_advcdc_features 1 goal specific option.

The following example illustrates the usage of this goal-specific option:

```
set_goal_option use_advcdc_features 1
```

Creating SpyGlass CDC Setup

Creating a setup means specifying design information, such as clocks, resets, and stop modules before *Performing CDC Verification*.

The quality of setup dictates the quality of SpyGlass CDC analysis. A wrong or incomplete setup may result in many false violations or mask a real design bug.

You create a setup by:

- *Specifying Clock Generation Blocks as Black Boxes*
- *Specifying Clocks and Resets for a Design*
- *Generating Clocks and Resets for a Design*
- *Using the Setup Manager*

Specifying Clock Generation Blocks as Black Boxes

Performing SpyGlass CDC analysis on internal of clock-generation blocks considerably complicates and adds little to the value of overall SpyGlass CDC analysis.

Mark such blocks as black boxes unless you have detailed SGDC constraints to define the clock characteristics of these blocks.

To mark such blocks as black boxes, specify them to the following command in a project file:

```
set_option stop <blocks>
```

Once you mark such blocks as black boxes:

- Specify the *clock* constraints on the output pins of these blocks.
- Define clock outputs in the same domain unless the clocks are not harmonically related.

  This is explained in the following figure:

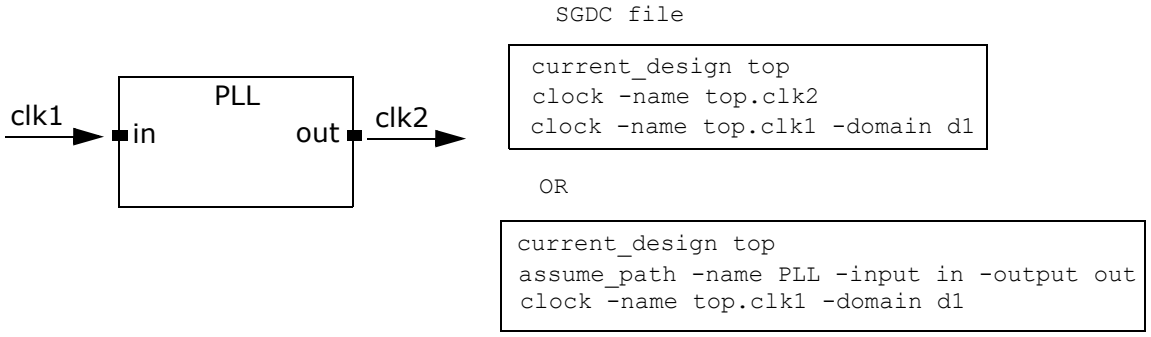


FIGURE 1. Defining Clock Outputs as in the Same Domain

Specifying Clocks and Resets for a Design

If you know clocks and resets in your design, specify them by performing the following steps:

1. Define clocks and resets by using the *clock* and *reset* constraints, respectively, in an SGDC file.
2. Analyze your design by running goals from the SpyGlass CDC solution methodology.
3. Examine *The Clock-Reset-Summary Report*.

   In this report, *Section D: Cases not checked for clock domain crossings Section* lists the unconstrained clocks.
4. Modify the SGDC file to specify the clock signals reported in *The Clock-Reset-Summary Report*.
5. Repeat step 2 with the SGDC file modified in the previous step.

Generating Clocks and Resets for a Design

If you do not know clocks and resets in the design, generate them by performing the following steps:

1. Run the cdc_setup goal.

This step creates the autoclocks.sgdc and autoresets.sgdc file containing SGDC constraints for inferred clocks and resets, respectively.

2. Review and modify the generated SGDC files.

   These files may include some control signals in addition to real clocks and resets. Therefore, you must review each inferred clock and reset in these files and remove signals that are not real candidates for clocks and resets.

   It is recommended that you view the *Setup_clock01* and *Reset_info01* rule messages to review such inferred signals.

3. Specify the modified autoclocks.sgdc and autoresets.sgdc files in the SpyGlass run, and analyze your design by running the required goals.

Understanding the Generated SGDC Files

Consider the following structure of the autoclocks.sgdc and autoresets.sgdc files generated after running the `structure_audit` goal:

```
//autoclocks.sgdc
current_design <du-name>
  {clock -name <clk-name> -domain domain <num>
      }

//autoresets.sgdc
current_design <du-name>
  {reset -name <async-rst-name> -value <value>}
  {reset -name -sync <sync-rst-name> -value <value>}
```

The arguments of the above commands are explained below:

Argument	Description
<du-name>	Specifies a module name (for Verilog designs) or a design unit name in the <entity-name>.<arch-name> format (for VHDL designs)
<clk-name>	Specifies a clock signal name

Argument	Description
<async-rst-name>	Specifies an asynchronous reset signal name. For each clock signal reported in the autoclocks.sgdc file, a unique domain name is assigned to that clock. For top-level nets, <clk-name> and <async-rst-name> are simple names. For nets at any other level, <clk-name> and <async-rst-name> are full hierarchical names. For each reset signal reported, the inferred active reset value is also reported.
<sync-rst-name>	Specifies a synchronous reset signal name. For each clock signal reported in the autoclocks.sgdc file, a unique domain name is assigned to that reset. For top-level nets, <clk-name> and <sync-rst-name> are simple names. For nets at any other level, <clk-name> and <sync-rst-name> are full hierarchical names. For each reset signal reported, the inferred active reset value is also reported.

Viewing the Generated clock Constraints

Each *clock* constraint specifies a new clock in a new clock domain (specified by the `-domain` argument). The clock is attached to the net specified by the `-name` argument. This net can be anywhere in the hierarchy.

The `-value` argument is provided for compatibility with SpyGlass DFT solution analysis and is not used by the SpyGlass CDC solution analysis.

NOTE: *The* `clock` *constraints information is printed in the Section B: Propagated Control Signals Section of The Clock-Reset-Summary Report.*

Viewing the Generated reset Constraints

Each *reset* constraint specifies a new reset. The reset is attached to the node specified by the name in the `name` argument. This name should be a net name that can be anywhere in the hierarchy.

The `-value` argument is provided for compatibility with SpyGlass DFT solution analysis and is not used by the SpyGlass CDC solution analysis. Possible values of the -value argument can be 0, 1, or x.

NOTE: *You normally will not be modifying the* `reset` *lines in the generated* autoresets.sgdc

file.

NOTE: *The* `reset` *constraint information is printed in the Section B: Propagated Control Signals Section of The Clock-Reset-Summary Report.*

Modifying Clock Domains in the Generated SGDC Files

By default, *clock* constraints generated in the autoclocks.sgdc file are assumed from a separate clock domain. In this case, the *CDC Verification Rules* report a violation for clock domain crossings between each pair of clock signals.

However, the tool may consider some clock signals in a design to be from the same domain. In this case, the tool assumes no synchronization issues in data transfer between flip-flops triggered by such clock signals.

You can supply such clock domain information by modifying the clock domain information to `clock` keyword lines in the Design Constraints file.

Modify the values of the `-domain` argument of `clock` keyword lines to a valid string or clock name so that all clocks of the same clock domain have the same value of the `-domain` argument.

For example, the following specification indicates that two clock signals `clk1` and `clk2` are of the same domain (named `A`) and one clock signal `clk3` is of a different domain (named `B`):

```
current_design myDU
  clock -name clk1 -domain A
  clock -name clk2 -domain A
  clock -name clk3 -domain B
```

Then, all clock crossings between one of `clk1` and `clk3` or `clk2` and `clk3` are candidates for clock domain crossings. Clock crossings between `clk1` and `clk2` are not reported because they are in the same clock domain (domain `A`).

The `-domain` argument is optional in the `clock` constraints. If you do not specify it, the name of the clock domain is assumed as the clock name. Therefore, the following specification indicates that the two clock signals `clk1` and `clk2` of the same domain (named `clk1`) and one clock signal

clk3 is of a different domain (named `clk3`):

```
current_design myDU
  clock -name clk1 -domain clk1
  clock -name clk2 -domain clk1
  clock -name clk3
```

Then, all clock crossings between one of `clk1` or `clk2` and `clk3` are candidates for clock domain crossings.

Using the Setup Manager

The setup manager enables you to create a setup by:

- Extracting and completing clocks and reset definitions.
- Configuring black boxes.
- Setting boundary (IO) assumptions.
- Defining acceptable synchronization practices for a design or project.

To create a setup, perform the following steps:

1. Load the design in SpyGlass Console.
2. Perform the design-read in process.
3. Select the Goal Setup and Run tab.
4. Select the `clock_reset_integrity` goal under the Select Goal tab.
5. Click on the Setup Goal tab.
6. Start the setup wizard of the setup manager.
7. Follow all the steps of the setup manager.

The setup is considered complete if there are zero errors and zero warnings.

CDC Analysis based on sg_clock_group

While performing timing analysis, a Static Timing Analysis (STA) tool considers all the clock pairs that are synchronous to each other unless an async relation is explicitly specified between them by using the `set_clock_group` or `set_false_path` constraint. As STA considers

only two clocks at a time, there is no ambiguity. SpyGlass, however, tries to compute clock domains by considering all the clocks and the involved constraints.

Consider the relationship between the three clocks as shown in the figure below.

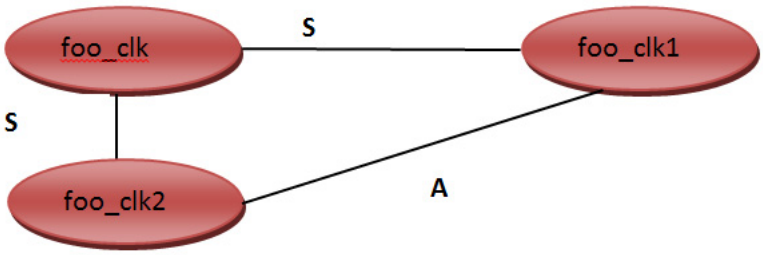


FIGURE 2.

In the above figure, since the `foo_clk` clock is synchronous to both the other clocks, `foo_clk1` and `foo_clk2`, it should be assigned in the same domain as the other two. However, `foo_clk1` and `foo_clk2` are asynchronous so they cannot be assigned in the same domain.

If you assign `foo_clk1` and `foo_clk2` in different domains to make them asynchronous, which would result in `foo_clk` being asynchronous to either `foo_clk1` or `foo_clk2`, the clock domains would look similar to as shown below:

```
clock  -name "clk"  -domain d0  -edge { "0.000000"
"2.500000"}  -period 5 -tag foo_clk
clock  -name "clk1"  -domain d0  -edge { "0.000000"
"10.000000"}  -period 20 -tag foo_clk1
clock  -name "clk2"  -domain d1  -edge { "0.000000"
"20.000000"}  -period 40 -tag foo_clk2
```

However, since this does not help specify the exact clock relationships, SpyGlass provides the *sg_clock_group* constraint that you can use to specify exact clock relationships. For the purpose of the example described above, use the following constraint specification to specify asynchronous

relationship between `foo_clk1` and `foo_clk2`.

```
sg_clock_group  -group1 foo_clk1 -group2 foo_clk2
```

The `clock_<top name>_clock_relationship_matrix_<n>.csv` file is generated and shows the relationship between the clocks as shown the figure below.

	A Clock Name	B Filename:Line	C foo_clk	D foo_clk1	E foo_clk2
1	foo_clk	test3.sdc:1	NA	S	S
2	foo_clk1	test3.sdc:3	S	NA	A (SCG)
3	foo_clk2	test3.sdc:4	S	A (SCG)	NA

Fixing Clock and Reset Integrity Problems

This step ensures that clocks and resets are properly defined, and they are free of glitches, race conditions, and other hazards.

You must fix clock and reset integrity problems by running the `clock_reset_integrity` goal.

Performing CDC Verification

CDC verification means detecting CDC problems in a design.

To perform CDC verification, perform the following steps:

1. Set the required parameters.

   For details on all parameters of SpyGlass CDC solution, see *Parameters in SpyGlass CDC*.
2. Run goals, such as `cdc_verify` and `cdc_verify_struct`, to detect a large spectrum of CDC problems.

You may initially find a large number of CDC violations. It is important to approach them in a systematic way so that you quickly reach a handful of issues that need consideration.

The following issues cover a majority of important violations:

- *Unsynchronized Crossings Issues*
- *Convergence Issues*
- *Reset Synchronization Issues*
- *Glitch Issues*
- *Signal Width Errors in Synchronized Control Crossings*
- *Data Hold Issues in Synchronized Data Crossings*

For all the other violations, search the SpyGlass CDC documentation for a violation by its rule name. For details, see *Rules in SpyGlass CDC*.

Unsynchronized Crossings Issues

First look at the unsynchronized domain crossings reported by *Ac_unsync01* and *Ac_unsync02* rules.

These rules report the areas of potential synchronization failures at clock domain crossings. To understand more about the likely problems, see *Reasons for Synchronized Crossings Reported by Ac_sync_group Rules* and *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules*.

To understand on how to debug these issues, see *Debugging CDC Issues*.

To get more specific details on rule violations, see *Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02*.

The following example shows the unsynchronized crossing issue:

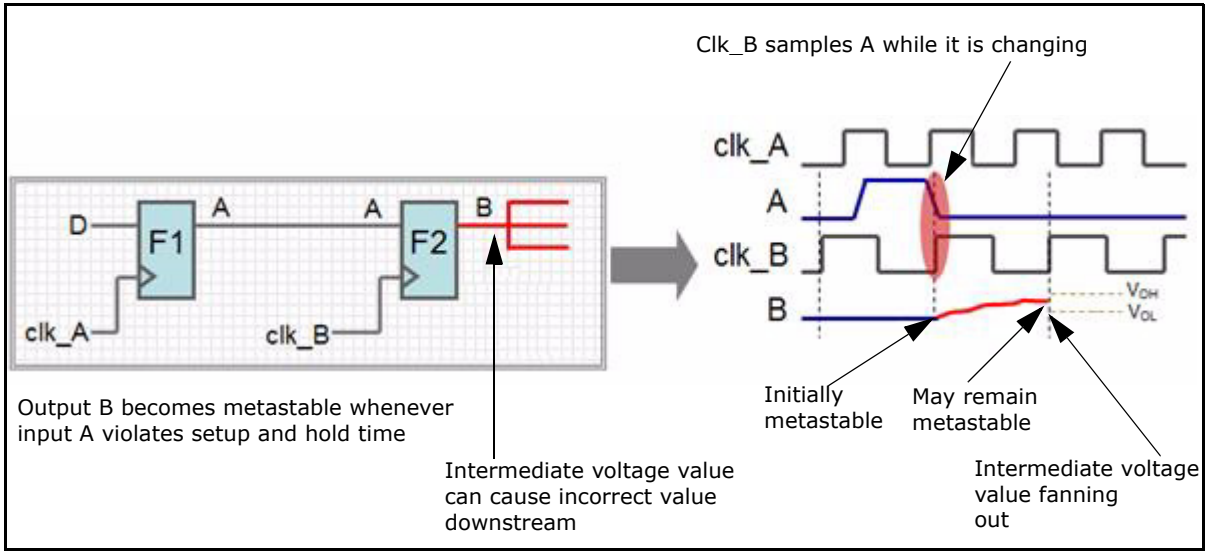


FIGURE 3. Unsynchronized crossing causing metastability problem

Convergence Issues

Convergence issues can occur when multiple signals cross from one domain to another but they are separately synchronized. For example, consider the following figures:

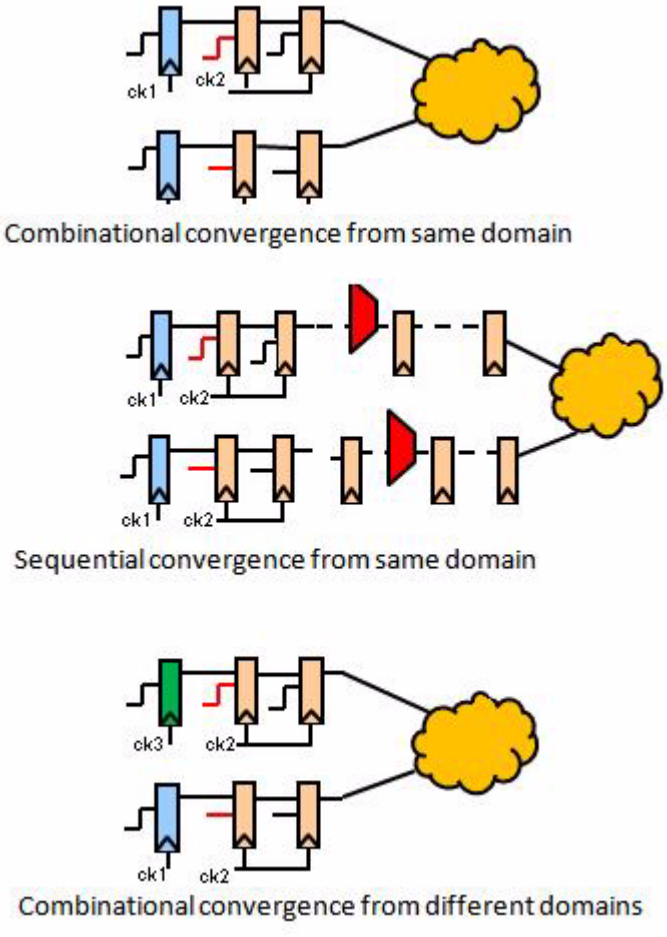


FIGURE 4. Convergence Issues

In the above figures, even though `X4` and `Y4` are separately and correctly synchronized, you cannot be sure if they will have simultaneously valid values when they reconverge.

In addition, convergence and coherency checks of reset control synchronizers are performed by the Ac_conv rules when the *coherency_check_type* parameter is set to `reset` as shown in the following

figure.

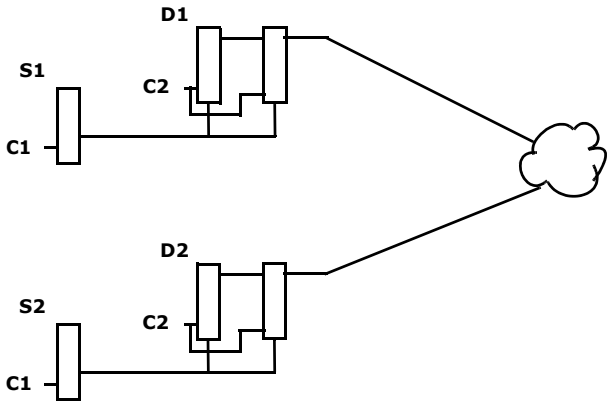


FIGURE 5. Convergence Issues of Reset Control Synchronizers

For information on such types of violations, see *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, and *Ac_conv05*.

For information on debugging such issues, see *Debugging CDC Issues*.

Reset Synchronization Issues

For such issues, check the `Ar_*` rule violations. These rules report violations for synchronizing asynchronous reset signals.

As resets are usually single-bit signals, you might expect them to be reported under *Ac_sync01*. However, resets typically require different synchronization techniques. For example, asynchronous resets can be asserted asynchronously, but they must be deasserted synchronously.

For example, the following figure shows a reset that deasserts synchronously:

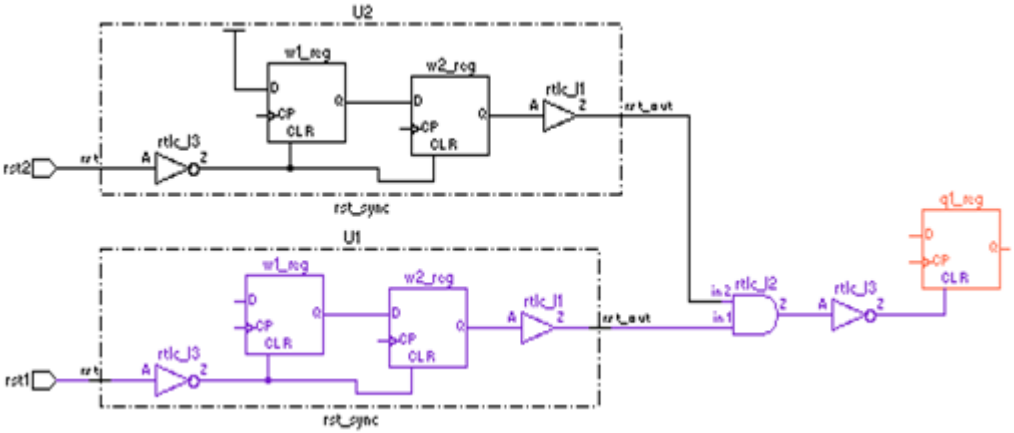


FIGURE 6. Reset that Deasserts Synchronously

In the above figure, the reset is properly synchronized, but it deasserts synchronously.

For information on these violations, see *Ar_asyncdeassert01*, *Ar_syncdeassert01*, *Ar_sync01*, and *Ar_unsync01*.

For information on debugging such issues, see *Debugging CDC Issues*.

The *Ar_cross_analysis01* rule performs crossing detection and synchronization checks and reports all the clock domain crossings in the reset path in a design. Users do not need to specify reset definitions in the constraints file as is required by the Ar_sync rules.

Glitch Issues

Check for any violation reported by `Ac_glitch*` or `Clock_glitch*` rules.

These rules highlight glitch-prone logic that can lead to problems very similar to synchronization issues.

For example, the following figure shows the reconverging combinational logic that is prone to glitch:

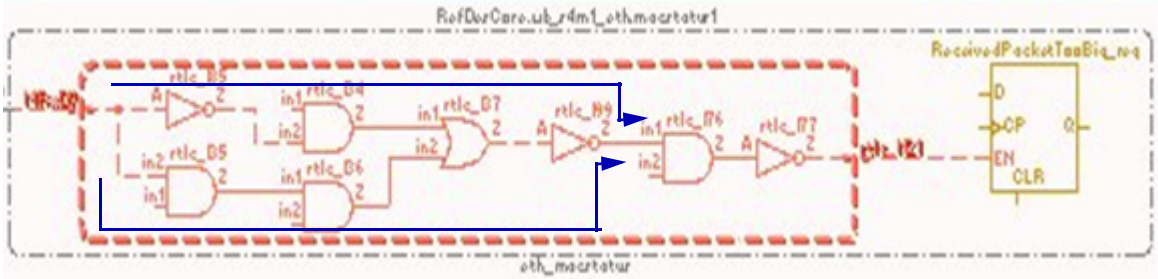


FIGURE 7. Glitch-prone reconverging combinational logic

For information on these violations, see *Ac_glitch01*, *Ac_glitch03*, *Clock_glitch02*, *Clock_glitch03*, *Clock_glitch04*, *Clock_converge01*, and *Reset_sync01*.

For information on debugging such issues, see *Debugging CDC Issues*.

Signal Width Errors in Synchronized Control Crossings

Check for the *Ac_cdc01* rule violation.

Such violations indicate potential problems in signals or data crossing typically from a fast clock domain to a slower clock domain where the data sent may have already changed by the time the capturing clock arrives.

The following figure shows the example of signal width issue:

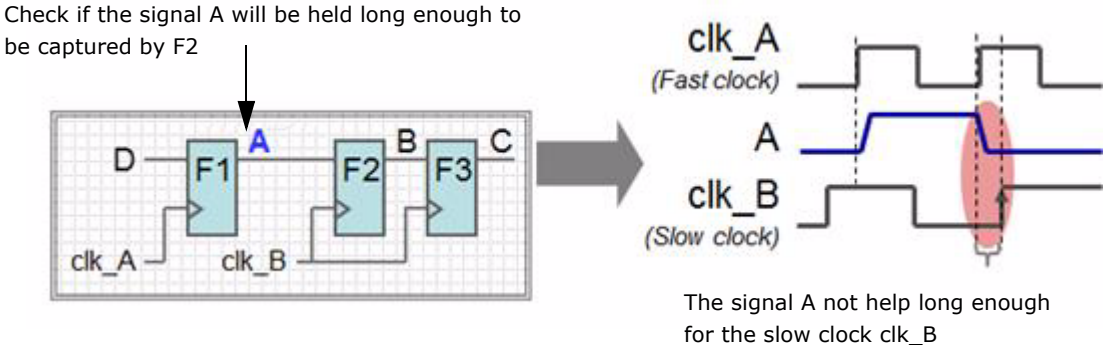


FIGURE 8. Example of a signal width issue

For information on debugging such issues, see *Debugging CDC Issues*.

Data Hold Issues in Synchronized Data Crossings

Check for the *Ac_datahold01a* violation.

The signals reported by such violations are present where a data synchronization structure is used but is not functioning correctly.

Consider the scenario shown in the following figure:

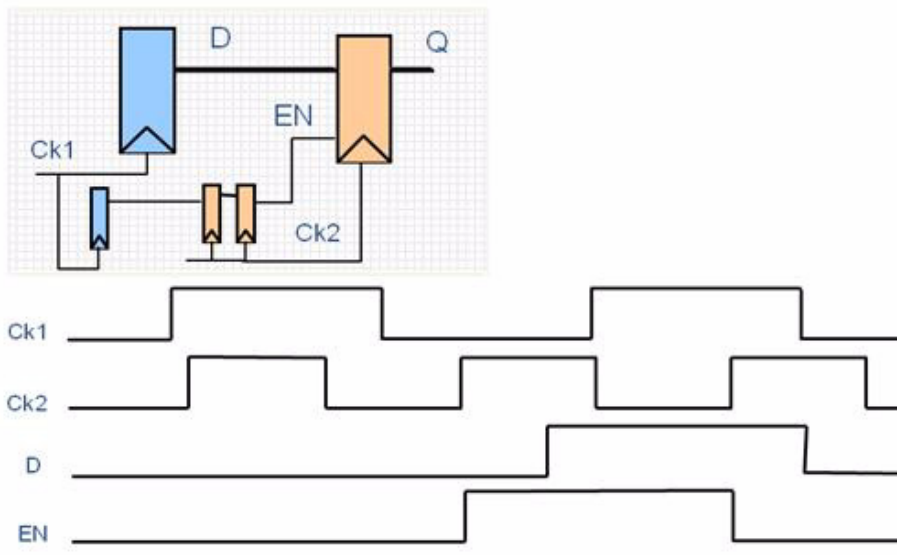


FIGURE 9. Incorrectly Synchronized Data Crossings

The above scenario depicts incorrectly synchronized data crossings. Here, the data is changing while the enable is active.

For information on debugging such issues, see *Debugging CDC Issues*.

Debugging CDC Issues

Before debugging, ensure that the analysis is set up correctly. For details, see *Creating SpyGlass CDC Setup*.

If you are running SpyGlass CDC on a reasonable size design, you are likely to see a large number of violations. Most of these violations are because of:

- Incorrect or incomplete setup.
- Configuration signals that should not typically be reported as CDC errors.

You can remove these violations in a systematic way, leaving only a handful of potential real problems that you need to consider.

NOTE: *Never deal with CDC issues by waiving violations. There is a significant danger that you will mask a real problem if you follow that approach.*

You can debug CDC issues by:

- *Using Spreadsheets*
- *Using Incremental Schematic*
- *Viewing Debug Data in Schematic*
- *Filtering Violations Based On Instances*
- *Solving CDC Issues Common to Multiple Violations*

Using Spreadsheets

When there are many violations, a significant percentage of these violations appear from a few number of root causes. The recommended way to analyze them is by using the spreadsheet viewer.

Perform the following steps to use the spreadsheet viewer for debugging many violations:

1. *Open the Spreadsheet Viewer*
2. *Filter and Sort Data*
3. *Check for Common Reasons or Sources*
4. *Filter Signals by Source*

Open the Spreadsheet Viewer

Open the spreadsheet by right-clicking on the violation header and selecting the *Spreadsheet Viewer* option from the shortcut menu.

Filter and Sort Data

Use filtering and sorting in the spreadsheet view to isolate common factors between violations.

To filter or sort data, right-click on a column header and select an appropriate option from the shortcut menu, as shown in the following figure:

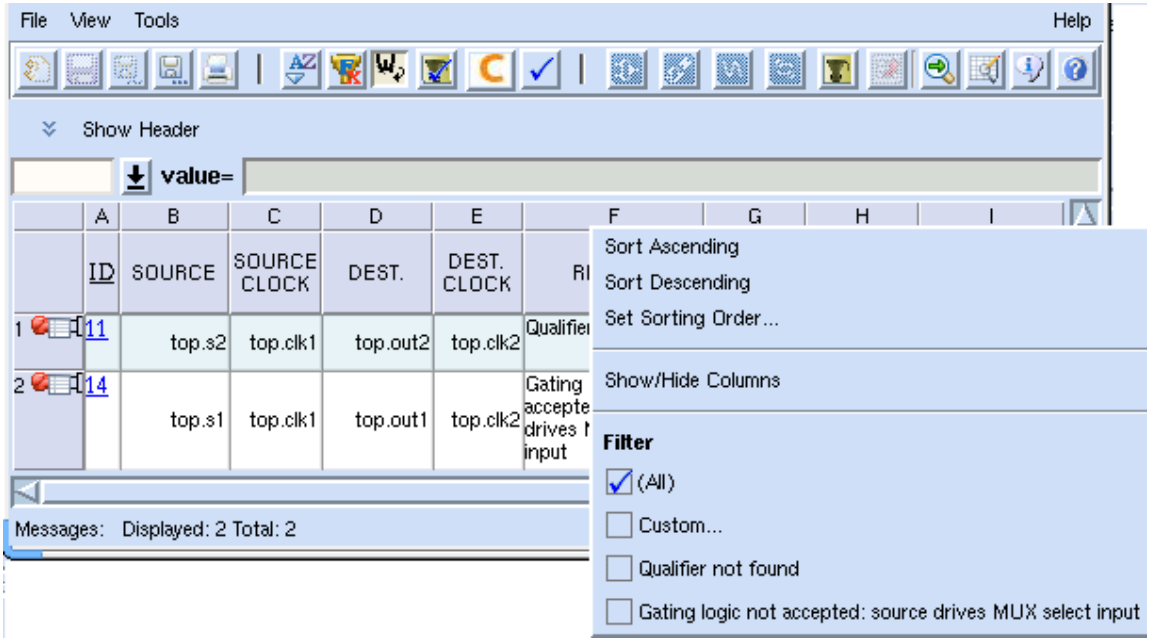


FIGURE 10. Filtering and Sorting in the Spreadsheet

Check for Common Reasons or Sources

Look for common reasons or common sources in the spreadsheet.

These will most probably point to a single root cause.

Filter Signals by Source

If you are using a naming methodology for static signals, filter by source name in the spreadsheet.

For example, the following figure shows how you can specify the filter criteria (`_cfg`) for sources:

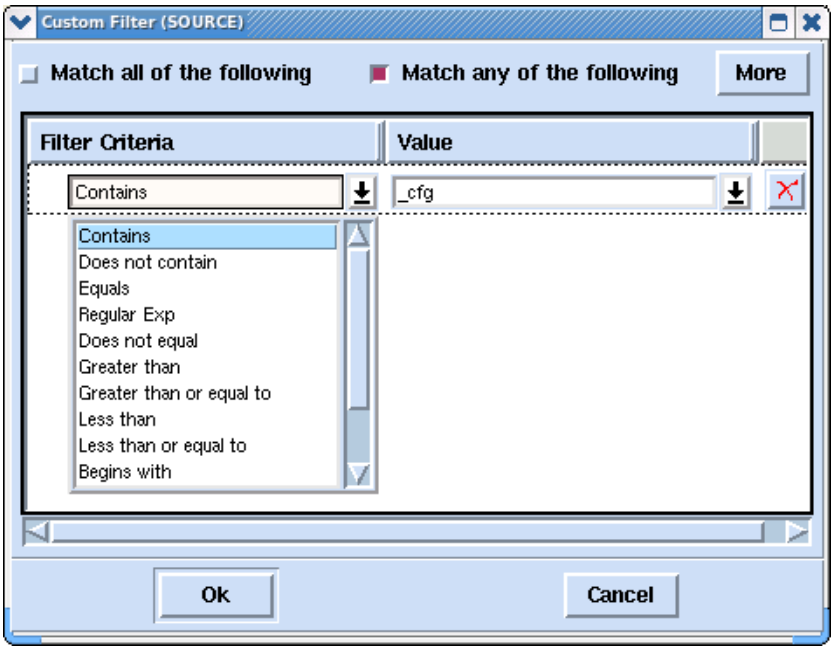


FIGURE 11. The Custom Filter Dialog

To open the above dialog, select the *Custom* option from the shortcut menu shown in *Figure 10*.

View the filtered list and address the root cause to eliminate a large number of violations.

Using Incremental Schematic

A schematic enables you to understand and isolate root cause of a violation.

You can view the schematic of multiple related violations appearing in a spreadsheet. To do so, perform the following steps:

1. Select the violations in the spreadsheet by keeping the <*Ctrl*> keyboard key pressed.
2. Open the *Incremental Schematic* by performing any of the following actions:
   - ❒ Click the link in the *Schematic* column of a row in the spreadsheet.
   - ❒ Click the *Incremental Schematic* button from the spreadsheet toolbar.

The following figure shows the incremental schematic:

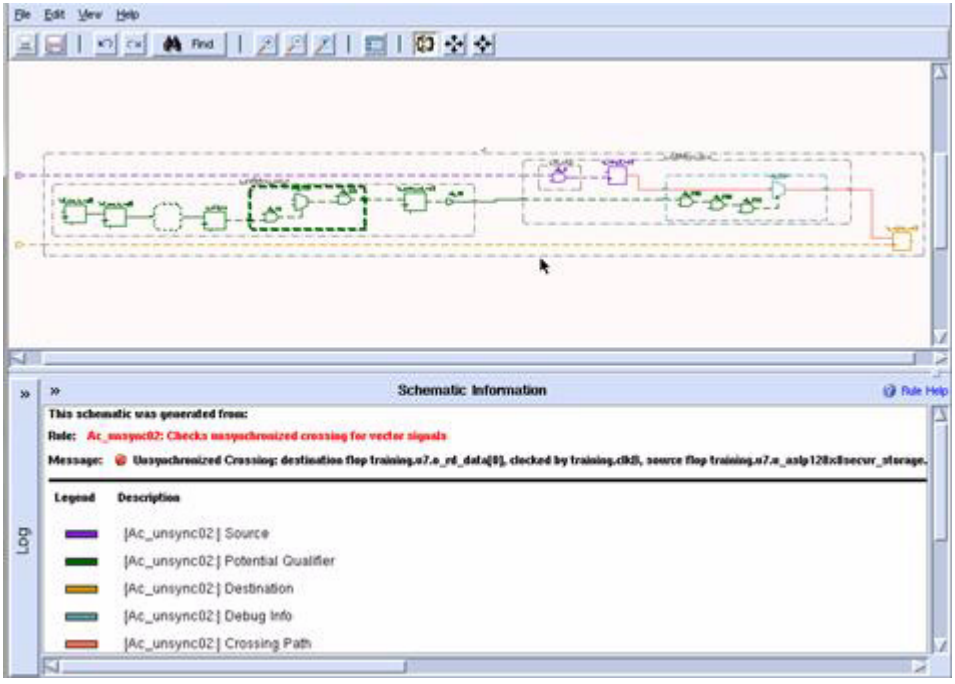


FIGURE 12. The Incremental Schematic Window

Cross probing from schematic to RTL and vice-versa occurs automatically.

Tips to Use the Incremental Schematic

Following are some useful tips:

- Use the legend to know the colors used to identify domains and qualifiers.
- Always run the *Info_Case_Analysis* rule to see propagation of constant values in the schematic.
- Right-click on any net (and not on a pin) and select the *Show Debug Data->Clock-reset* option to see clock and domain information.

  For details, see *Viewing Debug Data in Schematic*.
- Expand the hierarchy boundaries can by double-clicking on the boundary edges.
- Trace the inputs and outputs by double-clicking on the object whose inout/output needs to be traced.
- Trace the inputs and outputs by using appropriate right-click menu options in the schematic to trace to flip-flops, latches, inputs, outputs, and modules.

Viewing Debug Data in Schematic

While debugging SpyGlass CDC solution violations in the schematic, the following information is very useful:

- Clock domain information on nets in the clock path
- Reset domain information on nets in the reset path
- Clock domain information on nets in data or control paths in the design
- Quasi_static information on nets in data or control paths in the design

NOTE: *You must set the value of the enable_debug_data parameter to* `yes` *to view the above debug data for SpyGlass CDC solution rules.*

To view such debug data in the schematic, right-click on a net and select the *Show Debug Data->Clock-reset* option from the shortcut menu. When you select this option, a sub-menu appears displaying the following options:

- Clock

  Select this option to view clock information on nets present in the clock

path, as shown in the following figure:

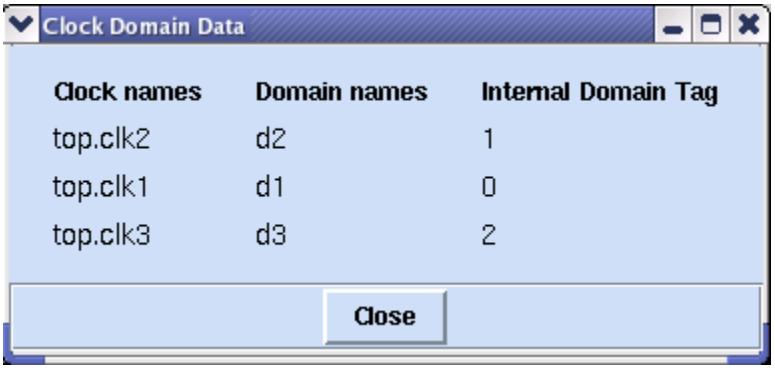


FIGURE 13. The Clock Domain Data Dialog

The *Propagate_Clocks* rule computes the above information.

This option is enabled only for nets that are present in the clock-path of the design.

- *Reset*

  Select this option to view reset information on nets present in the reset path, as shown in the following figure:

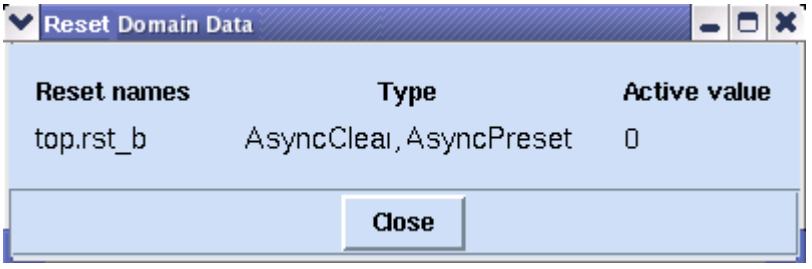


FIGURE 14. The Reset Domain Data Dialog

The *Propagate_Resets* rule computes the above information.

In the above figure, *Type* refers to the reset type, and *Active value* represents the initial value of reset sources.

This option is enabled only for nets that are present in the reset-path of

the design.

- Domain in data/control paths

  Select this option to view clock-domain information on nets in the data or control paths of the design, as shown in the following figure:

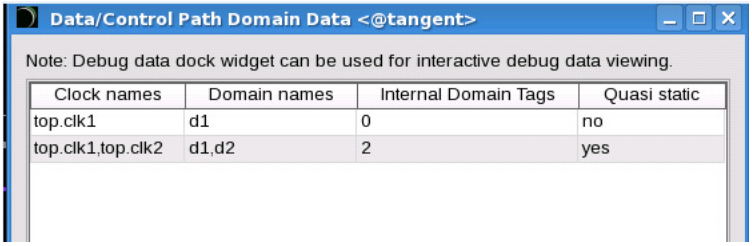


FIGURE 15. The Data/Control Path Domain Data Dialog

In the above figure, *Internal Domain Tag* refers to an internal tag that is computed while performing clock propagation and *Quasi static* refers to paths that have quasi static property.

Data/control path can have the following types of clock-domains:

- ❒ User-specified clock-domain: This category includes primary, black box and derived clocks. For such cases, SpyGlass displays the user-specified clock name.
- ❒ Merged clock domain: If more than one clock is converging on a gate, a merged domain is created internally. In such cases, SpyGlass displays the user-specified list of clocks.
- ❒ Virtual clock domain: If a virtual clock is associated with a primary port of a block instance port, SpyGlass displays the user-specified virtual clock name.

- Quasi_static

  Select this option to check if the net is quasi_static in the data or control path of the design, as shown in the following figure:

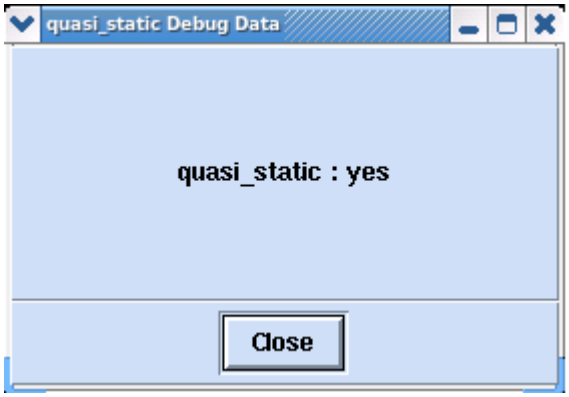


FIGURE 16. The quasi_static Debug Data Dialog

Filtering Violations Based On Instances

While working on large designs, designers are assigned certain design instances. In such cases, designers need to focus on the violations reported on specific instances.

To help designers quickly locate the violations on specific instances, SpyGlass CDC provides the instance-based filtering mechanism. In this mechanism, designers can filter violations based on an instance.

To filter violations based on an instance, perform the following steps:

1. Specify the following command in a project file:

   ```
   set_option enable_module_based_reporting yes
   ```

2. Select the *Tree Viewer* option from the *View* menu.

   The following figure shows the tree viewer:

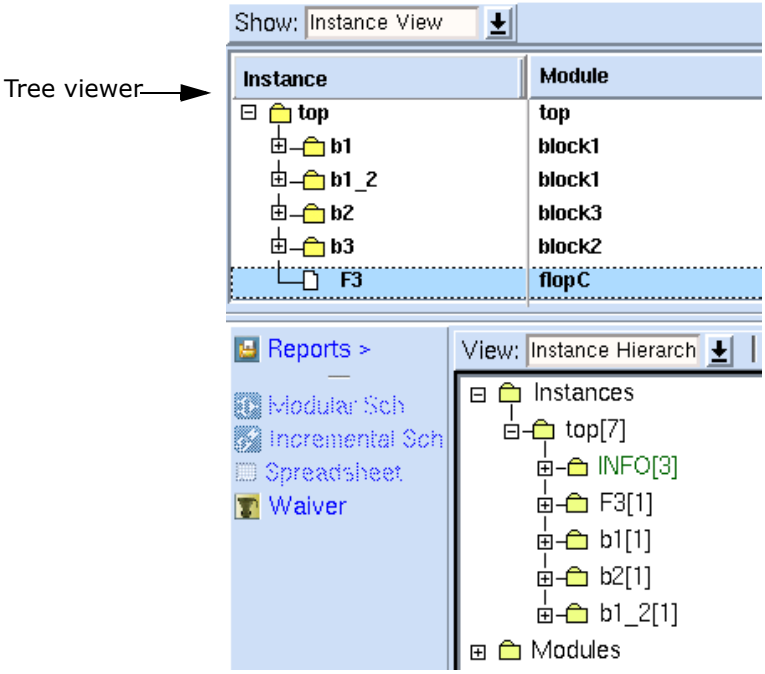


FIGURE 17. The tree viewer

Ensure that the *Instance View* option is selected in the tree viewer.

3. Right-click on an instance (say *F3* in *Figure 17*) in the instance view and select the *Show Messages of Instance* option from the shortcut menu.

   After performing this step, violations are filtered based on the *F3* instance. This is shown in the following figure:

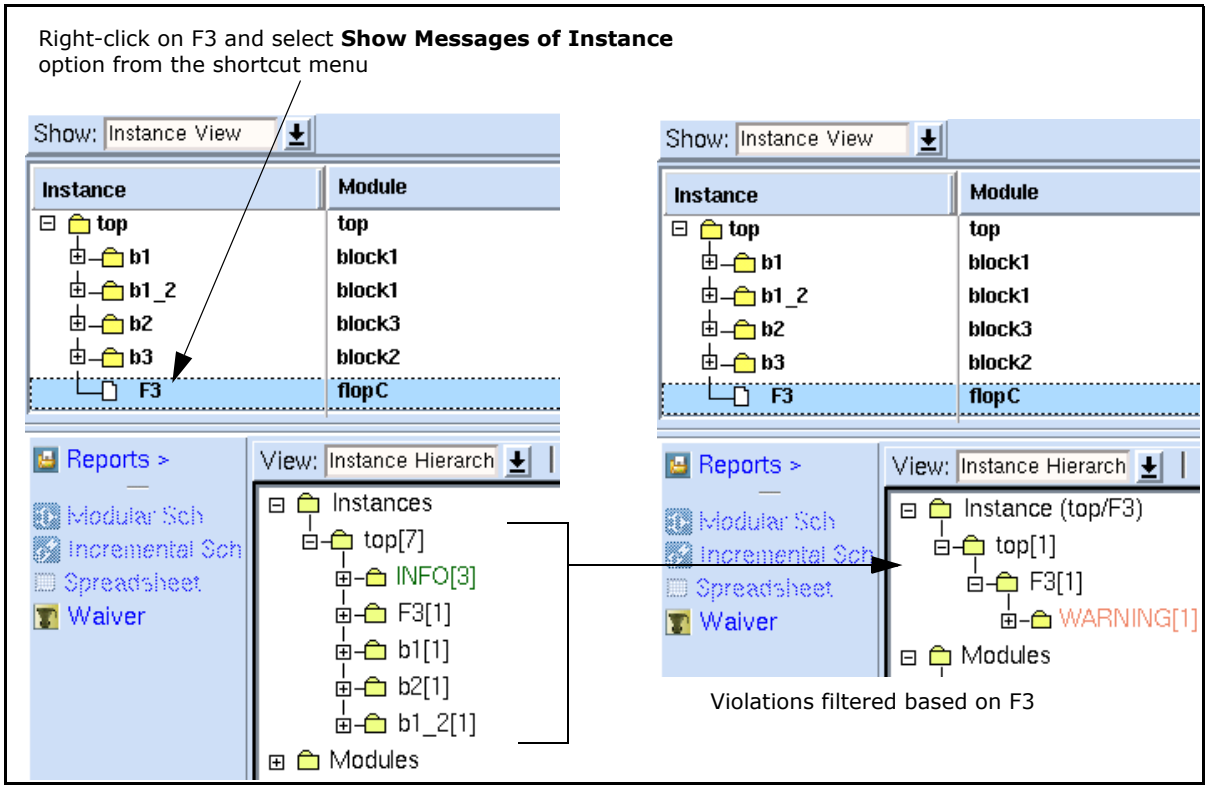


FIGURE 18. GUI view before filtering violations based on instances

NOTE: *You can filter violations based on the destination instance or based on the common module containing the destination and its source by using the msg_inst_mod_report parameter.*

In addition to the *Instance Hierarchy* view, violations also get filtered under *Msg Tree* view, as shown below:

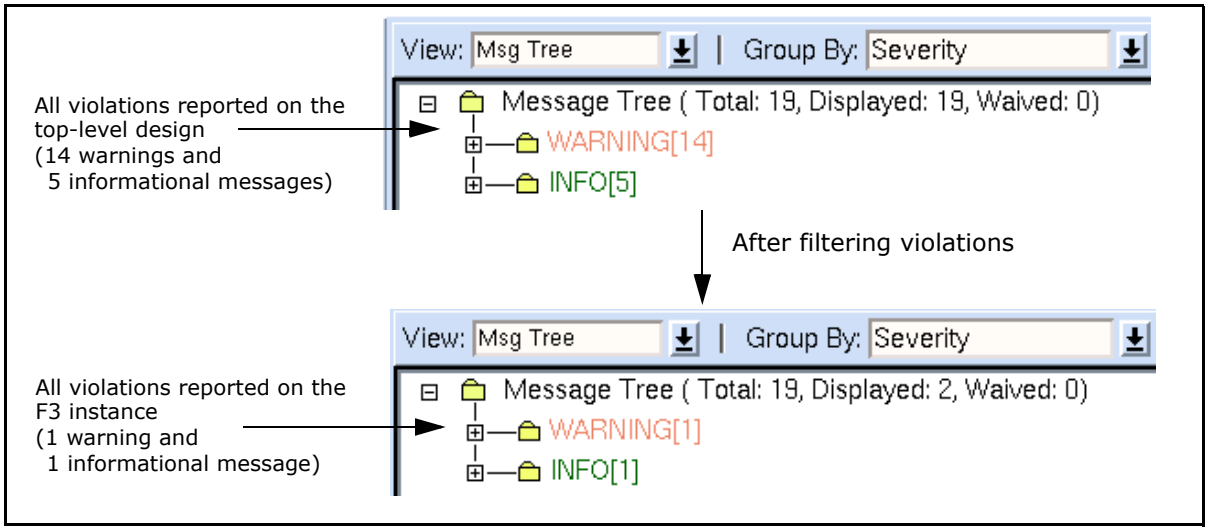


FIGURE 19.

Clearing the Filter

To clear the filter and restore the previous view in which all violations of the top-level design appear, right-click in the *Message Tree* and select the *Clear Message Filter* option from the shortcut menu, as shown in the following figure:

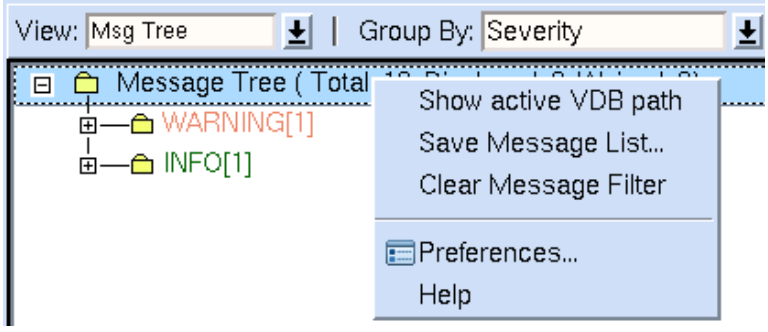


FIGURE 20. Clearing the filter

Saving Messages

To save the violation messages, right-click on the Message Tree pane and select the Save Message List option.

NOTE: *In SpyGlass Explorer, you can save messages for the selected hierarchies. For additional information, see SpyGlass Explorer User Guide.*

Solving CDC Issues Common to Multiple Violations

In case of large number of SpyGlass CDC violations, most of them appear from the following:

- Incorrect setup
- Apparent problems that can be safely ignored

This section describes the following root cause problems that are common to many violations:

- *Crossings Originating From or Ending on a Black Box*
- *Incorrect Case Analysis Settings*
- *Source Flip-Flops Generating Static Signals*
- *Noise*

You should consider addressing the above problems first before focusing on rule-specific problems.

Once you solve the above problems and rerun SpyGlass CDC, you should see substantially smaller and more manageable set of issues.

NOTE: *Never deal with CDC issues by waving violations. There is a significant danger you will mask a real problem if you follow that approach.*

Crossings Originating From or Ending on a Black Box

SpyGlass CDC analysis depends on the following:

- Being able to trace through paths
- Some level of functional understanding

A black box defeats both the above objectives in the upstream and downstream of the black box.

To remove this issue, specify constraints to provide SpyGlass CDC with a partial model, as described below:

- Assign a domain to the black box pin reported in a crossing by using:
  - The *abstract_port* constraint on the black box output.
  - The *signal_in_domain* constraint on the black box input.
- Model a feed through path from the input of the black box to the output of the black box by using the *assume_path* constraint.

Incorrect Case Analysis Settings

Check if the specified *set_case_analysis* constraints are set correctly for this analysis.

For example, you may see false violations because all functional and all test modes are simultaneously active when actually many of these modes will never be active simultaneously.

Source Flip-Flops Generating Static Signals

If a source flip-flop generates a predominantly static signal, no synchronization may be required. This is likely to be the case for configuration signals that are typically set up at power-on/boot time and then not changed again.

Ask the chip architect to get guidance on which signals fall in this group, and declare such signals by using the *quasi_static* constraint.

Noise

One of the major challenges in SpyGlass CDC verification is to manage high number of violations. You can reduce noise through specific setup and setup check steps.

Parameters in SpyGlass CDC

This section provides detailed information on the parameters used in SpyGlass CDC solution. You can set these parameters by using the following command in Atrenta Console and Tcl:

```
set_parameter <parameter_name> <parameter_value>
```

For more information on setting the parameters, refer to the SpyGlass Tcl Interface User Guide and Atrenta Console User Guide.

abstract_validate_express

Enables the *Ac_abstract_validation01* rule to validate only user-specified block assumptions with respect to the top-level block.

This way, missing block assumptions are not checked resulting in less noise during the *SpyGlass CDC Hierarchical Verification Flow*.

By default, the parameter is set to `none`. Set the parameter to one of the supported values to enable express mode for the specified mismatch type.

The following table shows the type of issues reported and not reported by the *Ac_abstract_validation01* rule when this parameter is set to `yes`:

Validation Issue	Violations Reported	Violations Not Reported
Qualifier Mismatch	Violation reported if block-level qualifier does not match with the top-level qualifier	• No violation reported if a qualifier reaches a block port from top level but it is not defined at the block level. • No **CDC SoC abstract auto update flow** occurs
Combo Check Mismatch	Violation reported if top level has a combinational logic but combo -no is specified at the block level	**CDC SoC abstract auto update flow** disabled
Virtual Clocks Mismatch	Violation reported if virtual clocks are not mapped because of conflicting domains	No violation reported if virtual clocks are not mapped because no top-level domain reaches to the block port
Case Analysis Mismatch	Violation reported if: • There is a conflicting constant value with respect to top level or block level. • *set_case_analysis* is specified at block level but no value reaches to the top-level port.	No violation reported when *set_case_analysis* is specified at top level but no *set_case_analysis* is defined at block level.

Validation Issue	Violations Reported	Violations Not Reported
Reset Mismatch	Violation reported if: • Top-level reset reaches at block level but no *reset* is specified at block level, and vice-versa. • Top level asynchronous reset reaches a port specified as a synchronous reset at block level. • There is a conflict in the value of reset with respect to the top level and block level.	No violation reported when a synchronous reset reaches a block port defined as an asynchronous reset.
Clocks Mismatch	Violation reported if: • Top-level clock reaches a block port for which no *clock* is defined, and vice-versa. • Same domain clock ports of an abstract view are driven from different top-level clock domains	No violation reported if different domain clock ports of an abstract view are driven from the same top-level clock domain.
Quasi Static Mismatch	Violation reported if *quasi_static* is defined at block level and no top-level *quasi_static* signal reaches that block port.	No violation reported if top-level *quasi_static* signal reaches a block port where *quasi_static* was not defined.
Issues related with *cdc_false_path*, *define_reset_order*, and *num_flops*	-	No violation reported for any issue with these constraints.

Used by	*Ac_abstract_validation01*, *Ac_abstract_validation02*
Options	none, all, clock_domain, quasi_static, case_analysis, reset, virtual_clock, auto_qualifier, num_flops, yes, no

Default value	no
Default Value in GuideWare2.0	yes
Example	
Console/Tcl-based usage	`set_parameter abstract_validate_express yes`
Usage in goal/source files	`-abstract_validate_express=yes`

ac_sync_mode

Specifies the mode in which *The Ac_sync_group Rules* should run for data crossings.

This parameter accepts the following values:

strict_gate	*soft_gate*	*strict_qual_logic*	*soft_qual_logic*

You must specify a specific combination of the above values to this parameter. For details, see *Valid Combination of Values Specified to the ac_sync_mode Parameter*.

Used by	*The Ac_sync_group Rules*
Options	*strict_gate*, *soft_gate*, *strict_qual_logic*, *soft_qual_logic*
Default value	*strict_gate*, *strict_qual_logic*
Example	
Console/Tcl-based usage	`set_parameter ac_sync_mode "soft_gate, soft_qual_logic"`
Usage in goal/source files	`-ac_sync_mode="soft_gate, soft_qual_logic"`

Valid Combination of Values Specified to the ac_sync_mode Parameter

Consider the following table:

	Column A (default values)	Column B
Row A	*strict_gate*	*soft_gate*
Row B	*strict_qual_logic*	*soft_qual_logic*

Based on the above table, the following points describe the usage of the values that should be specified to the *ac_sync_mode* parameter:

- You must specify one value from each row.
- Do not specify a combination of values from the same row.

For example, do not specify *strict_gate* and *soft_gate* together. Similarly, do not specify *strict_qual_logic* and *soft_qual_logic* together.

- If you do not specify any value to this parameter, default values under *column A* of each row are considered.
- If you specify only one value, SpyGlass internally picks a default value also from the other row.

  For example, if you specify *strict_gate* or *soft_gate* only, SpyGlass internally picks *strict_qual_logic* also. Similarly, if you specify *strict_qual_logic* or *soft_qual_logic*, SpyGlass internally picks *strict_gate* also.

Values used by the ac_sync_mode Parameter

You can set the *ac_sync_mode* parameter to any of the following values:

strict_gate	*soft_gate*	*strict_qual_logic*	*soft_qual_logic*

You must use a specific combination of the above options. For details, see *Valid Combination of Values Specified to the ac_sync_mode Parameter*.

strict_gate

When the *ac_sync_mode* parameter is set to `strict_gate`, *The Ac_sync_group Rules* consider the following schemes as valid synchronization schemes:

- *Synchronized Enable Synchronization Scheme*
- *Recirculation MUX Synchronization Scheme* and *MUX-Select Sync (Without Recirculation) Synchronization Scheme*
- *Clock-Gating Cell Synchronization Scheme*
- *AND Gate Synchronization Scheme*
- *Glitch Protection Cell Synchronization Scheme*

soft_gate

When the *ac_sync_mode* parameter is set to `soft_gate`, the following occurs:

- All synchronization schemes listed with the *strict_gate* option are considered valid, and the corresponding parameters are not required for the specified rules.
- A source signal is considered as synchronized even if a synchronizer is converging on any kind of instance before reaching to a destination instance.

For example, consider the following figure in which a synchronizer is converging with a source signal at a XOR gate before reaching to a destination instance:

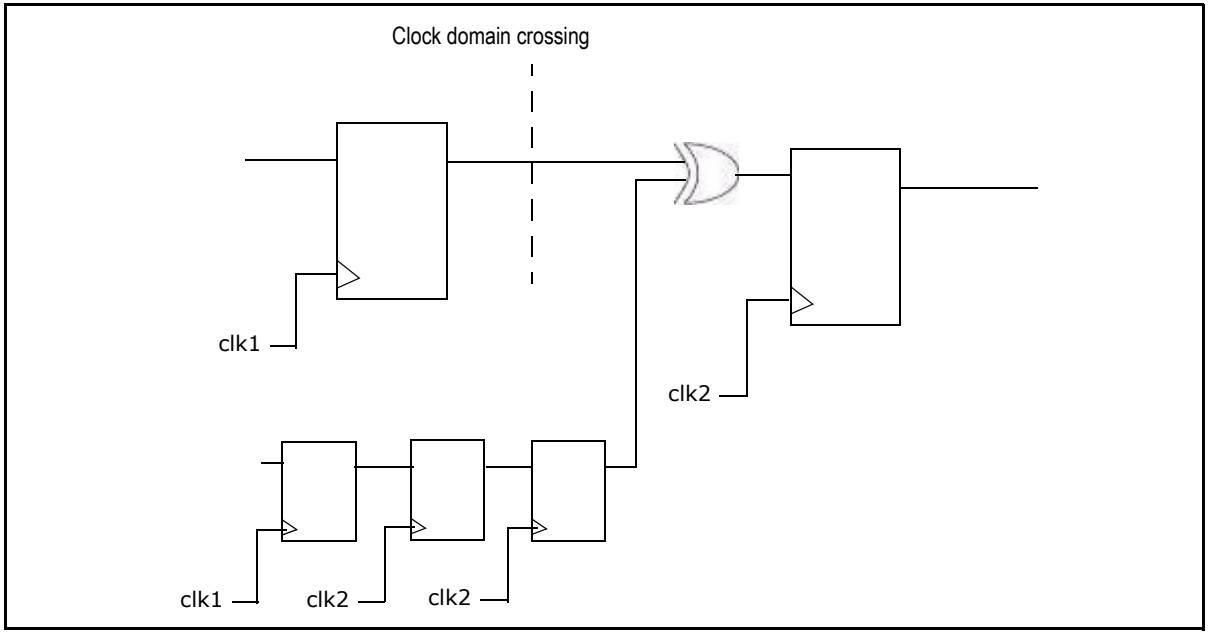


FIGURE 1. Setting the ac_sync_mode parameter is set to soft_gate

In the above figure, the source signal is considered valid if the `ac_sync_mode` parameter is set to `soft_gate`. By default, it is

considered as invalid synchronization and is reported by the *Ac_unsync01* and *Ac_unsync02* rules.

strict_qual_logic

Set the *ac_sync_mode* parameter to `strict_qual_logic` to ensure that a synchronizer should be converging directly with the source signal on a valid gate. A gate is considered as valid based on the *strict_gate* and *soft_gate* values of this parameter.

For example, the following figure is the valid scenario in which a synchronizer is converging directly with the source signal on the AND gate:

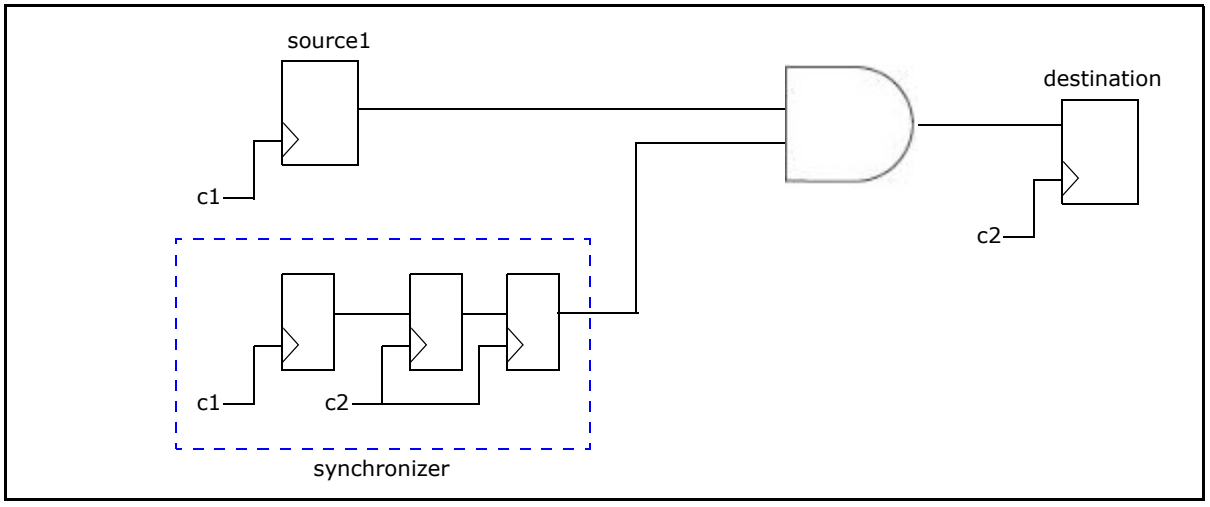


FIGURE 2. Valid scenario when ac_sync_mode is set to strict_qual_logic

However, the above scenario turns invalid if the synchronizer merges with some other same domain source before convergence. This is shown in the following figure:

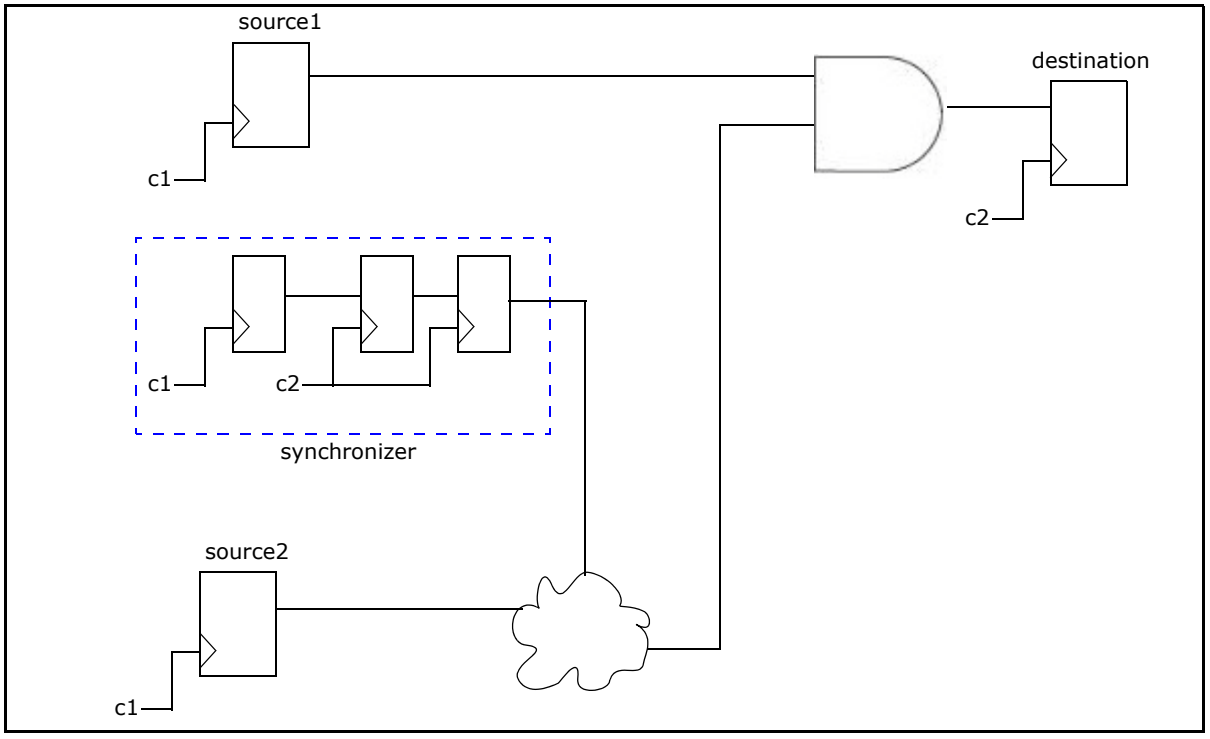


FIGURE 3. Invalid scenario when ac_sync_mode is set to strict_qual_logic

soft_qual_logic

Set the *ac_sync_mode* parameter to `soft_qual_logic` to allow a synchronizer to merge with some other same domain sources before convergence with the source signal.

For example, the scenario in *Figure 3* is a valid scenario when the `ac_sync_mode` parameter is set to `soft_qual_logic`.

all_potential_qual

Shows all *Potential Qualifier* signals, per destination, in the spreadsheet of *The Ac_sync_group Rules*.

By default, this parameter is set to `no` and only one of the potential qualifiers is shown in the spreadsheet.

Set this parameter to `yes` to show all the potential qualifiers in the spreadsheet.

Used by	*The Ac_sync_group Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter all_potential_qual yes`
Usage in goal/source files	`-all_potential_qual=yes`

allow_any_async_pin

Considers a flip-flop to be asserted if both reset and clear pins assert that flip-flop.

By default, this parameter is set to `yes` and a flip-flop is considered as asserted if any of the clear or reset pins assert that flip-flop.

Used by	*Reset_check12*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter allow_any_async_pin no`
Usage in goal/source files	`-allow_any_async_pin=no`

allow_clock_on_hier_term

In the qualifier constraint, the `-from_clk` and `-to_clk` arguments can be specified on hierarchical terminals for module scoping. Set the `allow_clock_on_hier_term` parameter to `yes` to enable this behavior. By default, this parameter is set to `no`.

Used by	All Clock Synchronization, Reset Synchronization, Ac_resetcross01, Ar_resetcross01
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter allow_clock_on_hier_term yes`
Usage in goal/source files	`-allow_clock_on_hier_term=yes`

allow_combo_logic

Specifies if combinational logic (including transparent latches, such as an enable latch) is allowed in the data transfer path between the flip-flops at the clock domain crossing synchronized by the *Conventional Multi-Flop Synchronization Scheme*.

NOTE: *If you specify the allow_combo_logic constraint, preference is given to the constraint instead of this parameter.*

Default Value (no)

By default, this parameter is set to `no` so that:

- No combinational logic is not allowed in the data transfer path.
- Such clock crossings are reported as unsynchronized.

NOTE: *It is recommended that the value of this parameter is set to yes to avoid missing glitch issues in the design.*

Alternate Value (yes)

Set this parameter to `yes` to ignore the combinational logic in the data transfer path between the flip-flops at clock domain crossings. In this case:

- Such crossings are checked for synchronization based on the *Conventional Multi-Flop Synchronization Scheme*.
- Run the *Ac_glitch03* rule to report the issues related to glitch prone combinational logic in synchronized crossings.

When to Use allow_combo_logic Constraint Instead of this Parameter

If you want to allow combinational logic in specific modules, use the *allow_combo_logic* constraint instead of the `allow_combo_logic` parameter.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_glitch03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, *Ac_crossing01*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01*
Options	yes, no
Default value	no
Default Value in GuideWare2.0	yes
Example	
Console/Tcl-based usage	`set_parameter allow_combo_logic yes`
Usage in goal/source files	`-allow_combo_logic=yes`

allow_combo_logic_repeater

Allows combinational logic between source/destination and *Repeaters*.

By default, such logic is not allowed and repeater insertion is reported as invalid by the *Ac_repeater01* rule if combinational logic is present between source/destination and *Repeaters*.

However, no combinational logic is allowed between *Repeaters* irrespective of the value of this parameter.

Used by	*Ac_repeater01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter allow_combo_logic_repeater yes`
Usage in goal/source files	`-allow_combo_logic_repeater=yes`

all_convergence_paths

Configures the *Clock_sync03a* and *Clock_sync03b* rules to report all the convergence points in a path.

By default, these rules report only the last convergence point in a path.

Using this Parameter May Result in Noise

If there are multiple convergences on the same path, the *Clock_sync03a* and *Clock_sync03b* rules report convergence on the last gate in the path, which covers all the converging signals.

Therefore, it is recommended not to use this parameter because it may lead to noise and increase in run time and memory.

Used by	*Clock_sync03a* and *Clock_sync03b*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter all_convergence_paths yes`
Usage in goal/source files	`-all_convergence_paths=yes`

all_converging_clocks

Configures the *Clock_info05* rule to report all the clocks converging on a mux.

By default, only two clocks converging on a mux are reported. If the `all_converging_clocks` parameter is set to `yes`, the *Clock_info05* rule reports information about all input pins of a mux, in the clock_info05 spreadsheet, if at least two clocks converge at two different input pins.

Used by	*Clock_info05*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter all_converging_clocks yes`
Usage in goal/source files	`-all_converging_clocks=yes`

allow_enabled_multiflop

Specifies if flip-flops with explicit enables should be considered as destination or synchronizer flip-flops in *Conventional Multi-Flop Synchronization Scheme*.

Set this parameter to *yes*, *no (default)*, or *same_enable* as explained below:

yes

Set this option to consider flip-flops with explicit enables or mux-enabled flip-flops as destination or synchronizer flip-flops in *Conventional Multi-Flop Synchronization Scheme*.

no (default)

Set this option to ignore flip-flops with explicit enables as destination or synchronizer flip-flops in *Conventional Multi-Flop Synchronization Scheme*.

For example, consider the scenario shown in the following figure (`allow_enabled_multiflop` set to `no`):

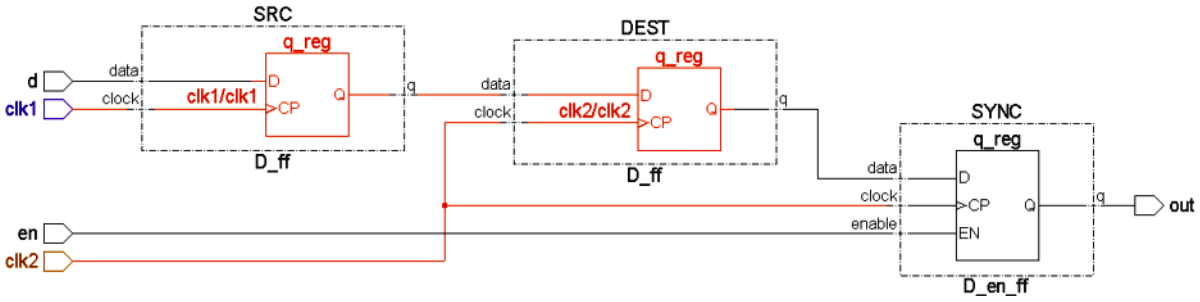


FIGURE 4. Flip-Flops with Explicit Enable

In the above scenario, the crossing is reported as unsynchronized because the synchronizer flip-flop has an explicit enable pin.

same_enable

Set this option to consider a multi-flop synchronizer as destination or a synchronizer flip-flop in *Conventional Multi-Flop Synchronization Scheme* if all the synchronizer flip-flops have the same enable signal.

In addition, mux-enabled flip flops are considered for the *Conventional Multi-Flop Synchronization Scheme* if select pin of all the muxes driven by same

enable signal.

For example, consider the scenario shown in the following figure (`allow_enabled_multiflop` set to `same_enable`):

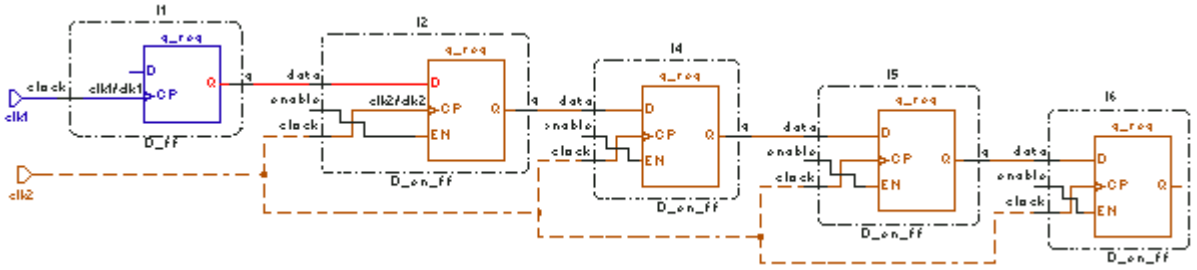


FIGURE 5. Multi-flop synchronizer as destination or synchronizer flip-flop

In the above scenario, the crossing is reported as synchronized because all the synchronizer flip-flops of the multi-flop structure have the same enable signal.

Used by	*Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_coherency06*
Options	yes, no, same_enable
Default value	no
Example	
Console/Tcl-based usage	`set_parameter allow_enabled_multiflop yes`
Usage in goal/source files	`-allow_enabled_multiflop=yes`

allow_half_sync

Specifies if half synchronizers should be considered as valid synchronizers in the *Conventional Multi-Flop Synchronization Scheme*.

When some of the synchronizer flip-flops (the destination flip-flop and the synchronizing flip-flop(s)) are triggered at different edges of the same clock, the synchronizer is called a half synchronizer.

By default, half synchronizers are allowed.

Set the `allow_half_sync` parameter to `no` so that half synchronizers are not treated as valid synchronizers.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc08*, *Ac_crossing01*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_coherency06*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter allow_half_sync no`
Usage in goal/source files	`-allow_half_sync=no`

allow_merged_qualifier

When a *Qualifier* merges with a source, then by default, the output of convergence is considered as a valid qualifier to qualify other sources.

For example, consider the scenario shown in the following figure:

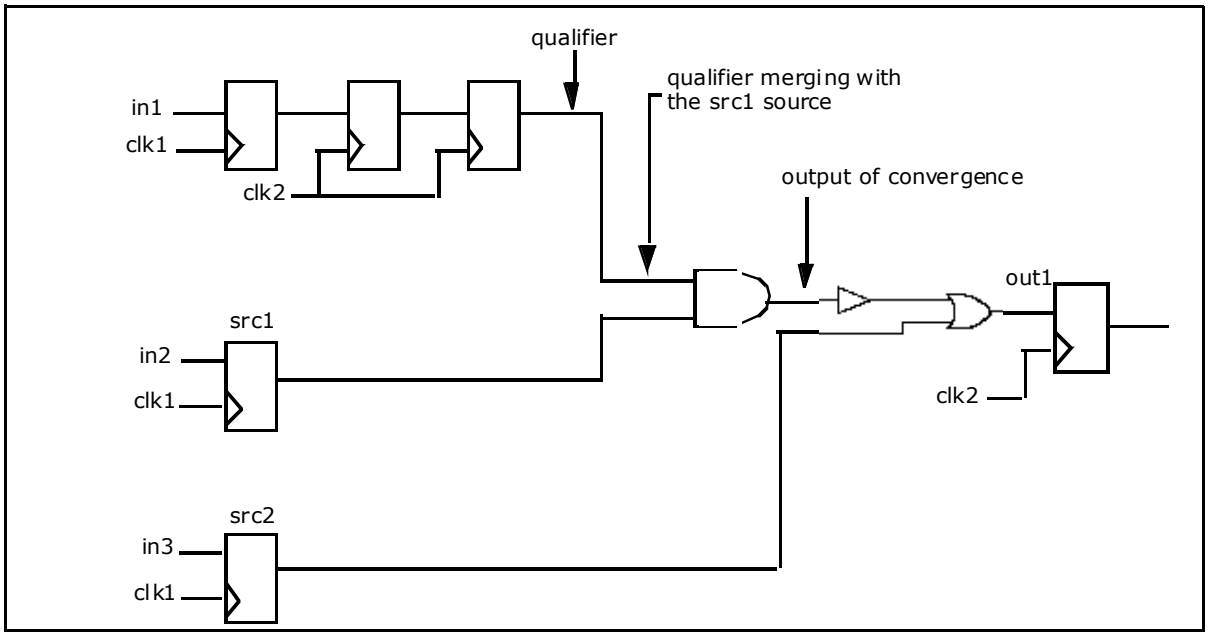


FIGURE 6. Qualifier Merging with a Source

In the above scenario, the *output of convergence* is considered as a valid qualifier by default such that when this output converges with the `src2` source, the `out1` output is considered as a valid qualifier.

The default behavior (`allow_merged_qualifier` parameter is set to `yes`) holds true for the sources that are *Control Signals*.

Set the `allow_merged_qualifier` parameter to `no` such that the `out1` output is not considered as a valid qualifier.

You can set the `allow_merged_qualifier` parameter to `strict` such that when this output converges with the `src2` source, the `out1`

output is considered as a valid qualifier if either:

- The blocking value on the convergence output (where a qualifier and source are merging) can also structurally block other sources. The blocking value on the convergence output is propagated through buffers and inverters only.
- Source is specified as a control signal by using the signal_type constraint.

Used by	*Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, *Ac_unsync02*, and *Ar_cross_analysis01*
Options	yes, no, strict
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter allow_merged_qualifier no`
Usage in goal/source files	`-allow_merged_qualifier=no`

allow_unconstrained_reset_in_rfp

By default, a net without reset constraint is not allowed in the *reset_filter_path* constraint. Set this parameter to `yes` to enable the *reset_filter_path* constraint to accept unconstrained resets.

Used by	*Ar_cross_analysis01*, *Reset_sync02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter allow_unconstrained_reset_in_rfp yes`
Usage in goal/source files	`-allow_unconstrained_reset_in_rfp=yes`

allow_vhdl_on_clock_path

Configures the *NoClockCell* rule to report Verilog constructs in clock trees.

By default, this parameter is set to `no` and the *NoClockCell* rule does not check the type of constructs in the clock trees.

Set this parameter to `yes` to enable the *NoClockCell* rule to report the use of Verilog constructs in the clock trees.

Used by	*NoClockCell*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter allow_vhdl_on_clock_path yes`
Usage in goal/source files	`-allow_vhdl_on_clock_path=yes`

async_reset_usage

Specifies signal types to be reported for non-reset usage by the *Reset_check10* rule.

The values accepted by this parameter are described in the following table (you can specify a comma-separated list of these values):

Value	Description
data (default value)	Specifies that the asynchronous reset signals used as data signals at flip-flops/sequential elements should be reported
control	Specifies that the asynchronous reset signals used as control signals at flip-flops/sequential elements should be reported
port	Specifies that the asynchronous reset signals reaching to primary ports should be reported
bbox	Specifies that the asynchronous reset signals reaching to input of black boxes should be reported
libcell	Specifies that the asynchronous reset signals reaching to a library cell instance that does not have a functional view should be reported
all	Specifies that the all the above types of reset signals should be reported

Used by	*Reset_check10*
Options	Comma-separated list of possible values
Default value	data
Example	
Console/Tcl-based usage	`set_parameter async_reset_usage "data,bbox"`
Usage in goal/source files	`-async_reset_usage="data,bbox"`

auto_detect_datahold01_enable

Enables the quantification flow in Ac_datahold01a rule. The quantification flow enables data synchronization analysis based on the qualifier search in the transitive input cone of a gate that receives the source. Set this parameter to `no` to reuse the enable expressions from the Ac_sync rule.

Used by	*Ac_datahold01a*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter auto_detect_datahold01_enable no`
Usage in goal/source files	`-auto_detect_datahold01_enable=no`

autofix_abstract_port

Enables the *SGDC_qualifier_validation02* and *SGDC_abstract_port_validation04* rules to automatically fix the reported *abstract_port* constraints in the context of SoC.

For details, see *Automatically Fixing the abstract_port Constraint of the Reported Port*.

Used by	*SGDC_qualifier_validation02* and *SGDC_abstract_port_validation04*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter autofix_abstract_port yes`
Usage in goal/source files	`-autofix_abstract_port=yes`

autofix_dump_allinputs

While using the **CDC SoC abstract auto-update flow** in the SoC validation run, you can control the generation of block *abstract_port* constraints by using this parameter.

By default, the value of this parameter is set to `yes` and all the input side *abstract_port* constraints and modified constraints are generated for abstract views.

Set this parameter to `no` to generate only modified constraints.

Used by	*SGDC_qualifier_validation02* and *SGDC_abstract_port_validation04*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter autofix_dump_allinputs no`
Usage in goal/source files	`-autofix_dump_allinputs=no`

cdc_bus_compress

Controls the number of bits of vector signals for rule checking.

The following table describes how the specification of different values to this parameter controls the number of bits of vector signals for rule checking:

TABLE 1 Possible values of the cdc_bus_compress parameter

Parameter Value	Functionality
The **Ac_glitch03** value is specified (default)	The *Ac_glitch03* rule performs glitch checks only on one bit of the destination flip-flop
The **Ac_cdc01** value is specified (default)	The *Ac_cdc01* rule group checks for data loss only on one bit of the destination flip-flop
The **Ac_glitch03** value is not specified	The *Ac_glitch03* rule performs glitch checks on all the bits of the destination flip-flop. This increases the run-time.
The **DeltaDelay02** value is specified	The *DeltaDelay02* checks only one bit of the source bus.
The **DeltaDelay02** value is not specified	The *DeltaDelay02* checks all the bits of the source bus. This may increase the memory consumption.
The **Ac_sync_data** value is specified	The *Ac_sync02* and *Ac_unsync02* rules report violations with respect to the merged destination vector signals of only **synchronized crossings**.
The **Ac_unsync** value is specified	The *Ac_sync02* and *Ac_unsync02* rules report violations with respect to the merged destination vector signals of only **unsynchronized crossings**.

TABLE 1 Possible values of the cdc_bus_compress parameter

Parameter Value	Functionality
The **Ac_sync_control** value is specified	The *Ac_sync02* and *Ac_unsync02* rules report violations with respect to the merged destination vector signals of only **control crossings**.
The **none** value is specified	This is equivalent to not specifying any of the values mentioned in this table

Used by	*Ac_glitch03*, *Ac_cdc01*, and *DeltaDelay02*
Options	Comma separated list of one or more of the *Possible values of the cdc_bus_compress parameter*.
Default value	Ac_glitch03, Ac_cdc01
Example	
Console/Tcl-based usage	`set_parameter cdc_bus_compress DeltaDelay02`
Usage in goal/source files	`-cdc_bus_compress=DeltaDelay02`

cdc_compatible

Makes the rules mentioned in the following table dependant on the *Clock_sync** rules data instead of *The Ac_sync_group Rules* data.

By default, data of *The Ac_sync_group Rules* is used.

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_glitch01*, *Ac_glitch02*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter cdc_compatible yes`
Usage in goal/source files	`-cdc_compatible=yes`

cdc_dump_assertions

Generates SystemVerilog Assertions (SVA) corresponding to SpyGlass CDC rules and *Design Assumptions* specified in an SGDC file.

SGDC Constraints for Which Assertions are Generated

Assertions are generated for the following constraints:

quasi_static	*input*	*reset* (Asynchronous reset)
clock	*abstract_port*	*signal_in_domain*
set_case_analysis		

Rules for Which Partially-Proved Assertions are Generated

Assertions are generated for partially proved properties of the following rules:

Ac_datahold01a	*Ac_cdc01*	*Ac_conv02*

Running the Design in the Audit Mode

On using the `cdc_dump_assertions` parameter with the *fa_audit* parameter, the design is run in the audit mode, and SVA is generated for the *Passed*, *Failed* and *Partially Proved* assertions.

Setting cdc_dump_assertions to sva

When you set the `cdc_dump_assertions` parameter to `sva`, SpyGlass generates the following files in the test_reports/clock-reset/assertions/ directory. These files contain all the data in the SVA format for the simulators.

Files for Rules	Files for Constraints	Description
sva_rules_prop_<top>_bind.sv	sva_assumptions_<top>_bind.sv	Common bind file for all simulators
sva_rules_prop _<top>_vcs.sv	sva_assumptions_<top>_vcs.sv	SVA file for VCS simulator

If the design is not a pure Verilog design, you need to specify the test bench name and design instance name in the SGDC file by using the *meta_design_hier* constraint.

You can choose the simulation initialization and functional window in which the assertions generated should be tested. See the documentation of the *monitor_time* constraint for details.

Pass this file to the third-party tools to check if the data is valid in the context of the design.

Used by	*Ac_datahold01a*, *Ac_cdc01*, and *Ac_conv02*
Options	sva
Default value	""
Example	
Console/Tcl-based usage	`set_parameter cdc_dump_assertions "sva"`
Usage in goal/source files	`-cdc_dump_assertions="sva"`

cdc_effective_bus_verif

Configures the *Ac_datahold01a* rule to perform efficient and complete SpyGlass CDC verification of all the bits of destination sources with the same destination enable expression.

When you set this parameter to `Ac_datahold01a`, *The Ac_sync_group Rules* report bus merged violations and the *Ac_datahold01a* rule generates a single property for all the bits of destination sources with the same destination enable expression.

When this parameter is set to `none`, the *Ac_datahold01a* rule checks for data hold on any one source bit.

Used by	*Ac_datahold01a*
Options	Ac_datahold01a, none
Default value	none
Example	
Console/Tcl-based usage	`set_parameter cdc_effective_bus_verif Ac_datahold01a`
Usage in goal/source files	`-cdc_effective_bus_verif=Ac_datahold01a`

cdc_express

Reduces SpyGlass runtime based on the following values specified to this parameter:

- `yes`

  Runtime is reduced by disabling the following:

  - Schematic and violation messages related to propagation of clocks (*Propagate_Clocks* rule)
  - Clock propagation related checks, such as *Clock_converge01*, *Clock_delay02*, *Clock_check06a*, *Clock_check06b*, and *Clock_check07* rules.

- `peakmem`

  SpyGlass CDC reduces the peak memory requirement of the SpyGlass run. This does not impact SpyGlass results, but this may slightly increase runtime.

Used by	All SpyGlass CDC rules
Options	yes, no, peakmem
Default value	no
Example	
Console/Tcl-based usage	`set_parameter cdc_express yes`
Usage in goal/source files	`-cdc_express=yes`

cdc_gen_unrelated_coherency

Generates the `unrelated_coherent_signals.sgdc` file that contains the *cdc_filter_coherency* and the *gray_signals* constraints for every *Ac_conv02* violation that has at least one vector bus along with either one or more scalar signal or one or more vector buses.

By default the sgdc file is not generated. Set this parameter to `yes` to generate the file.

Used by	*Ac_conv02*, *Ac_conv02Setup01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter cdc_gen_unrelated_coherency yes`
Usage in goal/source files	`-cdc_gen_unrelated_coherency=yes`

cdc_ignore_multi_domain

Set the cdc_ignore_multi_domain parameter to `data_path` to enable synchronization analysis of a data path clock domain crossing involving multiple source domains. By default, the parameter is set to `none`.

Used by	*The Ac_sync_group Rules*
Options	none, data_path
Default value	none
Example	
Console/Tcl-based usage	`set_parameter cdc_ignore_multi_domain data_path`
Usage in goal/source files	`-cdc_ignore_multi_domain=data_path`

cdc_qualifier_depth

Enables *The Ac_sync_group Rules* to search for *Qualifier* or *Potential Qualifier* through a sequential logic by specifying a **sequential depth**.

The following table describes the logic behind the usage of this parameter.

Sequential Depth	Sequential depth is the number of sequential elements between the **qualifier point** till the point of synchronization.
Qualifier Point	Qualifier point is the start point to search for a *Qualifier* or *Potential Qualifier* behind a crossing.
Setting a Qualifier Point	To set a qualifier point in a multi-flop control crossing, use the *cdc_qualifier_depth_start* parameter. In a crossing other than a multi-flop control crossing, a qualifier point is determined based on the following: • For a sync cell crossing, the qualifier point is the Q pin of the last sequential element within the sync cell boundary. • For the qualifier crossing (*Qualifier Synchronization Scheme Using qualifier -crossing*), the qualifier point is the Q pin of the destination instance. • For the specified *qualifier* constraint, the qualifier point is the point where this constraint is applied. • For the specified *abstract_port* constraint, the qualifier point is the point where this constraint is applied.
Values accepted by the cdc_qualifier_depth parameter	The *cdc_qualifier_depth* parameter accepts a *Positive Integer Value*, 0, or *-1*.

Positive Integer Value

Consider the following figure:

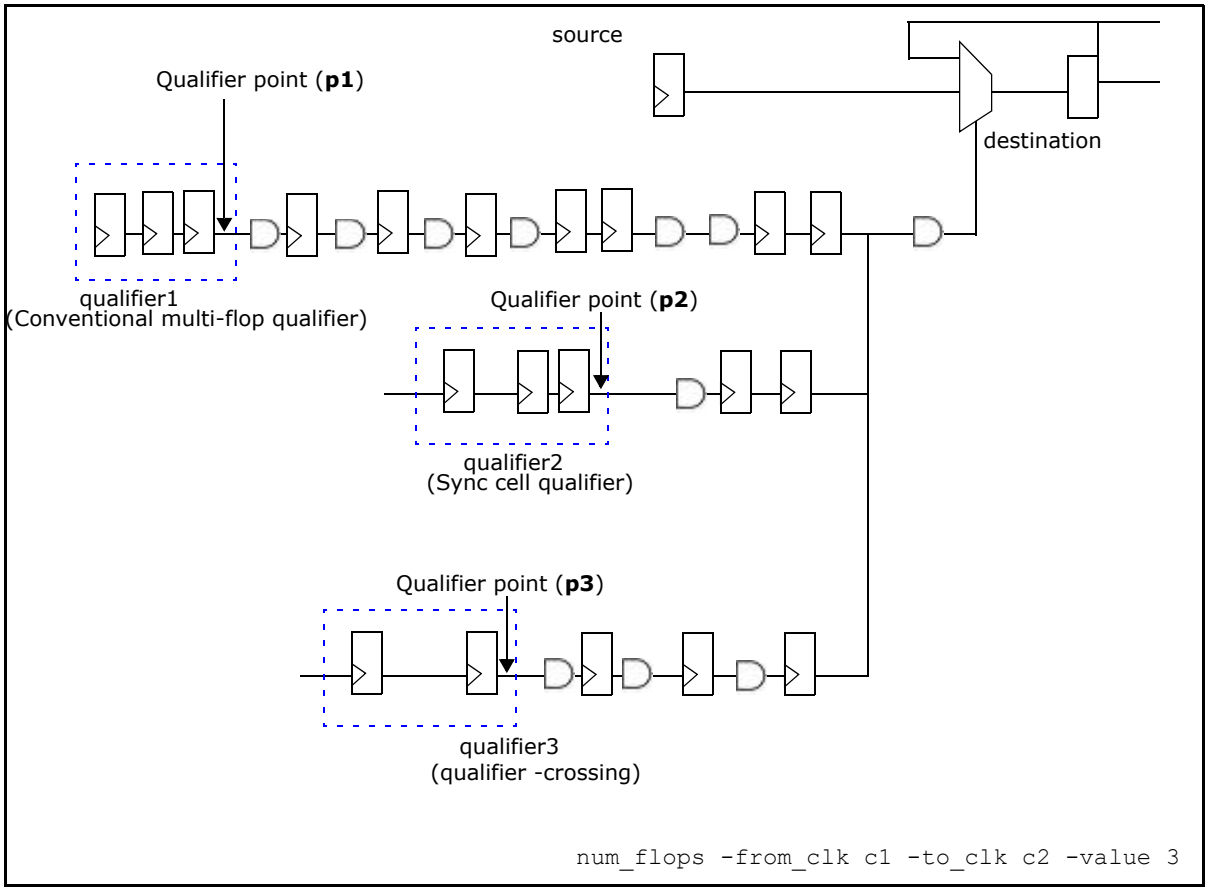


FIGURE 7. Example of using the cdc_qualifier_depth parameter

In the above figure:

- If the *cdc_qualifier_depth_start* parameter is set to *num_flop (default value)* then `p1` becomes the qualifier point till which a qualifier is searched while traversing back from destination.

  Therefore, the minimum depth specified to the `cdc_qualifier_depth` parameter should be 7 so that `qualifier1` can be searched.

- If the *cdc_qualifier_depth_start* parameter is set to *sync_chain* then `p2` is the qualifier point.

  Therefore, the minimum depth specified to the `cdc_qualifier_depth` parameter should be 2 so that `qualifier2` can be searched.
- If you have specified the *qualifier -crossing* constraint for `qualifier3`, specify the minimum depth as 3 to the *cdc_qualifier_depth_start* parameter to search for this qualifier.

-1

Disables the depth-based search scheme and reports first found *Qualifier*.

Used by	*The Ac_sync_group Rules*, *Ac_datahold01a*
Options	-1, 0, positive integer value
Default value	-1
Example	
Console/Tcl-based usage	`set_parameter cdc_qualifier_depth 6`
Usage in goal/source files	`-cdc_qualifier_depth=6`

cdc_qualifier_depth_start

Specifies the start point (**qualifier point**) in a multi-flop control crossing to search for a *Qualifier* or *Potential Qualifier* behind the crossing.

The value specified to this parameter is used by the *cdc_qualifier_depth* parameter to specify a depth for qualifier search.

This parameter accepts three values: *num_flop (default value)*, *sync_chain*, and *dest*.

Used by	*The Ac_sync_group Rules*, *Ac_datahold01a*
Options	num_flop, sync_chain, dest
Default value	num_flop
Example	
Console/Tcl-based usage	`set_parameter cdc_qualifier_depth_start` dest
Usage in goal/source files	`-cdc_qualifier_depth_start=`dest

num_flop (default value)

Specify `num_flop` so that the starting point of qualifier search is from the limit set by the *num_flops* constraint or the *num_flops* parameter.

For example, in the following figure, the Q pin of `f3` is the starting point (qualifier point) for qualifier search.

Assuming that there are two sequential elements in the combinational logic, the depth of the qualifier point is 4. Therefore, to search for the `q1` qualifier, the minimum depth specified to the *cdc_qualifier_depth* parameter should be 4.

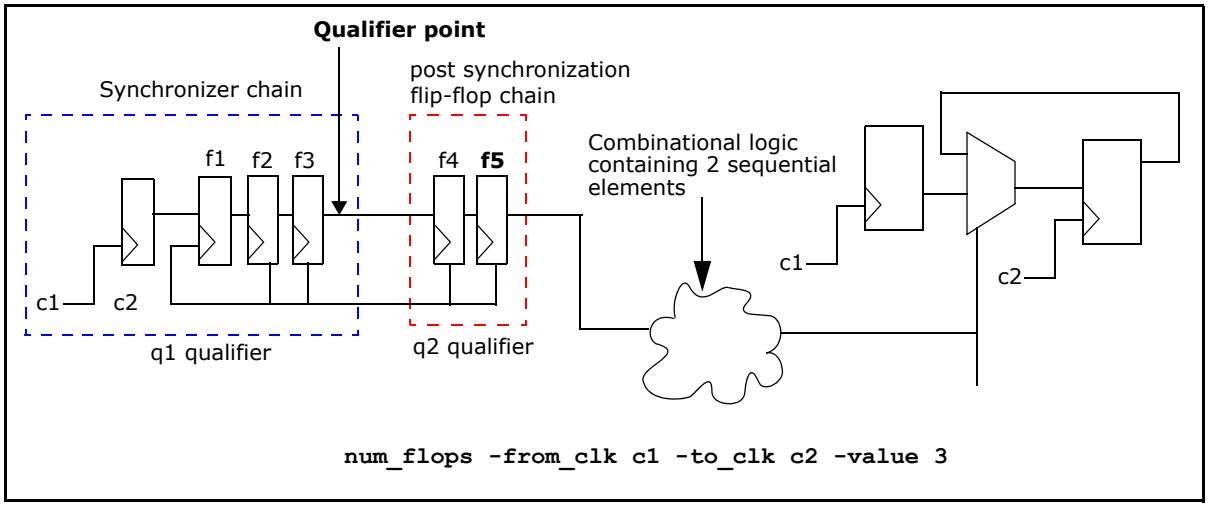


FIGURE 8. cdc_qualifier_depth_start parameter set to num_flop

sync_chain

Specify `sync_chain` so that the last flip-flop of the synchronization chain is the starting point beyond which a qualifier should be searched.

For example, in the following figure, the Q pin of `f5` is the starting point (qualifier point) for qualifier search.

Assuming that there are two sequential elements in the combinational logic, the depth of the qualifier point is 2. Therefore, to search for the `q2` qualifier, the minimum depth specified to the *cdc_qualifier_depth* parameter should be 2.

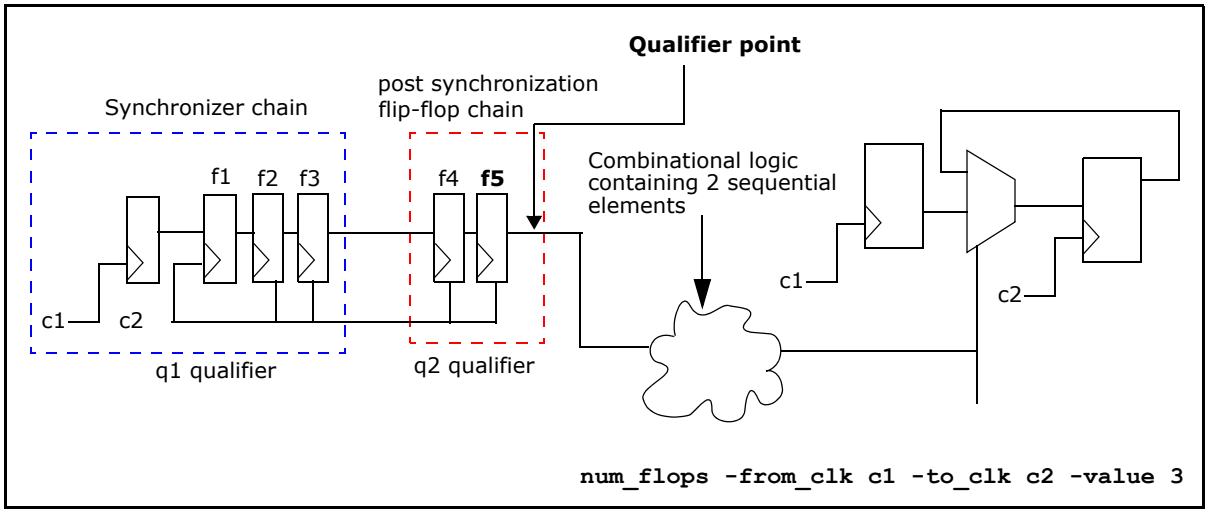


FIGURE 9. cdc_qualifier_depth_start parameter set to sync_chain

dest

Specify `dest` so that destination is the starting point beyond which a qualifier should be searched.

For example, in the following figure, the Q pin of `f1` is the starting point (qualifier point) for qualifier search.

Assuming that there are two sequential elements in the combinational logic, the depth of the qualifier point is 6. Therefore, to search for the `q1` qualifier, the minimum depth specified to the *cdc_qualifier_depth* parameter should be 6.

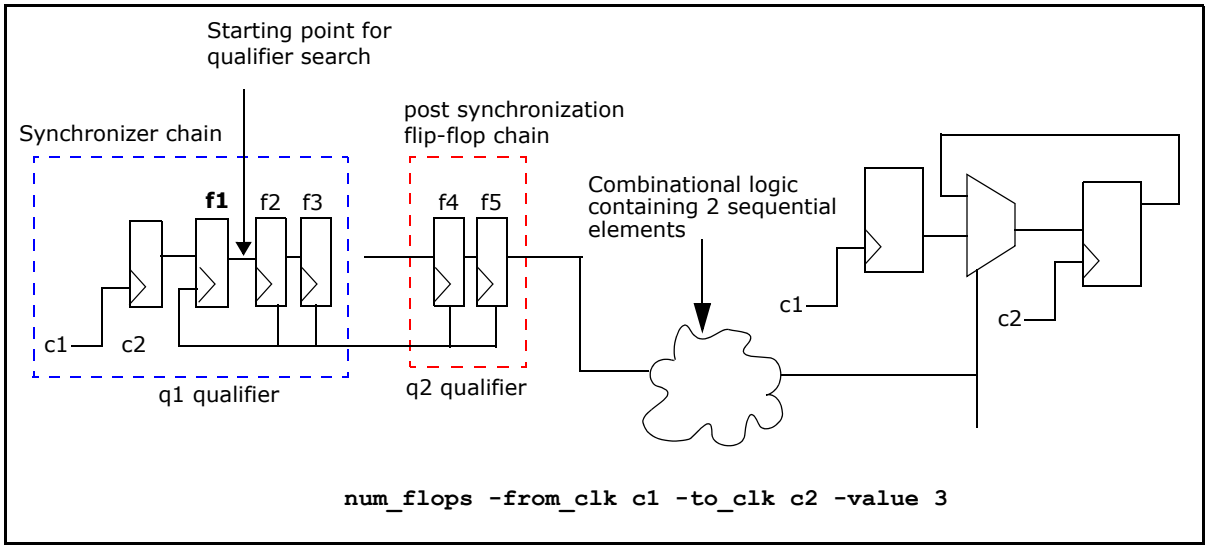


FIGURE 10. cdc_qualifier_depth_start parameter set to dest

cdc_reduce_pessimism

Specifies if:

- The *Clock_sync08a*, *Clock_sync09*, *Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02* rules ignore clock domain crossings involving black box instances or clock domain crossings with destinations having unused, hanging, or blocked outputs.

  The *Clock_sync03a*, *Clock_sync03b*, *Ac_conv01*, *Ac_conv02*, and *Ac_conv03* rules do not perform convergence checks on such ignored crossings.
- The *Clock_sync03a* and *Clock_sync03b* rules report convergence on multi-bit arithmetic macros, sync resets, or D-En convergence on flip-flops.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Clock_info05*, *Clock_info15*, *Setup_port01*, *Clock_info16*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_crossing01*, *Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, *Ac_unsync02*, *Ac_coherency06*, *Ar_sync01*, *Ar_unsync01*, *Reset_sync02*, *Reset_check10*, *Reset_info01*, *Ar_asyncdeassert01*, *Ar_syncdeassert01*, *Ac_glitch03*, *Setup_req01*, *Param_clockreset04*, *Propagate_Clocks*, *Ar_unsync01*, *Ac_glitch03*, *Clock_converge01*, *Ar_converge01*, *Clock_glitch05*, *Ar_resetcross01*
Options	Any combination of the *Allowed Values of the cdc_reduce_pessimism Parameter* as a comma-separated list or using the + (plus) character to append to the default value.
Default value	mbit_macro, no_convergence_at_syncreset, no_convergence_at_enable
Example	

Console/Tcl-based usage	The following specifications are the same:
	`set_parameter cdc_reduce_pessimism mbit_macro,no_convergence_at_syncreset,no_convergence_at_enable,bbox,hanging_net`
	
	`set_parameter cdc_reduce_pessimism +bbox,hanging_net`
	
	The + character must be the first character when specified. Therefore, bbox,hanging_net+ is incorrect.

Usage in goal/source files	`-cdc_reduce_pessimism= "mbit_macro,no_convergence_at_syncreset,no_convergence_at_enable,bbox,hanging_net"`

Allowed Values of the cdc_reduce_pessimism Parameter

The allowed values of the `cdc_reduce_pessimism` parameter are *bbox*, *output_not_used*, *hanging_net*, *mbit_macro (default)*, *no_convergence_at_syncreset (default)*, *no_convergence_at_enable (default)*, *skip_unused_paths*, *ignore_multi_domain*, *use_multi_arc*, *clock_crossing*, *const_source*, *glitch_on_vck_port*, *remove_redundant_logic*, *unmodeled_bbox*, *lockup_latch*, *no_unate_reconv*, *allow_quasi_static*, *clock_on_ports*, *stop_conv_at_seq_lib*, and *all*.

bbox

Clock domain crossings involving black box instances are ignored.

output_not_used

Clock domain crossings with destinations having unused and blocked outputs are ignored.

For example, consider the scenario shown in the following figure:

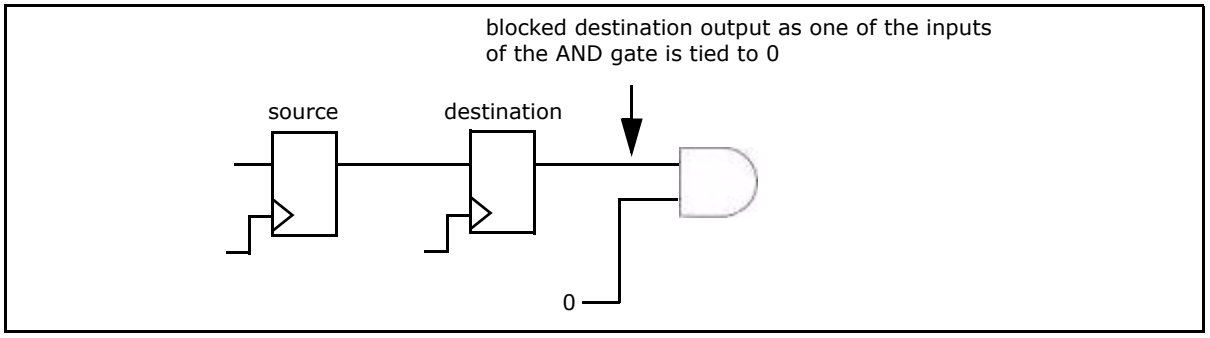


FIGURE 11. Blocked destination output

In the above scenario, if the `cdc_reduce_pessimism` parameter is set to `output_not_used`, the destination output is blocked and the crossing is not reported.

hanging_net

Clock domain crossings with destinations having hanging outputs are ignored.

mbit_macro (default)

Convergence on multi-bit arithmetic macros is not reported. (Traversal beyond the macro is also stopped.)

no_convergence_at_syncreset (default)

Convergence on sync reset is not reported

no_convergence_at_enable (default)

D-En convergence on flip-flop is not reported.

skip_unused_paths

Hanging or blocked paths from destination flip-flops are ignored while checking for multi-flop synchronizers. Any combinational logic is ignored while checking for hanging or blocked paths.

For example, consider the scenario shown in the following figure:

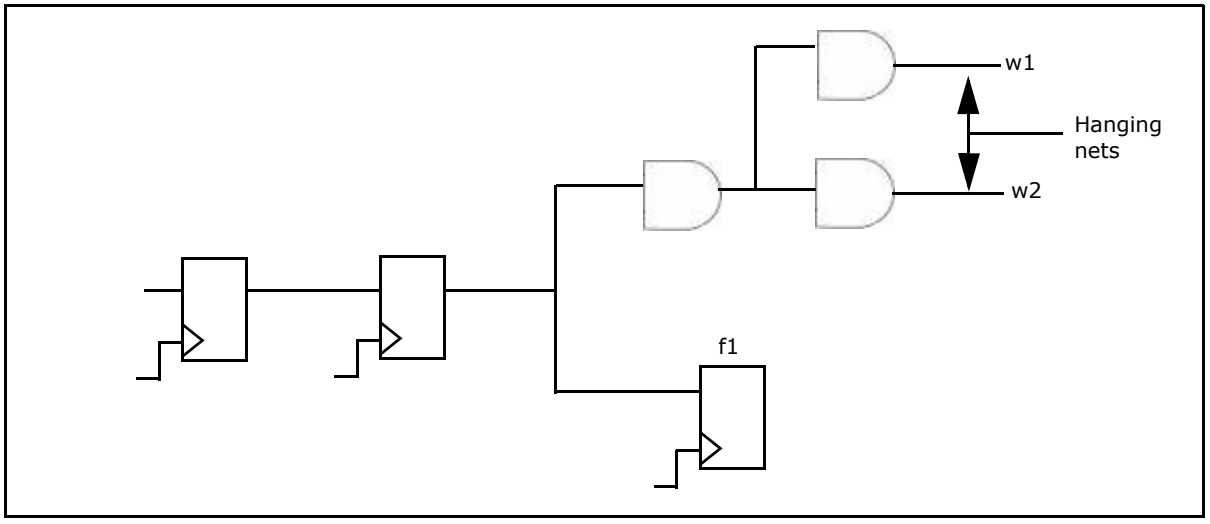


FIGURE 12. Hanging nets from destination flip-flop

In the above scenario, if the `cdc_reduce_pessimism` parameter is set to `skip_unused_paths`, the `w1` and `w2` hanging nets are ignored and the `f1` flip-flop is considered as a synchronizer flip-flop.

ignore_multi_domain

Crossings are reported as synchronized by the *Ac_sync01* and *Ac_sync02* rules when a synchronizer is present in the destination domain and sources from different clock domains converge before reaching the destination domain.

If you do not specify this option, such crossings are reported as unsynchronized by the *Ac_unsync01* and *Ac_unsync02* rules.

For example, the crossing shown in the following figure is considered as synchronized if `ignore_multi_domain` is specified to the `cdc_reduce_pessimism` parameter:

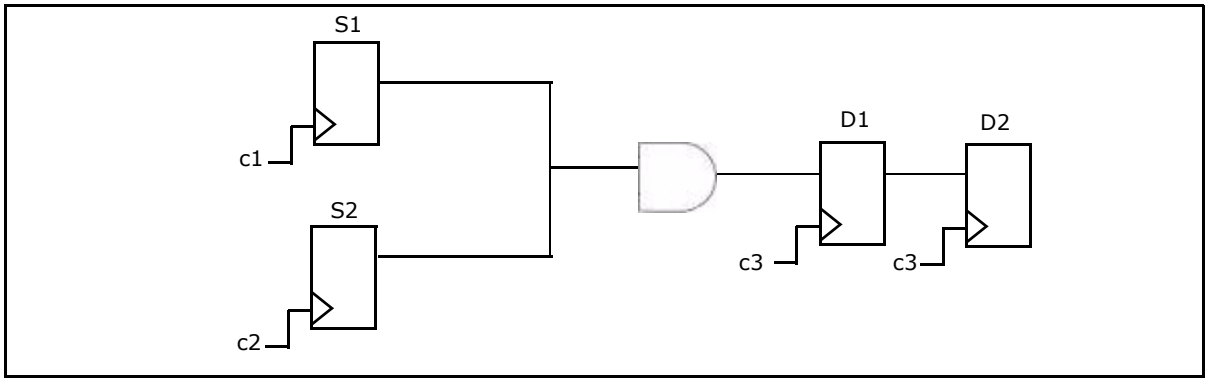


FIGURE 13. cdc_reduce_pessimism set to ignore_multi_domain

In addition, the above structure of *Conventional Multi-Flop Synchronization Scheme* is not allowed to be a valid qualifier for any other data crossing if `ignore_multi_domain` is specified to the `cdc_reduce_pessimism` parameter.

use_multi_arc

Allows multiple related-pin support in complex library cells.

For example, consider the following figure:

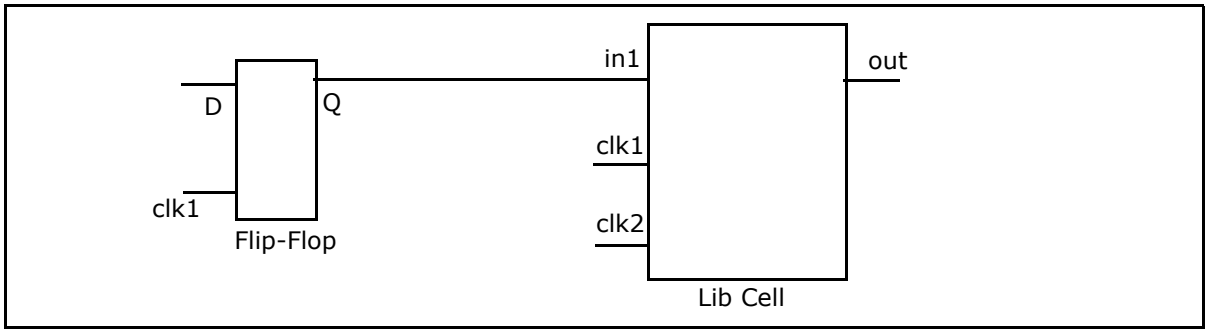


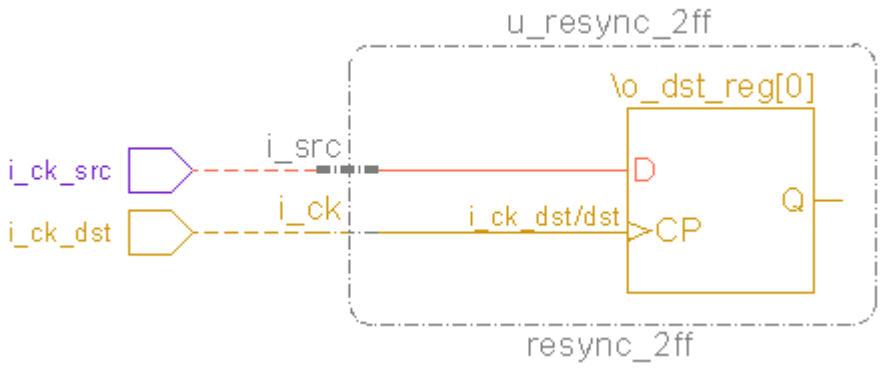
FIGURE 14.

In the above example, the crossing is missed if the related pin of the `in1` input pin is only `clk1`, although `in1` is related to both the `clk1` and `clk2` clock pins.

However, if you specify the `use_multi_arc` value to the *cdc_reduce_pessimism* parameter, a new merged domain is created internally for the `in1` input pin. Therefore, such crossings are detected.

clock_crossing

Detects the crossings for the cases where a defined clock (`i_ck_src` in the figure below) of a domain (`src` in this case) reaches the data pin of a sequential element that is driven by another defined clock (`i_ck_dst` in the figure below) of another domain (`dst` in this case).




```
// SGDC File
clock -name i_ck_src -domain src -period 2 -edge {0 1}
clock -name i_ck_dst -domain dst -period 10 -edge {0 5}
```

FIGURE 15.

const_source

Considers constant sequential elements for SpyGlass CDC analysis.

By default, SpyGlass CDC analysis is not performed on the data pin of a sequential element if that pin is tied to a constant value or is driven by the *set_case_analysis* constraint.

For the synchronized crossings containing such elements, the *Ac_sync01* and *Ac_sync02* reports the method "`Source is constant`" in the violation message and spreadsheet. For details, see *Constant Source Flop Synchronization Scheme*.

glitch_on_vck_port

Enables glitch checking by the *Ac_glitch03* rule on unsynchronized crossings where:

- The destination port has a virtual domain.

- The destination-port type is specified as `control` by using the *signal_type* constraint.

Figure 236 shows the example of such crossing.

remove_redundant_logic

Specify this option to:

- Ignore the reporting of the *Ac_conv01*, *Ac_conv02*, *Ac_conv03* and *Clock_glitch05* rule violations on converging nets if they are hanging.
- Ignore the crossings (synchronized or unsynchronized) containing hanging destinations.

  The following table describes the types of hanging destinations:

Hanging Destination Type	Description
Sequential cell	A sequential cell is considered as a hanging destination if: • All the outputs related to the input pin of the cell are hanging. • The output is related to the input pin through timing arc or functional arc. • The input pin makes a crossing with the source pin.
Black box	A sequential black box is considered as a hanging destination if: • No abstract port definition is present on the black box. • No *assume_path* constraint is defined on the black-box pin that makes a crossing. • All the outputs of the black box are hanging.
Flip-flop, latch, or a synchronous cell	A Flip-flop, latch, or a synchronous cell is considered as a hanging destination if its output is hanging.

The following rules are impacted if you specify the `remove_redundant_logic` option to the *cdc_reduce_pessimism* parameter.

Ac_conv01, *Ac_conv02*, *Ac_conv03*, *Ac_unsync01*, *Ac_unsync02*, *Ac_sync01*, *Ac_sync02*, *Ac_datahold01a*, *Ar_sync01*, *Ar_unsync01*, *Ar_asyncdeassert01*,

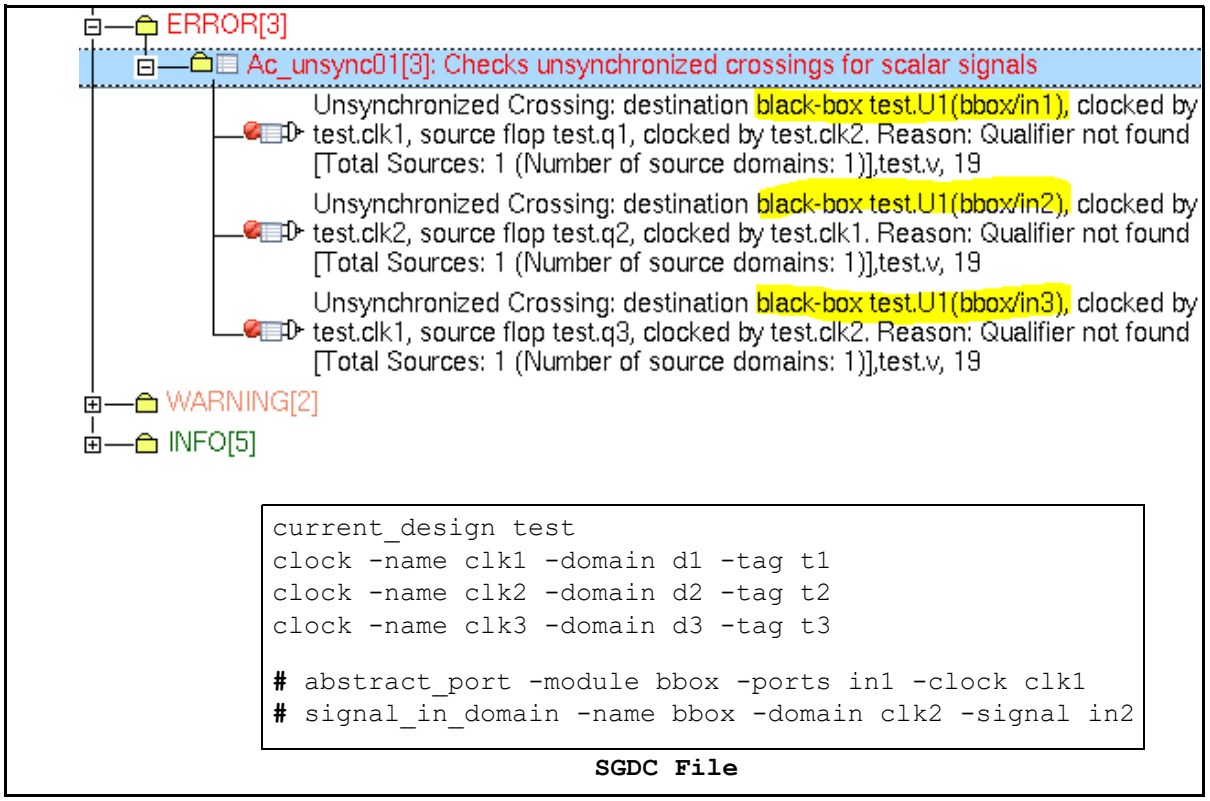
Ar_syncdeassert01, *Clock_sync05*, *Reset_check12*, *Clock_glitch05*, and *Reset_sync02*.

unmodeled_bbox

Suppresses *The Ac_sync_group Rules* violations for the crossings involving un-modeled black box pins.

A black box pin is considered as un-modeled if no SGDC constraint is defined on it.

For example, consider the following scenario in which the *Ac_unsync01* rule reports three violations for the crossings involving un-modeled black box pin `bbox`:




```
current_design test
clock -name clk1 -domain d1 -tag t1
clock -name clk2 -domain d2 -tag t2
clock -name clk3 -domain d3 -tag t3

# abstract_port -module bbox -ports in1 -clock clk1
# signal_in_domain -name bbox -domain clk2 -signal in2
```

SGDC File

FIGURE 16.

To suppress the above violations, set the *cdc_reduce_pessimism* parameter to `unmodeled_bbox`. This way, none of *The Ac_sync_group Rules* will report any violation for the crossings involving `bbox`.

lockup_latch

If a destination drives the path having a lockup latch that makes a crossing unsynchronized, you can ignore such path by specifying the `lockup_latch` value to the *cdc_reduce_pessimism* parameter.

For example, consider the following figure:

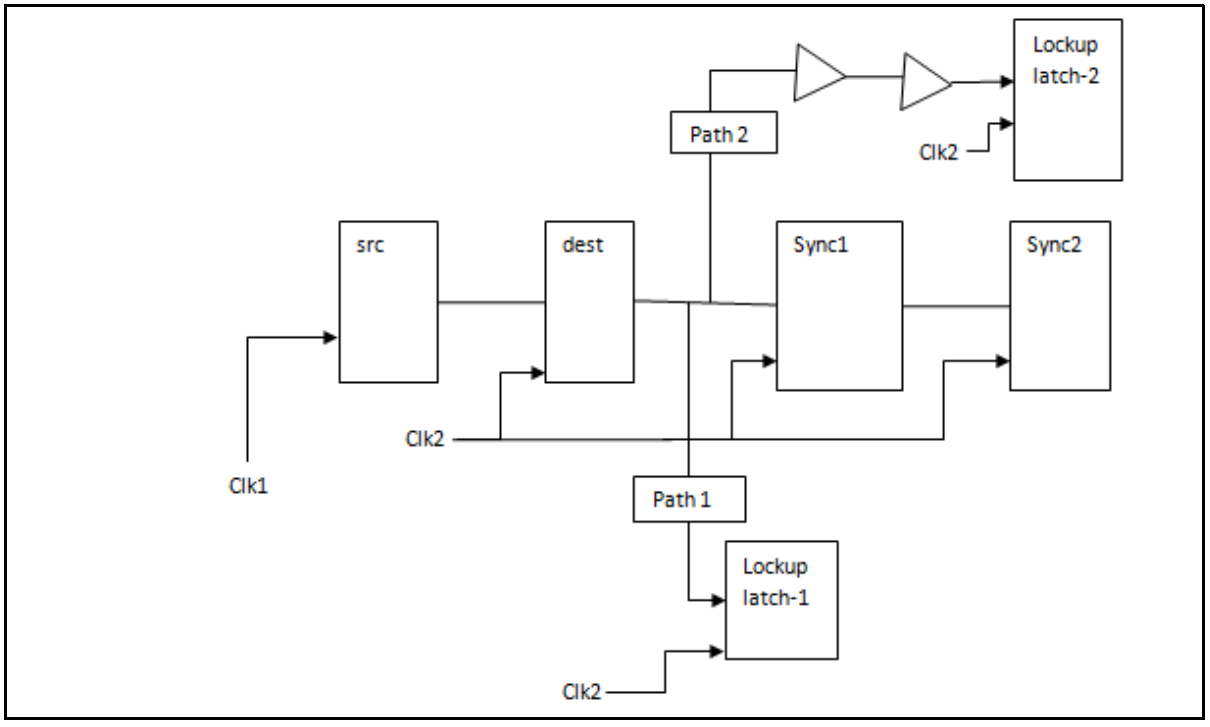


FIGURE 17.

In the above figure, the destination has multiple fan-outs having the following paths:

- `Path1` that is directly connected to a lockup latch

- `Path2` that is connected to a lockup latch through some allowable combinational logic

In the above case, specify the `lockup_latch` value to the *cdc_reduce_pessimism* parameter to suppress the *Ac_sync01* violation indicating that the destination is driving multiple paths.

For the multi-flop synchronizers, the *Ar_sync01* rule treats the lockup latch paths in the same way as they are treated by *The Ac_sync_group Rules*.

However, the above rules report a violation if the destination drives another path (say `Path3`) that is directly connected to another flip-flop or a non-allowable combinational logic.

no_unate_reconv

Specify the `no_unate_reconv` value to the *cdc_reduce_pessimism* parameter to configure the *Ac_glitch03*, *Clock_converge01*, and *Ar_converge01* rules to report violations related with same source reconvergence when the reconverging paths have different polarities or at least one path has an unknown polarity. This is explained in the following topics:

- *Case 1 - Reconverging Paths with Different Polarities*
- *Case 2 - At Least One Reconverging Path with an Unknown Polarity*
- *Case 3 - Reconverging Paths with the Same Polarity*

For information on determining the polarities of reconverging paths, see *Inferring Path Polarities After Same Source Reconvergence*.

Case 1 - Reconverging Paths with Different Polarities

Consider the following circuit for which the *Ar_converge01* rule reports a violation:

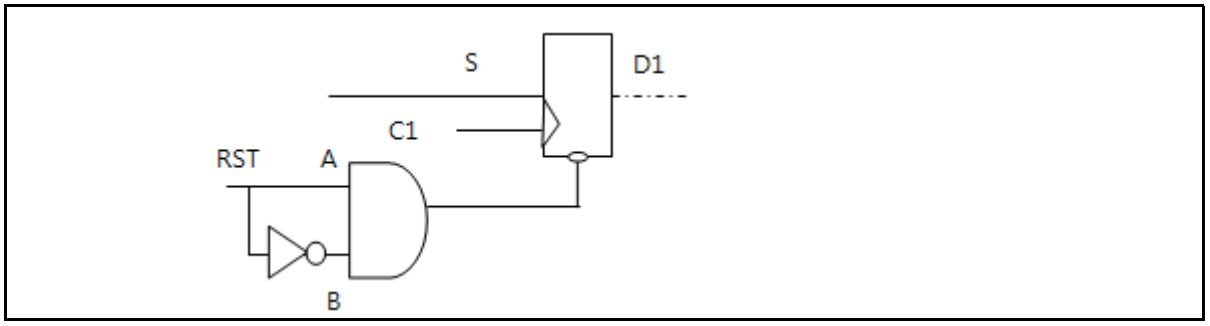


FIGURE 18. Example of Reconverging Paths with Different Polarities

In the above circuit, the reconverging paths A and B have different polarities. This is explained below.

Initially, consider RST is 0. Therefore:

- A of AND gate = 0, B of AND gate = 1
- Q of AND gate will be 0

When RST changes from 0 to 1:

- Due to different delay in the inverter path, the value 1 may not reach the B input of the AND gate at the same time as that of A.
- The output of AND gate is 1 as the intermediate value.
- When 0 reaches the B input, the output becomes 0.

This implies that the output of AND will be 010 whereas it was never expected to be 1.

NOTE: *For information on the cases in which a reconverging path attains an inverted polarity, see Inferring Path Polarities After Same Source Reconvergence.*

Case 2 - At Least One Reconverging Path with an Unknown Polarity

Consider the following circuit for which the *Ar_converge01* rule reports a violation:

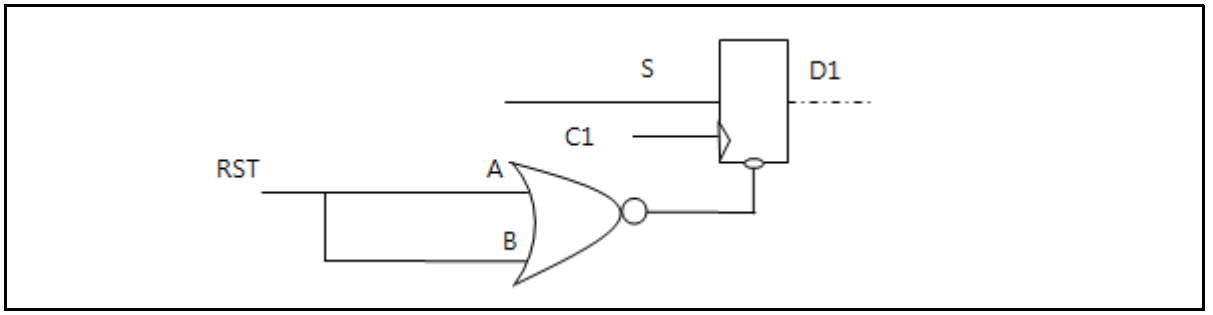


FIGURE 19.

In the above circuit, an unknown polarity reaches the path A because of some logic within the black box. Therefore, a violation is reported for such reconvergence.

NOTE: *For information on the cases in which a reconverging path attains an unknown polarity, see Inferring Path Polarities After Same Source Reconvergence.*

Case 3 - Reconverging Paths with the Same Polarity

Consider the following circuit for which the *Ar_converge01* rule does NOT report a violation:

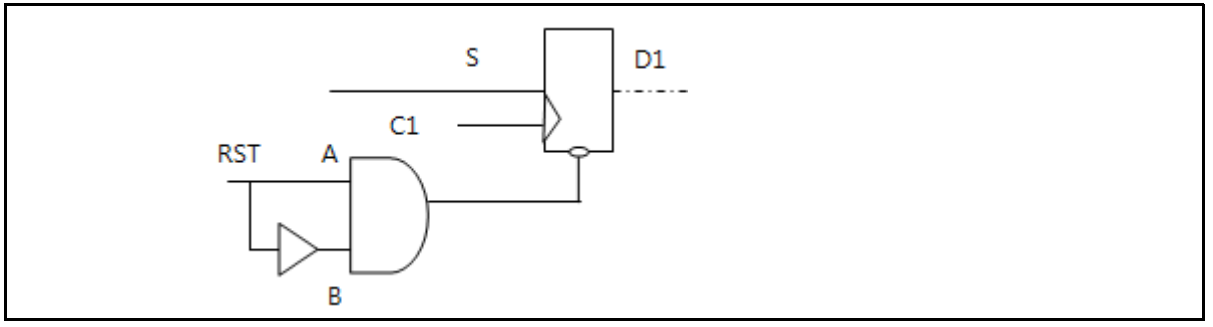


FIGURE 20. Example of Reconverging Paths with Same Polarities

In the above circuit, the reconverging paths A and B have the same polarity. This is explained below.

If RST changes from 0 to 1 then:

- Due to different delay in the buffer path, the value 1 may not reach the B input of the AND gate at the same time as that of A.
- The output of AND gate is 0 as the intermediate value.
- When 1 reaches to the B input, the output becomes 1.

This implies that the output of AND will be 001, which is a glitch-free output because the intermediate output does not take an unexpected value.

NOTE: *For information on the cases in which a reconverging path attains a non- inverted polarity, see Inferring Path Polarities After Same Source Reconvergence.*

allow_quasi_static

Specify this value to the *cdc_reduce_pessimism* parameter to allow combinational logic between synchronizer flip-flops if this combinational logic is a probable buffer path, that is one of the inputs of this logic is quasi static.

Consider the following schematic:

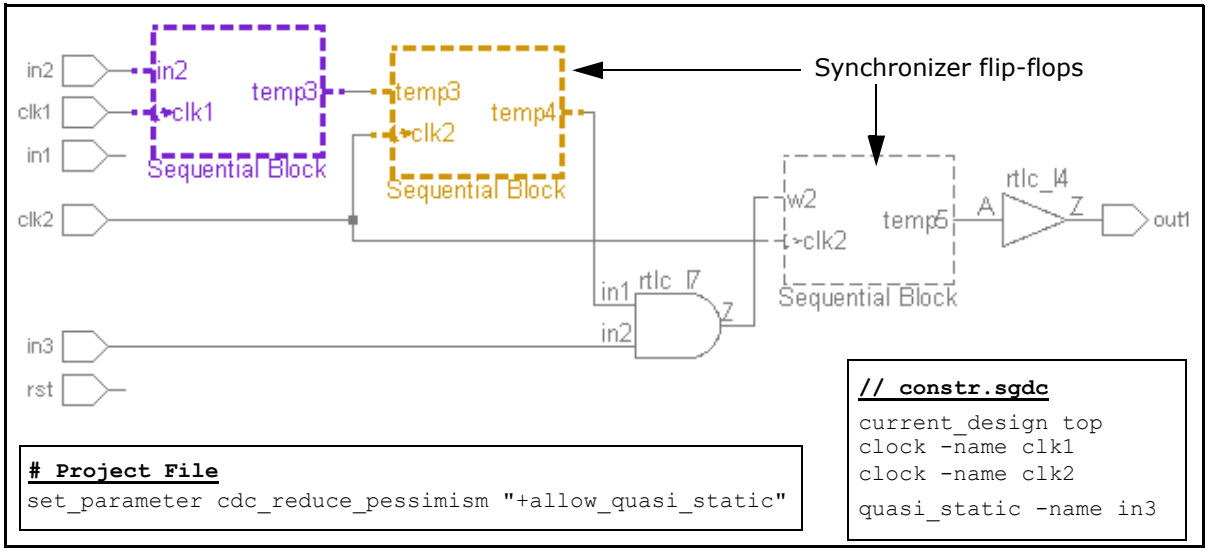


FIGURE 21.

In the above schematic, the `in3` input of the combinational logic between the synchronizer flips-flops is quasi static. Since the *cdc_reduce_pessimism* parameter is set to `allow_quasi_static`, *Ac_unsync01* does not report any unsynchronized crossing between the source and destination flip-flops.

clock_on_ports

Specify this value to the *cdc_reduce_pessimism* parameter to enable the *Clock_info05* and *Clock_info16* rules to report clock convergence on a mux if the output of the mux is captured by a port.

stop_conv_at_seq_lib

Specify the `stop_conv_at_seq_lib` value to the *cdc_reduce_pessimism* parameter to stop synchronizer propagation across sequential library cells in rules Ac_conv01 and Ac_conv03.

all

Considers all the *Allowed Values of the cdc_reduce_pessimism Parameter*.

Inferring Path Polarities After Same Source Reconvergence

The following table describes how the polarities of reconverging paths are determined:

Cases when polarity is inverting	Cases when polarity is non-inverting	Cases when polarity is unknown
Odd number of inverter, NAND, NOR gates (could be inferred from RTL or a lib cell)	Even number of inverter, NAND, or NOR gates	Black box pins specified without *abstract_port -path_logic <buf | inv>*
Any library cell (AND, OR, MUX, ANDOR, ORAND, buffer, and tristate) where the input pin is inverted	AND, OR, MUX, buffer, or tri-state	Library cells without functional arc

Cases when polarity is inverting	Cases when polarity is non-inverting	Cases when polarity is unknown
XOR for which another pin is defined as 1	XOR for which another pin is defined as 0	RTL macros, such as adder and comparator
XNOR for which another pin is defined as 0	XNOR for which another pin is defined as 1	XOR/XNOR without any constant pin
Library cells AOI (And-Or-inverter) or OAI (Or-And-inverter)	ANDOR or ORAND library cells. These are the cells that have only AND and OR gate.	A combinational or sequential loop
Black box or abstract block specified with *abstract_port -path_logic inv*	Sequential elements	Any combinational or sequential library cell that is not mentioned in the first two columns. For example, such cells could be a half adder or an arithmetic cell.
Inverted sequential element, such as CGC and PAD cell	Clock gating cells	
A library cell for which the inversion on the input/output is interpreted and taken into account during polarity calculation	PAD cells	
	Black box or abstract block specified with *abstract_port -path_logic buf*	
	A library cell for which the inversion on the input/output is interpreted and taken into account during polarity calculation	

check_bus_bit_convergence

Specifies whether bus signals should be checked for convergence by the *Clock_sync03a* rule.

By default, bus signals are not checked for convergence. Set the `check_bus_bit_convergence` parameter to `yes` to report converging bus signals.

Used by	*Clock_sync03a*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter check_bus_bit_convergence yes`
Usage in goal/source files	`-check_bus_bit_convergence=yes`

check_edge

Configures the *Ac_clockperiod01* rule to report a violation based on whether the `-period` or `-edge` argument of the *clock* constraint is specified.

By default, this parameter is set to `yes` and the *Ac_clockperiod01* rule reports a violation if any of the `-period` or `-edge` argument is not specified.

Set this parameter to `no` such that the *Ac_clockperiod01* rule reports a violation if none of the `-period` or `-edge` argument is specified.

Used by	*Ac_clockperiod01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter check_edge no`
Usage in goal/source files	`-check_edge=no`

check_input_coverage

Configures the *Clock_info18* rule to report a violation only for input ports of the top-level design unit.

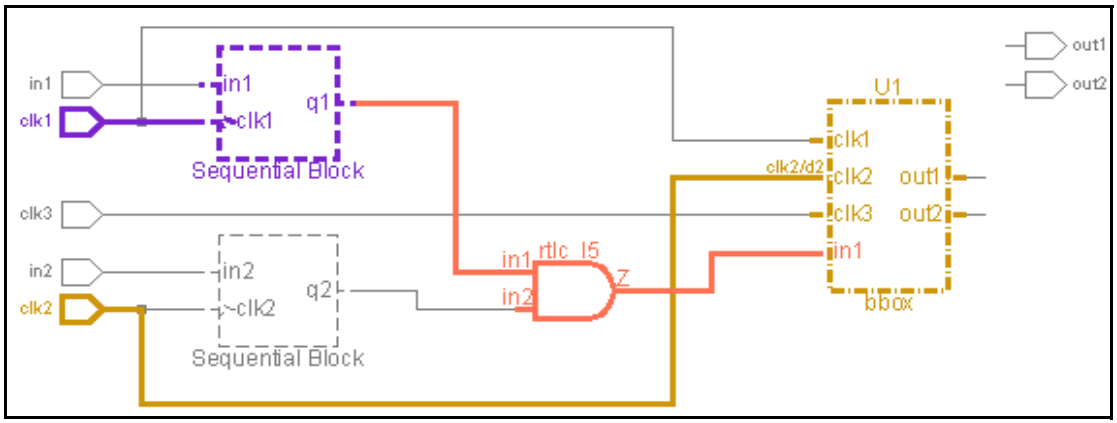
By default, this argument is set to `no` and this rule reports violations for unconstrained input and output ports in the top-level design.

Used by	*Clock_info18*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter check_input_coverage yes`
Usage in goal/source files	`-check_input_coverage=yes`

check_multiclock_bbox

Shows the violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.

For example, consider the following scenario in which the *Ac_unsync01* rule reports the crossing in which the destination black box (bbox) does not have any SGDC constraint specified on any of its data pins:




```
current_design test
clock -name clk1 -domain d1 -tag T1
clock -name clk2 -domain d2 -tag T2
clock -name clk3 -domain d3 -tag T3

# Constraints on the destination clock pins are commented out
#signal_in_domain -name bbox -domain clk1 -signal out1
#signal_in_domain -name bbox -domain clk2 -signal out2
#signal_in_domain -name bbox -domain clk1 -signal in1
```

SGDC File

FIGURE 22.

To show the above violation in which none of the data pins (in1) of the destination black box (bbox) is constrained, set this parameter to yes.

Used by	*Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, *Ac_unsync02*, *Ac_glitch03*, *Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Setup_req01*, *Ac_crossing01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter check_multiclock_bbox yes`
Usage in goal/source files	`-check_multiclock_bbox=yes`

check_single_source

Enables the *Ac_glitch02* to consider all the sources of a destination for rule checking.

By default, this rule considers one source per destination.

NOTE: *SpyGlass considers this parameter only when the cdc_compatible parameter is set to* no*.*

Used by	*Ac_glitch02*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter check_single_source no`
Usage in goal/source files	`-check_single_source=no`

check_port_setup

Configures the *Setup_port01* rule to check the constraining of input, inout, or output or all ports.

By default, the *Setup_port01* rule checks constraining of input and inout ports for top-design units. Use this parameter to configure the rule to perform the checking on input, inout, or output or all ports. Inout ports are always checked, regardless of the value of this parameter.

Used by	*Setup_port01*
Options	input|output|all
Default value	input
Example	
Console/Tcl-based usage	`set_parameter check_port_setup output`
Usage in goal/source files	`-check_port_setup=output`

check_reset_for_constclock

Specifies if the *Clock_info03c* rule checks for set and preset pins on flop whose clock pin is tied to constant value.

Used by	*Clock_info03c*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter check_reset_for_constclock yes`
Usage in goal/source files	`-check_reset_for_constclock=yes`

check_qualified_signal_at_soc

Configures the *Ac_abstract_validation01* rule to not report data-mismatch violations if a qualified signal reaches to abstract block input having same domain as the destination domain.

By default, the parameter is set to `no`. Set the parameter to `yes` to not report such data-mismatch violations.

In addition, if the parameter is set to `yes`, the *Ac_abstract01* rule reports `abstract_port -sync_inactive` for the qualified signal if a qualified signal reaches to output port.

Used by	*Ac_abstract_validation01*, *Ac_abstract01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter check_qualified_signal_at_soc yes`
Usage in goal/source files	`-check_qualified_signal_at_soc=yes`

clock_edge

Sets the edge type for the *Clock_check04* rule.

By default, the Clock_check04 rule expects the positive edge of clocks to be used in a design and reports all clock descriptions that have the negative edge specified. Set the `clock_edge` parameter to `negative` to report clock descriptions that have the positive edge specified.

NOTE: *Only one instance is reported for each clock. For example, if clk1 uses negative edge at several places, only one instance is reported.*

Used by	*Clock_check04*
Options	positive, negative
Default value	positive
Example	
Console/Tcl-based usage	`set_parameter clock_edge negative`
Usage in goal/source files	`-clock_edge=negative`

clock_fanout_max

Specifies the maximum fan-out limit of clocks in a design for the *Clock_check02* rule.

By default, the Clock_check02 rule reports clock nets that have a fan-out greater than 24 and are not driven by instances of cells specified by the *CTS_placeholder_cells* parameter.

You can change this fan-out limit as per your requirement.

NOTE: *The Clock_check02 rule is run only when placeholder cells are specified using CTS_placeholder_cells parameter.*

Used by	*Clock_check02*
Options	Positive integer value
Default value	24
Example	
Console/Tcl-based usage	`set_parameter clock_fanout_max 10`
Usage in goal/source files	`-clock_fanout_max=10`

clock_gate_cell

Specifies the clock-gating cell names for the *Clock-Gating Cell Synchronization Scheme*. These cell names exist in the clock path of the destination flip-flops in clock-domain crossings. A crossing is considered to be synchronized if a multi-flop synchronizer exists in the other fan-in (pin not connected to clock) of the clock-gating cell.

NOTE: *If the Clock-Gating Cell Synchronization Scheme is disabled using the enable_clock_gate_sync parameter, these clock gate cells will not be considered.*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Clock_hier01*, *Clock_hier02*, *Clock_hier03*
Options	Comma or space-separated list of clock-gating cell names. Wildcard characters (* [zero or more] and ? [single character match]) can be used in cell names.
Default value	NULL
Example	

Console/Tcl-based usage	`set_parameter clock_gate_cell CG01`
	
	You can specify multiple clock cell names as shown below:
	`set_parameter clock_gate_cell "CG01,CG02"`
	
	The following specification matches all cell names that start with CG:
	`set_parameter clock_gate_cell CG*`
	
	The following specification matches all three character cell names that start with CG:
	`set_parameter clock_gate_cell CG?`

Usage in goal/source files	`-clock_gate_cell=CG01,CG02`

Set the `enable_mux_dest_domain` parameter to eliminate the need

for a synchronizer where the clock-gating cell driving the first flip-flop in the destination domain has the other input pin (not connected to the clock) in the destination domain.

clock_reduce_pessimism

Use this parameter to:

- *Control Clock Propagation through MUX Select and Latch Enable Pin*
- *Control Clock Propagation from the Output of a Derived Flip-Flop Through a MUX Select Pin*
- *Infer Clocks in a Design*
- *Include the generated_clock Constraints for all the Master Clocks*
- *Check the Enable Condition of a CGC Cell*
- *Suppress Clock_info05 Violations for Converging Clocks of the Same Domain*
- *Control Clock propagation from the Output of a Flip-Flop*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Clock_check04*, *Clock_check05*, *Clock_glitch02*, *Clock_glitch03*, *Clock_converge01*, *Propagate_Clocks*, *Clock_info05*, *Clock_info16*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Clock_info03a* and *Ac_unsync01*
Options	latch_en, mux_sel, mux_sel_derived, all, all_potential_clocks, all_master_clocks, check_enable_for_glitch, single_input_output_bbox, stop_derived_at_random_logic, and ignore_same_domain You can specify multiple values through a comma-separated list. Alternatively, use the + (plus) character to append a value to the default value.
Default value	latch_en, check_enable_for_glitch, mux_sel_derived
Example	

Console/Tcl-based usage	Use any of the following specifications to disable clock propagation through both the latch enable pin and the MUX select pin:
	`set_parameter clock_reduce_pessimism 'all'`
	`set_parameter clock_reduce_pessimism 'latch_en,mux_sel'`
	`set_parameter clock_reduce_pessimism '+mux_sel'`
	
	(The + character must be the first character when specified. Therefore, `'mux_sel+'` is incorrect.)
	
	Note: To enable clock propagation through both the MUX select pin and the latch enable pin, set this parameter as a null value, as shown below:
	`set_parameter clock_reduce_pessimism ''`

Usage in goal/source files	`-clock_reduce_pessimism='all'`

Control Clock Propagation through MUX Select and Latch Enable Pin

To enable or disable clock propagation through MUX select pins and latch enable pins, specify any of the values to this parameter:

Allowed value	Clock propagation through latch enable pin	Clock propagation through MUX select pin
latch_en (default)	Disabled	Enabled
mux_sel	Enabled	Disabled
all	Disabled	Disabled

NOTE: *Clock propagation continues through the MUX select pin when the MUX inputs are tied to a constant value through the set_case_analysis constraint irrespective of any value specified to the clock_reduce_pessimism parameter.*

Control Clock Propagation from the Output of a Derived Flip-Flop Through a MUX Select Pin

To disable clock propagation from the output of a derived flip-flop through

the MUX select pin, specify the `mux_sel_derived` value to the *clock_reduce_pessimism* parameter.

By default, this value is already set for the *clock_reduce_pessimism* parameter, and therefore, clock propagation is disabled by default.

Consider the following figure:

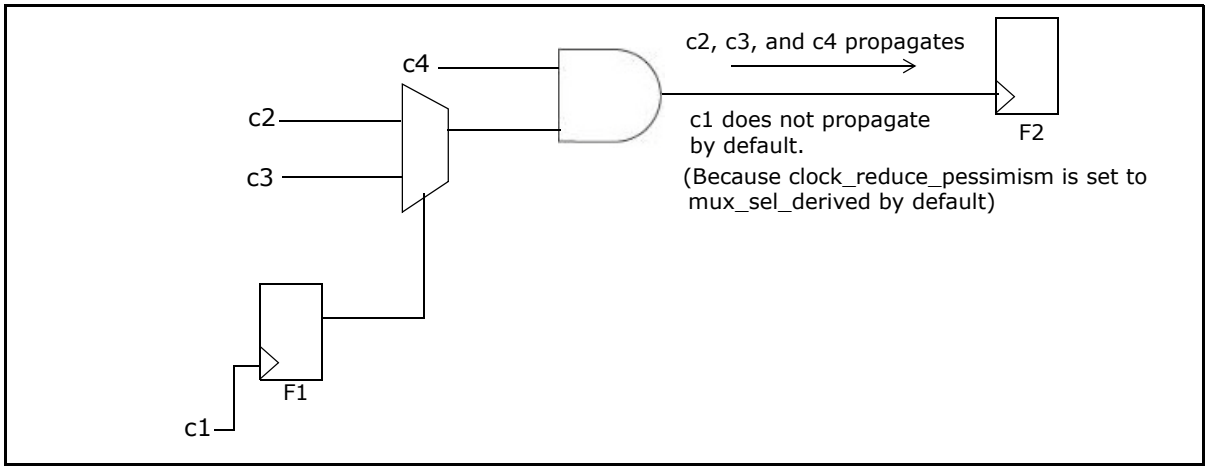


FIGURE 23.

In the above figure, the `c2`, `c3`, and `c4` clocks reach the `F2` flip-flop. However, the `c1` clock is not propagated through the MUX select pin because the default value of the *clock_reduce_pessimism* parameter is `mux_sel_derived`.

To enable the propagation of the `c1` clock in addition to the `c2`, `c3`, and `c4` clocks, remove the value `mux_sel_derived` from the `clock_reduce_pessimism` parameter.

Infer Clocks in a Design

To infer the clocks other than the clocks inferred by setting the *use_inferred_clocks* parameter to `yes` or running the *Clock_info01* rule:

- Specify the `all_potential_clocks` value to the *clock_reduce_pessimism* parameter.

- Set the value of the *use_inferred_clocks* parameter to `yes` or run the *Clock_info01* rule.

When you set the above values, then while inferring clocks, if SpyGlass encounters a two-input gate in which one of the input is a definite clock, it infers that clock as well as considers the other input to infer more clocks.

For example, consider the following figure:

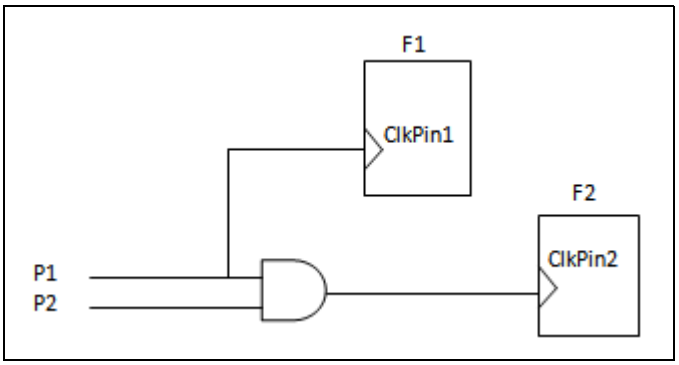


FIGURE 24. Input of the AND Gate is a Definite Clock

In the above figure, one of the inputs (`P1`) of the AND gate is a definite clock. Therefore, SpyGlass infers this clock. However, to also infer `P2` as a clock, set this parameter to `all_potential_clocks`.

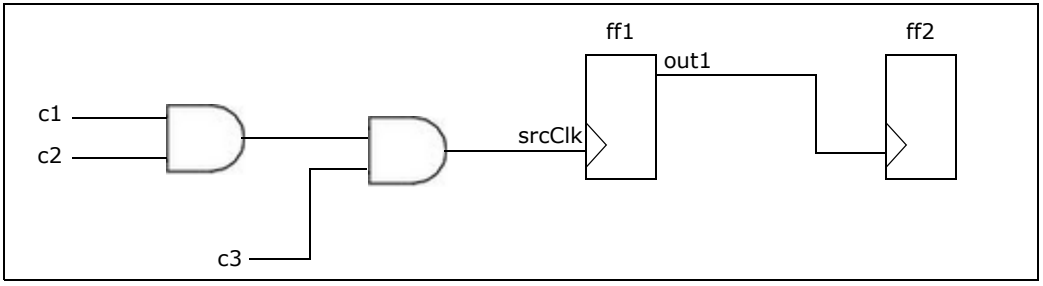
Include the generated_clock Constraints for all the Master Clocks

To include the *generated_clock* constraints for derived clocks with respect to all the master clocks reaching the source of derived clocks, set the *clock_reduce_pessimism* parameter to `all_master_clocks`.

By default, the *generated_clock* constraint for a derived clock is included with respect to any one master clock reaching to the source of the derived clock.

These constraints are included in the *generated_clocks.sgdc* and *cdc_setup_generated_clocks.sgdc* files when the *use_inferred_clocks* parameter is set to `yes`.

For example, consider the following figure in which multiple master clocks, `c1`, `c2`, and `c3` reach the source clock `srcClk`:




```
// SGDC File:
clock -name c1 -period 15 -domain d1 -tag T1
clock -name c2 -period 15 -domain d2 -tag T2
clock -name c3 -period 10 -domain d3 -tag T3
```

FIGURE 25. Multiple master clocks reaching a source clock

In the above scenario, if you set the *clock_reduce_pessimism* parameter to `all_master_clocks`, the *generated_clock* constraints are dumped with respect to all the master clocks, as shown below:

```
generated_clock -name out1 -source srcClk -master_clock T1
-divide_by 2 -tag GT1 -add

generated_clock -name out1 -source srcClk -master_clock T2
 -divide_by 2 -tag GT2 -add

generated_clock -name out1 -source srcClk -master_clock T3
 -divide_by 2 -tag GT3 -add
```

Check the Enable Condition of a CGC Cell

To enable the *Clock_glitch02* rule to check for the enable condition of a CGC cell, specify the `check_enable_for_glitch` value to the *clock_reduce_pessimism* parameter.

If you do not specify this value, only structure correctness of a CGC cell is checked and the enable condition is not checked.

Suppress Clock_info05 Violations for Converging Clocks of the Same

Domain

To suppress *Clock_info05* violations for the same domain clock signals that converge on a MUX when no *set_case_analysis* constraint is applied on the MUX select pin, specify the `ignore_same_domain` value to the *clock_reduce_pessimism* parameter.

Detect the Output of the Single-Input-Output Black Box As a Clock

By default, the output of the Single-Input-Single-Output pin (SISO) black box is not considered as a clock. Only the output of the Multiple-Input-Output-pin black box is considered as a clock.

To also consider the output of a SISO black box as a clock, set the *clock_reduce_pessimism* parameter to `single_input_output_bbox`.

For example, consider the following figure:

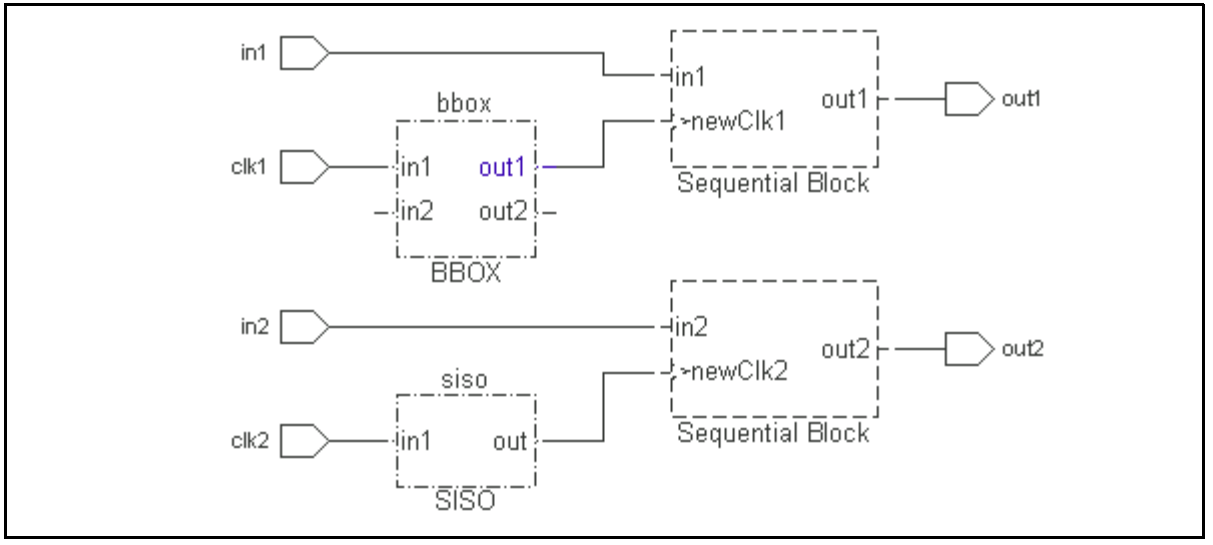


FIGURE 26.

In the above scenario, if you do not set the *clock_reduce_pessimism* parameter to `single_input_output_bbox`, the *Clock_info01* rule reports the `BBOX` output (`newClk1`) as the clock candidate.

However, if you set *clock_reduce_pessimism* parameter to

single_input_output_bbox, the *Clock_info01* rule reports the SISO output (newClk2) as the clock candidate (in addition to newClk1).

In this case, if you set the *use_inferred_clocks* parameter to yes, the *Propagate_Clocks* message appears indicating that the newClk2 clock is propagated.

However, if the input of the SISO black box is a clock (clk2 in *Figure 26*), *Clock_check07* message appears indicating that the clk2 clock has reached the clock of another domain, and therefore, halting its propagation.

Control Clock propagation from the Output of a Flip-Flop

To disable clock-propagation beyond a flip-flop that the clock reaches (except under conditions described below), specify the stop_derived_at_random_logic value to the clock_reduce_pessimism parameter.

By default, this value is NOT set for the clock_reduce_pessimism parameter.

The stop_derived_at_random_logic value impacts clock propagation and analysis as follows:

- Output of a sequential element is considered as derived clock only when it reaches the clock pin of other sequential elements directly or through the following gates:
  - ❒ Clock Gating Cell
  - ❒ Inputs of a MUX
  - ❒ Buffers and Inverters

  Propagation is halted for all other combinational logic. See *Examples*.
- In case of multiple paths, if the output of a sequential element (F1) reaches the clock pin in at least one path (F3) directly or through above mentioned gates, the output of the sequential element is considered a derived clock and the clock is propagated to all the sequential elements (F2 & F3) by skipping all the gates.

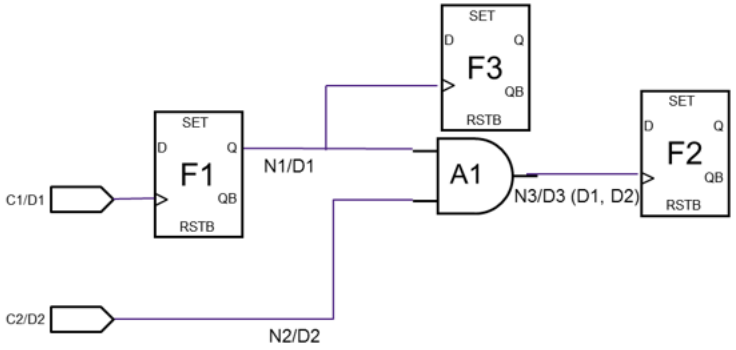


FIGURE 27.

- If any of the `disable_seq_clock_prop`, `sdc_generated_clocks`, or `enable_generated_clocks` parameters are enabled, then primary clocks are not propagated beyond sequential elements. Specify the generated_clock constraint on sequential outputs so that the generated_clocks are propagated irrespective of the `stop_derived_at_random_logic` switch.

Examples

Example 1 - Only CLK2/D2 propagates forward. CLK1/D1 is stopped at the flop.

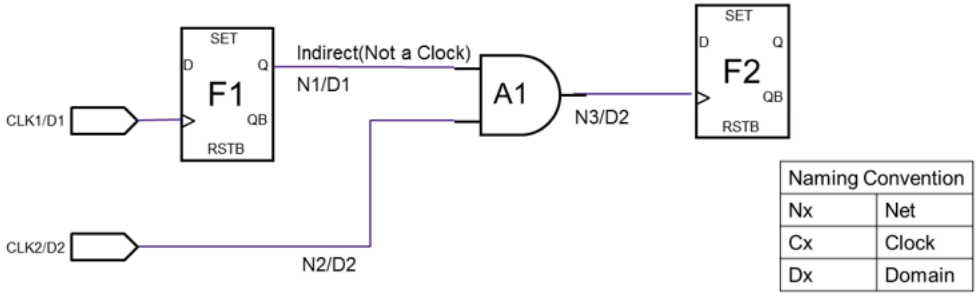


FIGURE 28.

Example 2 - Forward Propagation of two indirect clocks. Clock_info03a

flags flop(F3) not receiving the clock.

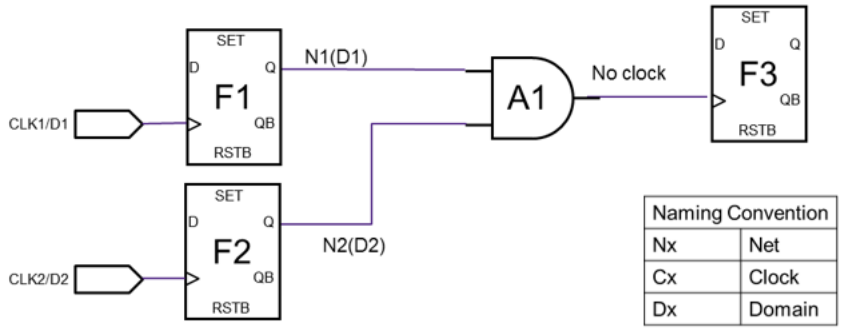


FIGURE 29.

Example 3 - N1(D1), N2(D2) and CLK3(D3) is propagated by default. You can stop CLK3(D3) in this case by using the following parameter:

```
set_parameter clock_reduce_pessimism +mux_sel
```

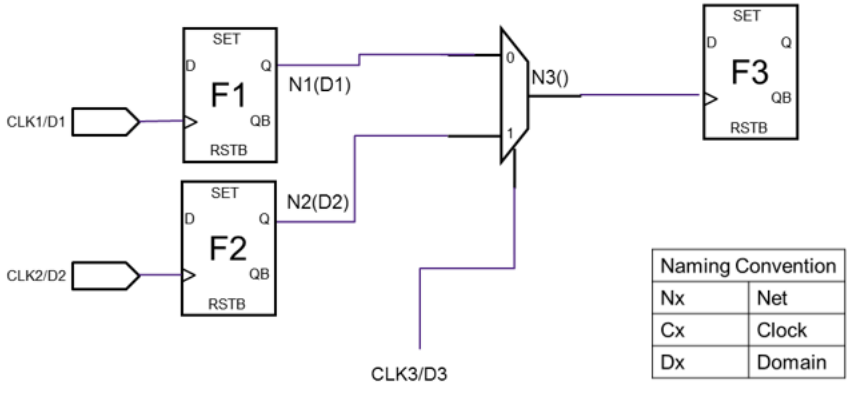


FIGURE 30.

Example 4 - All clocks, CLK1, CLK2, and CLK3, are propagated by default. You can stop propagation of clock on MUX-select (CLK3) by using the following parameter:

```
set_parameter clock_reduce_pessimism +mux_sel
```

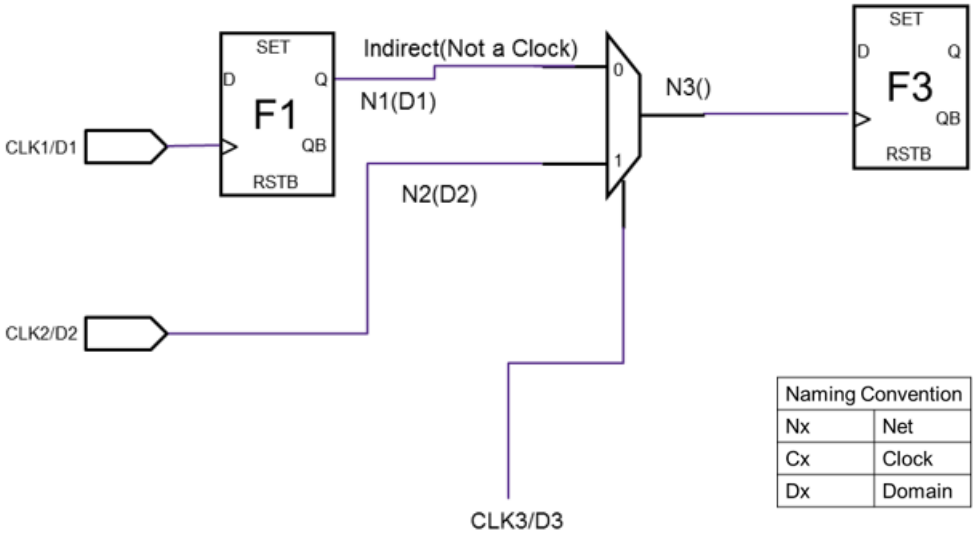


FIGURE 31.

Example 5 - Forward Propagation of 2 indirect clocks through Clock Gating Cell. CLK2/D2 propagates while CLK1 does not propagate

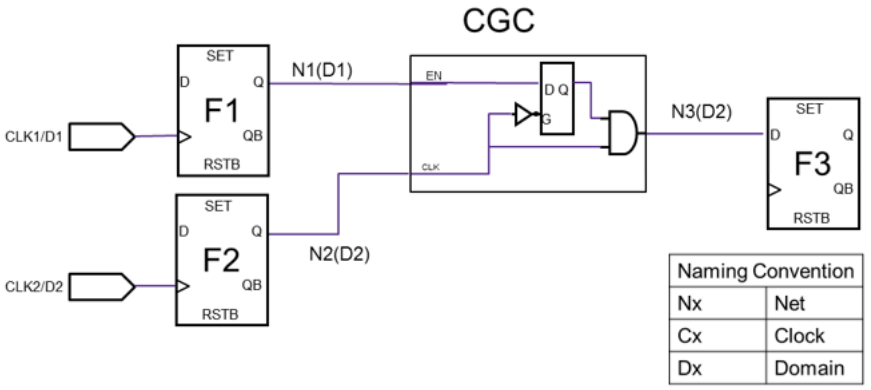


FIGURE 32.

Example 6 - Generated clocks C1/D1 and C2/D2 propagating through MUX but stopped when merging with direct clock C3/D3.

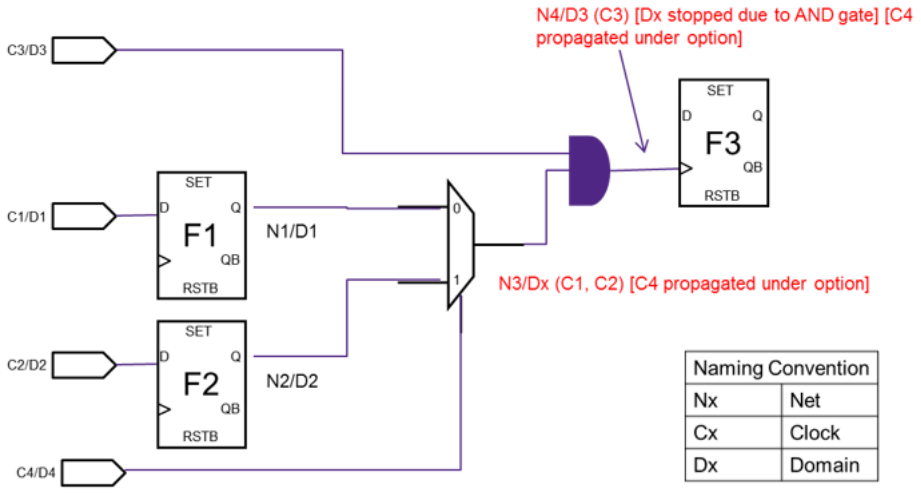


FIGURE 33.

clock_ripple_depth

Sets the maximum allowed ripple clock-divider depth for the *Clock_check05* rule.

By default, the *Clock_check05* rule reports ripple clock-dividers that are at least two levels deep. Use the `clock_ripple_depth` parameter to set a different number.

Used by	*Clock_check05*
Options	Positive integer value
Default value	2
Example	
Console/Tcl-based usage	`set_parameter clock_ripple_depth 4` In this case, the clock_ripple_depth parameter reports ripple clock-dividers with 5 or more flip-flops.
Usage in goal/source files	`-clock_ripple_depth=4`

clock_usage

Specifies the signal types to be reported for non-clock usage by the *Clock_check10* rule.

The `clock_usage` parameter can take the following values:

Value	Indicates
data	Clock signals used as data signals at flip-flops, latches, and sequential elements should be reported.
control	Clock signals used as control signals at flip-flops, tristates, and sequential elements should be reported.
reset	Clock signals used as reset signals at flip-flops, latches, and sequential elements should be reported.
port	Clock signals reaching to primary ports should be reported.
bbox	Clock signals reaching to input of black boxes that have *abstract_port* defined, should be reported.
others	Clock signals reaching to library cell inputs of type `others` (types excluding `clock`, `control`, and `reset`) should be reported. For example, the read address of a memory cell.
derived	Clock signals will be propagated further from derived flip-flops or latch enable pin (if the `clock_reduce_pessimism` parameter is set to `mux_sel`) and a violation is reported if the clock signals are used as non-clock signals. **Note**: This parameter must be specified in combination with at least one of the above values.
all	All the above types of clock signals should be reported.

By default, the `clock_usage` parameter is set to the following value:

```
data,control,reset,bbox,others
```

The above value specifies that the *Clock_check10* rule should report the clock signals that are used as data, control, reset, black box and other signals at flip-flops and sequential elements. You can set this parameter to a comma-separated list of values specified in the above table.

Used by	*Clock_check10*
Options	Comma separated list of one or more of the following values: data, control, reset, port, bbox, others, derived, all
Default value	data,control,reset,bbox,others
Example	
Console/Tcl-based usage	`set_parameter clock_usage "data,bbox"` In this case, the *Clock_check10* rule reports clock signals used as data signals or at black box ports where *abstract_port* is defined.
Usage in goal/source files	`-clock_usage="data,bbox"`

clocks_pair

Specifies clock signal pairs for the *Clock_info05* rule to check for convergence on a MUX.

For example, for pairs of clock signals `refclk`, `busclk`, `coreclk`, and `apiclk`, a message is reported only if pairs `refclk` and `busclk` or `coreclk` and `apiclk` converge on a MUX.

Used by	*Clock_info05*
Options	Comma-separated list of clock signal names in pairs
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter clocks_pair "refclk,busclk,coreclk,apiclk"`
Usage in goal/source files	`-clocks_pair=refclk,busclk,coreclk,apiclk`

coherency_check_type

Specifies if the control crossings on data path and reset path are checked for convergences and coherency issues.

By default only data path is checked for convergences/coherency. Set this parameter to `reset` to check control crossings on reset path for convergences/coherency.

Specify both the values in a comma-separated format to check the control crossings on both data path and reset path for convergences/coherency.

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_coherency06*
Options	control, reset
Default value	control
Example	
Console/Tcl-based usage	`set_parameter coherency_check_type control,reset`
Usage in goal/source files	`-coherency_check_type=control,reset`

convergence_stop_at_mux

Specifies that convergences should not be propagated to mux outputs.

If the value of this parameter is set to `yes`, propagation of relevant signals will stop whenever an RTL mux is encountered.

For example, consider the scenario shown in the following figure:

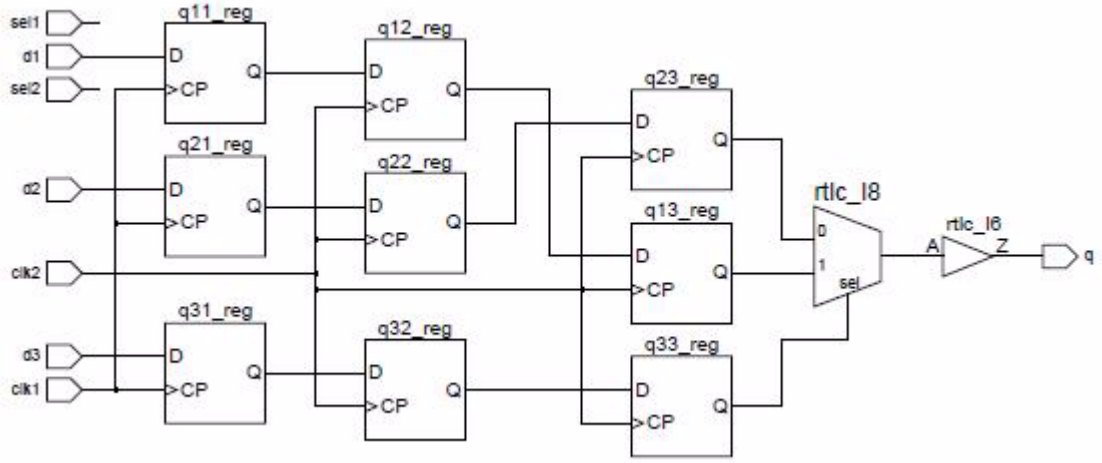


FIGURE 34. Controlling MUX Convergence

In the above scenario, if you set the `convergence_stop_at_mux` parameter to `yes`, convergence of `q23_reg`, `q13_reg`, and `q33_reg` on the MUX is not propagated further.

This parameter can be used to reduce the pessimism of propagation across muxes.

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*
Options	yes, no
Default value	no
Example	

Console/Tcl-based usage	`set_parameter convergence_stop_at_mux yes`
Usage in goal/source files	`-convergence_stop_at_mux=yes`

conv03_report_seq_conv

Specifies whether the *Ac_conv03* rule propagates synchronizers past sequential elements.

By default, the value of this parameter is set to `no`, and the Ac_conv03 does not propagate synchronizers past sequential elements. Set this parameter to `yes` to enable this rule to propagate synchronizers past sequential elements.

Used by	*Ac_conv03*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter conv03_report_seq_conv yes`
Usage in goal/source files	`-conv03_report_seq_conv=yes`

conv_all_mux_data_pins

Specifies if the *Ac_conv02* rule propagates synchronizers through all data pins of muxes for convergence detection.

By default, the value of this parameter is set to `no`, and the Ac_conv02 rule does not propagates synchronizers through all data pins of muxes for convergence detection.

Used by	*Ac_conv02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter conv_all_mux_data_pins yes`
Usage in goal/source files	`-conv_all_mux_data_pins=yes`

conv_clock_reset_path

Enables or disables the synchronizer propagation through the clock and reset pin for convergence detection.

For example, consider the scenario shown in the following figure:

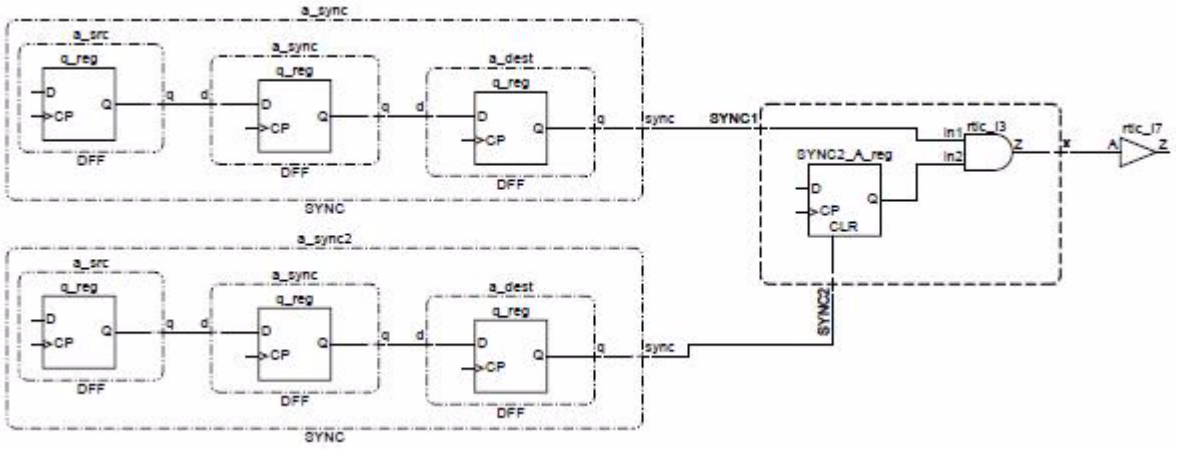


FIGURE 35. Enabling or disabling synchronizer propagation

In the above scenario, if the `conv_clock_reset_path` parameter is set to `yes`, `SYNC2` synchronizer propagates through the `CLR` and `Q` pin of `SYNC2_A_reg` flip-flop.

Set this parameter to `no` to stop the propagation at the `CLR` pin.

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter conv_clock_reset_path yes`
Usage in goal/source files	`-conv_clock_reset_path=yes`

conv_reset_seq_depth

Specifies the number of sequential elements beyond which a reset can propagate across a data terminal to determine reset convergence reported by the *Ar_converge02* rule.

For example, consider that this parameter is set to 3 in *Figure 297*. In this case, the *Ar_converge02* rule reports reset convergence after traversing through three sequential elements. However, if this parameter is set to 2, this rule stops traversal at Y and therefore, no reset convergence is reported.

Used by	*Ar_converge02*
Options	Positive integer value
Default value	2
Example	
Console/Tcl-based usage	`set_parameter conv_reset_seq_depth 4`
Usage in goal/source files	`-conv_reset_seq_depth=yes`

conv_reset_single_data_bit

When a reset signal reaches a data bus, this parameter controls if the *Ar_converge02* rule should traverse from one or all the bits of the data bus on sequential elements to detect reset convergence.

By default, this parameter is set to `no` and sequential traversal occurs from all the bits of a data bus.

Set this parameter to `yes` to allow sequential traversal from one bit of a data bus. Note that setting this value reduces runtime.

Used by	*Ar_converge02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter conv_reset_single_data_bit yes`
Usage in goal/source files	`-conv_reset_single_data_bit=yes`

conv_src_seq_depth

Specifies the maximum number of sequential elements (sequential depth) till which the *Ac_conv01*/*Ac_conv02* rules should traverse while detecting the common net of the source of synchronizers.

The following topics covers the details of this parameter:

- *Setting the Value -1 (default)*
- *Setting Value 0*
- *Setting a Positive Integer Value*
- *Points at Which Rule Traversal Stops*

Used by	*Ac_conv01*/*Ac_conv02*
Options	Positive integer value
Default value	-1
Example	
Console/Tcl-based usage	`set_parameter conv_src_seq_depth 4`
Usage in goal/source files	`-conv_src_seq_depth=4`

Setting the Value -1 (default)

The value -1 of the `conv_src_seq_depth` parameter implies that the *Ac_conv01*/*Ac_conv02* rule checks whether the same source signal diverges and is driving multiple converging synchronizers. This is shown in the following figure:

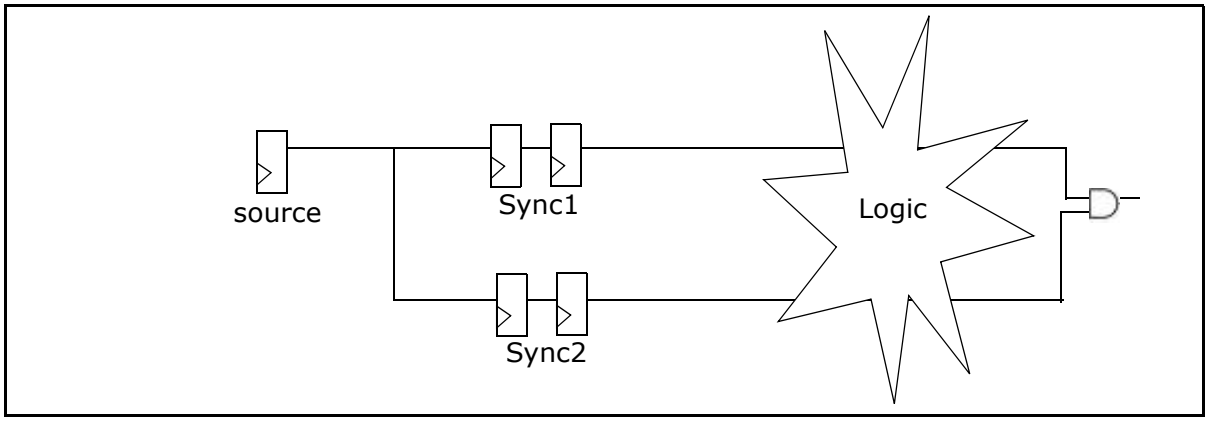


FIGURE 36. conv_src_seq_depth parameter set to -1

In the above case, the *Ac_conv01*/*Ac_conv02* rules do not traverse any sequential element including the source of synchronized crossings.

Setting Value 0

The value 0 of the `conv_src_seq_depth` parameter covers the functionality of the -1 value. In addition, the *Ac_conv01*/*Ac_conv02* rules traverse only combinational elements in the fan-in cone of sources of synchronizers to determine the common net driving these sources.

These rules do not traverse any sequential element except the source of synchronized crossing.

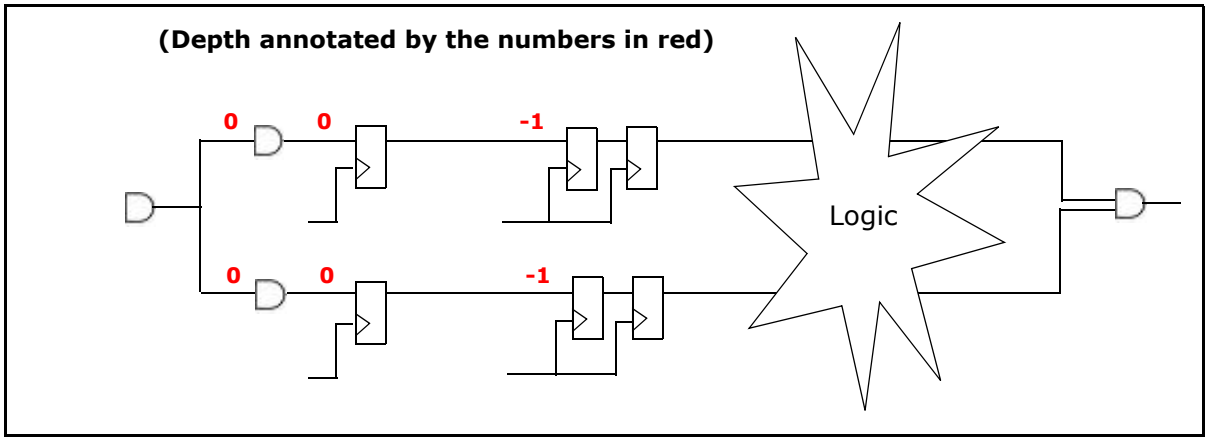


FIGURE 37. conv_src_seq_depth parameter set to 0

Setting a Positive Integer Value

Consider the following scenario in which this parameter is set to `4`:

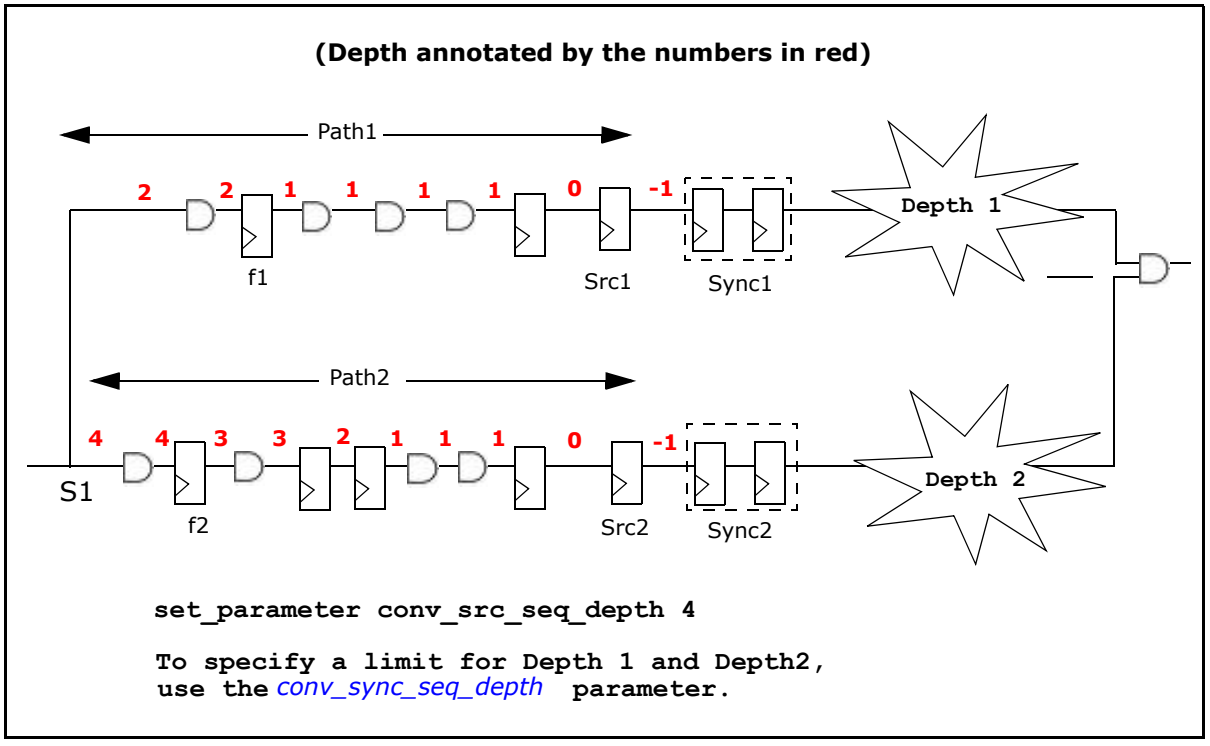


FIGURE 38. Example of using the conv_src_seq_depth parameter

In the above scenario, the depth of `Path1` (from `Src1` to `S1`) is 2, and the depth of `Path2` (from `Src2` to `S1`) is 4. Since both these depths lie within the depth (4) set by the `conv_src_seq_depth` parameter, `S1` is detected as the common net and the *Ac_conv01* rule reports `S1` as the common net in its violation.

However, if you specify the depth as 3 to this parameter, traversal on `Path2` stops at `f2` and, therefore, `S1` is not detected as the common net. In this case, the *Ac_conv01* rule reports a violation without reporting any common net.

Points at Which Rule Traversal Stops

The traversal stops at following design elements:

- Clock and reset pins of flip-flops
- Scan EN and scan input pins
- Black boxes that are not defined by the *assume_path* constraint
- Sequential arcs of complex library cells that are without a functional arc
- Blocking path

conv_sync_seq_depth

Specifies the maximum number of sequential elements (sequential depth) to be considered for the propagation of synchronizers.

A sequential depth is considered between the output pin of a destination flip-flop till the point of convergence. The flip-flop chain for a conventional multi-flop type of qualifier is considered as an additional depth.

For example, consider the following scenario in which this parameter is set to `3`:

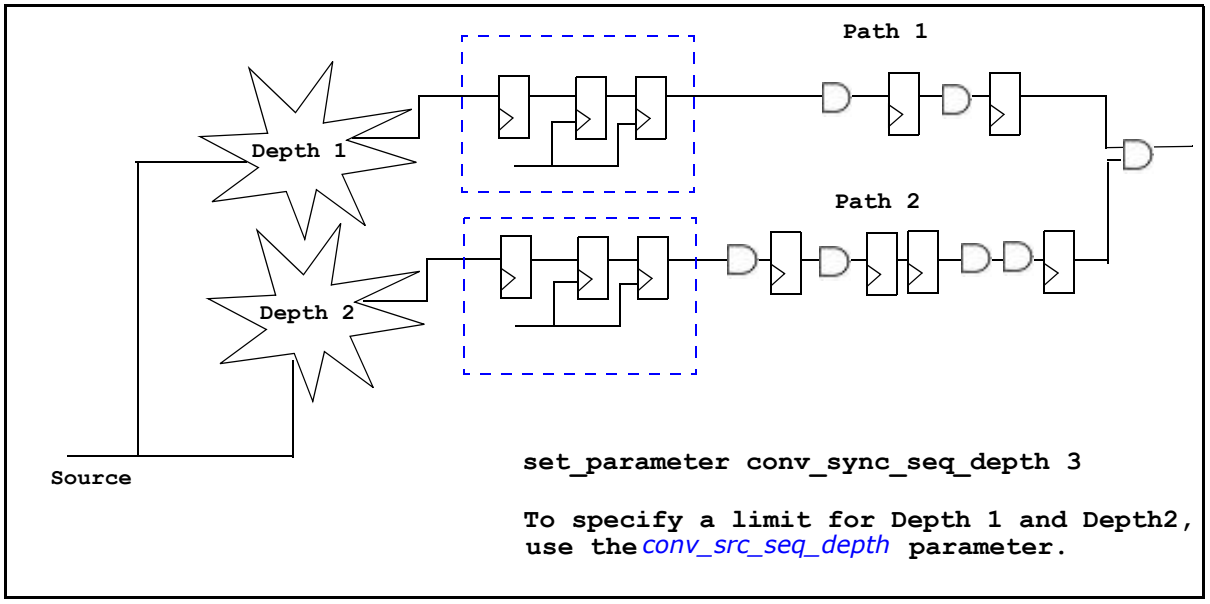


FIGURE 39. Example of using the conv_sync_seq_depth parameter

In the above scenario, propagation from `Path1` to the AND gate is valid as the sequential depth of `Path1` lies within the specified depth of 3.

However, the depth of `Path2` is greater than the specified depth. Therefore, no *Ac_conv01* violation is reported on the convergence of this path on the AND gate.

Used by	*Ac_conv01*
Options	Positive integer value
Default value	0
Example	
Console/Tcl-based usage	`set_parameter conv_sync_seq_depth 4`
Usage in goal/source files	`-conv_sync_seq_depth=4`

conv_sync_seq_depth_opt

Improves the runtime performance of the *Ac_conv01* rule when the *conv_sync_seq_depth* parameter is set to 1.

Set this parameter to `yes` to improve the runtime. By default, it is set to `no`.

Used by	*Ac_conv01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter conv_sync_seq_depth_opt yes`
Usage in goal/source files	`-conv_sync_seq_depth_opt=yes`

conv_sync_as_src

Checks convergence for synchronizers that are also used as a source in other crossings.

By default, the *Ac_conv01*, *Ac_conv02*, and *Ac_conv03* rules ignore such synchronizers.

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter conv_sync_as_src yes`
Usage in goal/source files	`-conv_sync_as_src=yes`

NOTE: *When this parameter is set to yes, SpyGlass run time is expected to increase.*

CTS_placeholder_cells

(Mandatory) Specifies names of the placeholder cells for the *Clock_check02* rule.

NOTE: *The Clock_check02 rule is not run if you do not specify the names of placeholder cells using the* `CTS_placeholder_cells` *parameter.*

Used by	*Clock_check02*
Options	Comma or space-separated list of placeholder cells
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter CTS_placeholder_cells "BUF1,BUF2"`
Usage in goal/source files	`-CTS_placeholder_cells=BUF1,BUF2`

compute_num_convergences

Specifies the number of convergences to be computed in *Ac_conv01*, *Ac_conv02*, and *Ac_conv03* rules for the same set of synchronizers.

By default this parameter is set to 1 and only one convergence is computed. Set this parameter to any positive integer less than 11 to specify the number of convergences to be computed for the same set of synchronizers.

NOTE: *If this parameter is not specified during SpyGlass goal run, the num_convergences argument in the get_cdc_coherency TCL command cannot be used.*

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*
Options	Any positive integer less than 11
Default value	1
Example	
Console/Tcl-based usage	`set_parameter compute_num_convergences 5`
Usage in goal/source files	`-compute_num_convergences =5`

deassert_mode

Configures the *Ar_asyncdeassert01* and *Ar_syncdeassert01* rules to perform different types of reset deassertion checking based on different values specified to this parameter.

Used by	*Ar_asyncdeassert01*, *Ar_syncdeassert01*
Options	Comma-separated list of *Possible Values of the deassert_mode Parameter*
Default value	none
Example	
Console/Tcl-based usage	`set_parameter deassert_mode "allow_divergence_convergence,allow_preset_domain"`
Usage in goal/source files	`-deassert_mode= "allow_divergence_convergence,allow_preset_domain"`

Possible Values of the deassert_mode Parameter

The *deassert_mode* parameter accepts the following values:

- *allow_divergence_convergence*
- *allow_preset_domain*
- *allow_assume_path_thru_bbox*
- *derived_flop*
- *none*
- *all*

allow_divergence_convergence

Specify this value to the *deassert_mode* parameter to enable SpyGlass to traverse all the paths (divergence and convergence cases). In this case, asynchronous paths of different domains from functional flip-flops are given priority.

For example, consider the following figure:

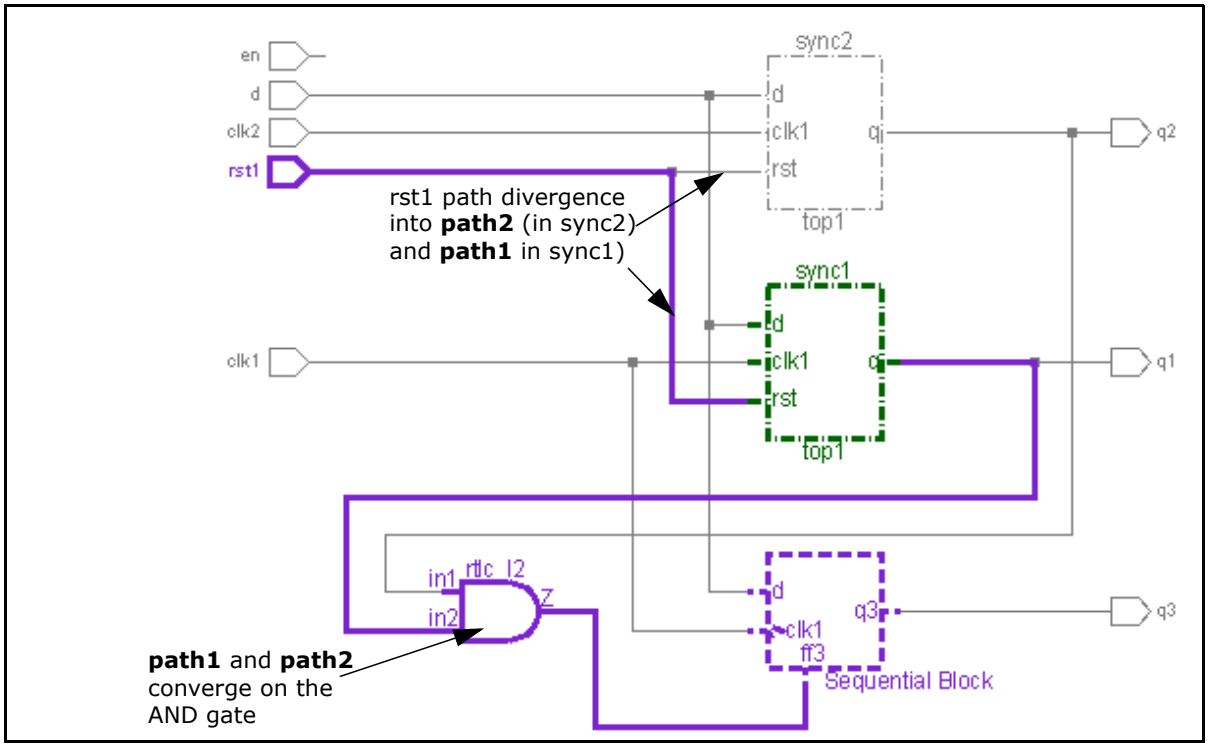


FIGURE 40.

In the above example, by default, SpyGlass traverses **path1** to reach `sync1` (of clock domain `clk1`). As **path1** converges on the AND gate that reaches the sequential block of the same clock domain `clk1`, SpyGlass reports the *Ar_syncdeassert01* violation.

To enable SpyGlass consider the alternate path, **Path2**, set the *deassert_mode* parameter to `allow_divergence_convergence`. SpyGlass then traverses **path2** to reach `sync2` (clock domain `clk2`). As **path2** converges on the AND gate that reaches the sequential block of the different clock domain `clk1`, SpyGlass reports the *Ar_asyncdeassert01* violation. The following figure shows the schematic of the *Ar_asyncdeassert01* rule for this example in which **path2** is highlighted in blue:

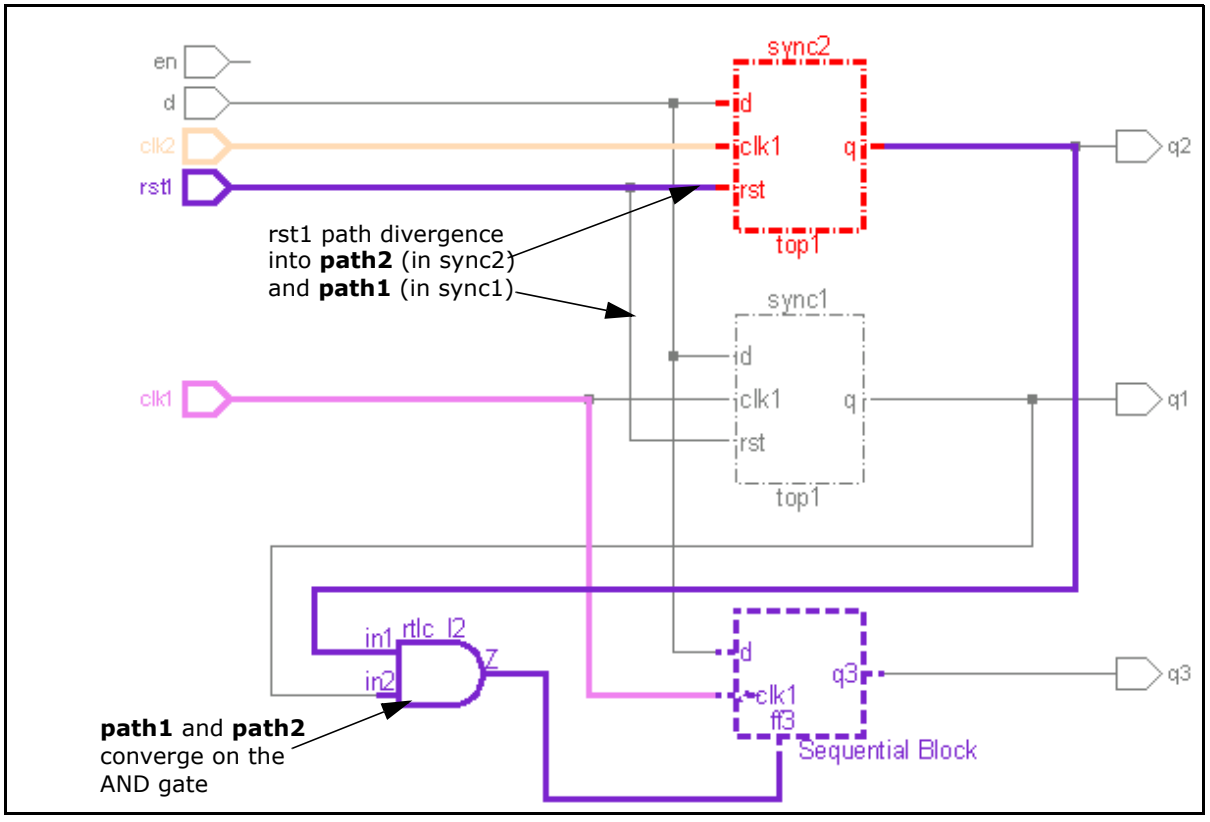


FIGURE 41.

allow_preset_domain

Specify this value to the *deassert_mode* parameter to enable SpyGlass check all the previous preset adjacent flip-flops.

For example, consider the following schematic of the *Ar_syncdeassert01* rule:

deassert_mode

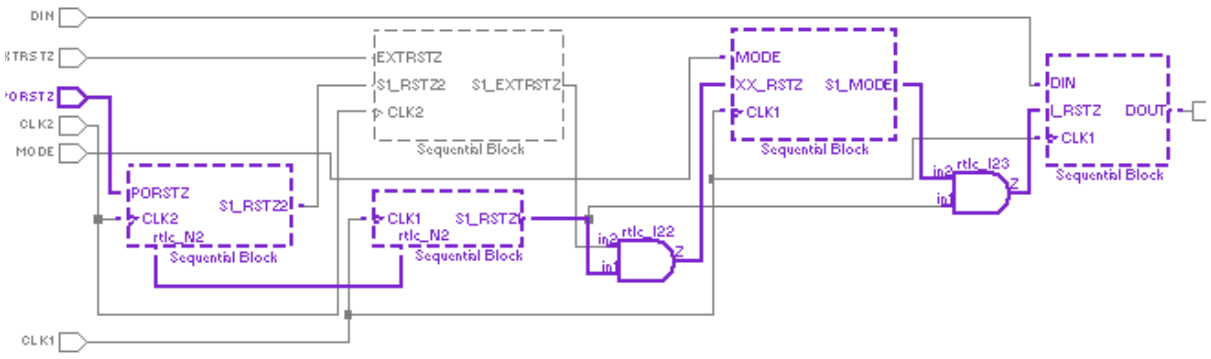


FIGURE 42.

In the above example, by default, the preset sequential block is not considered. Therefore, in this scenario the *Ar_syncdeassert01* message appears.

To consider the preset sequential block, set the *deassert_mode* parameter to `allow_preset_domain`. In this case, the *Ar_asyncdeassert01* violation appears. The following figure shows the schematic of the *Ar_syncdeassert01* rule in this case:

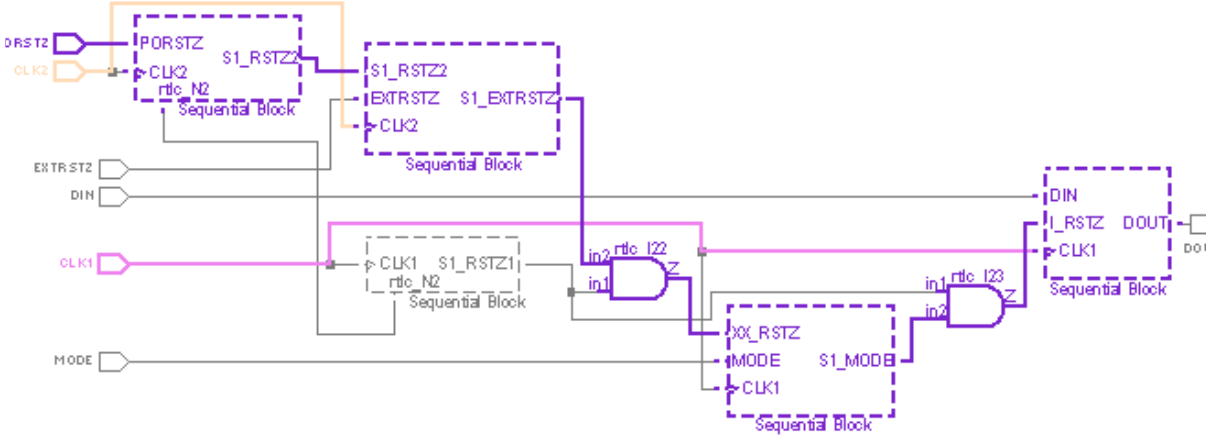


FIGURE 43.

allow_assume_path_thru_bbox

Specify this value to the *deassert_mode* parameter so that rule propagation does not stop at the following types of black boxes:

- The black box specified by the *assume_path* constraint
- Single-Input-Single-Output pin (SISO) black box

For example, consider the following schematic:

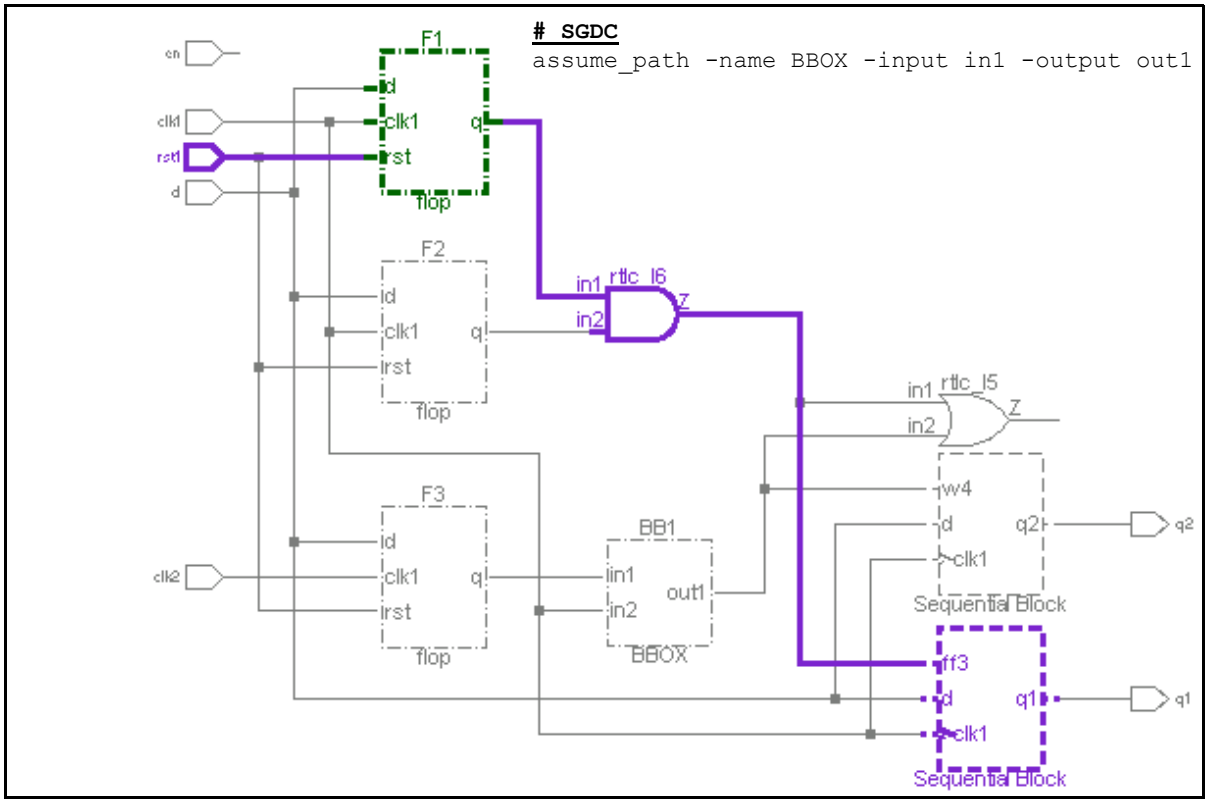


FIGURE 44.

In the above example, SpyGlass stops propagation beyond the BB1 black box, and therefore reports the *Ar_syncdeassert01* message.

To continue propagation beyond BB1, set the *deassert_mode* parameter to

`allow_assume_path_thru_bbox`. In this case, the *Ar_asyncdeassert01* reports a violation. The following figure shows the *Ar_syncdeassert01* rule schematic in this case:

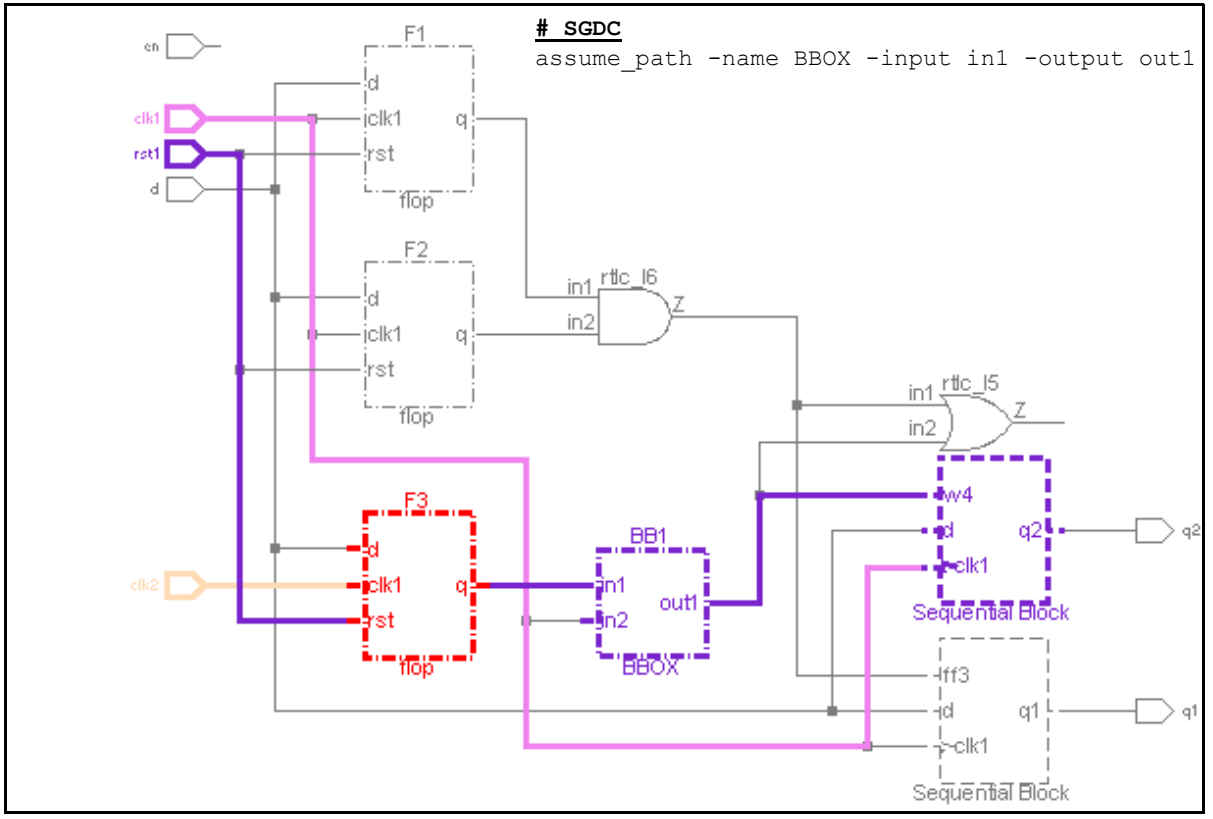


FIGURE 45.

derived_flop

Set the value of the *deassert_mode* parameter to `derived_flop` to enable the *Ar_asyncdeassert01* and the *Ar_syncdeassert01* rules to report violations on derived flops as well.

none

This is default value of the *deassert_mode* parameter that indicates that none of the following values are specified:

- *allow_divergence_convergence*
- *allow_preset_domain*
- *allow_assume_path_thru_bbox*

all

Set this value to the *deassert_mode* parameter to consider all of the following values to configure the behavior of *Ar_syncdeassert01* and *Ar_asyncdeassert01* rules:

- *allow_divergence_convergence*
- *allow_preset_domain*
- *allow_assume_path_thru_bbox*

delay_check_clk_list

Specifies a list of clock names to be checked by the *Clock_delay01* rule.

By default, the `delay_check_clk_list` parameter is set to `all` and the *Clock_delay01* rule checks for all clocks specified using the *clock* constraints.

Set the `delay_check_clk_list` parameter to a comma-separated list of clock source names to have the *Clock_delay01* rule to check for only these clocks.

Used by	*Clock_delay01*
Options	Comma or space-separated list of clock source names or all
Default value	all
Example	
Console/Tcl-based usage	`set_parameter delay_check_clk_list "clk1,clk2"`
Usage in goal/source files	`-delay_check_clk_list=clk1,clk2`

delayed_ptr_fifo

Enables detection of FIFOs with delayed read or write pointers.

Set the `delayed_ptr_fifo` parameter to `yes` when the read/write pointers are delayed and the multiplexer inside the memory is one-hot or implemented using gates, as shown in the following figure.

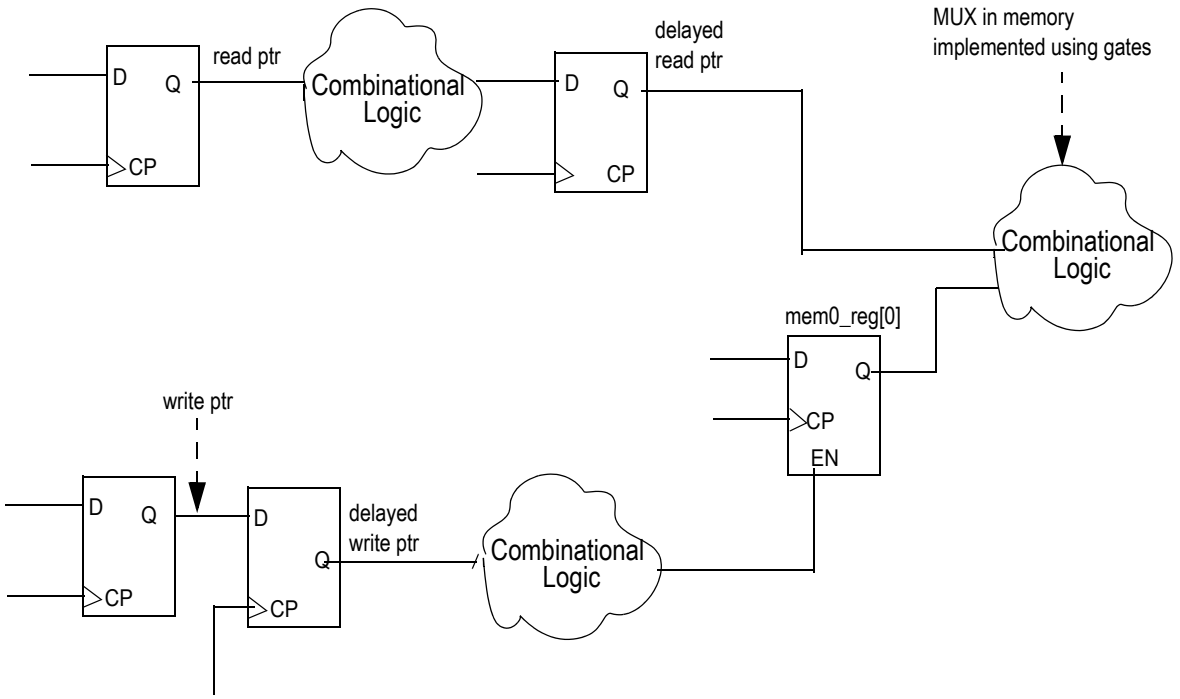


FIGURE 46. FIFO Detection with Delayed Read or Write Pointers

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv03*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*
Options	yes, no
Default value	no

Example	
Console/Tcl-based usage	`set_parameter delayed_ptr_fifo yes`
Usage in goal/source files	`-delayed_ptr_fifo=yes`

disable_inst_grouping

Specifies if *Instance-Based Grouping* should be disabled.

By default, this parameter is set to `no` and messages of the `Ac_sync_group` rules are grouped based on instance names of source and destination signal hierarchies.

Set this parameter to `yes` to disable such grouping.

Used by	*Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter disable_inst_grouping yes`
Usage in goal/source files	`-disable_inst_grouping=yes`

disable_seq_clock_prop

Disables propagation of clocks beyond flip-flops. In this case, you can specify generated clocks in an SGDC file.

By default, the `disable_seq_clock_prop` parameter is set to `no` and clock propagation continues beyond flip-flops.

Used by	*Propagate_Clocks*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter disable_seq_clock_prop yes`
Usage in goal/source files	`-disable_seq_clock_prop=yes`

dump_detailed_info

Specifies if the supported rules include detailed information in the corresponding rule/message-based spreadsheet.

By default, the `dump_detailed_info` parameter is set to `none`, and none of the supported rule includes detailed information in the generated spreadsheets.

Set the value of this parameter to one of the following values to enable the corresponding rule to include detailed information in the generated spreadsheets:

- `Ac_abstract_validation02_combo`: Set the `dump_detailed_info` parameter to the `Ac_abstract_validation02_combo` value to enable the `Ac_abstract_validation02` rule to generate a message-based spreadsheet for Combo check mismatch violations.
- `Ac_abstract_validation02_quasi`: Set the `dump_detailed_info` parameter to the `Ac_abstract_validation02_quasi` value to enable the `Ac_abstract_validation02` rule to generate a message-based spreadsheet for Quasi-static mismatch violations.
- `Clock_sync05`: Set the `dump_detailed_info` parameter to `Clock_sync05` to generate a message-based spreadsheet for `Clock_sync05` violations containing all the signals in which input port is sampled.
- `Clock_sync06`**:** Set the `dump_detailed_info` parameter to `Clock_sync06` to generate a message-based spreadsheet for `Clock_sync06` violations containing all the signals that are driving the output port.
- `Clock_sync05a`: Set the `dump_detailed_info` parameter to `Clock_sync05a` to generate a message-based spreadsheet for `Clock_sync05a` violations containing all the signals in which input port is sampled.
- `Clock_sync06a`**:** Set the `dump_detailed_info` parameter to `Clock_sync06a` to generate a message-based spreadsheet for

Clock_sync06a violations containing all the signals that are driving the output port.

- Reset_sync04**:** Set the dump_detailed_info parameter to Reset_sync04 to generate a message-based spreadsheet for Reset_sync04 violations listing the reset synchronizers.

Used by	*Ac_abstract_validation02*, *Clock_sync05*, *Clock_sync06*, *Reset_sync04*
Options	none, all, Ac_abstract_validation02_combo, Ac_abstract_validation02_quasi, Clock_sync05, Clock_sync06, Clock_sync05a, Clock_sync06a, Reset_sync04
Default value	none
Example	
Console/Tcl-based usage	`set_parameter dump_detailed_info Ac_abstract_validation02_combo`
Usage in goal/source files	`-dump_detailed_info Ac_abstract_validation02_combo=dump_inst_type`

dump_sync_info

Generates *The SynchInfo Report* and *The CrossingInfo Report*.

Based on the following values of this parameter, different information is generated in these reports:

- `no` (default)

  These reports are not generated.

- `yes`

  These reports show limited information for the crossings containing source and destination flip-flops.

  For example, when this parameter is set to `yes`, the *Section 1* of *The SynchInfo Report* shows only the last flip-flop in the synchronizer chain instead of all the flip-flops. To view all the flip flops, set this parameter to `detailed` or `detailed_mod`.

- `detailed`

  These reports show detailed information for the crossings containing source and destination flip-flops.

  For example, when this parameter is set to `detailed`, the *Section 1* of *The SynchInfo Report* shows all the flip-flops in the synchronizer chain.

- `detailed_mod`

  The SynchInfo Report shows detailed information for the crossings containing source and destination flip-flops. In addition, the flip-flops in the synchronizer chain are shown with their respective module names.

Used by	*Reset_sync03*, *Reset_sync04*, *The Ac_sync_group Rules*, *Ar_resetcross01*
Options	yes, no, detailed, detailed_mod
Default value	no
Example	
Console/Tcl-based usage	`set_parameter dump_sync_info yes`
Usage in goal/source files	`-dump_sync_info=yes`

dump_inst_type

Specifies the type of instances to be reported by *The SynchInfo Report* and *The CrossingInfo Report*.

By default, the `dump_inst_type` parameter is set to `all`, and the destinations and synchronizers that are flip-flops, latches, or sequential cells are dumped in these reports.

Set the value of this parameter to `flop` to dump only those destinations and synchronizers that are flip-flops.

Used by	*Reset_sync03*, *Ar_resetcross01*
Options	flop, all
Default value	all
Example	
Console/Tcl-based usage	`set_parameter dump_inst_type flop`
Usage in goal/source files	`-dump_inst_type=flop`

enable_ac_sync_qualdepth

Enables reporting of *Qualifier Name and Qualifier Depth in a Message-Based Spreadsheet* of *The Ac_sync_group Rules*.

By default, this parameter is set to `no`, and such information is not shown in the spreadsheet.

NOTE: *The value of this parameter is assumed to be* `yes` *when the cdc_qualifier_depth parameter is set to a value other than -1.*

Used by	*Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_ac_sync_qualdepth yes`
Usage in goal/source files	`-enable_ac_sync_qualdepth=yes`

enable_block_cfp

Generates block-level cdc_false_path constraints during block abstraction. By default, the parameter is set to `no` and the constraints are not generated during abstraction.

Used by	CDC verification rules
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_block_cfp yes`
Usage in goal/source files	`-enable_block_cfp=yes`

enable_and_sync

Enables the *AND Gate Synchronization Scheme*. In this scheme, the destination flip-flop should be driven directly by an AND gate where the other input (input pin not connected to the source flip-flop) of the AND gate should be driven by a primary port or should be synchronized to the destination clock domain.

For *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01* rules, this parameter also allows OR, NAND, and NOR gates in addition to an AND gate.

By default, this scheme is disabled.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ar_resetcross01*, and *Ac_unsync01*
Options	yes, no
Default value	no
Default Value in GuideWare2.0	yes
Example	
Console/Tcl-based usage	`set_parameter enable_and_sync yes`
Usage in goal/source files	`-enable_and_sync=yes`

enable_or_sync

Enables the OR Gate based synchronization in the *For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)*. In this scheme, the destination flip-flop should be driven directly by an OR gate where the other input (input pin not connected to the source flip-flop) of the OR gate should be driven by a primary port or should be synchronized to the destination reset or clock domain.

By default, this scheme is disabled.

Used by	*Ar_resetcross01*, *Ar_resetcross_matrix01*
Options	yes, no
Default value	no
Default Value in GuideWare2.0	no
Example	
Console/Tcl-based usage	`set_parameter enable_or_sync yes`
Usage in goal/source files	`-enable_or_sync=yes`

enable_clock_gate_sync

Enables the *Clock-Gating Cell Synchronization Scheme*. By default, the parameter is set to `yes` and the scheme is enabled.

Used by	*Clock_sync03a*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Clock_hier01*, *Clock_hier02*, *Ar_resetcross01*, and *Clock_hier03*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_clock_gate_sync no`
Usage in goal/source files	`-enable_clock_gate_sync=no`

enable_clock_path_crossings

Enables reporting of crossings for sources present in a clock path of a destination instance.

By default, this parameter is set to `no` and potential sources in a clock path of flip-flops are not identified.

Used by	*Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_clock_path_crossings yes`
Usage in goal/source files	`-enable_clock_path_crossings=yes`

enable_condition_based_sync

NOTE: *This parameter is deprecated. Please use the sync_check_type parameter instead of this parameter.*

Enables *The Enable Expression-Based Synchronization Analysis*.

By default, this parameter is set to `no`, and data synchronization analysis occurs based on the qualifier search in the transitive input cone of a gate that receives a source. This approach checks for proper enable condition that ensures correct transfer of source data.

Used by	*The Ac_sync_group Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_condition_based_sync yes`
Usage in goal/source files	`-enable_condition_based_sync=yes`

enable_debug_data

Enables annotation of debug information, such as clock, reset, quasi_static signals, and domain information on nets in the schematic. For details, see *Viewing Debug Data in Schematic*.

By default, the value of this parameter is set to `no`, and annotation of debug information in the schematic is disabled. Set the value of this parameter to `yes` to view such information.

Used by	All SpyGlass CDC solution rules
Options	yes, no
Default value	no
Default Value in GuideWare2.0	yes
Example	
Console/Tcl-based usage	`set_parameter enable_debug_data yes`
Usage in goal/source files	`-enable_debug_data=yes`

enable_delayed_qualifier

NOTE: *This parameter has been deprecated. Use the cdc_qualifier_depth parameter.*

Enables *The Ac_sync_group Rules* to search for *Qualifier* or *Potential Qualifier* through sequential and/or combinational logic.

This parameter accepts three values: *yes*, *no*, and *strict*.

Consider the following figure:

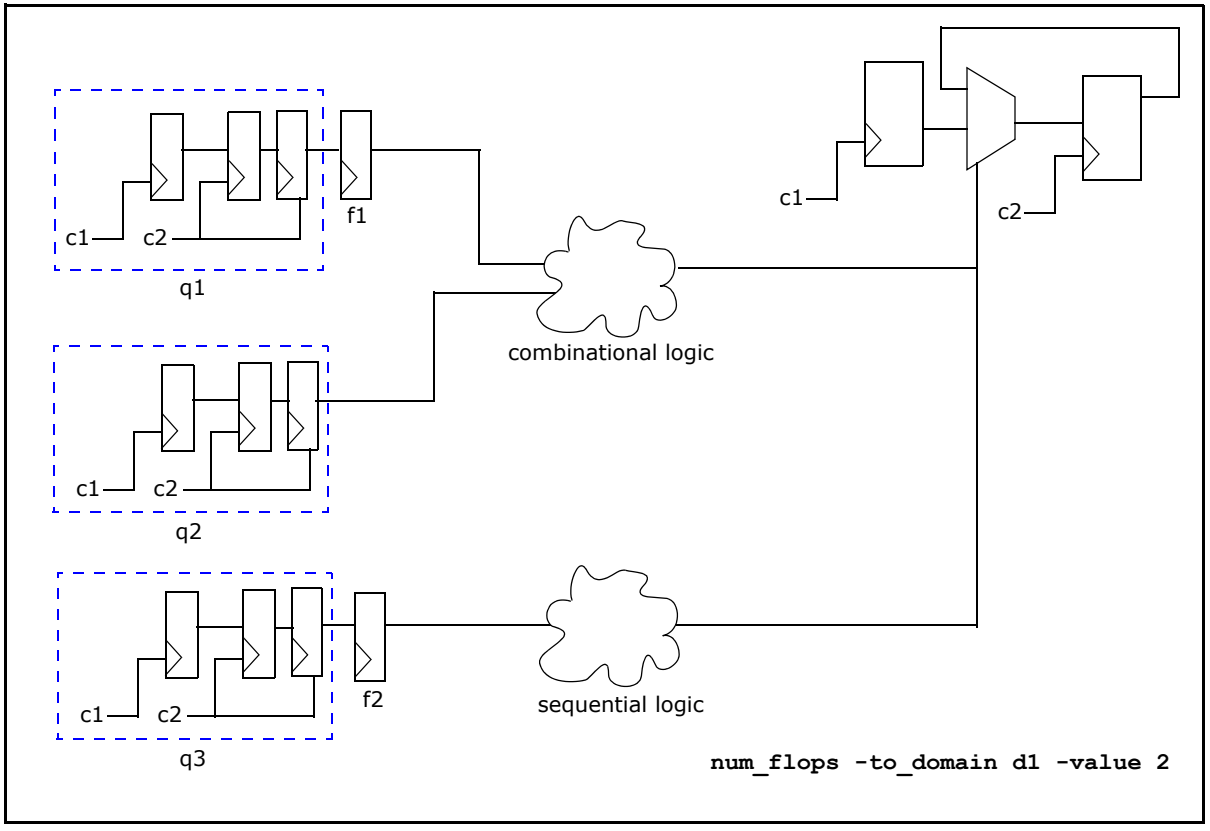


FIGURE 47. Example of using the enable_delayed_qualifier parameter

Based on the above scenario, *The Ac_sync_group Rules* search for different *Qualifier* or *Potential Qualifier*, which are connected to the destination, based on the following values of this parameter:

- `yes`

  In this case, *The Ac_sync_group Rules* search for *Qualifier* or *Potential Qualifier* beyond sequential and combinational logic.

  Therefore, in *Figure 47*, these rules detect the `q1`, `q2`, and `q3` qualifiers.

- `no`

  In this case, *The Ac_sync_group Rules* search for *Qualifier* or *Potential Qualifier* beyond combinational logic. The rules do not traverse beyond sequential logic in this case to search for qualifiers.

  In addition, the rules consider any flip-flop (in this case `f1`) that is beyond the synchronization chain limit (in this case 2) set by the *num_flops* constraint.

  Therefore, in *Figure 47*, these rules detect the `q1` and `q2` qualifiers.

- `strict`

  In this case, *The Ac_sync_group Rules* search for *Qualifier* or *Potential Qualifier* beyond combinational logic. The rules do not traverse beyond sequential logic in this case to search for qualifiers.

  In addition, the rules disallow any flip-flop (in this case `f1`) that is beyond the synchronization chain limit (in this case 2) set by the *num_flops* constraint.

  Therefore, in *Figure 47*, these rules detect the `q2` qualifier.

Used by	*The Ac_sync_group Rules*
Options	yes, no, strict
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_delayed_qualifier no`
Usage in goal/source files	`-enable_delayed_qualifier=no`

enable_derived_reset

Specifies if resets are propagated through derived resets.

By default, resets propagate through derived reset path. Set the value of the `enable_derived_reset` parameter to `no` to stop resets from propagating through possible derived reset paths.

Used by	*Propagate_Resets*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_derived_reset no`
Usage in goal/source files	`-enable_derived_reset=no`

enable_generated_clocks

Set this parameter to `yes` to:

- Consider *generated_clock* constraints specified in an SGDC file during SpyGlass analysis.
- Dump derived-clocks information in the form of *generated_clock* constraints in the *generated_clocks.sgdc* and *cdc_setup_generated_clocks.sgdc* files when the *use_inferred_clocks* parameter is set to `yes`.
- Stop clock propagation beyond sequential elements so that the derived clock specified by the *generated_clock* constraint is propagated beyond sequential elements.

Used by	All SpyGlass CDC rules
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_generated_clocks yes`
Usage in goal/source files	`-enable_generated_clocks=yes`

enable_glitchfreecell_detection

Enables reporting of glitch-free multiplexers in a design.

Information about glitch-free multiplexers is reported in the *The CDC-Detailed-Report* (*Section K*) and *The CDC-Summary-Report* (*Section I*).

Used by	*The Ac_sync_group Rules*, *Ar_sync01*, *Ar_unsync01*, *Clock_info05*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_glitchfreecell_detection yes`
Usage in goal/source files	`-enable_glitchfreecell_detection=yes`

enable_multiflop_sync

Controls the *Conventional Multi-Flop Synchronization Scheme*.

By default, this scheme is enabled. The following table describes all the possible options of the parameter:

Options	Description
control	Enables multiflop based synchronization for clock crossings/ Ac_sync rules.
reset	Enables multiflop based synchronization for reset synchronizers which affects Ar_sync rules.
all	Enables multiflop based synchronization for control and reset schemes.
yes	Enables multiflop based synchronization only for control schemes.
no	Disables multiflop based synchronization for control and reset

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01*
Options	control, reset, all, yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_multiflop_sync no`
Usage in goal/source files	`-enable_multiflop_sync=no`

enable_mux_dest_domain

Specifies that the following clock domain crossings should be considered synchronized under the *Recirculation MUX Synchronization Scheme, Synchronized Enable Synchronization Scheme*, *Glitch Protection Cell Synchronization Scheme*, *AND Gate Synchronization Scheme*, and *Clock-Gating Cell Synchronization Scheme*:

- Clock crossings where the first flip-flop in the destination clock domain is driven by a MUX where the select signal belongs to the destination domain (that is, driven by a flip-flop of the destination domain).
- Clock crossings where the enable signal of the first flip-flop in the destination clock domain belongs to the destination domain (that is, driven by a flip-flop of the destination domain).
- Clock crossings where the Glitch protection cell that is driving the first flip-flop in the destination domain has the other input pin (not connected to the source) in the destination domain.
- Clock crossing where the AND gate that is driving the first flip-flop in destination domain has the other input pin (not connected to source flip-flop) in the destination domain.
- Clock crossings where the clock-gating cell that is driving the first flip-flop in the destination domain has the other input pin (not connected to the clock) in the destination domain.

When the `enable_mux_dest_domain` parameter is set, it is sufficient to have a destination domain signal instead of the synchronizer driving the enable signal. This parameter accepts a comma-separated list of any of the following values:

Value	Description
mux	Works for Recirculation-mux Synchronization Scheme
enable	Works for Synchronized Enable Synchronization Scheme
gp	Works for Glitch Protection Cell Synchronization scheme
cg	Works for Clock-Gating Synchronization scheme
and	Works for AND Gate Synchronization scheme
all	All the above schemes
none	Indicates parameter is turned off

Value	Description
yes	Provided for backward compatibility; same as all
no	Provided for backward compatibility; same as none

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*
Options	all, none, or a comma-separated list of valid values
Default value	none
Example	
Console/Tcl-based usage	`set_parameter enable_mux_dest_domain "mux,gp"`
Usage in goal/source files	`-enable_mux_dest_domain="mux,gp"`

enable_mux_sync

Enables or disables the following schemes:

- *Recirculation MUX Synchronization Scheme*
- *MUX-Select Sync (Without Recirculation) Synchronization Scheme*.

Following are the valid values of the `enable_mux_sync` parameter:

- `recirculation` (default)

  Enables the *Recirculation MUX Synchronization Scheme*. See *Figure 5*.

- `mux_select`

  Enables the *MUX-Select Sync (Without Recirculation) Synchronization Scheme*. See *Figure 6*.

- `all`

  Enables both the above schemes.

- `strict`

  Enables both the above schemes with a restriction on the *MUX-Select Sync (Without Recirculation) Synchronization Scheme*. The restriction is that in this scheme, the data line that is free of source should not be tied to a constant or supply. It should only be coming from a destination domain.

  Therefore, the scenario shown in *Figure 5* is valid in this case. However, the scenario shown in the following figure is not valid in this case:

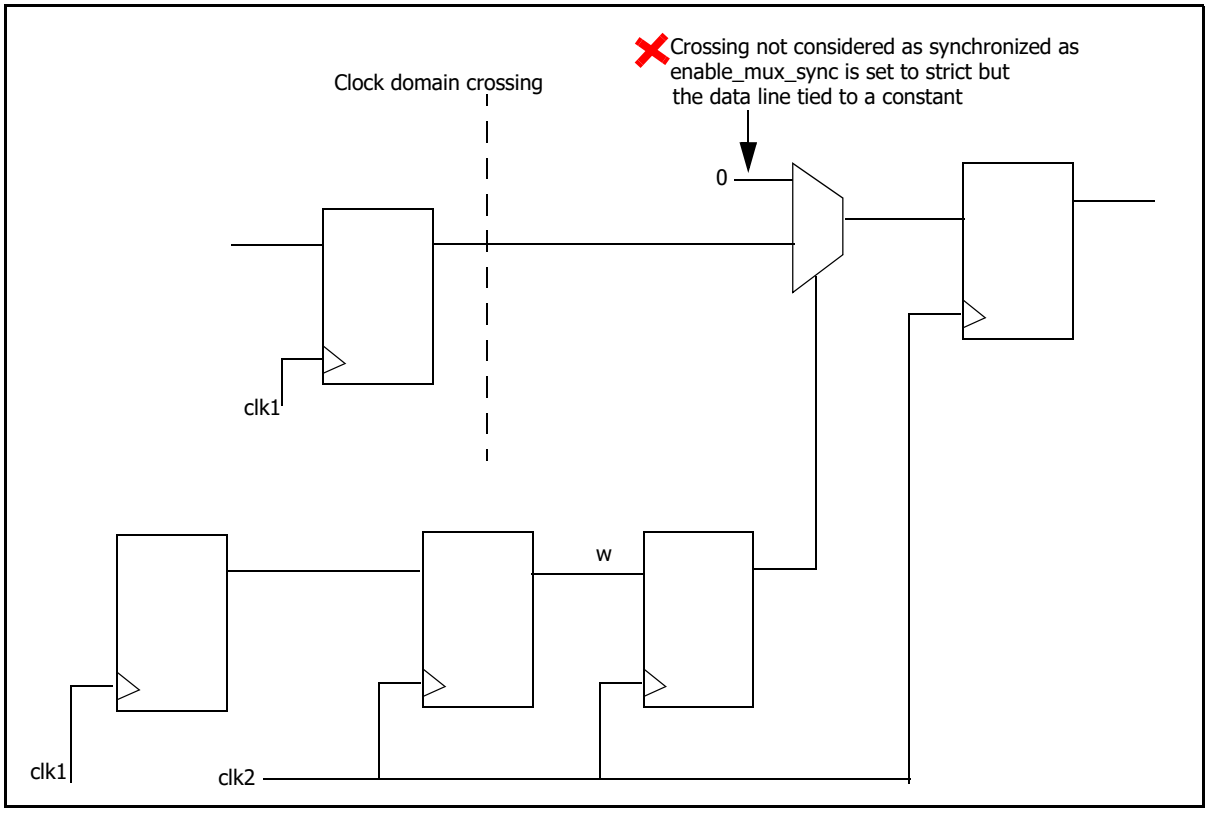


FIGURE 48. MUX-Select Sync (Without Recirculation) Synchronization Scheme

- `none`

  Disables both the specified schemes.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a* *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ar_resetcross01*
Options	none, recirculation, mux_select, all, strict
Default value	recirculation

Default Value in GuideWare2.0	all
Example	
Console/Tcl-based usage	`set_parameter enable_mux_sync all`
Usage in goal/source files	`-enable_mux_sync=all`

enable_reset_cone_spreadsheet

Specifies if a message-based spreadsheet is generated for each violation message reported by the *Ar_unsync01*, *Ar_asyncdeassert01*, and *Reset_sync02* rules. The generated spreadsheet includes all similar flops that are candidates for the reported violation.

By default, the spreadsheet is not generated. Set this parameter to `yes` to enable SpyGlass CDC to generate a spreadsheet for each violation message reported by the *Ar_unsync01*, *Ar_asyncdeassert01*, and *Reset_sync02* rules.

Used by	*Ar_unsync01*, *Ar_asyncdeassert01*, *Reset_sync02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_reset_cone_spreadsheet yes`
Usage in goal/source files	`-enable_reset_cone_spreadsheet=yes`

enable_selection

This parameter is deprecated. Use the *sync_point_selection* parameter instead of this parameter.

enable_sim_check_rdc

Specifies if the *Ar_resetcross01* rule should perform simulation to check that when the source reset is active, the destination clock is switched off so that no metastability is captured at the destination.

By default, such simulation checks are performed by this rule. Set this parameter to `no` to disable these simulation checks.

Used by	*Ar_resetcross01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_sim_check_rdc no`
Usage in goal/source files	`-enable_sim_check_rdc=no`

enable_sync_check_rdc

NOTE: *It is recommended that enable_sync_check_rdc is not used because rdc synchronization checks are enabled by default now. You can set multi-flop synchronizaton scheme to on/off for RDC by using the enable_multiflop_sync parameter. Use the qualifier -rdc and the sync_cell -rdc constraints to enable other synchronization schemes for RDC.*

NOTE:

Specifies if the *Ar_resetcross01* rule should check if the destination is synchronized by any of the following synchronization schemes:

- *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)*
- *For Control path crossings: Synchronizing Cell Synchronization Scheme (for RDC)*

Set this parameter to `no` to not consider the above schemes.

Used by	*Ar_resetcross01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_sync_check_rdc no`
Usage in goal/source files	`-enable_sync_check_rdc=no`

enable_diff_clkdom_rdc

Specifies if the *Ar_resetcross01* rule should report reset domain crossings having different clock domains in source and destination.

Set this parameter to `yes` to report reset domain crossings having different clock domains.

Used by	*Ar_resetcross01*, *Setup_rdc01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_diff_clkdom_rdc yes`
Usage in goal/source files	`-enable_diff_clkdom_rdc=yes`

ignore_qualifier_mismatch_rdc

Use this parameter to enable/disable checks to identify mismatches in qualifier's reset domains with that of destination object in reset domain crossing.

By default, if there is any mismatch in reset of the qualifier to that of destination object, the SGDC_qualifier23 rule reports an Error message and the qualifier is considered as invalid, and an unsynchronized RDC crossing is reported.

You can set the ignore_qualifier_mismatch_rdc parameter to `yes` and use the `reset_filter_path` constraint between the qualifier and destination object to ignore such reset mismatches. In this case, the qualifier is considered as valid and SpyGlass reports this crossing as synchronized. For example, consider the following schematic:

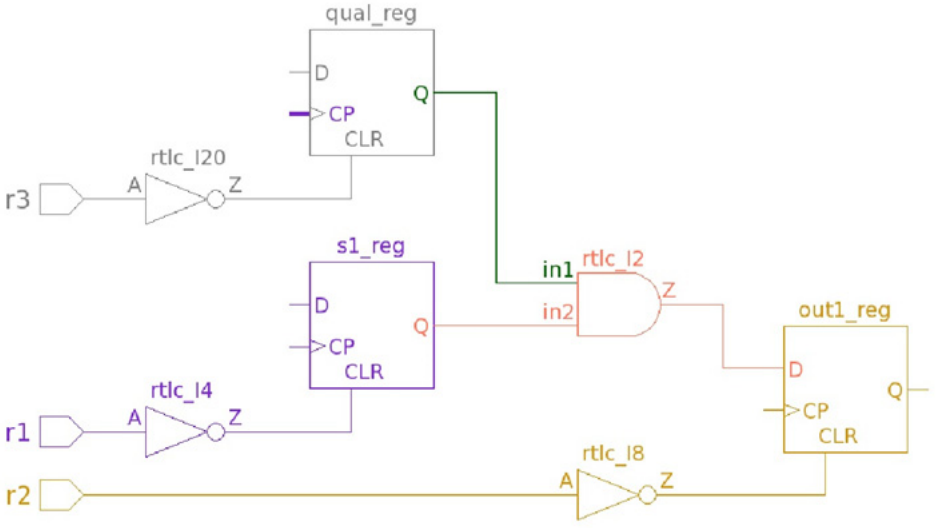


FIGURE 49.

In the above case, the qualifier is driven by the r3 reset and the destination is driven by the r2 reset. Therefore, this qualifier is considered as invalid. However, to enable SpyGlass consider this qualifier as valid and report the crossing as synchronized, you can use the ignore_qualifier_mismatch_rdc

parameter as shown below:

```
set_parameter ignore_qualifier_mismatch_rdc yes
reset_filter_path -from_rst r3 -to_rst r2
```

Used by	*Ar_resetcross01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter ignore_qualifier_mismatch_rdc yes`
Usage in goal/source files	`-ignore_qualifier_mismatch_rdc=yes`

rdc_reduce_pessimism

Use this parameter to enable the *Ar_resetcross01* rule to ignore reset-domain crossings in some scenarios.

The check to find reset-domain crossings between source and destinations is relaxed. For example, consider two source resets R1 and R2 and two destination resets R2 and R3. To ignore reset-domain crossings for each source destination pair, you would need to define either the `reset_filter_path` or the `define_reset_order` SGDC constraints for each of the following combinations:

- R1 -> R2
- R1-> R3
- R2 -> R3

Alternatively, you can set this parameter to `reset_filter`, define the `reset_filter_path` or the `define_reset_order` SGDC constraint for only one of the following combinations:

- R1 -> R2, or
- R1 -> R3

By default, the parameter is set to `none` and no reset-domain crossings are ignored.

Set this parameter to `qualifier` to relax the matching criteria of source resets and destination resets with the resets mentioned in `-from_rst` and `-to_rst` attribute of the qualifier constraints.

For example, consider the following qualifier constraint:

```
qualifier -name qual -from_rst R1 R2 -to_rst R3 R4 -rdc
```

In the above specification, if the parameter is not set to `qualifier`, the qualifier constraint will synchronize only the reset domain crossings with the specified combination of resets at the source and destination.

You can set the parameter to `qualifier` to relax the strict matching criteria and allow the qualifier constraint to synchronize any combination of reset domain crossings between the source (R1, R2) and the destination (R3, R4) resets.

Set this parameter to `exclusive` to waive or synchronize an RDC crossing by using a combination of different constraints such as quasi_static_rdc, reset_filter_path and define_reset_order.

Used by	*Ar_resetcross01*
Options	reset_filter, exclusive, qualifier, none
Default value	none
Example	
Console/Tcl-based usage	`set_parameter rdc_reduce_pessimism reset_filter`
Usage in goal/source files	`-rdc_reduce_pessimism=reset_filter`

rdc_allow_sync_reset

This parameter enables SpyGlass CDC to consider synchronous resets as equivalent of asynchronous resets for performing RDC checks. Use this parameter to specify synchronous resets, in addition to asynchronous resets, in the `-to_rst` argument of the `reset_filter_path` constraint.

By default, the value is set to `none` and synchronous resets specified in the `-to_rst` argument of the `reset_filter_path` constraint are not considered by the *Ar_resetcross01* rule.

Set the parameter to `endpoint` to enable this support at end points of RDC crossings.

Set this parameter to `both` to enable this support at end points as well as destination of RDC crossings.

Used by	*Ar_resetcross01*, *Ar_resetcross_matrix01*
Options	none, endpoint, both
Default value	none
Example	
Console/Tcl-based usage	`set_parameter rdc_allow_sync_reset both`
Usage in goal/source files	`-rdc_allow_sync_reset=both`

report_sync_rdc

Specifies the way sources should be reported by the *Ar_resetcross01* rule when the destination is synchronized by any RDC synchronization schemes.

Values Accepted by the report_sync_rdc Parameter

The following table describes the values accepted by this parameter:

Value	Description
all (default value)	Specify this value to report all the sources reaching the synchronized destination.
none	Specify this value to disable reporting of any source.

Example of the report_sync_rdc Parameter

Consider the following violations reported by the *Ar_resetcross01* rule when this parameter is set to `all`:

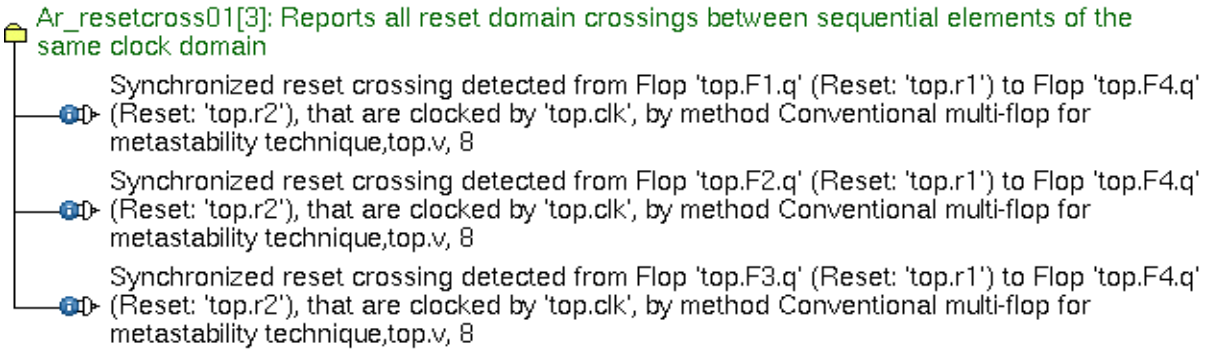


FIGURE 50.

In the above figure, all the sources reaching the synchronized destination `top.F4.q` are reported.

Used by	*Ar_resetcross01*
Options	all, none
Default value	all
Example	
Console/Tcl-based usage	`set_parameter report_sync_rdc none`
Usage in goal/source files	`-report_sync_rdc=none`

show_unsync_qualifier_rdc

Specifies if the reason and potential qualifiers for unsynchronized reset-domain crossings are reported by the *Ar_resetcross01* rule.

This parameter enables you to debug unsynchronized reset-domain crossings. Set this parameter to `yes` to report the reason and potential qualifiers for unsynchronized reset-domain crossings.

Set this parameter to `backward` to report a reduced set of reasons for invalid qualifier and invalid convergence.

Used by	*Ar_resetcross01*
Options	yes, no, backward
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter show_unsync_qualifier_rdc yes`
Usage in goal/source files	`-show_unsync_qualifier_rdc=yes`

enable_multiflop_sync

Controls the `Conventional Multi-Flop Synchronization Scheme`.

By default, this scheme is enabled. The following table describes all the possible options of the parameter:

Options	Description
control	Enables multiflop based synchronization for clock crossings/ Ac_sync rules.
reset	Enables multiflop based synchronization for reset synchronizers which affects Ar_sync rules.
rdc	Enables multiflop based synchronization for reset domain crossings which affects the Ar_resetcross01.
all	Enables multiflop based synchronization for control, reset, and rdc schemes.
yes	Enables multiflop based synchronization only for control and rdc schemes.
no	Disables multiflop based synchronization for control, reset, and rdc

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ar_resetcross01*, and *Ac_unsync01*
Options	control, reset, rdc, all, yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_multiflop_sync no`
Usage in goal/source files	`-enable_multiflop_sync=no`

enable_sync

Specifies whether the *Synchronized Enable Synchronization Scheme* is run. By default, the scheme is always run.

The *Synchronized Enable Synchronization Scheme* considers all those clock crossings where the destination domain flip-flop is an enabled flip-flop. A crossing is marked as synchronized if either of the following conditions is met:

- The enable pin is driven by a signal synchronized to the destination clock domain.
- A valid synchronizer exists in any one of the paths driving the enable pin and signals in all other paths are driven either by primary ports or by destination clock domain flip-flops.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ar_resetcross01*, and *Ac_unsync01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter enable_sync no`
Usage in goal/source files	`-enable_sync=no`

expected_ckcells_file

Specifies the name of the file containing the list of cells that are allowed in clock trees checked by the *Clock_check06a* rule.

By default, the `expected_ckcells_file` parameter is not set. You can set this parameter to one or more file names that contain names of allowed cells.

Specifying the List of Allowed Cells

Create an ASCII file containing the names of allowed cells (one name in each line) and specify it with the `expected_ckcells_file` parameter.

In the ASCII file:

- You can use Perl regular expressions to specify the names of multiple cells. For example, to refer to the cells `cell1`, `cell2`, `cell3`, and `cell4`, you can specify `cell.*`.
- You can use `//` style or # style comments.

Used by	*Clock_check06a*, *Ar_resetcross01*
Options	Comma or space-separated list of file names
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter expected_ckcells_file "cells1.list,cells1.list"`
Usage in goal/source files	`-expected_ckcells_file=cells1.list,cells1.list`

enable_sync_cell

Specifies the synchronizer cells that should be considered as valid synchronizers if they are driving the enable pin of the destination flip-flop in the *Synchronized Enable Synchronization Scheme* or the synchronizer cells driving the MUX select pin in the *Recirculation MUX Synchronization Scheme* and *MUX-Select Sync (Without Recirculation) Synchronization Scheme*.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, and *Ac_datahold01a*
Options	Comma or space-separated list of synchronizer cells
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter enable_sync_cell "Sync_cell1,Sync_cell2"`
Usage in goal/source files	`-enable_sync_cell=Sync_cell1,Sync_cell2`

fa_abstract

Functional analysis of a property depends on the size and complexity of its fan-in cone. Usage of this parameter applies a context-sensitive abstraction technique to reduce complex verification problem into a simpler and solvable problem.

If you have partially proved properties reported by the rules listed in the table below, this parameter may help in concluding such properties.

A list of valid values for the parameter, `fa_abstract`, is given below:

Values	Description
Ac_cdc08	Enable abstraction for the rule *Ac_cdc08*
Clock_sync03a	Enable abstraction for the rule *Clock_sync03a*
Ac_conv02	Enable abstraction for the *Ac_conv02* rule
Ac_glitch03	Enable abstraction for the rule *Ac_glitch03*
Ac_datahold01a	Enable abstraction for the rule *Ac_datahold01a* When you set the value of the *fa_abstract* parameter to Ac_datahold01a, failure cannot be determined, and such cases are reported as partially-proved. In such cases, check the failing properties without the *fa_abstract* parameter.
all	Enable abstraction for the rules *Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, and *Ac_glitch03*
none	Disable abstraction for the rules *Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, and *Ac_glitch03*

Used by	*Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, and *Ac_glitch03*
Options	all, none, or a comma-separated list of the following rule names: Ac_handshake01, Ac_cdc08, Ac_conv02, Clock_sync03a, Ac_fifo01, and Ac_glitch03
Default value	Ac_handshake01,Ac_glitch03
Example	

Console/Tcl-based usage	`set_parameter fa_abstract all`
Usage in goal/source files	`-fa_abstract=all`

fa_atime

Specifies the CPU time (in seconds) consumed by the tool to perform functional analysis per assertion.

The default run time is set to 20 seconds per assertion. This run time is for a 1GHz processor. The time is scaled for different processor speeds.

NOTE: *If you specify less time to this parameter, the RCA debug data spreadsheet may not get generated due to incomplete functional analysis.*

Used by	*CDC Verification Rules*
Options	Positive integer value
Default value	20
Example	
Console/Tcl-based usage	`set_parameter fa_atime 100`
Usage in goal/source files	`-fa_atime=100`

fa_atsrc

Specifies if the *Ac_cdc08* and *Clock_sync03a* rules should check for gray-encoding at the output of source instance.

By default, these rules check for gray-encoding at the input of the destination instance with respect to the source clock.

Used by	*Ac_cdc08* and *Clock_sync03a*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_atsrc yes`
Usage in goal/source files	`-fa_atsrc=yes`

fa_audit

NOTE: *This parameter will be deprecated in a future SpyGlass release. Use the fa_msgmode parameter instead of this parameter.*

Enables some SpyGlass CDC solution rules to explore assertion checking opportunities in a design without performing actual formal analysis.

When the `fa_audit` parameter is set to `yes`, SpyGlass does not perform functional analysis. However, messages of these rules are still reported.

When the `fa_audit` parameter is not set (default is `no`), SpyGlass CDC solution performs functional analysis.

On passing *cdc_dump_assertions* with `fa_audit`, the design is run in `audit` mode, and `SVA` for all assertions of the following rules is generated:

- *Ac_datahold01a*
- *Ac_cdc01*
- *Ac_conv02*

Used by	*CDC Verification Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_audit yes`
Usage in goal/source files	`-fa_audit=yes`

fa_c2c_max_cycles

Sets a limit for the *Maximum Cycle Count* checked by the *Ac_sanity07* rule.

Used by	*Ac_sanity07*
Options	Positive integer value
Default value	100
Example	
Console/Tcl-based usage	`-fa_c2c_max_cycles=50`
Usage in goal/source files	`set_parameter fa_c2c_max_cycles 50`

fa_enable_crpt

Configures SpyGlass CDC rules to generate a spreadsheet showing details of SVA constraints affecting each rule violation.

For information on this spreadsheet, refer to the *Using SystemVerilog Assertions application note*.

Used by	Refer to the Using SystemVerilog Assertions application note
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_enable_crpt yes`
Usage in goal/source files	`-fa_enable_crpt = yes`

fa_dump_hybrid

Specifies if SVA should be generated for all the types of assertions, such as pass, failed and partially-proved.

By default, SVA is generated only for partially-proved assertions.

Set this parameter to `all` to generate SVA for all the types of assertions, and set it to `none` to disable SVA generation. Set it to `+fail` to generate SVA for failed and partially-proved assertions.

Used by	*Ac_datahold01a*, *Ac_cdc01*, *Ac_conv02*
Options	none, all, or comma-separated list of partial, fail, or +fail
Default value	partial
Example	
Console/Tcl-based usage	`set_parameter fa_dump_hybrid partial,fail`
Usage in goal/source files	`-fa_dump_hybrid=partial,fail`

fa_flopcount

Specifies a maximum number of flip-flops so that an input cone of SpyGlass CDC solution properties can be abstracted by cutting the logic behind the specified number of flip-flops in that cone.

While running SpyGlass CDC solution analysis on full-chips, the cone of SpyGlass CDC solution properties can be very complex in terms of number of flip-flops. This results in significant time spent on verification.

To circumvent this problem, use this parameter to limit the number of flip-flops to abstract input cones. Using this parameter also increases the chances of getting properties concluded.

It is recommended that you use this parameter only for partially-proved properties because usage of this parameter may help in concluding such properties.

NOTE: *If you use this parameter for properties that are failing, such properties may be reported as partially proved. Therefore, it is recommended that you use this parameter only on partially proved properties by using the fa_propfile parameter.*

By default, this parameter is set to `-1`, which indicates that an input cone will not be abstracted by cutting the logic behind a specific number of flip-flops in the cone of SpyGlass CDC solution properties.

Used by	*CDC Verification Rules*
Options	-1 or a positive integer value greater than 0
Default value	-1
Example	
Console/Tcl-based usage	`set_parameter fa_flopcount 2`
Usage in goal/source files	`-fa_flopcount=2`

NOTE: *If you specify any invalid value to this parameter, the default value (`-1`) is considered.*

fa_hybrid_report_hier

Specifies if the supported rules report the top-level hierarchical names in the SVA Hybrid flow.

By default, the parameter is set to `no` and the object names generated in SVA are flat names. Set this parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Used by	*Ar_resetcross01*, *Propagate_Clocks*, *Propagate_Resets*, *Ac_datahold01a*, *Ac_cdc01*, *Ac_conv02*, *Ac_glitch03*, Ac_clock_relation01
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_hybrid_report_hier yes`
Usage in goal/source files	`-fa_hybrid_report_hier=yes`

fa_grayhold

Specifies if converging synchronizers are checked for data hold.

By default, the *Ac_conv02* and *Ac_conv04* rules only check the destination instance inputs for gray encoding with respect to the source clock. Set the value of the `fa_grayhold` parameter to `yes`, so that the *Ac_conv02* and *Ac_conv04* rules also check the destination instance outputs for gray-encoding with respect to the destination clock to ensure that the data is held properly.

Used by	*Ac_conv02*, *Ac_conv04*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_grayhold yes`
Usage in goal/source files	`-fa_grayhold=yes`

fa_hide_complex_enables

NOTE: *This parameter is deprecated. Please use the fa_hide_complex_expr parameter instead of this parameter.*

Hides complex enable expressions generated in The Hybrid CDC Flow.

During SVA generation in this flow, in some cases, the SVA generated for the *Ac_datahold01a* rule refers to an enable expression that is very complex and makes the hybrid model look unreadable. For such cases, the string **<complex enable expression>** is generated instead of the actual enable expression and the SVA assertion is commented. The following example shows this string:

```
//DATAHOLD_Check_mod  #(1, "SpyGlass - Ac_datahold01a failure
(./test.v:68): DataHold failure (Source:top.data11[0],
Destination:top.q1)") ADVCDC_datahold_1 (
.data(top.data11[0]),
.enable(<complex enable expression>),
.src_clk(top.clk1),
.des_clk(top.clk2),
.rst((!top.reset)) );

//DATAHOLD_Check_mod  #(1, "SpyGlass - Ac_datahold01a failure
(./test.v:132): DataHold failure (Source:top.data21,
Destination:top.q2)") ADVCDC_datahold_2 (
.data(top.data21),
.enable(<complex enable expression>),
.src_clk(top.clk1),
.des_clk(top.clk2),
.rst((!top.reset)) );
```

To generate the actual enable expression, set this parameter to `no`.

Used by	*Ac_datahold01a*
Options	yes, no
Default value	yes
Example	

Console/Tcl-based usage	`set_parameter fa_hide_complex_enables no`
Usage in goal/source files	`-fa_hide_complex_enables=no`

fa_hide_complex_expr

Hides complex expressions in the Hybrid flow.

By default, SpyGlass generates SVA data for partially analyzed assertions of the following rules:

- Ac_datahold01a
- Ac_cdc01
- Ac_conv02
- Ac_glitch03

In some cases, the SVA data might refer to an expression that is very complex and makes the hybrid model unreadable. The `fa_hide_complex_expr` parameter is used to configure if such expressions are generated or not. The following table shows the various values of the `fa_hide_complex_expr` parameter and the corresponding behavior.

Parameter Value	Description
yes (Default Value)	Does not generate data for expression containing complex macros such as adder and subtractor
suppress_macros	Does not generate data for expressions containing complex macros as well as simple macros such as incrementers
no	Generates data for all expressions

For the Ac_datahold01a rule, if the enable expression is complex, the string `<complex enable expression>` is written instead of the actual enable expression in the `Ac_datahold01a` SVA assertion. Set this parameter to `no` to generate the actual enable expression.

Used by	*Ac_datahold01a*
Options	yes, no, suppress_macros
Default value	yes
Example	

Console/Tcl-based usage	`set_parameter fa_hide_complex_expr no`
Usage in goal/source files	`-fa_hide_complex_expr=no`

fa_holdmargin

Controls the hold margins between data and clock (*Ac_cdc01* rule).

Value	Description
0	No hold margin is required. This means that the data should hold until the next active edge of the verification clock.
1	A minimum hold margin of one clock edge is required with respect to each data change. This means that the data should hold strictly over the next active edge of the verification clock.

In addition to the hold margin requirement, the tool requires the minimum setup margin of one clock edge.

For example, consider the scenario shown in the following figure:

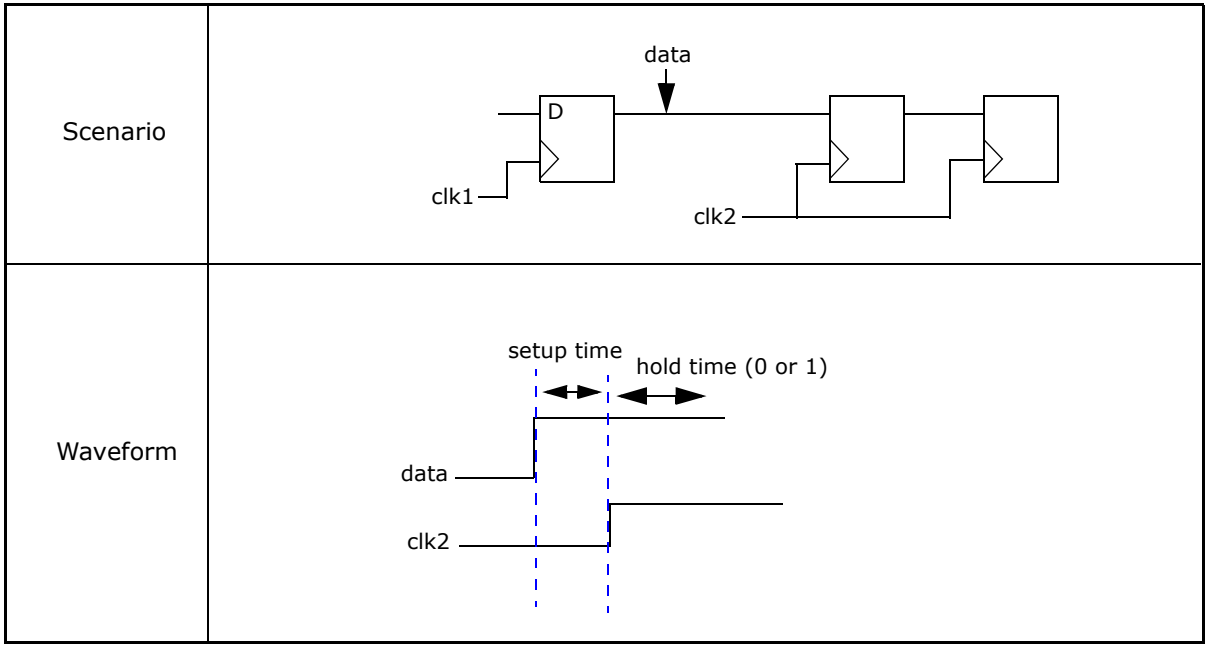


FIGURE 51. Controlling the data hold margin between data and clock

In the above scenario, you can specify the hold time so that after the clock

edge, data remains stable for the specified hold time.

Used by	*Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*
Options	0, 1
Default value	1
Example	
Console/Tcl-based usage	`set_parameter fa_holdmargin 0`
Usage in goal/source files	`-fa_holdmargin=0`

fa_holdmargin_window

Controls data enable sequencing, as checked by the *Ac_datahold01a* rule.

Used by	*Ac_datahold01a*
Options	0, 1
Default value	0
Example	
Console/Tcl-based usage	`set_parameter fa_holdmargin_window 0`
Usage in goal/source files	`-fa_holdmargin_window=0`

The following table describes the meaning of each value accepted by this parameter:

Value	Description
0	Data change can exactly coincide with the enable change.
1	A minimum margin of one clock edge is required between data and enable change.

NOTE: *Clock edges mentioned in the above refer to a positive or negative edge of any clock in the fan-in cone of the crossing.*

For example, consider the scenario shown in the following figure:

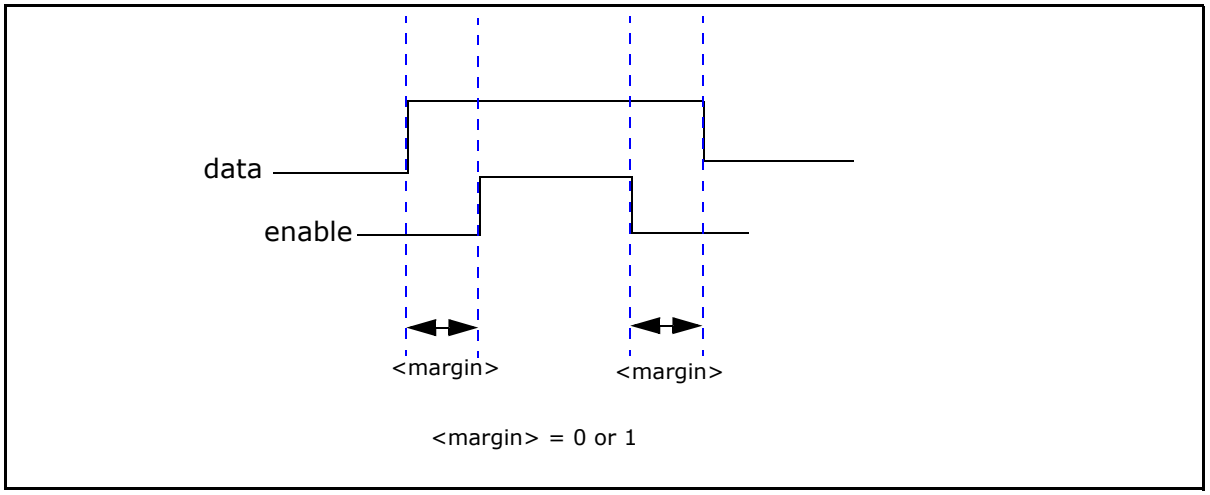


FIGURE 52. Controlling margin between the data and enable change

In the above scenario, you can use the `fa_holdmargin_window` parameter to control the margin (0 or 1) between the data and enable change.

fa_ieffort

This parameter is used to change the effort of the tool put in initial state search during design simulation.

By default, during initial state search, the tool first applies asynchronous set/resets on a design and then performs clocked simulation.

Clock Simulation

Clocked simulation is performed for a fixed number of cycles (200 cycles) until any of the following occurs:

- A non-X value reaches on all flip-flops.
- No improvement is observed for a fixed number of consecutive cycles (also referred to as waste cycles).

This waste cycle number is 10 for the reset simulation stage, and it is 20 for the data simulation stage.

If you set the value of the `fa_ieffort` parameter to a positive integer value (say, `N`), the tool performs the following steps:

1. It multiplies the simulation cycle count and waste cycle count with `N` (which in effect multiplies the time spent in initial state search by `N`).
2. It deactivates sets/resets, applies random values to the input ports, and performs clocked simulation for the total number of cycles calculated in the above step.

The tool performs the above steps over and above the default behavior of applying asynchronous set/resets on a design and then performing clocked simulation.

Setting the `fa_ieffort` parameter to a negative value (`-1` to `-3`) produces different results, as explained in the following table:

Value	Result
-1	Complete initial state search is skipped all together
-2	Initial state of all flip-flops to forced to 0
-3	Initial state of all flip-flops is forced to 1

Used by	*CDC Verification Rules*
Options	-3 to any positive integer value
Default value	0
Example	
Console/Tcl-based usage	`set_parameter fa_ieffort 2`
Usage in goal/source files	`-fa_ieffort=2`

fa_meta

Use this parameter to enable formal modeling of metastability in *CDC Verification Rules*.

If the fan-in of an assertion has a signal that is synchronized using the *Conventional Multi-Flop Synchronization Scheme*, SpyGlass injects metastability if the following conditions are met:

- There is a change in the asynchronous data and no change in the synchronous data at the source side of the crossing.
- The source and destination clock edges of the crossing occur at the same time.

Used by	*CDC Verification Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_meta yes`
Usage in goal/source files	`-fa_meta=yes`

fa_minimize_witness

Minimizes a witness when insignificant signals for a failure are removed. This enables you to identify the signals that should be ignored while analyzing a witness in the *Waveform Viewer*.

A signal may be insignificant for the entire witness depth and is shown red in the *Waveform Viewer*, or the signal may be significant for some cycles and is marked as *don't-care* for rest of the cycles.

Consider the design and its constraints as shown in the following figure:

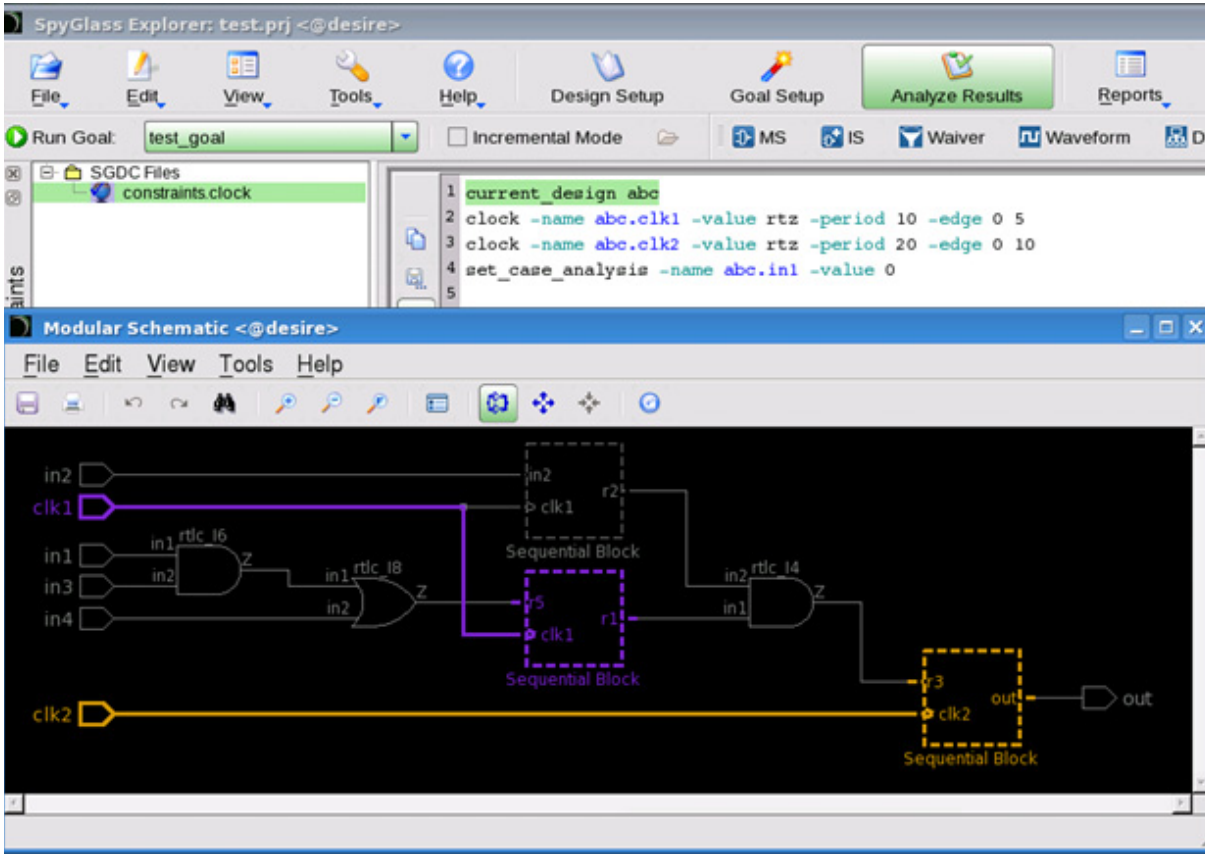


FIGURE 53.

In the above design, `in1` is constrained to `0` during a functional rule check.

When this parameter is set to `yes`, then after minimization:

- `in3` is shown as irrelevant for all the cycles because `in1` was constrained to 0.
- `in2` is shown as *don't care* for all the cycles except for third cycle.
- `in4` is shown as don't care for some cycles.

The above changes are shown in the *Waveform Viewer* in the following figures:

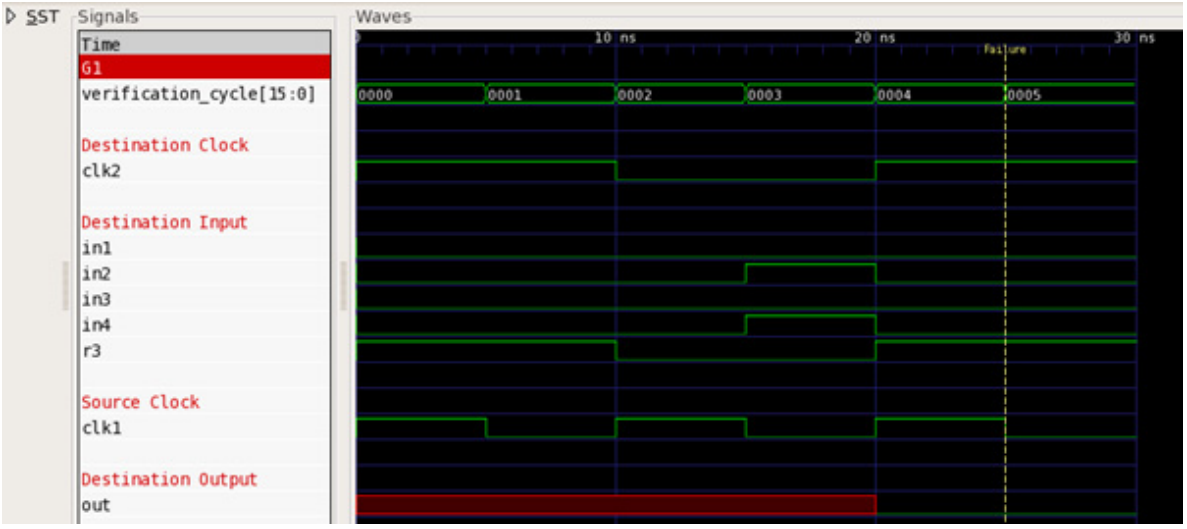


FIGURE 54.

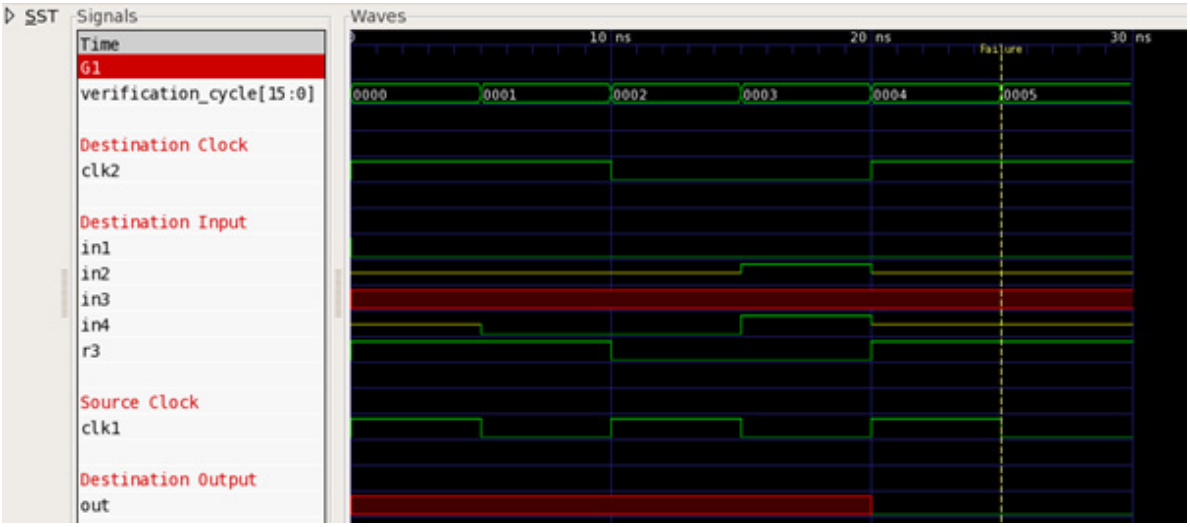


FIGURE 55.

Used by	*CDC Verification Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_minimize_witness yes`
Usage in goal/source files	`-fa_minimize_witness=yes`

fa_modulelist

Specifies a list of modules for which the functional analysis is to be performed.

By default, SpyGlass analyzes the user-defined properties and implicit properties for all design units in the user design. If you specify some particular design units with the `fa_modulelist` parameter, SpyGlass analyzes only the specified design units at the highest level. Properties at lower levels of the specified design units or in the remaining design units of the user design are not analyzed. However, the complete design is considered while determining the fan-in/fan-out of signals being checked.

Used by	*CDC Verification Rules*
Options	Space-separated or comma-separated list of design unit names enclosed in double quotes
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter fa_modulelist "Fsm Fsm_always"` `set_parameter fa_modulelist "Fsm,Fsm_always"`
Usage in goal/source files	`-fa_modulelist=Fsm Fsm_always` `-fa_modulelist=Fsm,Fsm_always`

fa_msgmode

Specifies how assertions reported by the *CDC Verification Rules* should be dumped.

This parameter accepts the following values:

- `fail`: Only the *Failed* assertions are reported.
- `pass`: Only the *Passed* assertions are reported.
- `pp`: Only the *Partially Proved* assertions are reported.
- `all`: All the assertions, such as *Failed*, *Passed*, and *Partially Proved* are reported.
- `no_msg`: Messages of *CDC Verification Rules* are dumped only in the *adv_cdc.rpt* and *adc_cdc.prp* files. No message of these rules is dumped in the *moresimple.rpt* and *spyglass.vdb* files.
- `none`: Messages of *CDC Verification Rules* are dumped in the *moresimple.rpt* and *spyglass.vdb* files. No message of these rules is dumped in the *adv_cdc.rpt* or *adv_cdc.prp* file.

  In addition, no *Assertions* are dumped in this case.
- `audit`: Messages of *CDC Verification Rules* are dumped in the *adv_cdc.rpt*, *adc_cdc.prp*, *moresimple.rpt*, and *spyglass.vdb* files.

  Assertions in this case are dumped only when the *cdc_dump_assertions* is set to `sva`.

Used by	*CDC Verification Rules*
Options	`all`, `none`, or a combination of `fail`, `pp` (partially proved), `pass`, `no_msg`, and `audit` as a comma-separated or space-separated list. The `none` option is used to disable functional analysis. This is most useful for the *Clock_sync03a*, *Ac_conv02*, and *Ac_glitch03* rules, which perform functional as well as structural checks. When you set the value of this parameter to `none`, these rules are reported with the `DISABLED` status.
Default value	fail, pp, coverage

Default Value in GuideWare2.0	all (in the *cdc/cdc_verify* goal) none (in the *cdc/cdc_verify_struct* goal)
Example	
Console/Tcl-based usage	`set_parameter fa_msgmode "pp,pass"`
Usage in goal/source files	`-fa_msgmode=pp,pass`

fa_multicore

Specifies if SpyGlass should invoke its multi core engine to solve complex assertions that are otherwise hard to solve.

It is recommended to use this feature on machines that have multiple cores.

Since using the multi core engine may be memory intensive, it is recommended to run SpyGlass initially without enabling this engine. After the first run is complete, use the Propfile flow (*fa_propfile*) to run only the unsolved assertions with this engine and with a higher value of *fa_atime*.

NOTE: *When this parameter to* `yes`*, an average depth of zero may get reported for some partially analyzed assertions.*

Used by	*CDC Verification Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_multicore yes`
Usage in goal/source files	`-fa_multicore=yes`

fa_num_cores

Optimizes multi-core engines for the given CPU cores.

The value specified to this parameter specifies the number of cores to be used by a multi-core engine.

The value of 0 means that all available cores can be used.

Used by	*CDC Verification Rules*
Options	0, 2, 4, 8
Default value	0
Example	
Console/Tcl-based usage	`set_parameter fa_num_cores 4`
Usage in goal/source files	`-fa_num_cores=4`

fa_opt_clock_fsm

Optimizes user-specified clock periods so that simpler clock FSMs can be created and the ratio between periods of synchronous clocks may be maintained.

Synchronous clocks are the clocks belonging to the same domain.

Used by	*CDC Verification Rules*
Options	Following are the options: • sync Specify this option to optimize synchronous clock periods such that the ratio between periods of clocks belonging to the same domain can be maintained. • both Specify this option to optimize synchronous clocks such that their original ratios are maintained. This option also optimizes asynchronous clocks. This optimization occurs to further reduce *Design Period* and *Design Cycle*. • none Specify this option to turn off clock FSM optimization. When you specify this option, SpyGlass rounds-off clock periods to the nearest multiple of 0.5, irrespective of whether clocks are synchronous or asynchronous. The drawback of specifying this option is that it rounds-off clock periods to lower the design cycle. Although this results in faster functional analysis, this also results in loss of original ratio between clock periods.
Default value	sync
Example	
Console/Tcl-based usage	`set_parameter fa_opt_clock_fsm both`
Usage in goal/source files	`-fa_opt_clock_fsm=both`

The following examples show how different values of this parameter modify

the original SGDC file:

Original SGDC	clock -name c1 -domain d1 -period 3.3 clock -name c2 -domain d1 -period 6.58 clock -name c3 -domain d1 -period 9.889 clock -name c4 -domain d2 -period 0.4 clock -name c5 -domain d2 -period 0.79
set_parameter fa_opt_clock_fsm sync (default) (Design Period: 79.2)	**Original SGDC modified to the following:** clock -name c1 -domain d1 -period 3.3 clock -name c2 -domain d1 -period 6.6 clock -name c3 -domain d1 -period 9.9 clock -name c4 -domain d2 -period 0.4 clock -name c5 -domain d2 -period 0.8
set_parameter fa_opt_clock_fsm both (Design Period: 19.2)	**Original SGDC modified to the following:** clock -name c1 -domain d1 -period 3.2 clock -name c2 -domain d1 -period 6.4 clock -name c3 -domain d1 -period 9.6 clock -name c4 -domain d2 -period 0.4 clock -name c5 -domain d2 -period 0.8
set_parameter fa_opt_clock_fsm none (Design Period: 910)	**Original SGDC modified to the following:** clock -name c1 -domain d1 -period 3.5 clock -name c2 -domain d1 -period 6.5 clock -name c3 -domain d1 -period 10 clock -name c4 -domain d2 -period 0.5 clock -name c5 -domain d2 -period 1.0

fa_parallelfile

Specifies a configuration file for distributed runs of Advanced SpyGlass CDC solution rules over several machines.

The configuration file is an ASCII text file that contains specific lines for different methods, as described below:

- The `lsf` method contains the following lines:

```
LOGIN_TYPE: lsf
MAX_PROCESSES: <num>
LSF_CMD: <bsub-command>
```

The following table describes the arguments and keywords of the above method:

Argument/Keyword	Description
<num>	Specifies the maximum number of processes to be spawned.
<bsub-command>	Specifies the LSF invocation command. (default is `bsub`). SpyGlass generates details of the bsub command, which is used in parallel LSF runs, in a log file. This information is useful while debugging. To generate complete information of the bsub command, set the *fa_verbose* parameter to 2.

NOTE: *To know the runtime details of SpyGlass CDC rules that are run on same or different machines, refer to The Distributed Time Report.*

NOTE: *In a parallel file specified by the* `fa_parallelfile` *parameter, the* `-I`*,* `-Ip`*, and* `-Is` *options of the* `bsub` *command are not allowed in the* `LSF_CMD` *keyword. This is because while running the* `bsub` *command, SpyGlass internally passes the* `-K` *option, which is mandatory for running parallel assertion runs. However, the* `bsub` *command does not allow the* `-K` *option along with the* `-I`*,* `-Ip`*, and* `-Is` *options. Therefore, if you specify these options, parallel assertions are not run, and the assertion status may remain partially-proved.*

The following is an example of the `lsf` method:

```
LOGIN_TYPE: lsf
MAX_PROCESSES: 3
LSF_CMD: bsub -q "normal | priority"
```

In the above example, the `-q` option is used to specify the queue as `normal` or `priority`.

The `LSF_CMD` command should contain necessary options required to run the `bsub` command in a particular environment. In most cases, the `bsub` options that are required to launch the main SpyGlass run should be passed through `LSF_CMD` so that child processes launched on `bsub` are run using the same `bsub` options.

- 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>]
...
```

The arguments and keywords of the above method are explained below:

- Specify the value of the `LOGIN_TYPE` keyword as `rsh` or `ssh` as per your requirement.
- The *<num>* argument for the `MAX_PROCESSES` keyword specifies the maximum number of processes to be spawned.
- The *<machine1-name>*, *<machine2-name>*,... arguments refer to the machine names.
- The *<num-process>* argument refers to the number of processes to be spawned on the specified machine. By default, the value of this argument is `1`.

NOTE: *Each spawned process uses one* `Advanced_CDC` *license.*

NOTE: *If any issues are found in the parallel file, the Ac_sanity06 reports a violation.*

By default, this parameter is not set to any value and, therefore, distributed runs are not enabled.

Used by	*CDC Verification Rules*
Options	File name
Default value	""
Example	
Console/Tcl-based usage	`set_parameter fa_parallelfile 'machinelist.txt'`
Usage in goal/source files	`-fa_parallelfile='machinelist.txt'`

fa_passfail

Specifies if SpyGlass CDC solution should check properties for proof, failure, or both.

The allowed values of the `fa_passfail` parameter are as follows:

Value	Description
pass	Enables pass-centric checking. Specify this value if you expect more properties to pass in your design.
fail	Enables fail-centric checking. Specify this value if you expect more properties to fail in your design.
both (Default)	Enables both pass-centric and fail-centric checking.

Use the `fa_passfail` parameter as per your design characteristics. Setting it to `pass` or `fail` may result in improved run-time performance. In all cases, both "Proved" and "Failed" cases are reported.

Used by	*CDC Verification Rules*
Options	pass, fail, both
Default value	both
Example	
Console/Tcl-based usage	`set_parameter fa_passfail pass`
Usage in goal/source files	`-fa_passfail=pass`

fa_preprocess_engine

Specifies the optimization techniques, such as isomorphic reduction and re timing to be used during functional verification.

By default, the retiming optimization technique is used.

Used by	*CDC Verification Rules*
Options	none, retime, iso, all
Default value	retime
Example	
Console/Tcl-based usage	`set_parameter fa_preprocess_engine iso`
Usage in goal/source files	`-fa_preprocess_engine=iso`

fa_propfile

Specifies a property file that contains properties to be checked for verification. Functional analysis is done only for the properties that are enabled in the specified property file.

SpyGlass CDC solution generates the *adv_cdc.prp* property file in the *<run-dir>/spyglass_reports/clock-reset* directory. This file contains all the properties analyzed in the current run. However, only the partially proved properties are enabled in the file so that you can perform incremental run by using this file through this parameter.

Used by	*CDC Verification Rules*
Options	Name of property file
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter fa_propfile adv_cdc.prp`
Usage in goal/source files	`-fa_propfile=adv_cdc.prp`

fa_resetoff

Disables all user-supplied `reset` constraints. By default, all user-supplied `reset` constraints are applied.

Used by	*CDC Verification Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter fa_resetoff yes`
Usage in goal/source files	`-fa_resetoff=yes`

fa_scope

Defines the scope of functional analysis.

By default, the `fa_scope` parameter is set to `chip` and the complete fan-in cone of the assertion is taken into account.

Set the `fa_scope` parameter to `block` to have SpyGlass CDC solution cut all signals in the fan-in cone (except clocks and resets) at the sub-module boundary in which the assertion is formed. All those signals at the boundary of the sub-module are then treated as primary inputs for functional analysis.

Used by	*CDC Verification Rules*
Options	chip, block
Default value	chip
Example	
Console/Tcl-based usage	`set_parameter fa_scope block`
Usage in goal/source files	`-fa_scope=block`

fa_seqdepth

Specifies a maximum sequential depth so that an input cone of SpyGlass CDC solution properties can be abstracted by cutting the logic behind the specified depth in that cone.

While performing SpyGlass CDC solution analysis on full-chips, the cone of SpyGlass CDC solution properties can be very complex in terms of a sequential depth. This results in significant time spent for verification.

To circumvent this problem, use this parameter to limit the sequential depth to abstract input cones. Limiting the sequential depth also increases the chances of getting properties concluded.

It is recommended that you use this parameter only for partially-proved properties because usage of this parameter may help in concluding such properties.

NOTE: *If you use this parameter for properties that are failing, such properties may be reported as partially proved. Therefore, it is recommended that you use this parameter only on partially proved properties by using the fa_propfile parameter.*

Setting this parameter to `-1` indicates that an input cone will not be abstracted by cutting the logic behind a specific depth in the cone of SpyGlass CDC solution properties.

Used by	*CDC Verification Rules*
Options	-1 or a positive integer value greater than 0
Default value	-1
Example	
Console/Tcl-based usage	`set_parameter fa_seqdepth 2`
Usage in goal/source files	`-fa_seqdepth=2`

NOTE: *If you specify any invalid value to this parameter, the default value (`-1`) is considered.*

fa_vcdtime

This parameter is deprecated. Use the *simulation_data* constraint instead of this parameter.

fa_vcdfile

This parameter is deprecated. Use the *simulation_data* constraint instead of this parameter.

fa_vcdfulltrace

Specifies the type of data that is to be dumped to the VCD file. It specifies whether all signals or only user signals in the fan-in cone of an assertion are dumped in the VCD file. You can set the `fa_vcdfulltrace` parameter to the following values:

Value	Description
no	Only the flip-flop output signals and primary inputs in the fan-in cone of an assertion are written to the VCD file
usernets (default)	All the user nets in the fan-in cone of an assertion are written to the VCD file
allnets	All internally generated nets along with the user-defined signals in the fan-in cone of an assertion are written to the VCD file

Used by	*CDC Verification Rules*
Options	no, usernets, allnets
Default value	usernets
Default Value in GuideWare2.0	allnets
Example	
Console/Tcl-based usage	`set_parameter fa_vcdfulltrace allnets`
Usage in goal/source files	`-fa_vcdfulltrace=allnets`

fa_verbose

Specifies the verbosity level of SpyGlass CDC solution messages printed at the standard output.

You can set the `fa_verbose` parameter to values 0 (default), 1, 2, and 3. The higher the value, more messages are printed.

Used by	*CDC Verification Rules*
Options	0, 1, 2, 3,4
Default value	0
Example	
Console/Tcl-based usage	`set_parameter fa_verbose 2`
Usage in goal/source files	`-fa_verbose=2`

fa_verif_cycles

Specifies the maximum number of verification cycles for a clock.

By default, the value of this parameter is set to `1024`, and the *Ac_clockperiod03* rule reports clocks for which total number of verification cycles is greater than 1024.

You can specify any positive integer value to specify a different maximum number of verification cycles.

Used by	*Ac_clockperiod03*
Options	Positive integer value greater than 2 and less than 65535
Default value	1024
Example	
Console/Tcl-based usage	`set_parameter fa_verif_cycles 100`
Usage in goal/source files	`-fa_verif_cycles=100`

fa_verify_slow_to_fast

By default, data loss is checked only for fast-to-slow crossings. When this parameter is set to a specific number, data loss is also checked for slow-to-fast crossings where the percentage of the slow clock-period with reference to the fast clock-period is greater than this number.

Used by	*CDC Verification Rules*
Options	Positive integer value greater than or equal to 0
Default value	100
Example	
Console/Tcl-based usage	`set_parameter fa_verify_slow_to_fast 80`
Usage in goal/source files	`-fa_verify_slow_to_fast=80`

fa_vcdscopename

This parameter is deprecated. Use the *simulation_data* constraint instead of this parameter.

false_path_enable_hier_view

By default, hierarchical terminals are not supported correctly in the *cdc_false_path* constraint. For example, if the *cdc_false_path* constraint is specified on a hierarchical terminal, SpyGlass CDC moves that constraint on the connected net of the hierarchical terminal.

Set the `false_path_enable_hier_view` parameter to `yes` to correctly support hierarchical terminals where the specified *cdc_false_path* constraint is retained on the hierarchical terminal itself.

Used by	*Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02*, and *Ar_cross_analysis01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter false_path_enable_hier_view yes`
Usage in goal/source files	`-false_path_enable_hier_view=yes`

filter_named_clocks

Specifies a list of strings such that SpyGlass does not infer clocks whose names contain the specified strings.

By default, SpyGlass infers clocks if their names contain the string `clk`, `clock`, or `ck`, even if you specify these strings to the `filter_named_clocks` parameter. To disable the inferring of such clocks, specify `no_default_list` to this parameter along with the other strings to be ignored. Refer to the example in the table below for further clarity.

Used by	*Clock_info01*, *Clock_info03c*, *Clock_info05*, *Clock_info05a*, *Clock_info06*, *Clock_info07*, *Clock_info14*, *Clock_info16*, *Propagate_Clocks*, *Clock_sync03a*, *Clock_sync03b*, *Clock_sync05*, *Clock_sync06*, *Clock_sync08a*, *Clock_sync09*, *Reset_sync01*, *Reset_sync02*, *Reset_sync03*, *Reset_sync04*, *DeltaDelay01*, *DeltaDelay02*, *Clock_check02*, *Clock_check03*, *Clock_check04*, *Clock_check05*, *Clock_check06a*, *Clock_check06b*, *Clock_check07*, *Clock_glitch01*, *Clock_glitch02*, *Clock_glitch03*, *Clock_Reset_info01*, *Clock_converge01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Clock_info05b*, *Clock_Reset_check01*, *Clock_Reset_check01*, *Clock_info02*, *Clock_info03a*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_xclock01*, *Ac_coherency06*, *Ar_resetcross01*
Options	Comma or space-separated list of strings
Default value	rst, reset, scan, set
Example	

Console/Tcl-based usage	`set_parameter filter_named_clocks "rst,scan"`
	
	In this case, if the name of a clock contains the rst or scan string and it does not contain the clk, clock, or ck string, SpyGlass does not infer such clocks.
	
	However, if the name contains the rst or scan string as well as the clk, clock, or ck string, SpyGlass infers such clocks. In this case, specify no_default_list along with rst and scan to this parameter, as shown below, to disable the inferring of such clocks:
	`set_parameter filter_named_clocks "rst,scan", no_default_list`

Usage in goal/source files	`-filter_named_clocks=rst,scan`

filter_named_resets

Specifies a list of strings such that SpyGlass does not infer asynchronous resets whose names contain the specified strings.

By default, SpyGlass infers resets if their names contain the string `rst`, `reset`, `set`, or `res`, even if you specify these strings to the `filter_named_resets` parameter. To disable the inferring of such resets, specify `no_default_list` to this parameter along with the other strings to be ignored. Refer to the example in the table below for further clarity.

Used by	*Clock_Reset_info01*, *Propagate_Resets*, *Reset_info01*, *Reset_info02*, *Reset_info09a*, *Reset_sync01*, *Reset_sync03*, *Reset_sync04*, *Reset_check04*, *Reset_check06*, *Reset_check10*, *Reset_check11*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_conv02*, *Ar_sync01*, *Ar_unsync01*, *Ar_asyncdeassert01*, *Ar_resetcross01*, and *Ar_syncdeassert01*
Options	Comma or space-separated list of strings, no_default_list
Default value	clk, clock, scan
Example	

Console/Tcl-based usage	`set_parameter filter_named_resets "clk,scan"`
	
	In this case, if the name of a reset contains the clk or scan string and it does not contain the rst, reset, set, or res string, SpyGlass does not infer such resets.
	
	However, if the name contains the clk or scan string as well as the rst, reset, set, or res string, SpyGlass infers such resets. In this case, specify no_default_list along with clk and scan to this parameter, as shown below, to disable the inferring of such resets:
	`set_parameter filter_named_resets "clk,scan", no_default_list`

Usage in goal/source files	`-filter_named_resets=clk,scan`

filter_clock_converge_on_cdc

Filters clock signals that converge on a mux that is reaching the source or destination of a CDC crossing.

Set the `filter_clock_converge_on_cdc` parameter to `yes` to enable the *Clock_info05* rule to report only the clock signals that converge on a Mux that is reaching the source or destination of a CDC crossing.

Used by	*Clock_info05*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter filter_clock_converge_on_cdc yes`
Usage in goal/source files	`-filter_clock_converge_on_cdc=yes`

formal_setup_rules_check

Runs the *Ac_clockperiod02*, *Ac_initstate01*, *Ac_sanity01*, *Ac_sanity02*, *Ac_sanity06*, and *Ac_init01* rules, even without enabling the advanced SpyGlass CDC solution rules. This is used in formal setup step.

By default, the value of this parameter is set to `no`, and the above-specified rules are not run. Set the value of this parameter to `yes` to run these rules.

Used by	*Ac_clockperiod02*, *Ac_initstate01*, *Ac_sanity01*, *Ac_sanity02*, *Ac_sanity06*, and *Ac_init01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter formal_setup_rules_check yes`
Usage in goal/source files	`-formal_setup_rules_check=yes`

format_report

Specifies that the text in the reports of SpyGlass CDC solution should get wrapped up. By default, the width of each column in the reports is dependent on the length of information. You can format the display of information by wrapping up the text by setting the `format_report` parameter to `yes`.

Used by	*Propagate_Clocks*, *Clock_info01*, *Clock_info02*, *Reset_info01*, *Clock_info03a*, *Clock_info03b*, *Clock_info03c*, *Clock_info15*,*Clock_Reset_info01*, *Reset_check04*, *Reset_sync03*, *Setup_clockreset01*, *Clock_check06a*, *Clock_check06b*, *Clock_check07*, *Ac_crossing01*, *Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter format_report yes`
Usage in goal/source files	`-format_report=yes`

gen_sync_reset_style_info

Specifies whether the spreadsheet generated by the *Reset_info01* should contain the following information about synchronous resets:

- Load
- Presence of combinational logic in reset path
- Polarity
- Presence of the `sync_set_reset` pragma
- Reset usage in first if of sequential block

You can use this information to specify the *sync_reset_style* constraint for pruning synchronous reset detection.

Used by	*Reset_info01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter gen_sync_reset_style_info yes`
Usage in goal/source files	`-gen_sync_reset_style_info=yes`

generate_rfp_suppressed_violations

By default, the *reset_filter_path* constraint waives the rule violations for the rules specified in the `-type` argument of the constraint. For example, if the `-type` argument is set to `reset_sync02`, the *Reset_sync02* rule will not report violations for the reset paths specified in this constraint.

Set the `generate_rfp_suppressed_violations` parameter to a supported rule to generate a report of the such suppressed rule violations. The report is generated with the `<rule-name>.filtered.csv` name.

Used by	Ar_resetcross01, Ar_sync01, Ar_unsync01, *Reset_sync02*, Ar_syncdeassert01, Ar_unsyncdeassert01
Options	none, sync, deassert, reset_sync02, all
Default value	none
Example	
Console/Tcl-based usage	`set_parameter generate_rfp_suppressed_violations sync, reset_sync02`
Usage in goal/source files	`-generate_rfp_suppressed_violations=sync, reset_sync02`

The information included in the `<rule-name>.filtered.csv` file is similar to the information in the corresponding rule-based spreadsheet except for the `WAIVED` column which is replaced by the `SGDC-REF` column in the `<rule-name>.filtered.csv` file.

However, the `Reset_sync02.filtered.csv` report includes the following additional columns:

- `RESET-DRIVER` - Lists the driver flop/black box/port that drives the RESET signal
- `SOURCE-RESET` - Lists primary resets, if it is defined on the RESET-DRIVER or is driving the reset/clear pin of RESET-DRIVER
- `SGDC-REF` - Lists the file name and line number of all *reset_filter_path* constraints that helped to filter the corresponding violation

glitch_check_type

Defines the type of crossings that need to be considered by the *Ac_glitch02* and *Ac_glitch03* rules.

The allowed values for the `glitch_check_type` parameter are as follows:

Value	Indicates
sync_control (default)	Consider both synchronized and unsynchronized crossings using *Conventional Multi-Flop Synchronization Scheme* or *Synchronizing Cell Synchronization Scheme* along with a combinational logic in the crossing path. In these crossings, the Ac_glitch02 rule checks are performed irrespective of whether the *allow_combo_logic* constraint is specified.
unsync	Consider all unsynchronized crossings.
all	sync_control+unsync (The *Ac_glitch02* and *Ac_glitch03* rules are checked on all the unsynchronized crossings as well as synchronized crossings using multi-flop or synchronize_cells schemes along with a combinational logic in the crossing path)

Used by	*Ac_glitch02*, *Ac_glitch03*
Options	A combination of allowed values as a comma-separated list or using the + (plus) character to append to the default value
Default value	sync_control
Example	

Console/Tcl-based usage	The following specifications are equivalent:
	`set_parameter glitch_check_type all`
	`set_parameter glitch_check_type "sync_control,unsync"`
	`set_parameter glitch_check_type +unsync`
	
	The + character must be the first character when specified. Therefore, 'unsync+' is incorrect.

Usage in goal/source files	`-glitch_check_type=all`
	`-glitch_check_type=sync_control, unsync`
	`-glitch_check_type=+unsync`

glitch_on_sync_src

Specifies if synchronous sources present in the input cone of a destination signal should be considered for glitch checking.

Used by	*Ac_glitch03*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter glitch_on_sync_src yes`
Usage in goal/source files	`-glitch_on_sync_src=yes`

glitch_on_unconstrained_src

Specifies if unconstrained ports present in the input cone of a destination signal should be considered for glitch checking.

Used by	*Ac_glitch03*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter glitch_on_unconstrained_src yes`
Usage in goal/source files	`-glitch_on_unconstrained_src=yes`

glitch_protect_cell

Specifies glitch protection cell names for the *Glitch Protection Cell Synchronization Scheme*. These cell names exist in the clock path of the destination flip-flops in clock-domain crossings. A crossing is considered as synchronized if a multi-flop synchronizer exists in the other fan-in (pin not connected to the source flip-flop) of the glitch protect cell.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ar_resetcross01*, and *Ac_unsync01*
Options	Comma or space-separated list of cell names enclosed in double quotes. You can also use wildcard characters (* [zero or more] and ? [single character match]) in the cell names.
Default value	NULL
Example	

Console/Tcl-based usage	`set_parameter glitch_protect_cell GP01`
	
	You can specify multiple cell names as shown below:
	`set_parameter glitch_protect_cell "GP01,GP02,GP03"`
	
	The following specification matches all cell names that start with GP:
	`set_parameter glitch_protect_cell GP*`
	
	The following specification matches all 3 character cell names that start with GP:
	`set_parameter glitch_protect_cell GP?`

Usage in goal/source files	`-glitch_protect_cell=GP01`
	`-glitch_protect_cell="GP01,GP02,GP03"`
	`-glitch_protect_cell=GP*`
	`-glitch_protect_cell=GP?`

Set the `enable_mux_dest_domain` parameter to eliminate the need for a synchronizer where the glitch protection cell driving the first flip-flop

in the destination domain has the other input pin (not connected to the source) in the destination domain.

handle_combo_arc

Enables clock and reset propagation through a combinational arc present in the sequential library cells.

By default, this parameter is set to `no` and clock/reset propagation is stopped if SpyGlass encounters any pin other than the clock of a sequential library cell.

Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Used by	*Clock and Reset Information Rules*, *Clock Information Rules*, *CDC Verification Rules*, *Clock Checking Rules*, *Clock and Reset Checking Rules*, *Clock Glitch Checking Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter handle_combo_arc yes`
Usage in goal/source files	`-handle_combo_arc=yes`

ignore_bus_clocks

Specifies whether the *Clock_info01* rule ignores vector signals of bus-width more than or equal to the specified number while deciding clock candidates during auto detection of clocks.

By default, the `ignore_bus_clocks` parameter is set to `1024` and only vector signals of bus width less than `1024` bits are considered as clock candidates.

Set the `ignore_bus_clocks` parameter to any positive integer number to have the Clock_info01 rule ignore all vector signals of bus width equal to or more than that number.

Set the `ignore_bus_clocks` parameter to `no` to consider all the vector signals.

Set the `ignore_bus_clocks` parameter to `yes` to have the Clock_info01 rule ignore all vector signals.

Used by	*Clock_info01*
Options	yes, no, a positive integer value
Default value	1024
Example	

Console/Tcl-based usage	`set_parameter ignore_bus_clocks 512`
	The above example considers vector signals of width less than 512
	
	`set_parameter` ignore_bus_clocks yes
	The above example ignores all vector signals
	
	`set_parameter ignore_bus_clocks no`
	`The above example considers all the vector signals.`

Usage in goal/source files	`-ignore_bus_clocks=512`
	`-ignore_bus_clocks=yes`
	`-ignore_bus_clocks=no`

ignore_bus_resets

By default (`ignore_bus_resets` parameter is `yes`), asynchronous vector resets, which are not struct nets, are:

- Not reported in the `autoresets.sgdc` and `generated_resets.sgdc` files by the *Reset_info01* rule
- Not automatically detected if the *use_inferred_resets* is set to `yes`
- Not automatically detected by the *Ar_converge02* rule

Set the `ignore_bus_resets` parameter to `no` to enable detection and reporting of asynchronous vector resets.

Used by	*Reset_info01*, *Propagate_Resets*, *Ar_converge02*, *Ar_glitch01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter ignore_bus_resets no`
Usage in goal/source files	`-ignore_bus_resets=no`

ignore_set_case

Specifies that the `set_case_analysis` constraint be ignored for simulation or block path traversal for the specified rules.

Currently, only the Ar_glitch01 rule uses this parameter.

NOTE: *The* `ignore_set_case` *parameter takes rule names in comma separated format. The rules specified with the* `ignore_set_case` *parameter ignore the* `set_case_analysis` *constraint for simulation or block path traversal.*

Used by	*Ar_glitch01*
Options	Ar_glitch01, none
Default value	none
Example	
Console/Tcl-based usage	`set_parameter ignore_set_case Ar_glitch01`
Usage in goal/source files	`-ignore_set_case=Ar_glitch01`

ignore_latches

Specifies whether the rules using this parameter should ignore signals ending on latch enable terminals while deciding candidate clocks.

By default, the `ignore_latches` parameter is set to `yes` and such signals ending on the latch enable terminals are ignored. Set this parameter to no to consider such signals as clock candidates.

Used by	*Clock_info01*, *Clock_info03a*, *Clock_info03b*, *Clock_info03c*, *Clock_check01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter ignore_latches no`
Usage in goal/source files	`-ignore_latches=no`

ignore_nets_clock_path_file_name

Specifies the file containing hierarchical names of nets (one name per line) so that SpyGlass halts clock propagation along the path when any of these nets is encountered.

By default, SpyGlass reads a file named ignore_nets_clock_path.txt if found in the current directory for the net names. Use the `ignore_nets_clock_path_file_name` parameter to specify a different file in the same or different location.

NOTE: *Wildcard characters (* [zero or more] and ? [single character match]) and escape characters can be used to specify net names in this file. For example, the names,* `clk*1` *and* `\clk@2`*, are valid net names.*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*
Options	File name
Default value	ignore_nets_clock_path.txt in the current working directory
Example	
Console/Tcl-based usage	`set_parameter ignore_nets_clock_path_file_name nets.txt`
Usage in goal/source files	`-ignore_nets_clock_path_file_name=nets.txt`

ignore_num_rtl_buf_invs

Specifies the maximum number of inferred buffers or inverters that are allowed in a data transfer path between flip-flops at a clock domain crossing or between flip-flops in a multi-flop synchronizer.

By default, all buffers and inverters are allowed.

Set the `ignore_num_rtl_buf_invs` parameter to `one` to allow one, `two` to allow two, or `none` to disallow all buffers and inverters.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01*, *Ar_resetcross01*
Options	one, two, many, or none
Default value	many
Example	
Console/Tcl-based usage	`set_parameter ignore_num_rtl_buf_invs two` The above example specifies that a maximum of two buffers or inverters are allowed.
Usage in goal/source files	`-ignore_num_rtl_buf_invs=two`

ignore_race_thru_latch

Specifies whether the *Clock_Reset_check02* rule should ignore cases where a latch exists in the feedback path between the flip-flop's output pin and its clock pin while detecting race conditions.

By default, the `ignore_race_thru_latch` parameter is set to `no` and the Clock_Reset_check02 rule reports such cases.

Used by	*Clock_Reset_check02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter ignore_race_thru_latch yes`
Usage in goal/source files	`-ignore_race_thru_latch=yes`

infer_constraint_from_abstract_blocks

Specifies if the supported rules infer clock, reset, set_case_analysis from similar constraint defined in abstracted blocks.

By default, the `infer_constraint_from_abstract_blocks` parameter is set to `none` and the constraints are not inferred.

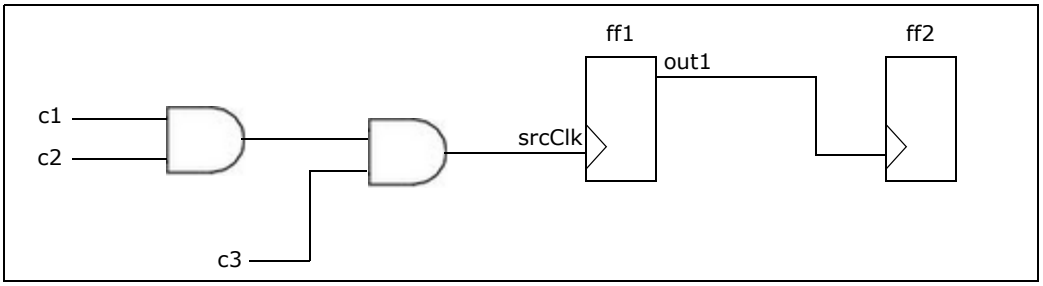
Used by	*Clock_info01*, *Reset_info01*
Options	none, clock, reset, sca
Default value	none
Example	
Console/Tcl-based usage	`set_parameter infer_constraint_from_abstract_blocks clock`
Usage in goal/source files	`-infer_constraint_from_abstract_blocks=clock`

master_clock_limit

Specifies the maximum number of master clocks for which the *generated_clock* constraints should be dumped for derived clocks when multiple master clocks reach the source of derived clocks.

These constraints are dumped in the *generated_clocks.sgdc* and *cdc_setup_generated_clocks.sgdc* files when clocks are inferred automatically after you set the *use_inferred_clocks* parameter to `yes`.

Consider the following figure in which multiple clocks, `c1` (tag `T1`), `c2` (tag `T2`), and `c3` (tag `T3`) reach the source of the derived clock `out1`:




```
// SGDC File:
clock -name c1 -period 15 -domain d2 -tag T1
clock -name c2 -period 15 -domain d2 -tag T2
clock -name c3 -period 10 -domain d2 -tag T3
```

FIGURE 56. Multiple master clocks reaching a source clock

In the above example, if you set the `master_clock_limit` parameter to 2, the *generated_clock* constraints are dumped with respect to any two master clocks:

```
generated_clock -name out1 -source srcClk -master_clock T1
-divide_by 2 -tag GT1 -add

generated_clock -name out1 -source srcClk -master_clock T2
-divide_by 2 -tag GT2 -add
```

This parameter is applicable when the value of the *clock_reduce_pessimism*

parameter is set to `all_master_clocks`.

Set this parameter to -1 if *generated_clock* constraints for a derived clock is to be generated in *generated_clocks.sgdc* with respect to all master clocks reaching its source.

Used by	All SpyGlass CDC rules
Options	Integer greater than 0
Default value	1000
Example	
Console/Tcl-based usage	`set_parameter master_clock_limit 800`
Usage in goal/source files	`-master_clock_limit=800`

msg_inst_mod_report

Specifies whether *Filtering Violations Based On Instances* should be done on the destination instance/module or the common instance/module containing both source and destination.

By default, filtering is done based on the destination instance/module. Set this parameter to `all` to filter violations based on the common instance/module containing the destination and its source.

For example, consider the following schematic of the design in which CDC reports a violation on the destination present in the `F3` instance:

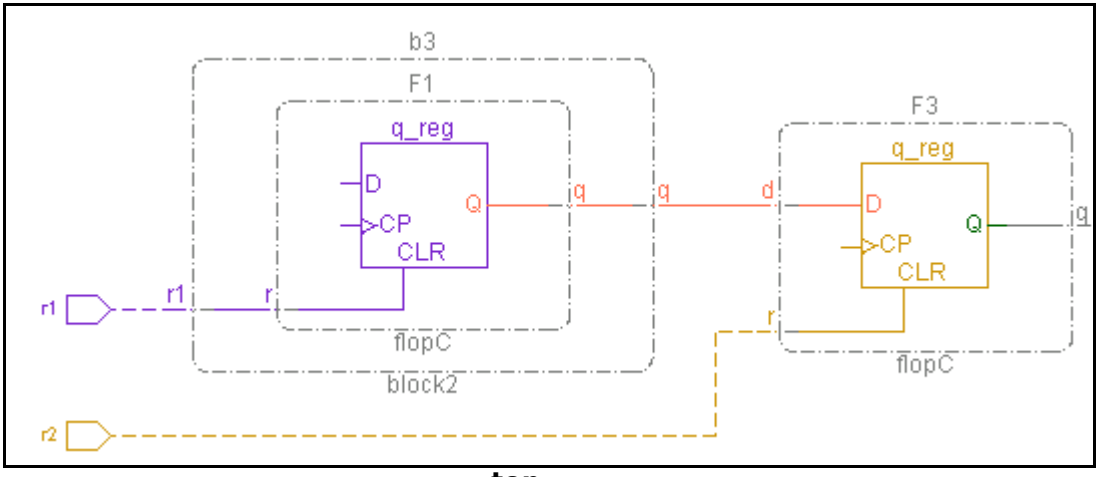


FIGURE 57.

Now, if the `msg_inst_mod_report` parameter is set to `auto`, violations are filtering based on the destination instance `F3`, as shown in the following figure:

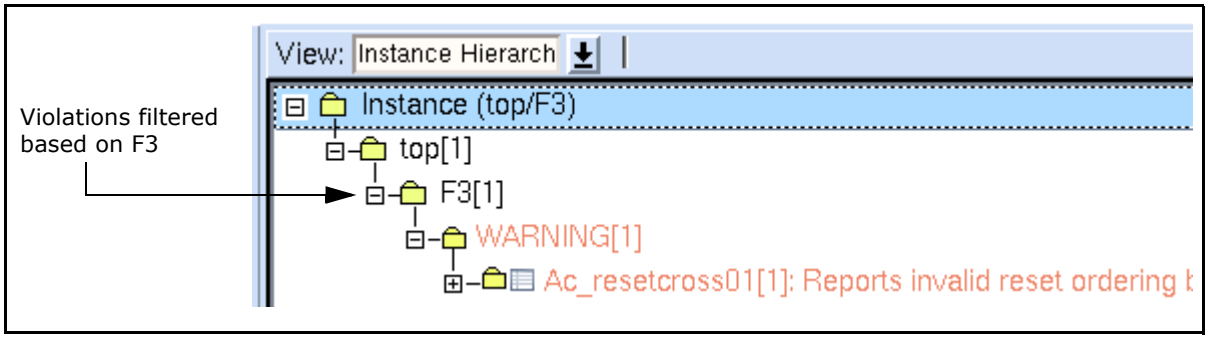


FIGURE 58. Filtering when msg_inst_mod_report is set to auto

When you set this parameter to `all`, violations are filtered based on the `top` module because it is the common module containing the `F3` destination and its source `F1` (see *Figure 57*). The following figure shows the tree view in which filtering occurs based on `top` (instead of `F3` as shown in *Figure 58*):

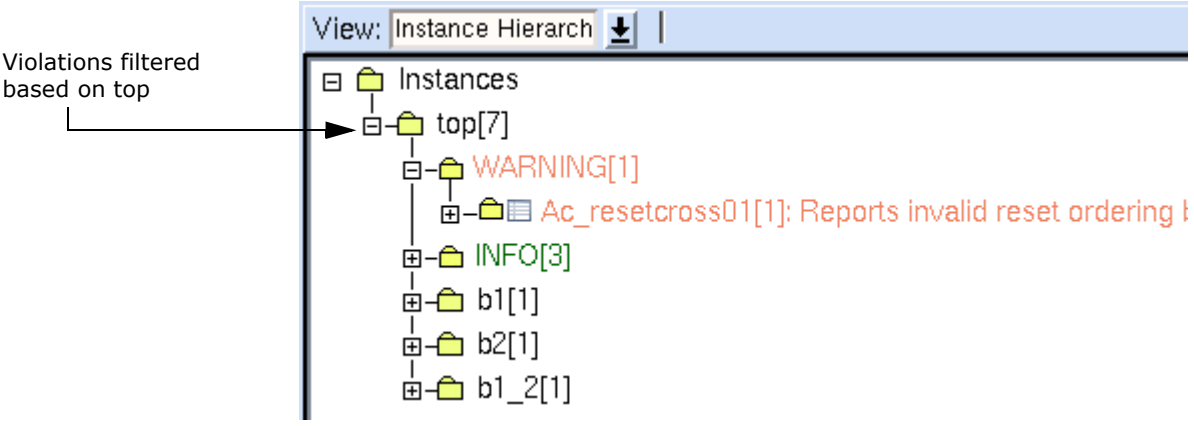


FIGURE 59. Filtering when msg_inst_mod_report is set to all

The `msg_inst_mod_report` parameter for the `Ar_resetcross01` rule supports the following options:

- `des` - Reports the name of the destination parent module only
- `src` - Reports the name of the source parent module only

- ❒ `all` - Reports the common parent for the destination and the source
- ❒ `any` - Reports the name of the source and the destination parent modules
- ❒ `auto` - Same as the value `des`. Reports the name of the destination parent module only.

Used by	All SpyGlass CDC rules
Options	auto, all
Default value	auto
Example	
Console/Tcl-based usage	`set_parameter msg_inst_mod_report all`
Usage in goal/source files	`-msg_inst_mod_report=all`

mux_search_depth

Specifies the logic depth that will be explored by the *Ac_glitch01* rule for implicit MUX detection. The logic search is performed on all possible combinations of the gates within a crossing.

By default, the parameter value is set to `6`. Increasing the value of this parameter may cause run-time increase.

Used by	*Ac_glitch01*
Options	Positive integer value greater than 1
Default value	6
Example	
Console/Tcl-based usage	`set_parameter mux_search_depth 4`
Usage in goal/source files	`-mux_search_depth=4`

netlist_name_convention

Specifies a naming convention of generated net in netlist designs.

Based on the naming convention specified, the *Ac_unsync01*/*Ac_unsync02* and *Ac_sync01*/*Ac_sync02* rules merge the violating nets.

Used by	*Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, and *Ac_unsync02*
Options	String value
Default value	No value (this parameter is inactive by default)
Example	
Console/Tcl-based usage	`set_parameter netlist_name_convention "_qx1"`
Usage in goal/source files	`-netlist_name_convention="_qx1"`

The following examples explain the usage of this parameter:

- Consider a scenario in which violations are reported for `abc_1_qx1` and `abc_2_qx1` nets. If you specify `netlist_name_convention=" _qx1"`, only one violation is reported for the `abc_[1:2]_qx1` net.
- Consider a scenario in which violations are reported for `w1` and `w2` nets. If you specify `netlist_name_convention=""`, only one violation is reported for the `wr[1:2]` net.

NOTE: *By default, this parameter is not active. You must specify it explicitly to enable naming convention.*

NOTE: *Do not specify netlist bus-merged names in an SGDC file because they are irrelevant to RTL.*

no_convergence_check

Specifies a comma or space-separated list of net names that should not be checked for convergence by the *Clock_sync03a*, *Clock_sync03b*, *Ac_conv01*, *Ac_conv02*, and *Ac_conv03* rules.

If convergence is found on a specified gate, violation is not reported on that gate if its output net has been specified through this parameter.

For example, consider the scenario shown in the following figure:

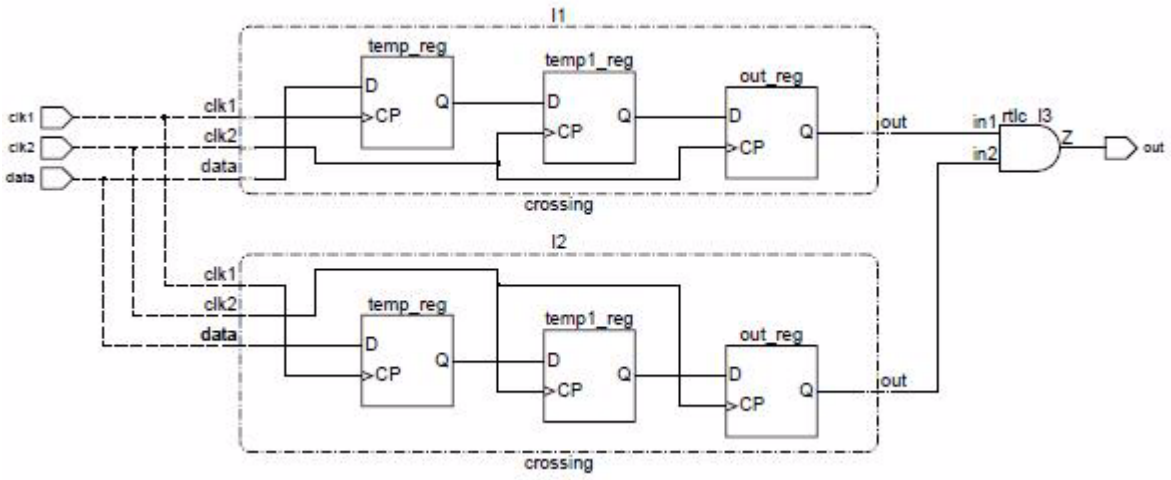


FIGURE 60. Specifying nets not to be checked for convergence

For the above scenario, consider that you set the following parameter:

```
set_parameter no_convergence_check "out"
```

In this case, the convergence on the AND gate is not reported.

NOTE: *If you specify nets by using the no_convergence_check constraint as well as the* `no_convergence_check` *parameter, SpyGlass considers the nets specified by both the constraint and parameter.*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Ac_conv01*, *Ac_conv02*, and *Ac_conv03*
Options	Comma or space-separated list of net names

Default value	NULL
Example	
Console-based usage	`set_parameter no_convergence_check "top.U1.net1,top.net2"`
Usage in goal/source files	`-no_convergence_check=top.U1.net1,top.net2`

num_flops

Specifies the minimum number of flip-flops required for synchronizing a signal using the *Conventional Multi-Flop Synchronization Scheme*.

By default, the *Conventional Multi-Flop Synchronization Scheme* marks those clock crossings as synchronized where two flip-flops are in a synchronization flip-flop arrangement. Set the `num_flops` parameter to a positive integer number to specify a different number (more than 2).

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_coherency06*
Options	Positive integer value greater than 1
Default value	2
Example	
Console/Tcl-based usage	`set_parameter num_flops 4` The above example specifies that at least four flip-flops should be used in the *Conventional Multi-Flop Synchronization Scheme*.
Usage in goal/source files	`-num_flops=4`

num_quasi_seq_elem

Specifies the sequential element depth, in numbers, that should be considered when traversing the fanout traversal for the *quasi_static* constraint. By default, the `quasi_static` constraint does not propagate through any flops/sequential elements.

Set the `num_quasi_seq_elem` parameter to a positive integer greater than or equal to zero to specify the depth. Set the parameter to `-1` to propagate the constraint through infinite number of flops/sequential elements.

Used by	All rules that use the quasi_static constraint
Options	An integer greater than or equal to -1
Default value	0
Example	
Console/Tcl-based usage	`set_parameter num_quasi_seq_elem 2` The above example specifies that the quasi_static fanout traversal happens for 2 level depth of sequential elements in the path. At the 3rd level of depth, it stops the traversal.
Usage in goal/source files	`-num_quasi_seq_elem=2`

one_cross_per_dest

By default, only one clock crossing (first found) is marked for each destination (flip-flop, latch, black box, or primary output port) by the rules even if there are more than one possible crossing from different sources.

Set the `one_cross_per_dest` parameter to `no` to report all clock crossings on each destination. Then, the Clock Synchronization rules report a synchronization status message between each source and destination.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter one_cross_per_dest no`
Usage in goal/source files	`-one_cross_per_dest=no`

prefer_abstract_port

Specifies the process of finding the source of a crossing reported by *The Ac_sync_group Rules*.

For details, see the following topics:

- *Finding the Source when prefer_abstract_port=yes*
- *Example - prefer_abstract_port=yes*
- *Finding the Source when prefer_abstract_port=no (Default mode)*
- *Example - prefer_abstract_port=no*

Used by	*The Ac_sync_group Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter prefer_abstract_port yes`
Usage in goal/source files	`-prefer_abstract_port=yes`

Finding the Source when prefer_abstract_port=yes

If both the *assume_path* and *abstract_port* constraints defined on the output pin of a black box, preference is given to the *abstract_port* constraint if the domain of the black box output pin is different from the destination domain. In this case, the black is considered as the source.

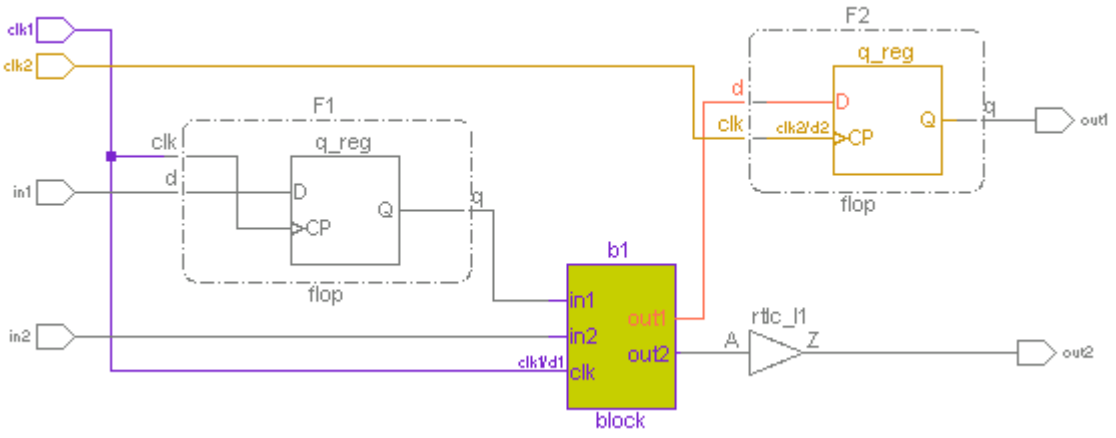
However, if the domain of the black box output pin is same as the destination domain, the *assume_path* constraint is considered to find the sources behind the black box.

Prerequisites: The *assume_path* and *abstract_port* constraints should be defined on the black box.

For details, see the example below.

Example - prefer_abstract_port=yes

Consider the following figure showing the design and SGDC files:




```
# Top-level SGDC
current_design top
clock- name clk1 -domain d1
clock -name clk2 -domain d2
sgdc -import block block.sgdc
```

```
# block.sgdc
current_design block
clock -name clk -domain d1
assume_path -name block -input in1 -output out1
abstract_port -module block -ports out1 -clock clk
```

FIGURE 61.

In the above example, to find a source, *The Ac_sync_group Rules* traverse back from the destination (`F2.q`) and stops traversal at the `out1` pin of `b1`.

Reason to stop traversal at out1: The `out1` port of `b1` receives the `clk1` clock of the `d1` domain. Although, `in1` and `out1` are connected through the path defined by the *assume_path* constraint, preference is given to the *abstract_port* constraint and `out1` is considered it as the source.

Crossing reported between F2.q and out1: The *Ac_unsync01* reports the following violation in this case:

`Unsynchronized Crossing: destination flop` **`top.F2.q`**`, clocked by top.clk2, source black-box` **`top.b1_out1`**`, clocked by top.clk1. Reason: Qualifier not found [Total Sources: 1 (Number of source`

```
domains: 1)]
```

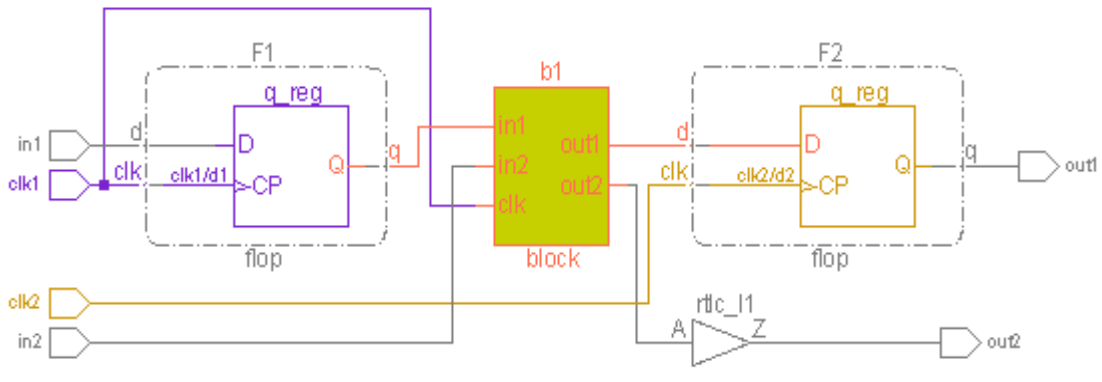
Finding the Source when prefer_abstract_port=no (Default mode)

While traversing back from the destination to find a source, preference is always given to the *assume_path* constraint even if the *abstract_port* constraint is defined.

For details, see the example below.

Example - prefer_abstract_port=no

Consider the following figure showing the design and SGDC files:



 # Top-level SGDC

```
current_design top
clock- name clk1 -domain d1
clock -name clk2 -domain d2
sgdc -import block block.sgdc
```

 # block.sgdc

```
current_design block
clock -name clk -domain d1
assume_path -name block -input in1 -output out1
abstract_port -module block -ports out1 -clock clk
```

FIGURE 62.

In the above example, to find a source, *The Ac_sync_group Rules* traverse

back from the destination (`F2.q` of the `d2` domain), ignores the `b1` black box, and stops traversal at `F1.q` of the `d1` domain.

Reason to continue traversal at out1: The `prefer_abstract_port` parameter is set to `no`. Therefore, *The Ac_sync_group Rules* ignore the *abstract_port* constraint applied on `out1` encountered in the path and preference is given to the *assume_path* constraint.

Crossing reported between F2.q and F1.q: *Ac_unsync01* reports the following violation in this case:

```
Unsynchronized Crossing: destination flop top.F2.q, clocked by
top.clk2, source flop top.F1.q, clocked by top.clk1. Reason:
Qualifier not found [Total Sources: 1 (Number of source
domains: 1)]
```

prop_clock_thru_quasi_static

Enables the propagation of clock signals through quasi static path.

By default, the parameter is set to `no`. Set this parameter to `yes` to allow propagation of clock signal through quasi static path.

Used by	*Clock_info03a*, *Clock_info03c*, *Clock_info05*, *Clock_info05a*, *Clock_info06*, *Clock_info07*, *Clock_info14*, *Clock_info16*, *Propagate_Clocks*, *Clock_sync03a*, *Clock_sync03b*, *Clock_sync05*, *Clock_sync06*, *Clock_sync08a*, *Clock_sync09*, *Reset_sync01*, *Reset_sync02*, *Reset_sync03*, *Reset_sync04*, *Reset_check07*, *DeltaDelay01*, *DeltaDelay02*, *Clock_check02*, *Clock_check03*, *Clock_check04*, *Clock_check05*, *Clock_check06a*, *Clock_check06b*, *Clock_check07*, *Clock_glitch01*, *Clock_glitch02*, *Clock_glitch03*, *Clock_Reset_info01*, *Clock_converge01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Clock_info05b*, *Clock_Reset_check01*, *Clock_Reset_check01*, *Clock_info02*, *Clock_info03a*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_xclock01*, *Setup_quasi_static01*, *Clock_hier01*, *Clock_hier02*, *Clock_hier03*, *Ac_coherency06*, *SGDC_clock_path_wrapper_module01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter prop_clock_thru_quasi_static yes`
Usage in goal/source files	`-prop_clock_thru_quasi_static=yes`

rdc_report_all_resets

Enables the *Ar_resetcross01* rule to report all resets in the reset domain crossing in rule-based spreadsheet.

By default, the parameter is set to `no` and the rule does not report all resets in the rule-based spreadsheet.

Used by	*Ar_resetcross01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter rdc_report_all_resets yes`
Usage in goal/source files	`-rdc_report_all_resets=yes`

reconvergence_stages

Specifies the maximum number of flip-flops allowed in the fan-out path of a synchronized signal as checked by the *Clock_sync03a* and *Clock_sync03b* rules.

By default, the `reconvergence_stages` parameter is set to 0. In this case, only combinational logic is allowed between the synchronized signals and the instance on which the convergence is taking place.

In case there is a sequential logic between the synchronized signals and the convergence point, set the `reconvergence_stages` parameter to a positive integer number to specify the maximum depth of sequential logic allowed in the path.

The above rules do not report a violation if the number of flip-flops in the re-convergence path is more than the specified value.

Consider the following example:

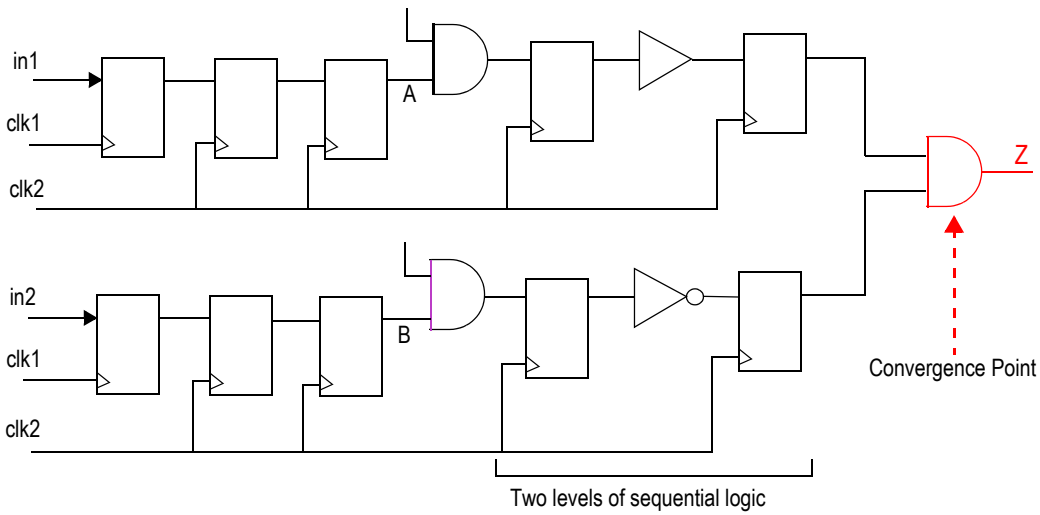


FIGURE 63. Allowing maximum two flip-flops in the fan-out path of a synchronized signal

The above figure shows two synchronized signals A and B that are converging on an AND gate after traversing two levels of sequential logic. In this case, the Clock_sync03a rule reports a violation when the `reconvergence_stages` parameter is set to 2. The rule does not report a violation when the parameter is set to a value less than equal to 1.

Used by	*Clock_sync03a* and *Clock_sync03b*
Options	0 or positive integer value
Default value	0
Example	
Console/Tcl-based usage	`set_parameter reconvergence_stages 2`
Usage in goal/source files	`-reconvergence_stages=2`

report_all_clockgate_enables

Enables the *Clock_glitch01* and *Clock_glitch02* rules to report all the enable nets that are directly merging with a clock signal.

By default, the parameter is set to no and the rules do not report all the enable nets.

Used by	*Clock_glitch01* and *Clock_glitch02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_all_clockgate_enables yes`
Usage in goal/source files	`-report_all_clockgate_enables=yes`

report_all_flops

Enables the *Reset_sync02* and *Ar_converge02* rules to list all affected flip-flops that are reset in an asynchronous domain.

By default, these rules report one message for each asynchronous reset signal that is generated in every asynchronous clock domain. Using the `report_all_flops` parameter enables these rules to report all the flip-flops whose resets are generated in asynchronous domains.

Used by	*Reset_sync02* and *Ar_converge02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_all_flops yes`
Usage in goal/source files	`-report_all_flops=yes`

report_all_sync

Generates a spreadsheet corresponding to each violation message of the *Clock_sync09* rule. The generated spreadsheet displays all destinations where source is being synchronized.

By default, the value of this parameter is set to `no`, and no such spreadsheet is generated. Set the value of this parameter to `yes` to generate such a spreadsheet.

Used by	*Clock_sync09*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_all_sync yes`
Usage in goal/source files	`-report_all_sync=yes`

report_common_reset

Enables the *Reset_info09a* rule to find the common reset source by skipping buffers/inverters and MUX/combo gates acting as buffer. By default, the parameter is set to `no`.

Used by	*Reset_info09a*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_common_reset yes`
Usage in goal/source files	`-report_common_reset=yes`

report_conv_type

Enables the *Clock_sync03b* rule to report convergence from the synchronized, unsynchronized, and standalone flip-flops or a combination of any of these flip-flops.

Possible Values of the report_conv_type Parameter

The possible values are described in the following table:

Value	Indicates
sync (default value)	Convergence from synchronized signals are reported
unsync	Convergence from unsynchronized signals are reported
nocross	Convergence from standalone flip-flops are reported
all	All the values are applied

Used by	*Clock_sync03b*
Options	Any combination of the allowed values as a comma-separated list or using the + (plus) character to append to the default value.
Default value	sync
Example	

Console-based usage	`set_parameter report_conv_type sync,unsync`
	
	The above statement is equivalent to the following:
	`set_parameter report_conv_type +unsync`

Tcl-based usage	`set_parameter report_conv_type sync unsync`
Usage in goal/source files	`-report_conv_type=sync,unsync`

report_derived_reset

Specifies if the *Reset_check04* and *Reset_check10* rules should report violations for asynchronous *Derived Resets*.

See *Cases of Reporting Asynchronous Derived Resets by the Reset_Check10 Rule*.

By default, none of these rules report violations for such resets.

Used by	*Reset_check04*, *Reset_check10*
Options	Reset_check04, Reset_check10, none, or comma-separated list of Reset_check04 and Reset_check10
Default value	none
Example	
Console/Tcl-based usage	`set_parameter report_derived_reset Reset_check04`
Usage in goal/source files	`-report_derived_reset=Reset_check04`

report_detail

Specifies if all the violations should be reported for each clock or only one violation should be reported per clock.

By default, this parameter is set to `all` and the *Clock_check10* and the *Setup_library01* rules report all the violations.

Set this parameter to `Clock_check10` to report all the violations of the *Clock_check10* rule and a reduced number of violations of the *Setup_library01* rule.

Set this parameter to `Setup_library01` to report all the violations of the *Setup_library01* rule and a reduced number of violations of the *Clock_check10* rule.

Set this parameter to `none` to report a reduced number of violations of both the rules.

Used by	*Clock_check10*, *Setup_library01*
Options	Comma-separated list of rule names, `all`, `none`
Default value	all
Example	
Console/Tcl-based usage	`set_parameter report_detail none`
Usage in goal/source files	`-report_detail=none`

report_user_defined_clock

By default, the violation messages reported by the *Clock_Reset_check02* rule includes the clock object of the parent hierarchy instead of the clock that is highlighted in the RTL viewer and the incremental schematic.

Set the report_user_defined_clock parameter to `yes` to enable the *Clock_Reset_check02* rule to report the clock that is highlighted in the RTL viewer and schematic.

Used by	*Clock_Reset_check02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_user_defined_clock yes`
Usage in goal/source files	`-report_user_defined_clock=yes`

reset_cross_seq

Specifies whether clock-domain crossings to the asynchronous reset pins of complex library cells should be reported. Complex library cells are cells that have sequential attributes but are not detected as a simple flop/latch or clock gating cell.

By default, such crossings are not reported. To report such crossings, set this parameter to `yes`.

Example of Using the reset_cross_seq Parameter

In the following figure, the *Ac_unsync01* rule reports the crossing to the asynchronous reset pin `CN` of the library cell when this parameter is set to `yes`:

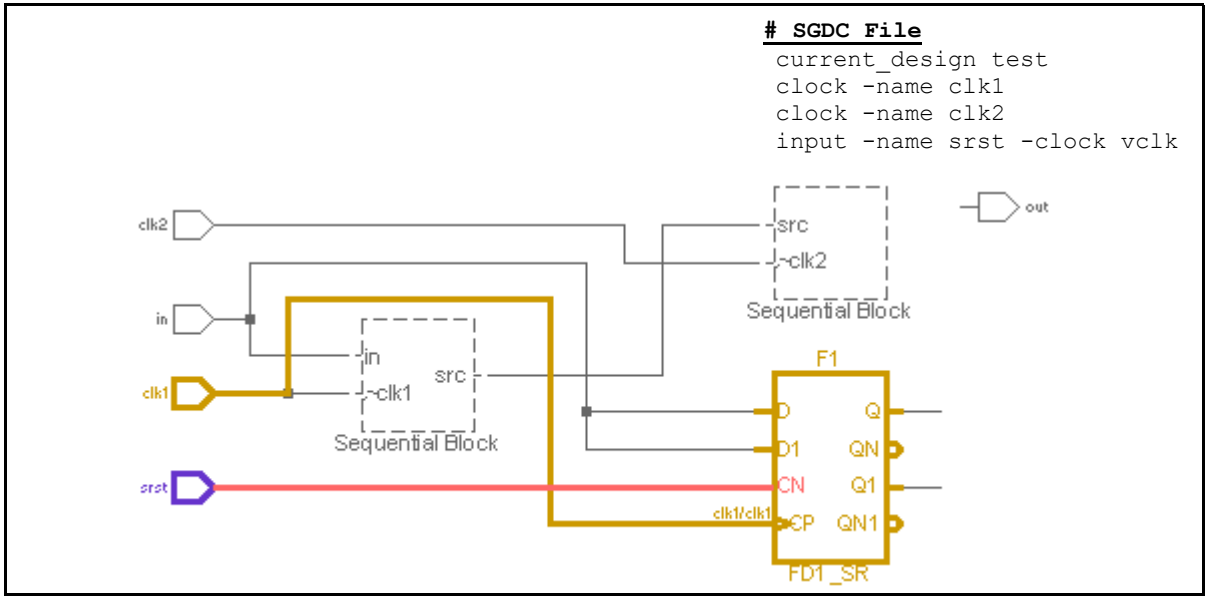


FIGURE 64.

Used by	*CDC Verification Rules*
Options	yes, no

Default value	no
Example	
Console/Tcl-based usage	`set_parameter reset_cross_seq yes`
Usage in goal/source files	`-reset_cross_seq=yes`

reset_fanout_max

Specifies the maximum fan-out limit of the resets in a design.

The default fan-out limit is 24, which means that and the *Reset_check06* rule reports the reset nets that have the fan-out greater than 24 and they are not driven by the instances of the cells specified by the *reset_placeholder_cells* parameter.

NOTE: *The Reset_check06 rule runs only when you specify the names of placeholder cells using reset_placeholder_cells parameter.*

Used by	*Reset_check06*
Options	Positive integer value
Default value	24
Example	
Console/Tcl-based usage	`set_parameter reset_fanout_max 10`
Usage in goal/source files	`-reset_fanout_max=10`

reset_reduce_pessimism

Specifies the criteria to infer resets other than the resets inferred by setting the *use_inferred_resets* parameter to `yes` or by running the *Reset_info01* rule.

For information on the values accepted by this parameter, see *Possible Values to the reset_reduce_pessimism Parameter*.

Used by	*Propagate_Resets*, *Reset_info01*, *Reset_info02*, *Reset_check03*, *Reset_check04*, *Reset_check06*, *Reset_check10*, *Reset_check11*, *Reset_sync01*, *Reset_sync03*, *Reset_sync04*, *Ar_sync01*, *Ar_unsync01*, *Ar_syncdeassert01*, *Ar_asyncdeassert01*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_conv02*, *Ac_glitch03*, *Ac_datahold01a*, *Ac_conv04*, *Ac_conv05*, *Setup_req01*, *Param_clockreset04*, *Ar_resetcross01*
Options	Combination of *Possible Values to the reset_reduce_pessimism Parameter* as a comma-separated list or using the + (plus) character to append to the default value.
Default value	filter_unused_synchronizer, same_data_reset_flop
Example	
Console/Tcl-based usage	`set_parameter reset_reduce_pessimism "+all_potential_resets"`
Usage in goal/source files	`-reset_reduce_pessimism="+all_potential_resets"`

Possible Values to the reset_reduce_pessimism Parameter

The *reset_reduce_pessimism* parameter accepts the following values:

filter_unused_synchronizer (default value)	*same_data_reset_flop (default value)*
all_potential_resets	*remove_overlap*

filter_reset_resync	*syncrst_gate_const_check*
none	*all*

filter_unused_synchronizer (default value)

When this value is specified to the *reset_reduce_pessimism* parameter, multi-flop structures using the same reset are not reported by the *Ar_unsync01* and *Ar_asyncdeassert01* rules when the output of such structures is not further used as a reset in a design.

For example, consider the following scenario:

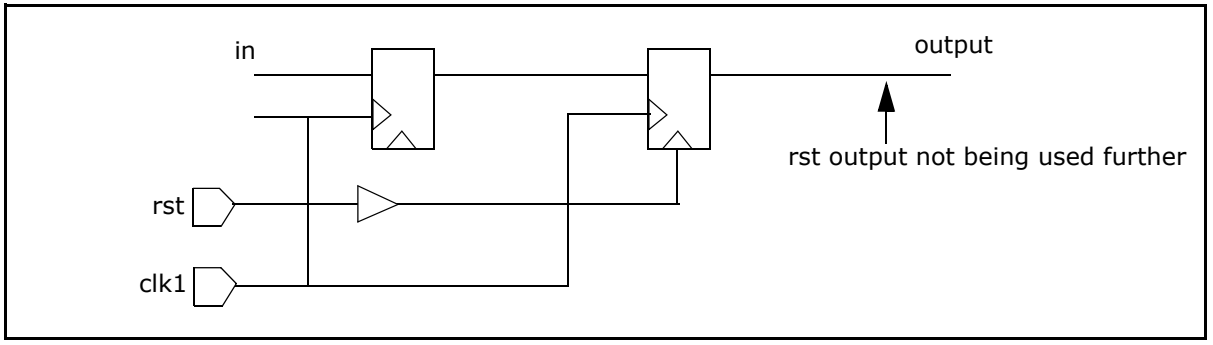


FIGURE 65. reset_reduce_pessism set to filter_unused_synchronizer

In the above scenario, by default, the *Ar_unsync01* and *Ar_asyncdeassert01* rules do not report a violation when the output is not used as a reset in the design.

same_data_reset_flop (default value)

When this value is specified to the *reset_reduce_pessimism* parameter, the *Reset_check10* violations are not reported if both the data pin and the clear pin/preset pin of a flip-flop is receiving the same reset.

all_potential_resets

When this value is specified to the *reset_reduce_pessimism* parameter and

the value of the *use_inferred_resets* parameter to `yes` (or run the *Reset_info01* rule), then while inferring resets, if SpyGlass encounters a two-input gate in which one of the inputs is a definite reset, it infers that reset as well as considers the other input to infer more resets.

For example, consider the following figure:

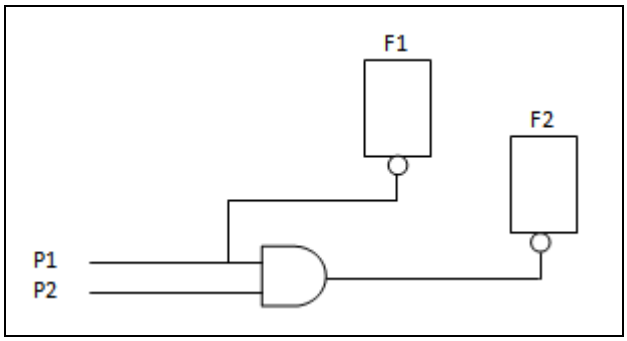


FIGURE 66. Specifying Criteria to Infer Resets in a Design

In the above figure, one of the inputs of the AND gate is a definite reset, that is `P1`. Therefore, SpyGlass infers this reset. However, to also enable SpyGlass infer `P2` as the reset, set this parameter to `all_potential_resets`.

remove_overlap

When this value is specified to the *reset_reduce_pessimism* parameter:

- The following violations are suppressed as they are already covered by the *Reset_sync02* rule:
  - *Different domain synchronizer* violation of the *Ar_unsync01* rule
  - *Domain-mismatch* violation of the *Ar_asyncdeassert01* rule

  By default, when a reset drives different domain flip-flops, the *Reset_sync02*, *Ar_unsync01*, and *Ar_asyncdeassert01* rules report violations. This may result in noise.

- Deassertion checks on end flip-flops by the *Ar_asyncdeassert01* / *Ar_syncdeassert01* rules are suppressed when the flip-flops are not synchronized.

  The deassertion checks are performed only when a proper reset synchronizer is in place.

filter_reset_resync

When this value is specified to the *reset_reduce_pessimism* parameter, SpyGlass CDC does not report the *Reset_sync04* rule violations if the reset signal and the synchronizer are in the same clock domain.

syncrst_gate_const_check

When this value is specified to the *reset_reduce_pessimism* parameter, SpyGlass CDC treats AND/NOR gates as equivalent to buffer if other gate pin is sensitized to constant due to set_case_analysis while propagating the synchronized resets in the design.

all

When this value is specified to the *reset_reduce_pessimism* parameter, all the *Possible Values to the reset_reduce_pessimism Parameter* are considered.

none

When this value is specified to the *reset_reduce_pessimism* parameter, none of the *Possible Values to the reset_reduce_pessimism Parameter* is considered.

report_clock_names_sgdc_qualifier10

Enables the SGDC_qualifier10 rule not include the clock/domain names in the violation message.

By default, the parameter is set to `yes` and the SGDC_qualifier10 rule includes the clock/domain names in the violation message.

Used by	*SGDC_qualifier10*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter report_clock_names_sgdc_qualifier10 no`
Usage in goal/source files	`-report_clock_names_sgdc_qualifier10=no`

report_abstract_module_coverage

Enables the Setup_blackbox01 rule to report coverage of abstracted modules.

By default, the parameter is set to no and SpyGlass CDC does not report the coverage of abstracted module by rule Setup_blackbox01.

Used by	*Setup_blackbox01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_abstract_module_coverage yes`
Usage in goal/source files	`-report_abstract_module_coverage=yes`

report_indirect_port_clock

Generates the enhanced *PortClockMatrix Report* that should generate the following:

- Information of the clocks indirectly connected to the input/output ports
- Information of the clocks directly connected to the input/output ports

By default, this parameter is set to `no`, and the *PortClockMatrix Report* generates information of the clocks directly connected to the input/output ports only.

Used by	*Clock_info15*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_indirect_port_clock yes`
Usage in goal/source files	`-report_indirect_port_clock=yes`

report_inst_for_netlist

Specifies whether the output net name of an instance should be reported.

By default, this parameter is set to `no`, and the output net name of an instance is reported.

Set this parameter to `yes` to enable reporting in the following manner:

- For netlist designs, violating instance name is reported instead of the net name.
- For RTL designs, leaf-level net name is reported instead of the top-level net.
- For vector terminals, the rules report a violation at the connected nets.

NOTE: *The violations reported by the rules on the instance names are bit blasted because the instance names cannot be merged.*

Used by	The parameter is used by the following rules: *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Clock_check01*, *Clock_check05*, *Clock_converge01*, *Clock_delay01*, *Clock_delay02*, *Clock_glitch04*, *Clock_info03b*, *Clock_info03c*, *Clock_info05*, *Clock_info16*, *Clock_Reset_check02*, *Clock_Reset_check03*, *Clock_sync03a*, *Clock_sync03b*, *Clock_sync05*, *Clock_sync06*, *Clock_sync08a*, *Clock_sync09*, *DeltaDelay01*, *DeltaDelay02*, *Reset_sync01*, *Reset_sync02*, *Reset_sync03*, *Reset_sync04*, *Reset_check02*, *Reset_check03*, *Reset_check04*, *Reset_check07*, *Reset_check09*, *Reset_check10*, *Reset_check11*, *Reset_check12*, *NoClockCell*, *Clock_info17*, *Clock_info02*, *Reset_info02*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, and *Ar_converge02*, *Ar_resetcross01* The parameter is also used by the following reports: *The Clock-Reset-Summary Report*, *The CKTree Report*, *The Clock-Reset-Detail Report*, *The RSTree Report*, *The Advanced CDC Report*, and *The DeltaDelay-Detailed Report*
Options	yes, no, inst_name **Note:** The value *inst_name* is applicable only when the *report_uniform_name* parameter is set to yes.

Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_inst_for_netlist yes`
Usage in goal/source files	`-report_inst_for_netlist=yes`

report_instance_pin

Specifies if instance pins of a netlist design should appear in the violation message of SpyGlass CDC rules.

Allowed Values of the report_instance_pin Parameter

The allowed values of this parameter are described in the following table:

Value	Description
yes	Specify this value to enable *The Ac_sync_group Rules* report instance-pin names.
no (default value)	Specify this value to report the output-net name of an instance instead of the instance-pin name.
bbox	Specify this value so that *The Ac_sync_group Rules* report the black-box pin of the source and destination of a crossing of an RTL or a netlist design.
seqCell	Specify this value so that *The Ac_sync_group Rules* report the library sequential-cell pin of the source and destination of a crossing of an RTL or a netlist design.
flop	Specify this value so that *The Ac_sync_group Rules* report the flip-flop pin of the source and destination of a crossing for a netlist design. This value has no impact on RTL designs.
latch	Specify this value so that *The Ac_sync_group Rules* report the latch pin of the source and destination of a crossing for a netlist design. This value has no impact on RTL designs.

Used by	*The Ac_sync_group Rules*, *Ac_coherency06*, and *Ac_datahold01a*
Options	yes, no, bbox, seqCell, flop, latch, all
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_instance_pin yes`
Usage in goal/source files	`-report_instance_pin=yes`

reset_num_flops

Specifies the minimum number of flip-flops required for synchronizing a reset signal.

By default, the value of this parameter is set to `2`, and SpyGlass uses two flip-flops to synchronize reset signals.

You can specify any positive integer value, greater than one, to this parameter to specify a different number of flip-flops.

Used by	*Clock Checking Rules*
Options	Positive integer value greater than 1
Default value	2
Example	
Console/Tcl-based usage	`set_parameter reset_num_flops 3`
Usage in goal/source files	`-reset_num_flops=3`

reset_placeholder_cells

Specifies a comma or space-separated list of placeholder cells driving reset nets with a higher fan-out.

NOTE: *The Reset_check06 rule is not run if you do not specify placeholder cells by using this parameter.*

Used by	*Reset_check06*
Options	Comma or space-separated list of placeholder cells
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter reset_placeholder_cells "BUF1,BUF2"`
Usage in goal/source files	`-reset_placeholder_cells=BUF1,BUF2`

reset_sync_check

Disables deassertion checks in reset synchronizers.

By default, this parameter is set to `strict` and the *Reset_sync01* and *Reset_sync03* rules check for deassertion on the reset source.

Set this parameter to `soft` to perform deassertion checks on the reset/clear pin of flip-flops instead of reset source. In this case, do not provide the *set_case_analysis* constraint settings on the other signals present in cone of reset/clear pins of flip-flops.

Used by	*Reset_sync01*, *Reset_sync03*
Options	strict, soft
Default value	strict
Example	
Console/Tcl-based usage	`set_parameter reset_sync_check soft`
Usage in goal/source files	`-reset_sync_check=soft`

Reset_info09a_filter_on_constant_clock

Filters violation messages of the *Reset_info09a* rule for flops whose clock pin is receiving a constant value.

By default, this parameter is set to `no` and the violation messages of the *Reset_info09a* rule for flops whose clock pin is receiving a constant value are reported.

Set this parameter to `yes` to filter such messages.

Used by	*Reset_info09a*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter Reset_info09a_filter_on_constant_clock yes`
Usage in goal/source files	`-Reset_info09a_filter_on_constant_clock=yes`

report_common_clock

Reports the common clock source by ignoring buffer or inverter, CGC clock pin, MUX/combo gates that act as buffer.

Set this parameter to `yes` to report common clock sources.

Used by	*Clock_info03a*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_common_clock yes`
Usage in goal/source files	`-report_common_clock=yes`

report_common_reset

Reports the common reset source skipping buf/inv and MUX/combo gates acting as buffer.

By default, this parameter is set to no. Set this parameter to `yes` to report the common reset source skipping buf/inv and MUX/combo gates acting as buffer.

Used by	*Reset_info09a*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_common_reset yes`
Usage in goal/source files	`-report_common_reset=yes`

report_clock_tag_names

Reports the clock tag name in the CKTree.rpt report. By default, the parameter is set to `no` and the clock object name is included in the CKTree.rpt report.

Set this parameter to `yes` to report the clock tag name in the CKTree.rpt report.

If this parameter is set to yes, the *Clock_info05*, *Clock_info05a*, *Clock_info05b*, *Clock_info05c* rules report the logical names of the clocks in the violation message. In addition, a new column, named CLOCK TAG NAMES, is added in the rule-based spreadsheet of the rules which lists the clock tag names.

Used by	*The CKTree Report*, *Clock_info05*, *Clock_info05a*, *Clock_info05b*, and *Clock_info05c*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_clock_tag_names yes`
Usage in goal/source files	`-report_clock_tag_names=yes`

report_matched_attributes

Enables the *Ac_abstract_validation02* rule report matched clocks in the messages related to data path domain mismatch issues.

Set this parameter to `yes` to enable the rule report the matched clocks as shown in the following message:

```
Data Path Domain Mismatch: Top-level clocks 'vck2vck3',
blocklevel clocks 'NA', matched-clocks 'top.block_vck1(vck1)',
block port 'rst1', block instance 'top.block' (block: 'BLOCK1')
```

Used by	*Ac_abstract_validation02*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report__matched_attributes yes`
Usage in goal/source files	`-report__matched_attributes=yes`

report_quasi_static_on_clock

Specifies the rule that should report a violation when a quasi_static signal is reaching to the clock pin of a flop.

By default, the *Clock_info03a* rule reports a violation when a quasi_static signal is reaching to the clock pin of a flop. Set the `report_quasi_static_on_clock` parameter to `Clock_info03c` to enable the *Clock_info03c* rule to report a violation in this case.

Used by	*Clock_info03a Clock_info03c*
Options	Clock_info03a, Clock_info03c
Default value	Clock_info03a
Example	
Console/Tcl-based usage	`set_parameter report_quasi_static_on_clock Clock_info03c`
Usage in goal/source files	`-report_quasi_static_on_clock=Clock_info03c`

report_reset_path_mux

Specifies if the *Reset_check07* rule should report a violation when the asynchronous set/reset pins of a sequential element are driven by a multiplexer.

By default, such violations are not reported. Set this parameter to `yes` to report such violations.

Used by	*Reset_check07*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_reset_path_mux yes`
Usage in goal/source files	`-report_reset_path_mux=yes`

report_sync_clk_for_hier

Specifies the list of hierarchies for which top-level synchronous clock signals should be reported.

Using Wildcard Expressions While Specifying Hierarchies

Consider the following example:

```
set_parameter report_sync_clk_for_hier "top.U*.U?"
```

Based on the expression specified in the above example, the matching and non matching hierarchies are specified in the following table:

Matching Hierarchies	Non Matching Hierarchies
top.U1.U2 top.U11.U2	top.X1.U2 top.U1.U

Used by	*Clock_info17*
Options	Comma-separated list of hierarchy names
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter report_sync_clk_for_hier "top.U1.U2,top.U3.U4"`
Usage in goal/source files	`-report_sync_clk_for_hier=top.U1.U2,top.U3.U4`

report_top_block_info

Specifies if the *SGDC_abstract_mapping01* rule reports clock domain and tag information for an abstracted instance in the clock mapping spreadsheet.

By default, the clock mapping spreadsheet shows clock domains and tags.

Set this parameter to 'no' to stop reporting of information about the clock domains and tags in the clock mapping spreadsheet.

Used by	*SGDC_abstract_mapping01*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter report_top_block_info yes`
Usage in goal/source files	`-report_top_block_info=yes`

reset_synchronize_cells

Specifies a comma or space-separated list of synchronizer cells that are considered as valid synchronizers for asynchronous reset signals.

By default, the `reset_synchronize_cells` parameter is not set.

NOTE: *Combinational logic is not allowed in the reset path before it is synchronized using the synchronizer cells specified by this parameter.*

When is a Reset Considered as Synchronized?

If an asynchronous reset is generated from a synchronizer module specified by this parameter, the reset is considered as synchronized and is not reported by the *Reset_sync01* rule.

When is a Reset Not Considered as Synchronized?

If all the paths from a reset signal do not have synchronizers, the reset signal is not considered as synchronized and the *Reset_sync01* rule reports violations for all the flip-flops, which are using this reset signal.

Using Wildcard Expressions to Specify Synchronizer Cells

The following example shows the usage of wildcard expressions:

```
set_parameter reset_synchronize_cells "sy?c2"
set_parameter reset_synchronize_cells "sync*"
```

For details on using wildcard characters, refer to the *Using Regular Expressions and Wildcard Characters* topic of the *Atrenta Console User Guide*.

Used by	*Reset_sync01*, *Reset_sync02*, *Reset_sync03*, *Reset_sync04*, *Reset_check07*, *Reset_check10*, *Ar_asyncdeassert01*, *Ar_syncdeassert01*, *Ar_sync01*, and *Ar_unsync01*
Options	Comma or space-separated list of synchronizer cells
Default value	NULL
Example	

Console/Tcl-based usage	`set_parameter reset_synchronize_cells "sync1,sync2"`
Usage in goal/source files	`-reset_synchronize_cells=sync1,sync2`

report_uniform_name

By default, the naming format of design objects, such as flip-flops, latches, and black boxes reported in the violation messages and reports of SpyGlass CDC are not consistent. SpyGlass CDC uses the top-level name or the hierarchical name of the design object depending on the object and the rule reporting the violation.

The `report_uniform_name` parameter enables a consistent naming format of design objects in all the violation messages and reports.

The following table describes the naming format for **RTL** design objects under the influence of this parameter and other parameters:

report_uniform_name =yes	report_uniform_name =yes and *report_inst_for_netlist* =yes	*report_inst_for_netlist* =inst_name and report_uniform_name =yes	*report_instance_pin* =yes and report_uniform_name=yes
RTL name format for a flip-flop			
Leaf-level output net name	Leaf-level output net name	Instance name	Pin name (<hier-inst>.<pin>)
RTL name format for a complex library cell with or without functional arc			
Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)
RTL name format for a black box			
Hierarchical pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)
RTL name format for a primary port			
Port name	Port name	Port name	Port name

The following table describes the naming format for **Netlist** design objects under the influence of this parameter and other parameters:

report_uniform_name=yes	*report_inst_for_netlist*=yes and report_uniform_name=yes	*report_instance_pin* =yes and report_uniform_name=yes
Netlist name format for a flip-flop		
Leaf-level output net name	Instance name	Pin name (<hier-inst>.<pin>)
Netlist name format for a complex library cell with or without functional arc		
Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)
Netlist name format for a black box		
Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)	Pin name (<hier-inst>.<pin>)
Netlist name format for a primary port		
Port name	Port name	Port name

Used by	All SpyGlass CDC Rules
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter report_uniform_name yes`
Usage in goal/source files	`-report_uniform_name=yes`

run_cells_in_cktree_rules

Generates *The CKPathInfo Report*.

By default, this parameter is set to `no` and this report is not generated.

Used by	*The CKPathInfo Report*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter run_cells_in_cktree_rules yes`
Usage in goal/source files	`-run_cells_in_cktree_rules=yes`

same_domain_at_gate

Optimizes the inference of domain on clock merging gates. Syou can specify the following allowed values:

- `same_mux_sel_gate`: MUX which are receiving same set of input domains and have same select signal are given same domain
- `any_gate`: Any gate where same set of clocks of different domains are merging are given same domain
- `no`: No optimization is performed

By default, this parameter is set to `no` and no optimization is performed.

Used by	*Clock_info03a*, *Clock_info03c*, *Clock_info05*, *Clock_info05a*, *Clock_info06*, *Clock_info07*, *Clock_info14*, *Clock_info16*, *Propagate_Clocks*, *Clock_sync03a*, *Clock_sync03b*, *Clock_sync05*, *Clock_sync06*, *Clock_sync08a*, *Clock_sync09*, *Reset_sync01*, *Reset_sync02*, *Reset_sync03*, *Reset_sync04*, *Reset_check07*, *DeltaDelay01*, *DeltaDelay02*, *Clock_check02*, *Clock_check03*, *Clock_check04*, *Clock_check05*, *Clock_check06a*, *Clock_check06b*, *Clock_check07*, *Clock_glitch01*, *Clock_glitch02*, *Clock_glitch03*, *Clock_Reset_info01*, *Clock_converge01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Clock_info05b*, *Clock_Reset_check01*, *Clock_Reset_check01*, *Clock_info02*, *Clock_info03a*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_xclock01*, *Setup_quasi_static01*, *Clock_hier01*, *Clock_hier02*, *Clock_hier03*, *SGDC_clock_path_wrapper_module01*, *Ac_coherency06*, *Ar_resetcross01*
Options	same_mux_sel_gate, any_gate, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter same_domain_at_gate same_mux_sel_gate`
Usage in goal/source files	`-same_domain_at_gate=same_mux_sel_gate`

same_domain_at_gate

same_sync_reset

Configures *CDC Verification Rules* to report a violation for the cases where different synchronous reset is used to synchronize destination domain flip-flops.

For example, consider the following scenario:

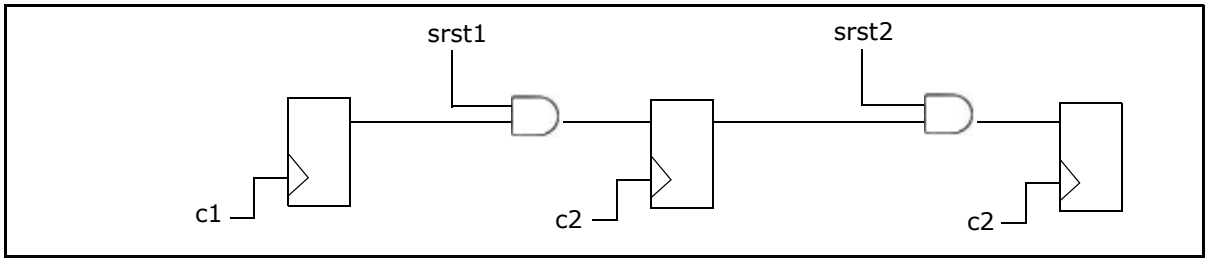


FIGURE 67. same_sync_reset example

For the above scenario, if you set the `same_sync_reset` parameter to `yes`, the *CDC Verification Rules* report a violation because a different synchronous reset `srst1` reaches the destination.

NOTE: *For this parameter to work, set the sync_reset parameter to yes.*

Used by	*CDC Verification Rules*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter same_sync_reset yes`
Usage in goal/source files	`-same_sync_reset=yes`

same_threshold_all_cktree

Enables the *Clock_check06b* rule to check if master library cells of cell instances in all clock trees have a common value for the `threshold_voltage_group` library attribute.

By default, the *Clock_check06b* checks for each clock tree separately. Set the `same_threshold_all_cktree` parameter to `yes` to check for all clock trees together.

Used by	*Clock_check06b*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter same_threshold_all_cktree yes`
Usage in goal/source files	`-same_threshold_all_cktree=yes`

sel_case_analysis_mode

Causes the *Setup_clock01* rule to detect probable unconstrained mux-select signals in the clock path where the *set_case_analysis* settings are required.

This parameter accepts the following values:

- **sequential (default value)**

  By default, this parameter is set to `sequential`, and the *Setup_clock01* rule generates the following objects in the fan-in cone of mux-select signals in the *auto_case_analysis.sgdc* file under the spyglass_reports/clock-reset/ directory:

Primary ports	Black box outputs
Undriven nets	Output of sequential elements

- **source**

  When this parameter is set to `source`, the *Setup_clock01* rule goes beyond sequential elements to find primary ports, black box outputs, and undriven nets.

  If the input of a sequential element is tied to a constant, the *Setup_clock01* rule stops propagation on its output, and the output is saved in the *auto_case_analysis.sgdc* file.

- **direct**

  When this parameter is set to `direct`, the *Setup_clock01* rule dumps the nets that are directly connected to the mux select pins.

  If a connected net is a SpyGlass-generated internal net, it is not dumped.

Used by	*Setup_clock01*
Options	direct, source, sequential
Default value	sequential
Example	

Console/Tcl-based usage	`set_parameter sel_case_analysis_mode "direct"`
Usage in goal/source files	`-sel_case_analysis_mode="direct"`

show_all_xclock_flops

Specifies whether the *Ac_xclock01* rule should generate a spreadsheet (see *Figure 280*) showing violations on a per net basis.

If you set this parameter to `no`, this rule does not generate any spreadsheet. In this case, this rule reports only one of the rule-violating instances along with the total violation count.

Used by	*Ac_xclock01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter show_all_xclock_flops yes`
Usage in goal/source files	`-show_all_xclock_flops=yes`

show_derived_busclocks

Enables the *Clock_check03* rule to report violations for the bus-bit signals present in the derived clock path.

By default, the value of this parameter is set to `no`, and the *Clock_check03* rule reports violations for only those bus-bit signals that are in the primary clock path.

Set this parameter to `yes` to also report violations for bus-bit signals that are in derived clock path.

Used by	*Clock_check03*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter show_derived_busclocks yes`
Usage in goal/source files	`-show_derived_busclocks=yes`

show_module_in_spreadsheet

Generates the following data in the spreadsheet of the *The Ac_sync_group Rules*.

- The module containing both the source and destination of a crossing
- Parent instances of the source and destination instances that are interacting with each other

By default, the value of this parameter is set to `no`, and the above data is not generated. Set the value of this parameter to `yes` to generate such data.

Used by	*Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter show_module_in_spreadsheet yes`
Usage in goal/source files	`-show_module_in_spreadsheet=yes`

show_parent_module_in_spreadsheet

Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

By default, the value of this parameter is set to `no`, and the PARENT_MODULE column is not included in the spreadsheet.

The parameter reports the `SOURCE_PARENT_MODULE`, `DESTINATION_PARENT_MODULE`, and `PARENT_MODULE` depending on the value of *msg_inst_mod_report* parameter.

Used by	*Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, *Ac_conv05*, *Ac_glitch01*, *Ac_glitch02*, *Ac_glitch03*, *Ac_glitch04*, *Ac_coherency06*, *Ac_sync01*, *Ac_sync02*, *Ac_unsync01*, *Ac_unsync02*, *Clock_info05*, *Clock_info05a*, *Clock_info05b*, *Clock_info05c*, *Ar_resetcross01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter show_parent_module_in_spreadsheet yes`
Usage in goal/source files	`-show_parent_module_in_spreadsheet=yes`

show_reconv_paths

Causes the *Clock_sync03a* and *Clock_sync03b* to highlight only converging signals in the schematic and the gate where they are converging.

By default, these rules highlight the complete path from converging signals to the gate where they are converging. Highlighting complete path may be run-time intensive.

Therefore, to reduce the run time by avoiding schematic data generation for the convergence paths, set this parameter to `no`. In this case, only converging signals and the gate where they are converging are highlighted.

Used by	*Clock_sync03a* and *Clock_sync03b*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter show_reconv_paths no`
Usage in goal/source files	`-show_reconv_paths=no`

show_source_in_spreadsheet

Controls whether the sources of synchronizers should be displayed in the message-based spreadsheet of the *Ac_conv01*, *Ac_conv02*, or *Ac_conv03* rules or linked to the spreadsheet of *The Ac_sync_group Rules*.

NOTE: *This parameter works only when you run The Ac_sync_group Rules.*

The following topics describe the impact of the values (`yes` or `no`) of this parameter on SpyGlass CDC analysis:

- *Setting the show_source_in_spreadsheet Parameter To no*
- *Setting the show_source_in_spreadsheet Parameter To yes (default)*

Setting the show_source_in_spreadsheet Parameter To no

If this parameter is set to `no`, a link appears in the spreadsheet of the *Ac_conv01*, *Ac_conv02*, or *Ac_conv03* rules. This link points to the message-based spreadsheet of *The Ac_sync_group Rules* showing the source of synchronizers. This is shown in the following figure:

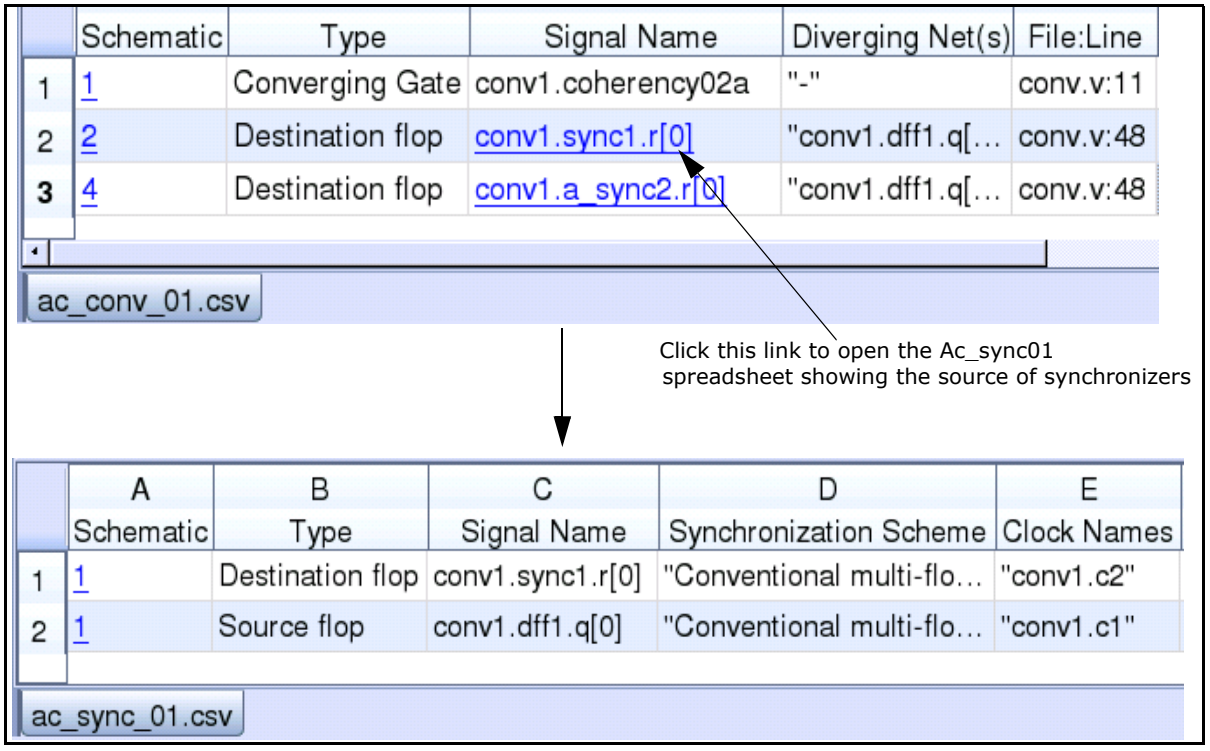


FIGURE 68. Link in spreadsheet when show_source_in_spreadsheet is set to no

Setting the show_source_in_spreadsheet Parameter To yes (default)

If this parameter is set to `yes` or if *The Ac_sync_group Rules* are not run, the sources of synchronizers appear in the message-based spreadsheet of the *Ac_conv01*, *Ac_conv02*, or *Ac_conv03* rules. This is shown in the following figure:

	C Signal Name	D Source(s)	E Diverging Net(s)	F Destination Clock(s)	G Source Clock(s
1	conv1.coherency02a	"-"	"-"	"-"	"-"
2	conv1.sync1.r[0]	"conv1.dff1.q[0]"	"conv1.dff1.q[0]"	"conv1.c2"	"conv1.c1"
3	conv1.a_sync2.r[0]	"conv1.dff1.q[0]"	"conv1.dff1.q[0]"	"conv1.c2"	"conv1.c1"

ac_conv_01.csv

FIGURE 69. Sources of synchronizers appearing in Ac_conv02 spreadsheet

Used by	*Ac_conv01*, *Ac_conv02*, and *Ac_conv03*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter show_source_in_spreadsheet no`
Usage in goal/source files	`-show_source_in_spreadsheet=no`

reset_sync_depth

Specifies the number of flip-flops that are the part of the longest reset synchronizer chain in a design.

This parameter is used for propagation of an active value of a reset in a design. The *Ar_asyncdeassert01* and *Ar_syncdeassert01* rules internally use this information for classification of resets to synchronous or asynchronous deasserted resets.

Used by	*Ar_asyncdeassert01* and *Ar_syncdeassert01*
Options	Positive integer value greater than 0
Default value	8
Example	
Console/Tcl-based usage	`set_parameter reset_sync_depth 20`
Usage in goal/source files	`-reset_sync_depth=20`

simulator_file_name

Specifies a *Simulator File* that contains simulator-specific delta-delay information for RTL constructs.

The *DeltaDelay01* and *DeltaDelay02* rules assume that the simulator-specific delta delay for the constructs other than those specified in the simulator mode file is zero.

Used by	*DeltaDelay01*, *DeltaDelay02*
Options	Simulator file name
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter simulator_file_name delayfile`
Usage in goal/source files	`-simulator_file_name=delayfile`

skip_samedom_syncpath

Skips the same domain sequential elements while searching synchronizers for particular synchronization schemes.

You can specify a synchronization scheme by setting this parameter to the following values:

Value	Synchronization Scheme
`mux`	*Recirculation MUX Synchronization Scheme* and *MUX-Select Sync (Without Recirculation) Synchronization Scheme*
`enable`	*Synchronized Enable Synchronization Scheme*
`gp`	*Glitch Protection Cell Synchronization Scheme*
`and`	*AND Gate Synchronization Scheme*
`cg`	*Clock-Gating Cell Synchronization Scheme*
`all`	All the above-mentioned synchronization schemes
`none`	(Default) None of the synchronization schemes

You can specify a comma-separated list of any of the above values (except `all` and `none`) to specify multiple synchronization schemes.

When this parameter is set, all the same domain sequential cells are skipped in the fan-in to find the synchronizer.

Used by	*CDC Verification Rules*
Options	all, none, or comma-separated list of any of the following values: mux, enable, and, gp, and cg
Default value	none
Example	
Console/Tcl-based usage	`set_parameter skip_samedom_syncpath "cg,gp"`
Usage in goal/source files	`-skip_samedom_syncpath="cg,gp"`

stop_at_reset

Specifies whether to continue propagation of the reset reaching another reset in its path.

By default, reset propagation does not happen.

Used by	All the SpyGlass CDC rules that use resets
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter stop_at_reset no`
Usage in goal/source files	`-stop_at_reset=no`

strict_double_flop

Marks those clock crossings as synchronized under the *Conventional Multi-Flop Synchronization Scheme* where another flip-flop exists between the source flip-flop and the destination flip-flop and its clock signal is inverse of the clock signal of the source flip-flop.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter strict_double_flop yes`
Usage in goal/source files	`-strict_double_flop=yes`

strict_sync_check

By default, the `strict_sync_check` parameter is set to `no`. If the parameter is set to `yes`, the synchronization schemes are affected as described below:

- *Conventional Multi-Flop Synchronization Scheme*
  - ❒ Allows combinational logic between the synchronizer flip-flops only if the combinational logic is case-sensitized to be equivalent to buffers or inverters. By default, combinational logic is not allowed.

    For example, consider the following figure in which one of the inputs of the AND gate is tied to a constant (case-sensitized):

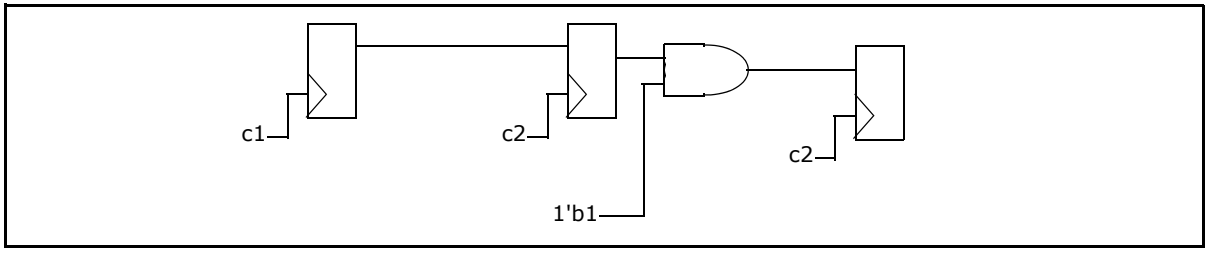


FIGURE 70. Combinational logic case-sensitized to be equivalent to buffer or inverter

    In the above scenario, the AND gate is considered as a buffer.
  - ❒ Allows combinational logic between the source and destination flip-flops if the combinational logic is case-sensitized to be equivalent to buffers or inverters.

    For example, consider the following figure in which one of the inputs of the AND gate is tied to a constant (case-sensitized):

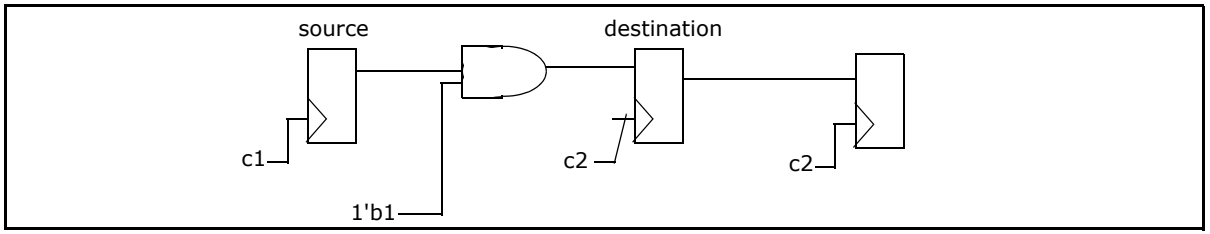


FIGURE 71. Combinational logic case-sensitized to be equivalent to buffer or inverter

In the above scenario, the AND gate is considered as a buffer. Transparent latches (enabled latch) are also considered as combinational elements.

- *Synchronizing Cell Synchronization Scheme*

  Allows combinational logic between the source and destination flip-flops if the combinational logic is case-sensitized to be equivalent to buffers/ inverters. See *Figure 71*.

  Transparent latches (enabled latch) are also considered as combinational elements.

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_datahold01a*, *Ac_glitch03*, *Ac_coherency06*
Options	yes, no
Default value	no
Default Value in GuideWare2.0	yes
Example	
Console/Tcl-based usage	`set_parameter strict_sync_check yes`
Usage in goal/source files	`-strict_sync_check=yes`

sync_check_type

Specifies the signal type that is considered as a valid candidate for synchronizing the source of a data crossing reported by the clock synchronization rules.

Supported Values

This parameter accepts the following values:

- `qual_only`: Enables data synchronization analysis based on the qualifier search in the transitive input cone of a gate that receives the source. This allows crossing detection synchronization using schemes specified in *Clock Domain Crossing Synchronization Schemes*.
- `enable_with_qual`: Allows a qualifier as a valid enable for *The Enable Expression-Based Synchronization Analysis*.
- `enable_with_des_dom`: Allows the destination of a crossing as a valid enable for *The Enable Expression-Based Synchronization Analysis*.

Used by	*The Ac_sync_group Rules*
Options	*Supported Values*
Default value	qual_only
Example	
Console/Tcl-based usage	`set_parameter sync_check_type enable_with_qual`
Usage in goal/source files	`-sync_check_type=enable_with_qual`

synchronize_cells

Specifies cells that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.

If the cell is soft (contains logic), the domain crossing must occur on a flip-flop for this method to be recognized.

You can use wildcard characters while specifying cell names. For details, refer to the *Using Regular Expressions and Wildcard Characters* topic of the *Atrenta Console User Guide*.

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 parameter is not applicable for data crossings. It is applicable for control crossings.*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_coherency06*
Options	Comma or space-separated list of synchronizer cells enclosed in double quotes
Default value	NULL
Example	

Console/Tcl-based usage	`set_parameter synchronize_cells "sync1,sync2"`
	
	`set_parameter synchronize_cells "sy?c2"`
	`set_parameter synchronize_cells "sync*"`

Usage in goal/source files	`-synchronize_cells="sync1,sync2"`

synchronize_data_cells

Specifies the cells that are considered as valid synchronizers for source domain vector control signals for the *Synchronizing Cell Synchronization Scheme*.

If the cell is soft (that is, contains logic), the domain crossing must occur on a flip-flop for this method to be recognized.

You can use wildcard characters while specifying cell names. For details on using wildcard characters, refer to the *Using Regular Expressions and Wildcard Characters* topic of the *Atrenta Console User Guid*e.

NOTE: *This parameter is not applicable for data crossings. It is applicable only for control crossings.*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*
Options	Comma or space-separated list of synchronizer cells enclosed in double quotes
Default value	NULL
Example	

Console/Tcl-based usage	`set_parameter synchronize_data_cells "sync1,sync2"`
	
	`set_parameter synchronize_data_cells "sy?c2"`
	`set_parameter synchronize_data_cells "sync*"`

Usage in goal/source files	`-synchronize_data_cells="sync1,sync2"`

sync_point_report_limit

Specifies the maximum number of enable points (to be shown in the *Message-Based Spreadsheet for the Enable Condition Based Flow*) at which a crossing could be synchronized.

Use this parameter during *The Enable Expression-Based Synchronization Analysis*.

This parameter is effective when both the following conditions are true:

- The *sync_point_selection* parameter is set to `none`.
- The *sync_check_type* parameter is set to `enable_with_qual` or `enable_with_des_dom`

Used by	*The Ac_sync_group Rules*
Options	Integer greater or equal to 2
Default value	5
Example	
Console/Tcl-based usage	`set_parameter sync_point_report_limit 3`
Usage in goal/source files	`-sync_point_report_limit=3`

sync_point_selection

Specifies the enable signal that should be considered for synchronization in *The Enable Expression-Based Synchronization Analysis*.

NOTE: *This parameter is effective only when the sync_check_type parameter is set to* `enable_with_qual` *or* `enable_with_des_dom`.

Used by	*The Ac_sync_group Rules*
Options	*Possible Values of the sync_point_selection Parameter*
Default value	first
Example	
Console/Tcl-based usage	`set_parameter sync_point_selection last`
Usage in goal/source files	`-sync_point_selection=last`

Possible Values of the sync_point_selection Parameter

This parameter accepts the following values:

first	*last*	*none*	*gp_sync*	*all*

first

Set this value to consider the first possible enable signal (from source) where a crossing could be synchronized. For example, consider the following schematic:

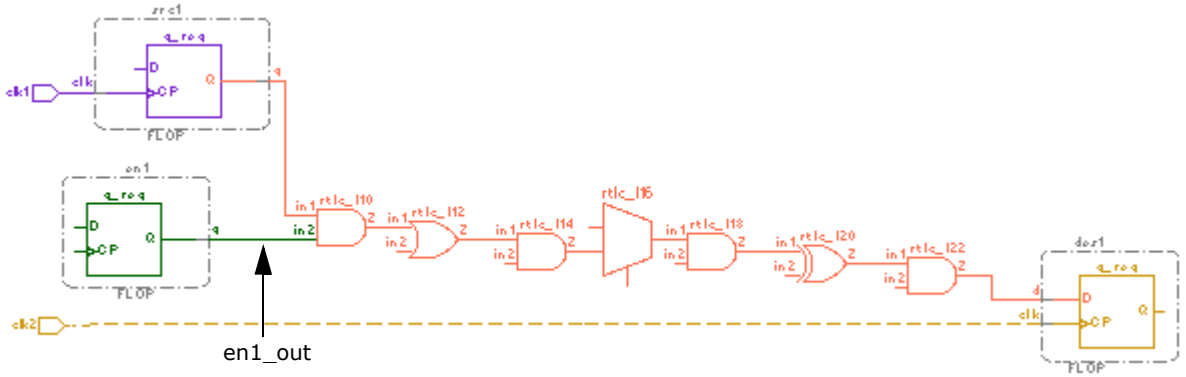


FIGURE 72.

In the above schematic, `en1_out` is considered for synchronization.

last

Set this value to consider the last possible enable signal (from source) where a crossing could be synchronized. For example, consider the following schematic:

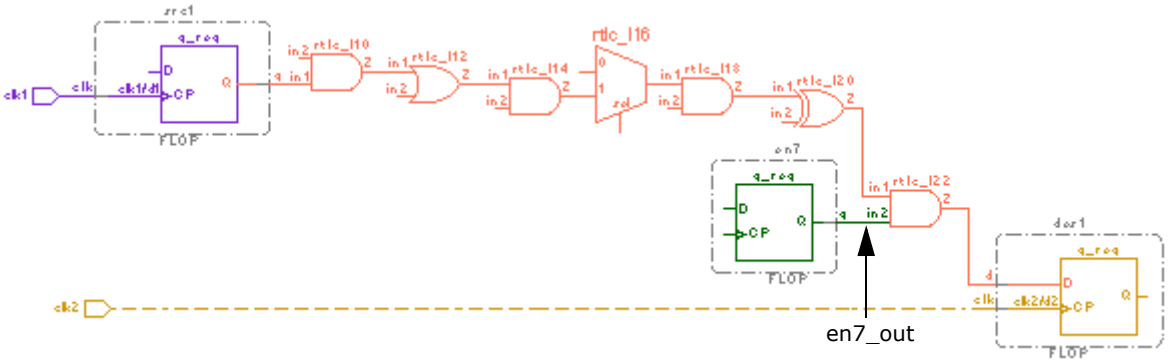


FIGURE 73.

In the above schematic, `en7_out` is considered for synchronization.

none

Set this value to consider multiple possible enable signals (from destination) at which the crossing could be synchronized. To set the number of such enable signals, use the *sync_point_report_limit* parameter.

From the multiple enable signals reported, select the enable signal which is right candidate for synchronization for the crossing. Specify that enable signal by using the *qualifier* constraint in the next SpyGlass run to get the crossing synchronized.

For example, consider the following spreadsheet when the value of the *sync_point_report_limit* parameter is set to 3:

A	B	C	D	E	F	G	H	I
Schematic	Type	Signal Name	Failure Reason	Synchronization Scheme	Enable Candidate	Enable Count	Clock Names	Internal Clock Domain Tag
1	Destination flop	top.des1.q	unsynchronized destination	N.A.	-	-	top.clk2	1
1	Source flop	top.src1.q	No enable condition selected	N.A.	-	3	top.clk1	0

FIGURE 74.

In the above spreadsheet, note the value `3` reported in the *Enable Count* column. This value indicates three possible enable points (from destination) where a source can get synchronized. To view the details of these points, click on `3`. This displays the following spreadsheet showing the details of these points:

Schematic ID	Enable Analysis Point	Enable Condition	Type
1	top.rtlc_l14.in1	(top.en3_out)	Enable
2	top.rtlc_l18.in1	(top.en5_out)	Enable
3	top.rtlc_l22.in1	(top.en7_out)	Enable

FIGURE 75.

gp_sync

This value is equivalent to the *none* value of this parameter (*sync_point_selection*) such that SpyGlass does not report any synchronized crossing but reports possible enable signals that could synchronize a crossing.

Specifying this value additionally enables SpyGlass to report crossings as synchronized (instead of unsynchronized) in which an enable signal merges with the source at the gate specified by the *glitch_protect_cell* parameter.

For example, consider the design shown in the following figure:

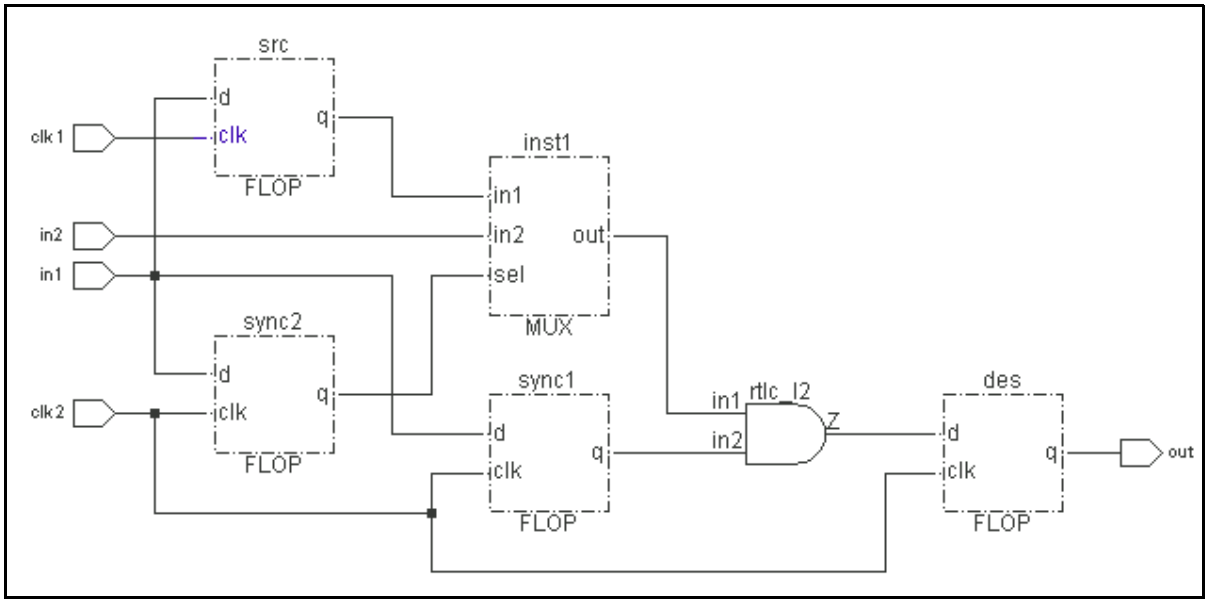
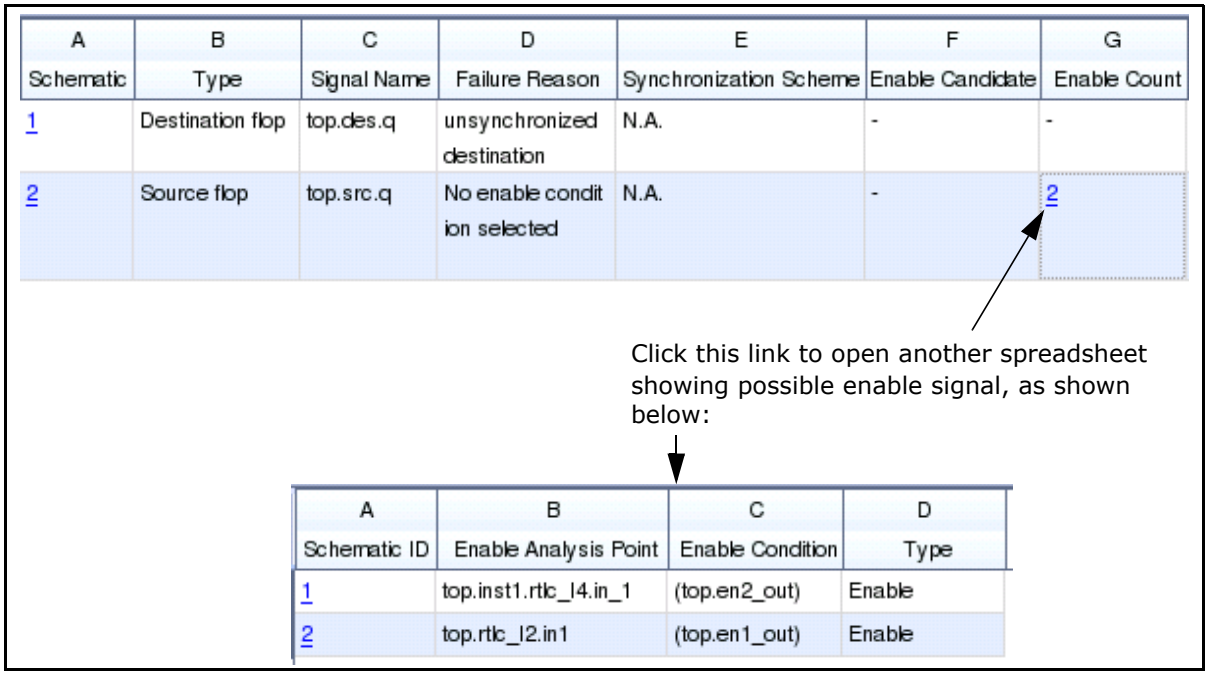


FIGURE 76.

For the above design, the *Ac_unsync01* rule reports an unsynchronized crossing when the *sync_point_selection* parameter is set to `none`. In this case, this rule generates the following spreadsheets showing the possible enable signal that could synchronize the crossing:



A Schematic	B Type	C Signal Name	D Failure Reason	E Synchronization Scheme	F Enable Candidate	G Enable Count
1	Destination flop	top.des.q	unsynchronized destination	N.A.	-	-
2	Source flop	top.src.q	No enable condition selected	N.A.	-	2

A Schematic ID	B Enable Analysis Point	C Enable Condition	D Type
1	top.inst1.rtlc_l4.in_1	(top.en2_out)	Enable
2	top.rtlc_l2.in1	(top.en1_out)	Enable

FIGURE 77. Spreadsheets of unsynchronized crossings when sync_point_selection is set to none

To report the crossing shown in *Figure 76* as synchronized, set the *sync_point_selection* parameter is set to `gp_sync`. The following figure shows the spreadsheet generated by the *Ac_sync01* rule reporting synchronized crossing in this case:

A Schematic	B Type	C Signal Name	D Synchronization Scheme	E Enable Candidate	F Enable Count	G Clock Names
1	Destination flop	top.des.q	Valid enable condition found	-	-	top.clk2
1	Source flop	top.src.q	Valid enable condition found	(top.en2_out)	N.A	top.clk1

FIGURE 78. Spreadsheet of synchronized crossing when sync_point_selection is set to gp_sync

all

This value is used only in the Hybrid CDC flow and the effective bus verification flow. Use this parameter to list the complete enable expression in the above two flows.

Note that the effective bus verification flow is enabled when you use the cdc_effective_bus_verif parameter.

sync_reset

Allows a synchronous reset to be used for destination domain flip-flops including the synchronizer flip-flops. As a result, maximum of one gate of the type AND/NAND/OR/NOR is allowed at the crossing and between the flip-flops of a multi-flop synchronizer.

If same synchronous resets are not used in the destination domain flip-flops including synchronizer flip-flops, capture such cases by setting the *same_sync_reset* parameter to `yes`.

By default, no gates are allowed in the data transfer path between the flip-flops at clock domain crossings or the data paths around synchronizing flip-flops.

NOTE: *It is recommended that you specify a synchronous reset using the reset constraint with* `-sync` *argument (*`reset -sync`*) instead of using the sync_reset parameter. This is because this parameter allows any single combinational gate that might not be a synchronous reset gate.*

Used by	*Clock_sync03a*, *Clock_sync03b*, *Clock_sync08a*, *Clock_sync09*, *Ac_crossing01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Ac_conv04*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_coherency06*, *Ar_resetcross01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter sync_reset yes`
Usage in goal/source files	`-sync_reset=yes`

thru_reset_synchronizer

Specifies if the *Reset_check07* rule reports a violation for reset synchronizers that are driven by a combinational logic or mux.

Set this parameter to `no` so that the *Reset_check07* rule does not report violations for any reset synchronizers that are driven by a combinational logic or mux.

Used by	*Reset_check07*
Options	yes, no
Default value	yes
Example	
Console/Tcl-based usage	`set_parameter thru_reset_synchronizer no`
Usage in goal/source files	`-thru_reset_synchronizer=no`

unexpected_ckcells_file

Specifies the name of the file containing the list of cells that are not allowed in the clock trees checked by the *Clock_check06a* rule.

Specifying the List of Disallowed Cells

Create an ASCII file containing the names of disallowed cells (one name in each line) and specify it with the `unexpected_ckcells_file` parameter.

In the ASCII file:

- You can use Perl regular expressions to specify the names of multiple cells. For example, to refer to cells `cell1`, `cell2`, `cell3`, and `cell4`, you can specify `cell.*`.
- You can use `//` style or # style comments.

Used by	*Clock_check06a*
Options	Comma or space-separated list of file names
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter unexpected_ckcells_file "cells1.list,cells2.list"`
Usage in goal/source files	`-unexpected_ckcells_file=cells1.list,cells2.list`

unex_reset_gate_list

Specifies a list of disallowed cell names on a reset net.

The disallowed cell names should be given as they appear in schematic. For netlist designs, specify the cell names as mentioned in the .lib files. For black boxes, specify the name of black box master module.

NOTE: *The Reset_check09 rule checks for any cells with the XOR, XNOR, AND, and NAND functions even if you specify the list of disallowed gates. Checking for these cell types cannot be disabled.*

Used by	*Reset_check09*
Options	Comma or space-separated list of disallowed cell names
Default value	NULL
Example	
Console/Tcl-based usage	`set_parameter unex_reset_gate_list "RTL_AND,RTL_NOT"`
Usage in goal/source files	`-unex_reset_gate_list=RTL_AND,RTL_NOT`

user_group_str

Specifies a comma or space-separated list of strings based on which the messages of *The Ac_sync_group Rules* are grouped. For details on this type of grouping, see *User-Specified String-Based Grouping*.

NOTE: *Strings specified in this parameter are case-sensitive. Therefore, specifying* `user_group_str=cfg` *will not match the strings, such as* `Cfg` *and* `CFG`*.*

Used by	*The Ac_sync_group Rules*
Options	Comma or space-separated list of strings
Default value	""
Example	
Console/Tcl-based usage	`set_parameter user_group_str cfg`
Usage in goal/source files	`-user_group_str=cfg`

use_inferred_clocks

Specifies whether SpyGlass CDC solution rules should use auto-generated clock information (generated by the *Clock_info01* rule) *in addition* to any user-defined clocks (specified using the *clock* constraint in a *.sgdc* file).

Setting use_inferred_clocks to yes

When you set this parameter to `yes`, SpyGlass CDC infers clock candidates by going backward from the clock pins of sequential elements.

However, if a combinational logic is present in the clock path, all its inputs may be considered as the clock candidates and using them in analysis may produce faulty results. Therefore, it is not recommended that you use this parameter.

If you do not know the clocks in your design, find the clocks by using the methodology described in the *SpyGlass CDC Methodology User Guide*.

Used by	*Clock_info03a*, *Clock_info03c*, *Clock_info05*, *Clock_info05a*, *Clock_info06*, *Clock_info07*, *Clock_info14*, *Clock_info16*, *Propagate_Clocks*, *Clock_sync03a*, *Clock_sync03b*, *Clock_sync05*, *Clock_sync06*, *Clock_sync08a*, *Clock_sync09*, *Reset_sync01*, *Reset_sync02*, *Reset_sync03*, *Reset_sync04*, *Reset_check07*, *DeltaDelay01*, *DeltaDelay02*, *Clock_check02*, *Clock_check03*, *Clock_check04*, *Clock_check05*, *Clock_check06a*, *Clock_check06b*, *Clock_check07*, *Clock_glitch01*, *Clock_glitch02*, *Clock_glitch03*, *Clock_Reset_info01*, *Clock_converge01*, *Ac_conv01*, *Ac_conv02*, *Ac_conv03*, *Clock_info05b*, *Clock_Reset_check01*, *Clock_Reset_check01*, *Clock_info02*, *Clock_info03a*, *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, *Ac_unsync01*, *Ac_xclock01*, *Setup_quasi_static01*, *Clock_hier01*, *Clock_hier02*, *Clock_hier03*, *SGDC_clock_path_wrapper_module01*, *Ac_coherency06*, *Ar_resetcross01*
Options	yes, no
Default value	no
Example	

Console/Tcl-based usage	`set_parameter use_inferred_clocks yes`
Usage in goal/source files	`-use_inferred_clocks=yes`

use_inferred_resets

By default, the `use_inferred_resets` parameter is not set.

Set the parameter to `yes` to specify that the other reset rules should use the auto-generated reset information *in addition* to any user-defined resets (specified using the *reset* constraint in a *.sgdc* file).

Used by	*Propagate_Resets*, *Reset_info02*, *Reset_sync01*, *Reset_sync03*, *Reset_sync04*, *Reset_check03*, *Reset_check04*, *Reset_check06*, *Reset_check07*, *Reset_check10*, *Reset_check11*, *Ac_cdc01a*, *Ac_cdc01b*, *Ac_cdc01c*, *Ac_cdc08*, *Ac_conv02*, *Ar_sync01*, *Ar_unsync01*, *Ar_asyncdeassert01*, *Ar_syncdeassert01*, *Reset_sync01*, *Ar_resetcross01*
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter use_inferred_resets yes`
Usage in goal/source files	`-use_inferred_resets=yes`

validate_reduce_pessimism

Configures the *Ac_abstract_validation01* and *SGDC_virtualclock_validation01* rules to ignore reporting on the block ports that are hanging, or have a constant or quasi static signal reaching on them.

Specify any of the following values to this parameter:

Value	Description
hanging_net	No *Clocks Mismatch*, *Clock Domain Mismatch*, *Reset Mismatch*, *Data Path Domain Mismatch*, or *Qualifier Mismatch* checks are performed on the hanging block ports.
constant	No *Clocks Mismatch*, *Clock Domain Mismatch*, *Reset Mismatch*, *Data Path Domain Mismatch*, or *Qualifier Mismatch* checks are performed on block ports that are receiving a constant signal.
quasi_static	No *Clocks Mismatch*, *Clock Domain Mismatch*, *Reset Mismatch*, *Data Path Domain Mismatch*, or *Qualifier Mismatch* checks are performed on block ports that are receiving quasi static signal.
ignore_domain_overconstraint	No *Data Path Domain Mismatch* is reported when all the top-level domains reaching to the block-level port gets mapped to the block level *abstract_port* constraint specified on that port, but there are some extra unmapped block-level *abstract_port* constraints remaining on block port. For details, see *Example 2*.
none	All validation checks are performed.
all	No *Clocks Mismatch*, *Clock Domain Mismatch*, *Reset Mismatch*, *Data Path Domain Mismatch*, or *Qualifier Mismatch* checks are performed on block ports that are hanging or are receiving constant or quasi static signal.

Example 1

Consider the following figure:

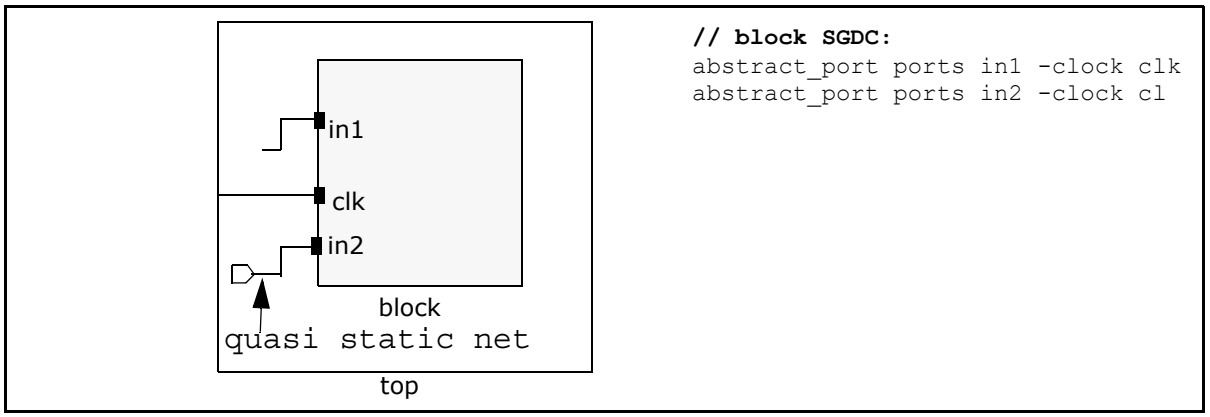


FIGURE 79. Example of the validate_reduce_pessimism parameter

For the above example, consider that the `validate_reduce_pessimism` parameter is set as:

```
validate_reduce_pessimism hanging_net,quasi_static
```

In this case, the *Ac_abstract_validation01* rule does not report *Data Path Domain Mismatch* for `in1` (hanging port) and `in2` (quasi static signal reaching on it).

However, the *Ac_abstract_validation01* rule will report *Quasi Static Mismatch* for the `in2` net.

Example 2

Consider the following figure in which two *abstract_port* constraints with the `clk1` and `clk2` clocks are defined on the `p1` abstract-block port:

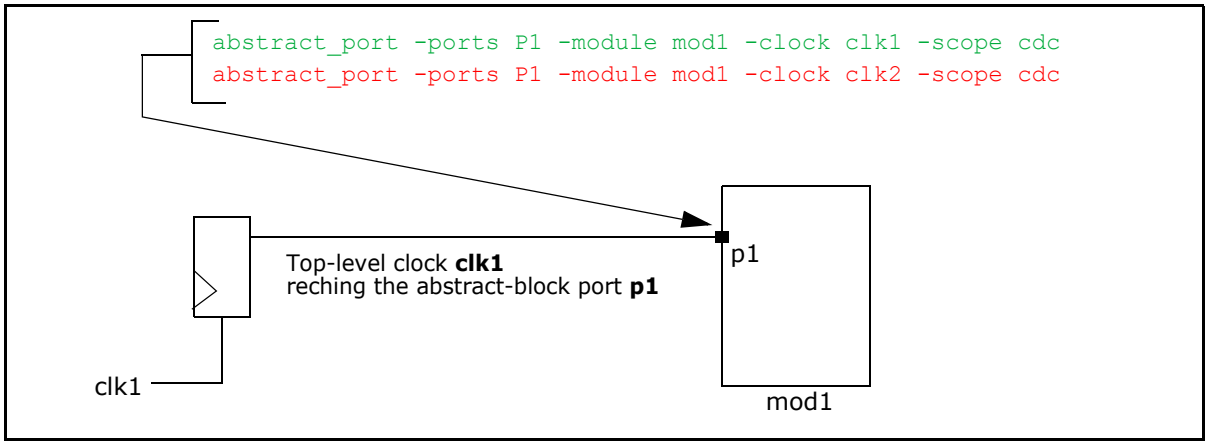


FIGURE 80.

In the above figure, `p1` receives the signal of the `clk1` domain that matches with the following *abstract_port* constraint on `p1`:

```
abstract_port -port p1 -clock clk1 -scope cdc
```

Therefore, all the top-level domains get mapped to the block-level constraints. However, the following extra *abstract_port* constraint is present at the block level:

```
abstract_port -port p1 -clock clk2 -scope cdc
```

Therefore, *Data Path Domain Mismatch* violation is reported for `clk2`.

To suppress the violation, set the `validate_reduce_pessimism` parameter to `ignore_domain_overconstraint`.

Used by	*Ac_abstract_validation01* and *SGDC_virtualclock_validation01*
Options	hanging_net, constant, quasi_static, none, all
Default value	none
Default Value in GuideWare2.0	all
Example	

Console/Tcl-based usage	`set_parameter validate_reduce_pessimism constant`
Usage in goal/source files	`-validate_reduce_pessimism=constant`

valid_enable_type

NOTE: *This parameter is deprecated. Use the sync_check_type parameter instead of this parameter.*

Specifies the signal that should be considered as a valid enable in *The Enable Expression-Based Synchronization Analysis*.

By default, a qualifier is considered as a valid enable. Set this parameter to `dest` to consider a destination as a valid enable.

NOTE: *This parameter works only when the enable_condition_based_sync parameter is set to yes.*

Used by	*The Ac_sync_group Rules*
Options	dest, qual
Default value	qual
Example	
Console/Tcl-based usage	`set_parameter valid_enable_type dest`
Usage in goal/source files	`-valid_enable_type=dest`

Tcl Commands in SpyGlass CDC

The following Tcl commands are pertaining to SpyGlass CDC:

Command	Description
get_cdc	Creates a list of clock domain crossings (in the current design) that match certain criteria
get_cdc_coherency	Returns the collection of *Ac_conv* issues based on field values
get_cdc_glitch	Creates a collection of clock domain crossings (in the current design) that may have glitches and match certain criteria
get_paths	Reports complete paths between the specified start and end points
report_cdc	Reports clock domain crossing details
report_cdc_coherency	Displays the collection of coherency/convergence issues reported by the *get_cdc_coherency* Tcl command
report_cdc_glitch	Reports clock domain crossing with glitches
report_paths	Reports elements in a defined path in the current design

Clock Domain Crossing Synchronization Schemes

A synchronization scheme is a method to synchronize a clock-domain crossing in a design.

The following synchronization schemes are considered by SpyGlass CDC:

Synchronization scheme	Crossings that are considered as synchronized
Conventional Multi-Flop Synchronization Scheme	Where flip-flops are in a synchronization flip-flops arrangement
Synchronizing Cell Synchronization Scheme	Where the destination object is an instance of a synchronizing cell specified by the *sync_cell* constraint, *synchronize_cells* parameter, or *synchronize_data_cells* parameter
Synchronized Enable Synchronization Scheme	Where the first flip-flop in the destination clock domain is enabled by a signal synchronized to the destination clock, and the clock crossing is in the data path.
Recirculation MUX Synchronization Scheme	Where the first flip-flop in the destination clock domain is driven by a MUX.
MUX-Select Sync (Without Recirculation) Synchronization Scheme	Similar to the *Recirculation MUX Synchronization Scheme*, but the MUX is not a recirculation MUX
Delay Signals Synchronization Scheme	Where the crossing is in the fan-out of the signals specified by the *quasi_static* constraint.
AND Gate Synchronization Scheme	Where AND gates exist in the data path of the crossing

Synchronization scheme	Crossings that are considered as synchronized
Glitch Protection Cell Synchronization Scheme	Where the first flip-flop in the destination domain is driven by the instance of a glitch protection cell and an input pin of the cell is synchronized by synchronizer flip-flops
Clock-Gating Cell Synchronization Scheme	Where the clock path of the flip-flop has a clock-gating cell and an input pin of the cell is synchronized
Qualifier Synchronization Scheme	Where a valid qualifier specified by the *qualifier* constraint reaches the source or destination of the crossing depending on its type
Qualifier Synchronization Scheme Using qualifier -crossing	Where the *qualifier* *-crossing* constraint is specified on the crossing output

Conventional Multi-Flop Synchronization Scheme

This scheme marks those clock crossings as synchronized where flip-flops are in a synchronization flip-flops arrangement. You can set the number of flip-flops in the synchronization chain by using the *num_flops* constraint. For details, see *Controlling the Number of Flip-Flops in a Synchronizer Chain*.

The following figure shows the example of this scheme:

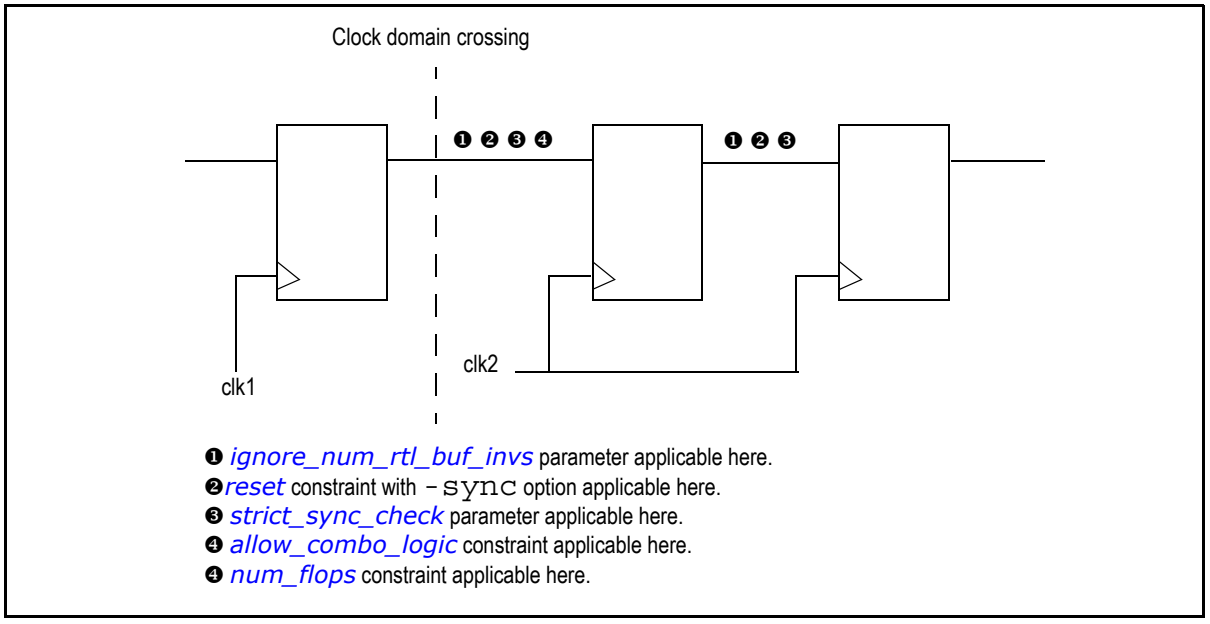


FIGURE 1. Conventional Multi-flop Synchronization Scheme

In the above scheme, the destination object can also be a latch.

To disable this scheme, set the *enable_multiflop_sync* parameter to `no`.

The following table shows how you can customize this scheme by using SpyGlass-CDC parameters and constraints:

Parameter / Constraint	Customizing the scheme
num_flops constraint	By default, this scheme marks those clock crossings as synchronized where two flip-flops are in a synchronization flip-flops arrangement. Use the *num_flops* constraint to specify a different number. For details, see *Controlling the Number of Flip-Flops in a Synchronizer Chain*.
allow_combo_logic constraint	By default, this scheme reports clock crossings with combinational logic in the data transfer path between flip-flops at clock domain crossing as unsynchronized. Use the *allow_combo_logic* constraint to ignore combinational logic. Transparent latches (enabled latch) are also considered as combinational elements.
output_not_used constraint	By default, the two-flip-flop synchronization strategy fails in the following cases: ● If the output of a flip-flop in the synchronization chain is connected to a primary output (ignoring buffers and inverters). Use *output_not_used* to specify the name of the primary output port so that the connection is ignored while checking for synchronization. ● If the output of the destination flip-flop has more than one fan-out. However, this scheme allows multiple fan-outs when a fan-out is blocked or when a fan-out is connected to a primary output port and that port is specified using *output_not_used* constraint.-

Parameter / Constraint	Customizing the scheme
allow_half_sync parameter	Unset the *allow_half_sync* parameter (that is, set to no) to ignore half synchronizers. By default, this scheme allows half synchronizers as valid synchronizers.
strict_double_flop parameter	Set the *strict_double_flop* parameter to mark only those clock crossings as synchronized where another flip-flop exists between the source flip-flop and the destination flip-flop and its clock signal is the inverse of the source flip-flop's clock signal. This scheme requires that all objects at the clock crossings must be flip-flops except the source object that can be a black box instance if the *strict_double_flop* parameter is not specified.

Controlling the Number of Flip-Flops in a Synchronizer Chain

Consider the following figure:

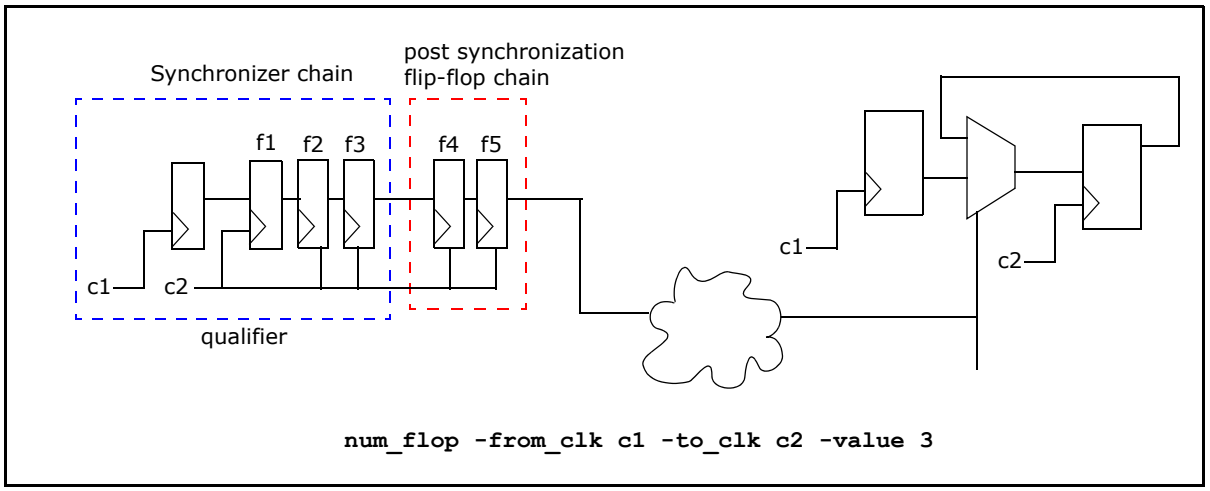


FIGURE 2. Controlling the number of flip-flops in a synchronizer chain

In the above scenario, the number of flip-flops allowed in the synchronizer chain is 3 as the value of the *num_flops* constraint is 3. Therefore, `f1`, `f2`, and `f3` are considered the part of the synchronizer chain, and the structure in the blue dotted line in the above figure is considered as a qualifier.

In this case, the `f4` and `f5` flip-flops are the part of post synchronization flip-flop chain, and are overall considered as a part of the synchronization flip-flop chain of this multi-flop synchronous qualifier.

Synchronizing Cell Synchronization Scheme

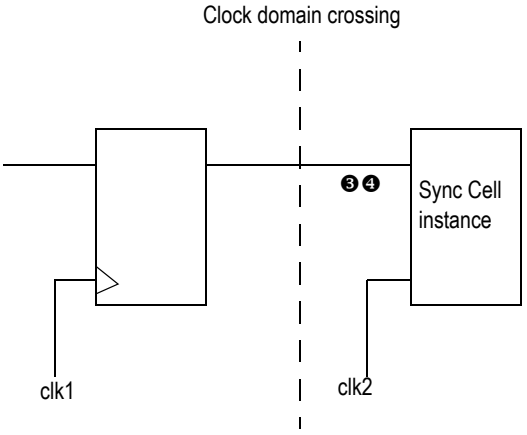
This scheme marks those clock crossings as synchronized in which the destination object is an instance of a synchronizing cell specified by any of the following:

- The *sync_cell* constraint
- The *synchronize_cells* parameter

  Use this parameter to specify control synchronization cells for scalar control crossings, that is crossings in which the destination signal is described with no bus index or range.
- The *synchronize_data_cells* parameter

  Use this parameter to specify data synchronizer cells for vector control crossings, that is crossings in which the destination signal is described with a bus index or range.

The following figure shows the crossing synchronized by this scheme:



❸ *strict_sync_check* parameter applicable here.
❹ *allow_combo_logic* constraint applicable here.

FIGURE 3. Synchronizing Cell Synchronization Scheme

In this scheme:

- The source object should be a flip-flop or a black box instance

- The destination object should be any of the following:
  - A design unit instance (soft instance)

    That is, a design object can be a cell instance that contains a functional description.
  - A black box instance (hard instance)

    That is, a design object can be a cell instance that contains no functional description.

  This distinction is significant because unlike hard instances, soft instances specified by using the *sync_cell* constraint or by using the *synchronize_cells* or *synchronize_data_cells* parameters must be determined to describe a flip-flop for each destination bit. Otherwise, the control crossing is not considered as synchronized by this scheme.

Synchronized Enable Synchronization Scheme

This scheme marks those clock crossings as synchronized where the first flip-flop in the destination clock domain is enabled by a signal synchronized to the destination clock and the clock crossing is in the data path.

The clock domain crossing is marked as synchronized if either of the following conditions is met:

- The enable pin is driven by a signal synchronized to the destination clock domain.
- A valid synchronizer exists in any one of the paths driving the enable pin and signals in all other paths are driven either by primary ports or destination clock domain flip-flops and there is no unsynchronized crossing in any of the paths.

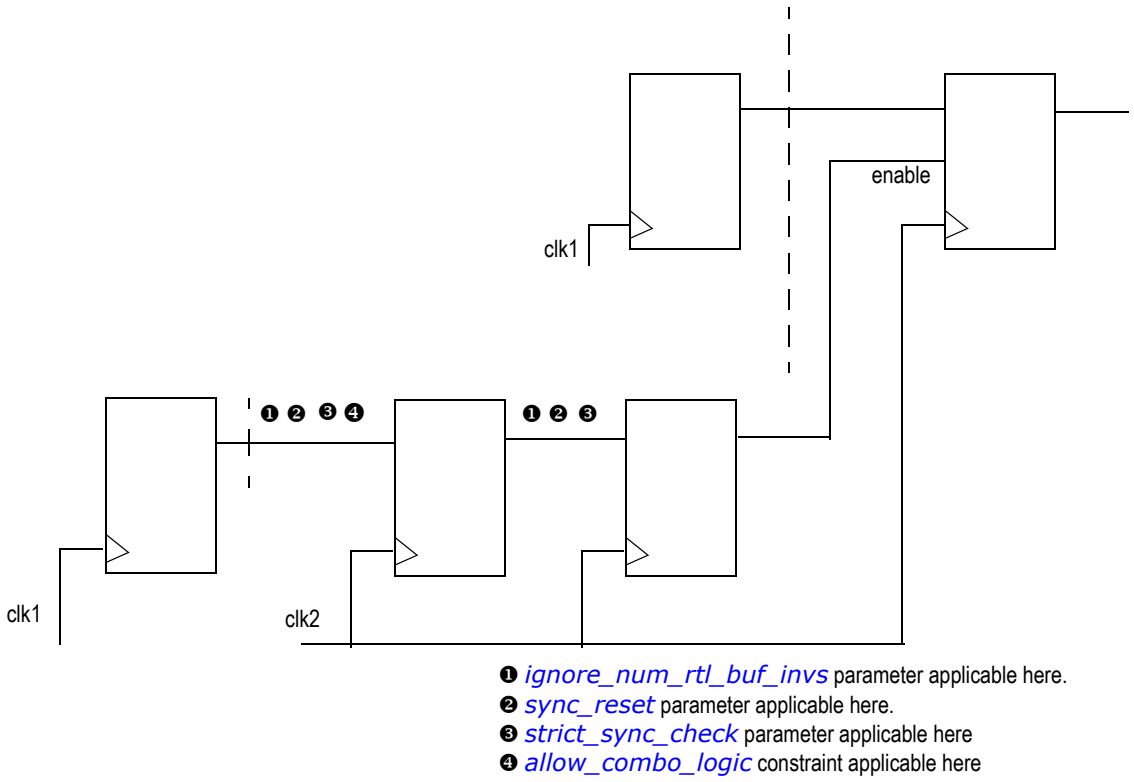


FIGURE 4. Synchronization Through Common Enable Scheme

This scheme allows the source object to be a flip-flop or a black box instance. All other objects must be flip-flops.

This scheme also allows transparent latches between the enable synchronizer and the destination flip-flop enable.

This scheme also considers a clock crossing to be synchronized if:

- The enable pin of the destination flip-flop is driven by nets coming from the same clock domain as the destination flip-flop when the *enable_mux_dest_domain* parameter is set.

- The enable pin of the destination flip-flop is driven by an instance of a synchronizer cell specified using the *enable_sync_cell* parameter.

By default, this scheme is always run. Unset the *enable_sync* parameter to disable this scheme.

Recirculation MUX Synchronization Scheme

This scheme marks those clock crossings as synchronized where the first flip-flop in the destination clock domain is driven by a MUX. The clock domain crossing happens through one of the MUX input pins and the other MUX input pin is driven by the destination flip-flop output.

The clock domain crossing is marked as synchronized if either of the following conditions is met:

- The MUX select pin is driven by a signal synchronized to the destination clock domain.
- A valid synchronizer exists in any one of the paths driving the MUX select pin and signals in all other paths are driven either by primary ports or the destination clock domain flip-flops and there is no unsynchronized crossing in any of the paths.

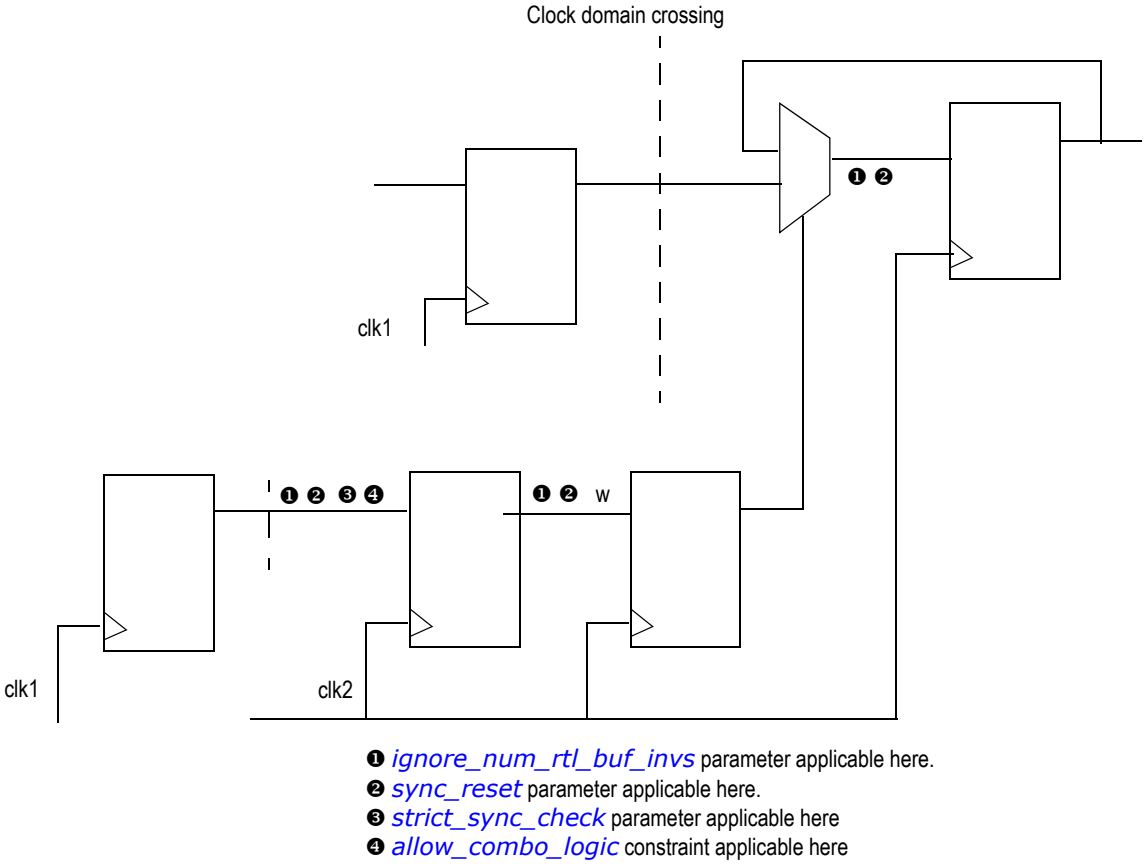


FIGURE 5. Synchronization Through Common Select Scheme

This scheme allows the source object to be a flip-flop or a black box instance. All other objects must be flip-flops.

This scheme also allows transparent latches between the select pin synchronizer and the select pin of MUX.

By default, this scheme is always run. Use the *enable_mux_sync* parameter to disable this scheme.

Set the *enable_mux_dest_domain* parameter to consider those clock crossings as synchronized where the first flip-flop in the destination clock

domain is driven by a MUX where the select signal belongs to the destination domain (that is, driven by a flip-flop of the destination domain).

This scheme also considers a clock crossing to be synchronized if the select pin of MUX is driven by an instance of a synchronizer cell specified using the *enable_sync_cell* parameter.

This scheme rule should be run when the corresponding synchronization strategy is acceptable.

MUX-Select Sync (Without Recirculation) Synchronization Scheme

This scheme is the same as the *Recirculation MUX Synchronization Scheme* except for the following differences:

- The MUX need not be a recirculation MUX.
- All select signals of the MUX must be individually synchronized in the destination domain.
- At least one data line of the MUX should be free of source and that data line should be either coming from a destination domain or should be tied to a constant or supply. See *Figure 6*.

  To restrict this behavior by allowing only destination domain and not allowing a constant or supply, set the *enable_mux_sync* parameter to `strict`.

The following figure shows the example of a crossing synchronized by this scheme:

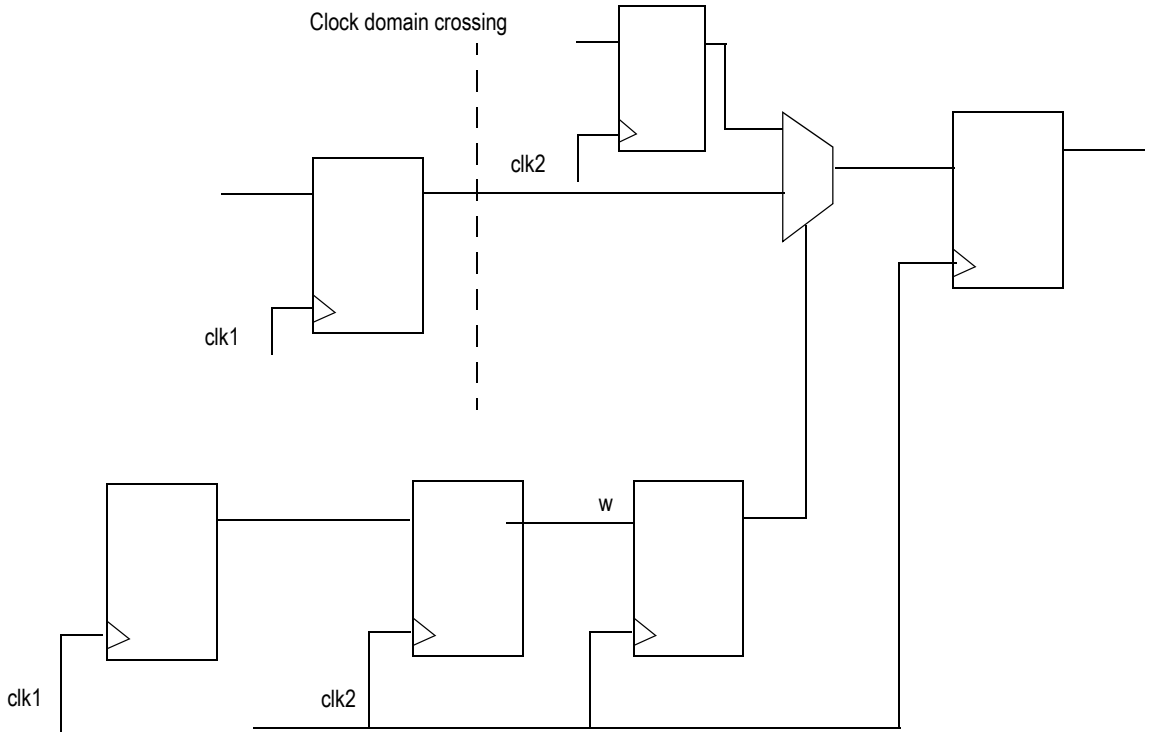


FIGURE 6. MUX-Select Sync (Without Recirculation) Synchronization Scheme

This scheme allows transparent latches between the select pin synchronizer and the select pin of MUX.

By default, this scheme is not run. Set the *enable_mux_sync* parameter to `mux_select`, `strict`, or `all` to run this scheme.

Delay Signals Synchronization Scheme

This scheme marks those clock crossings as synchronized that are in the fan-out of signals specified by using the *quasi_static* keyword in a design constraints file.

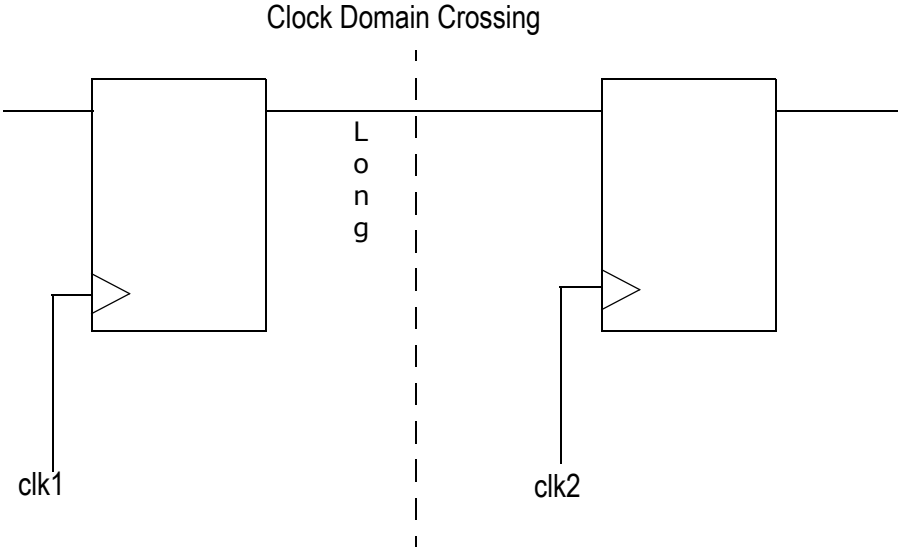


FIGURE 7. Synchronization of Long Delay Signals Scheme

This scheme allows the source object and destination object to be a flip-flop or a black box instance.

This scheme is run only when you supply a valid `quasi_static` constraint in a design constraints file.

NOTE: *The common parameters are not applicable for this scheme.*

AND Gate Synchronization Scheme

The *AND Gate Synchronization Scheme* checks for synchronization at AND gates in the data path of a clock domain crossing.

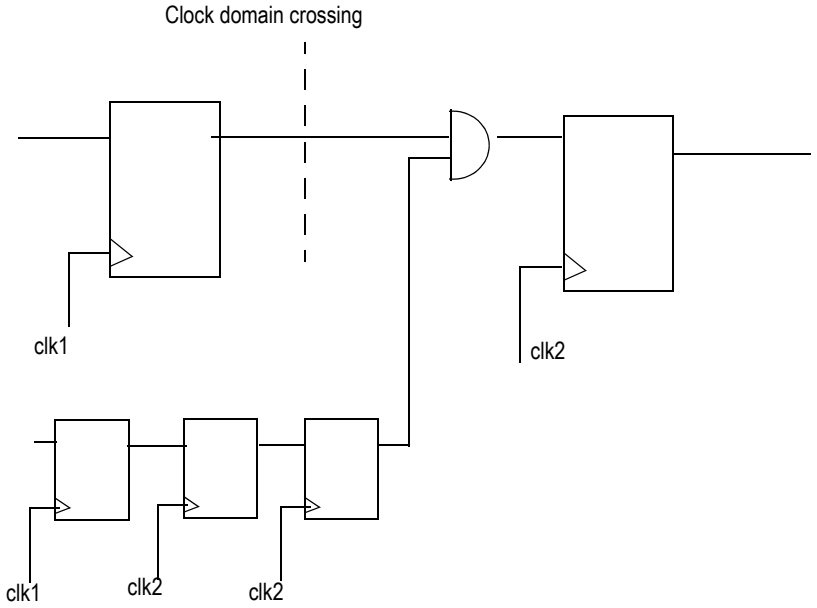


FIGURE 8. AND Gate Synchronization Scheme

This scheme marks those clock crossings as synchronized where:

- The first flip-flop in the destination domain is directly driven by an AND gate.
- The other input (input pin not connected to source flip-flop) of the AND gate is synchronized by synchronizer flip-flops in the destination domain clock or any destination domain signal without synchronizer if *enable_mux_dest_domain* parameter is set to `yes`.

This scheme allows combinational logic between the synchronizer and the input pin of the AND gate in data path. In this case, the fan-in cone of the combinational logic should be coming either from the destination domain or

from the primary ports. Transparent latches (enabled latch) are also considered as combinational elements.

This synchronization scheme is enabled by the *enable_and_sync* parameter.

Glitch Protection Cell Synchronization Scheme

The *Glitch Protection Cell Synchronization Scheme* marks those clock domain crossings as synchronized where:

- The first flip-flop in the destination domain is driven by an instance of a glitch protection cell (specified using the *glitch_protect_cell* parameter)
- The other input (input pin not connected to source flip-flop) of the glitch protection cell instance is synchronized by the synchronizer flip-flops in the destination domain clock or any destination domain signal without synchronizer if *enable_mux_dest_domain* parameter is set to `yes`.

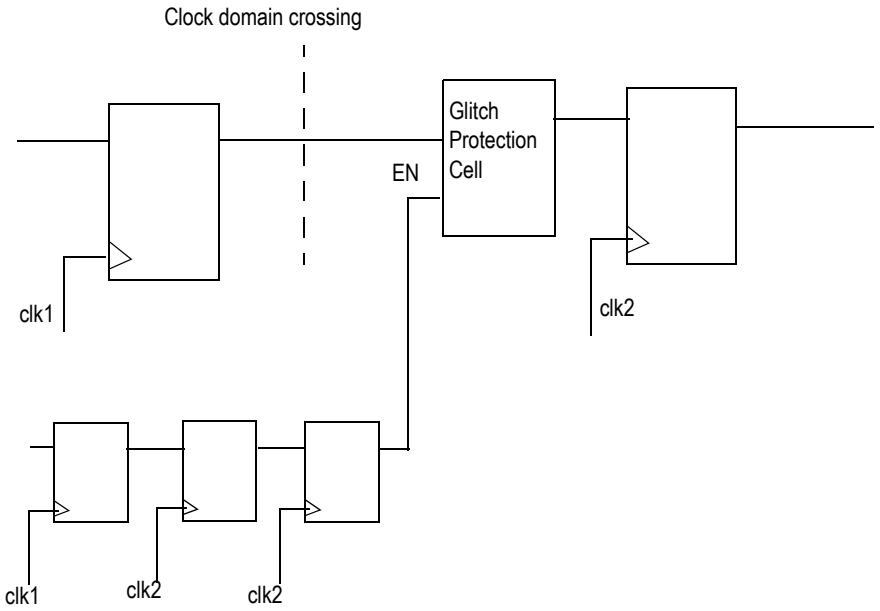


FIGURE 9. Glitch Protection Cell Synchronization Scheme

This scheme allows combinational logic between the synchronizer and the input pin of the glitch protection cell instance in data path. However, there should be no other glitch protection cell instance between the glitch protection cell connected to the synchronizer and the destination flip-flop. Transparent latches (enabled latch) are also considered as combinational elements.

This scheme runs only when you specify the *glitch_protect_cell* parameter with valid value.

The scheme supports the `input` and *set_case_analysis* constraints.

Clock-Gating Cell Synchronization Scheme

The *Clock-Gating Cell Synchronization Scheme* marks those clock domain crossings as synchronized where:

- The clock path of the destination flip-flop has a clock-gating cell, and
- The other input (input pin not connected to clock) of the clock-gating cell instance is synchronized or is coming from destination domain signals, if *enable_mux_dest_domain* parameter is set to `yes`.

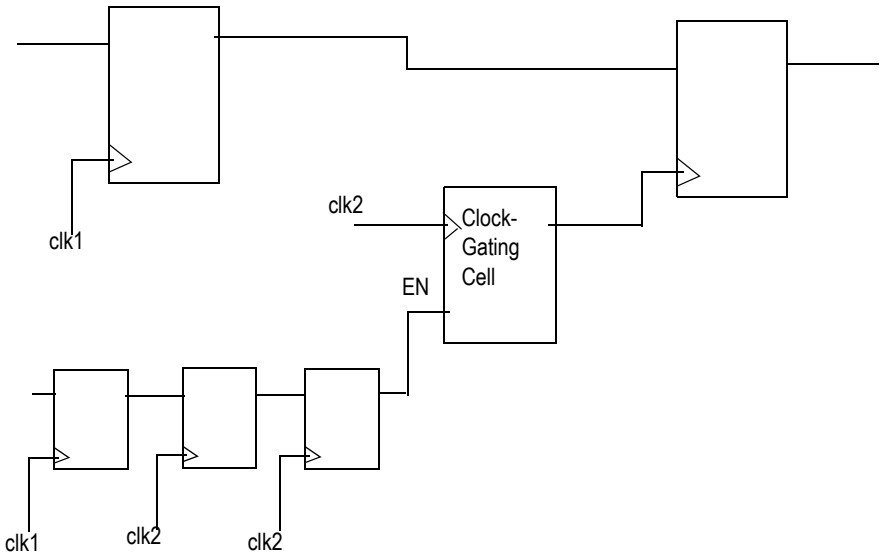


FIGURE 10. Clock-Gating Cell Synchronization Scheme

This scheme allows combinational logic between the synchronizer and the input pin of the clock-gating cell instance in the clock path. There should be no other combinational gate between the clock-gating cell instance and the clock pin of the destination flip-flop. Transparent latches (enabled latch) are also considered as combinational elements.

The clock-gating cells are identified in any of the following three ways:

- A clock-gating cell instantiated from a .lib file has an attribute, `clock_gating_integrated_cell`.

- Clock-gating cell names are specified with the *clock_gate_cell* parameter.
- Clock-gating structures are automatically identified from the RTL. SpyGlass identifies the following structures at RTL:
  - CGLP structure (Positive edge triggered clock-gating), as shown below:

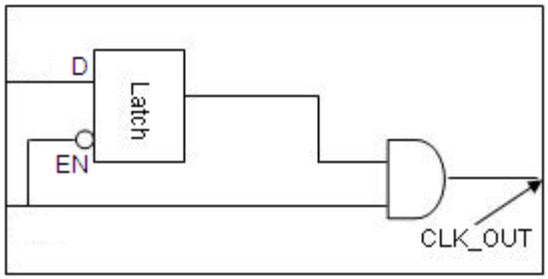


FIGURE 11. CGLP Structure

  - CGLN structure (Negative edge triggered clock-gating), as shown below:

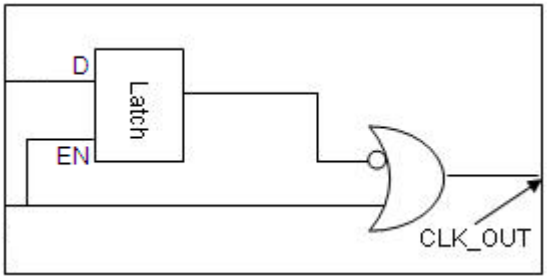


FIGURE 12. CGLN Structure

Output of the latch may also have an extra OR gate for scan-enable. Such a structure will also be recognized.

This synchronization scheme is enabled by the *enable_clock_gate_sync* parameter.

Qualifier Synchronization Scheme

The *Qualifier Synchronization Scheme* marks those clock domain crossings as synchronized where a valid qualifier specified by the *qualifier* constraint reaches the source or destination of the crossing depending on its type.

A valid qualifier can be:

- A primary port.
- Output of a black box.
- Output of a sequential element belonging to source domains if the type specified in the *qualifier* constraint is `src`.
- Output of a sequential element belonging to destination domains if the type specified in the *qualifier* constraint is `des`.
- Output of a combinational element that has only source or destination domains in the fan-in cone, depending on the type of the qualifier

This scheme helps in reducing false violations by providing qualifiers that may not be automatically identified. It is checked after checking all the other synchronization schemes.

The following figure shows the crossing synchronized by this scheme:

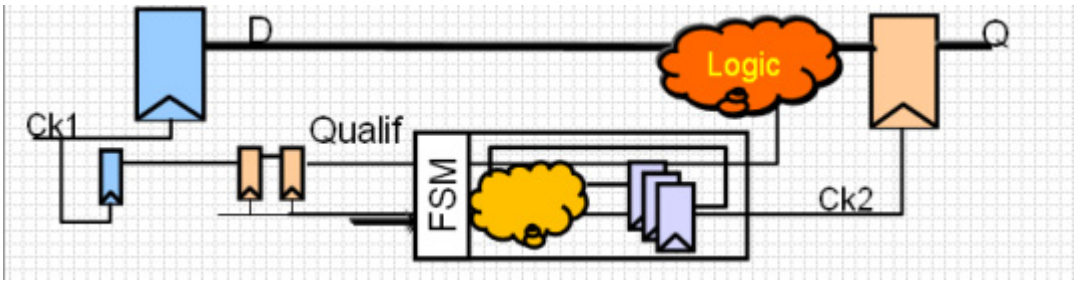


FIGURE 13. Qualifier Synchronization Scheme

If a qualifier reaches the pin of a source or destination instance other than preset/clear, that crossing is considered as synchronized by this scheme.

The instance where a qualifier reaches should be a sequential element or an output port. If the instance is a black box, this scheme ignores it. However, this scheme considers cases where a qualifier reaches a

destination flip-flop instance and source is a black box or vice-versa.

Design Areas where a Qualifier is Not Propagated

A qualifier propagates forward in the design to trace crossings. Propagation stops at the following design areas:

- At source or destination flip-flops of clock-domain crossings
- At the output of a sequential element belonging to a domain other than source if the type specified in the *qualifier* constraint is `src`
- At the output of a sequential element belonging to a domain other than destination if the type specified in the *qualifier* constraint is `des`
- At blocked paths
- At black boxes specified without *assume_path*
- At memories
- At the reset pin of sequential elements

Crossings with Qualifier Specified for Strict Checking

If you specify the `-strict` option of the *qualifier* constraint for a qualifier, the constraint behavior will be different from the default behavior of this scheme.

- It requires enable, recirculation, AND-based logic, GP cell, and clock gate cell controlled by qualifier; no other logic will be accepted.

  Use the *enable_and_sync*, *enable_mux_sync*, *enable_clock_gate_sync*, *enable_sync*, and *glitch_protect_cell* parameters to enable/disable particular schemes.
- If the qualifier is output of destination flip-flop or synchronizer output of a crossing, then it would match the source instance domain with `-from_clk`/`-from_domain` specified in *qualifier* constraint.

Qualifier Synchronization Scheme Using qualifier -crossing

Consider that you use a qualifier specified by the *qualifier* `-crossing` constraint on a crossing output. In this case, the crossing is considered as synchronized.

For example, consider the following figure:

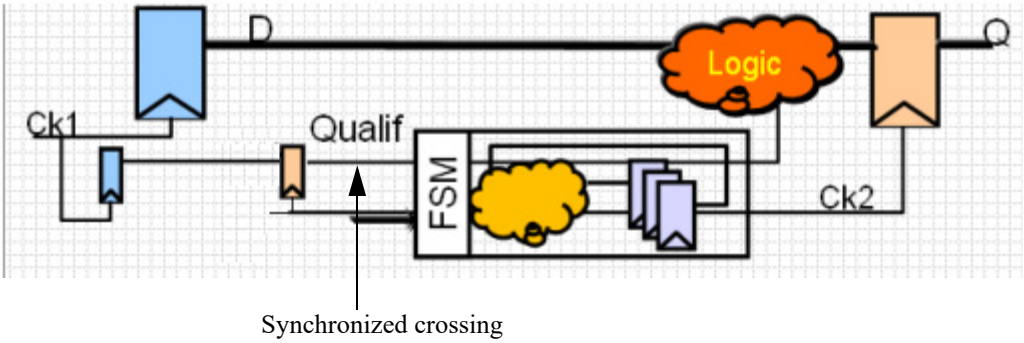


FIGURE 14. Qualifier Synchronization Scheme Using qualifier -crossing

In the above figure, the crossing is considered as synchronized when the `-crossing` argument of the `qualifier` constraint has been specified.

Using the Clock Setup Window

The clock setup information refers to the information related to *Clock Sources Section of the Clock Setup Window* and *Clock Cones Section of the Clock Setup Window* in a design. Viewing this information enables you to understand the clock architecture of your design.

To view the clock setup information, double-click on the violation of the *Setup_clock01* rule. The *SpyGlass Clock Setup* window appears showing the clock setup information.

The following figure shows the *SpyGlass Clock Setup* window:

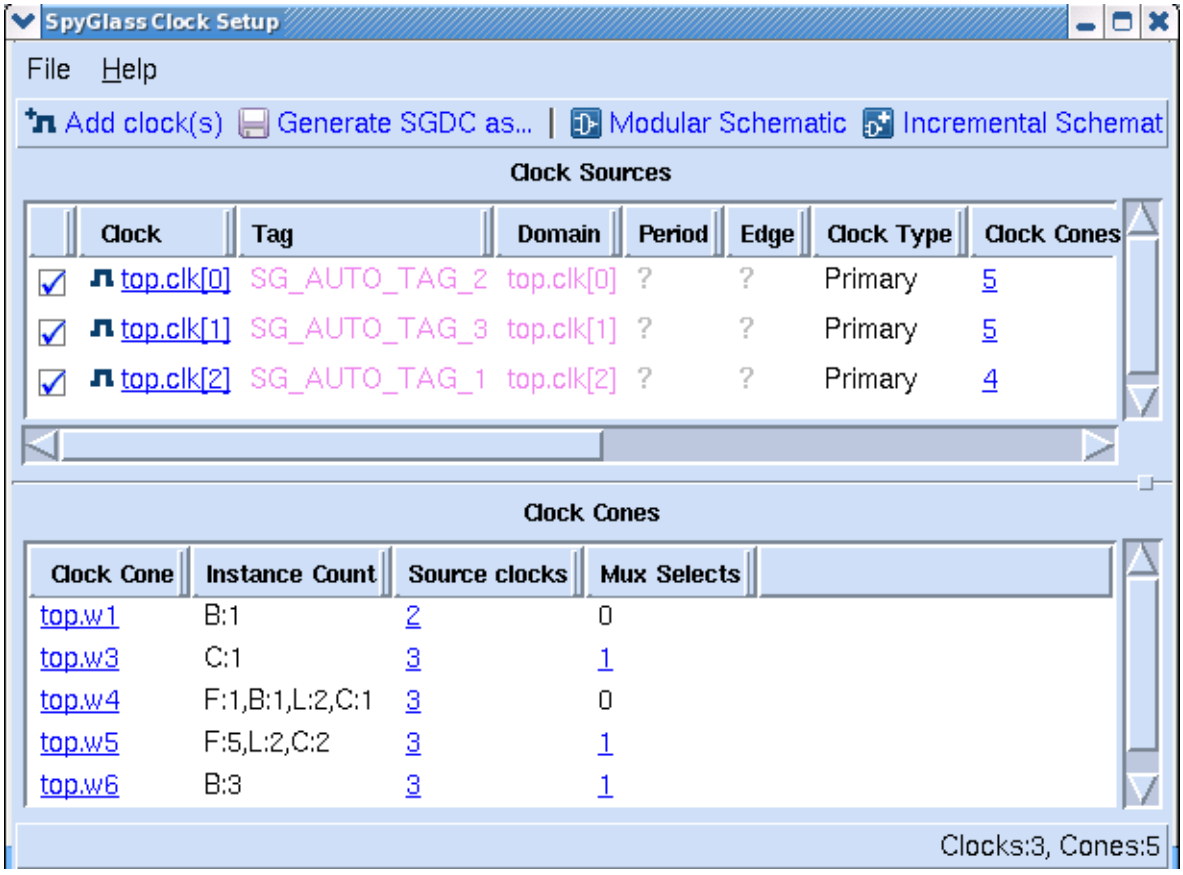


FIGURE 1. The Clock Setup Window

In the above window, when you click on a clock in the *Clock Sources Section of the Clock Setup Window* view, its corresponding clock cones appear in the *Clock Cones Section of the Clock Setup Window* view. Similarly, when you select a clock cone in the *Clock Cones Section of the Clock Setup Window* view, the clocks of the selected cone appear in the *Clock Sources Section of the Clock Setup Window* view.

Viewing the Clock Setup Information

The clock setup information in *The Clock Setup Window* is divided in the following sections:

- *Clock Sources Section of the Clock Setup Window*
- *Clock Cones Section of the Clock Setup Window*

Clock Sources Section of the Clock Setup Window

The *Clock Sources* view shows information about clocks in a design. This information is categorized under the following fields:

Field	Description
Clock	Specifies the clock name.
Domain	Specifies the name of the clock domain. If the domain name is not specified in the *clock* constraint, the domain name is the same as the clock name. If the domain name is not specified for a virtual clock, *sg_virtual* string gets appended to the domain name. In this field, when you click on a domain name, the button appears. Click this button to view all domain names in a drop-down list.
Period	Displays the clock period (in nanoseconds) if the period is specified in the *clock* constraint.
Edge	Displays the edge list specified by the *clock* constraint.
Clock Type	Displays the clock type, such as *Primary*, *Black box*, *derived*, and *virtual*. If the clock type is *Black box*, you can set constraints, such as *assume_path* and *abstract_port* on it. To set constraints, click on *Black box* to display the *Constraints Editor* window.
Clock Cones	Displays the number of clock cones associated with the source clock. When you click the clock cone number, the clock cones associated with the source clock appears in the *Clock Cones Section of the Clock Setup Window* window.

Field	Description
Mux Selects	Displays the count of select pins of all muxes that lie in the clock path. When you click the mux-select number, the mux selects associated with the source clock appear in the *Mux Selects* window. These mux selects are determined based on the value specified by the *sel_case_analysis_mode* parameter.
Latch/Flop	Specifies if the clock is driving flip-flops, latches, or both. Possible values appearing in this field are *latch*, *flop*, *all*, and *none*.
Source	Displays the SGDC/SDC file containing the clock. When you click the file name, the line of the source code in the SGDC/SDC file containing the clock is highlighted in the HDL window. However, if the clock is inferred automatically, the text *Auto-Inferred* appears in this field.

DFT mode	Displays a drop-down list showing the *testclock* and *atspeed* options.
	Based on the value selected in this list, SpyGlass adds either of the following arguments to the *clock* constraint while generating SGDC:
	`-testclock`
	`-testclock -atspeed`

Additional Option	Enables you to specify additional options, such as *-fflimit* and *value* to the *clock* constraint.

Clock Cones Section of the Clock Setup Window

The *Clock Cones* view displays information about the clock cones present in a design. This view appears when you double-click a clock in the *Clock Sources Section of the Clock Setup Window* view.

The *Clock Cones* view contains information under the following fields:

Field	Description
Clock Cone	Shows the name of the clock cone associated with the clock selected in the *Clock Sources Section of the Clock Setup Window* view.
Num Flops	Shows the number of flip-flops (F)/ latches (L)/ black boxes (B), or sequential cells (C) connected to the clock cone.
Source clocks	Shows the number of source clocks associated with the clock cone. When you click on a value in this field, the source clocks appear in the *Clock Sources Section of the Clock Setup Window* view.
Mux Select	Shows the count of select pins of all muxes that lie in the fan-in cone of the clock cone node. When you click a value in this field, the mux selects associated with the clock cone appear in the *Mux Selects* window

Adding Clocks in the Clock Setup Window

A design may contain elements for which no clock is defined.

To define clocks on such elements using *The Clock Setup Window*, select the *File* > *Add Clocks* menu option in this window or click  on the tool bar of this window.

The *Add Clocks* dialog appears in which you specify details, such as name of the clock, domain, and clock period.

Generating SGDC Files From the Clock Setup Window

To generate an SGDC file containing the *clock* constraints for the clocks in *The Clock Setup Window*, perform any of the following actions in this window:

- Select the *File* > *Save As SGDC* menu option.
- Click the *Save As SGDC* link in the tool bar.

On performing the above actions, a dialog appears in which specify the name and path of the file to be generated.

The following figure shows *The Clock Setup Window* and the SGDC file generated for the information in this window:

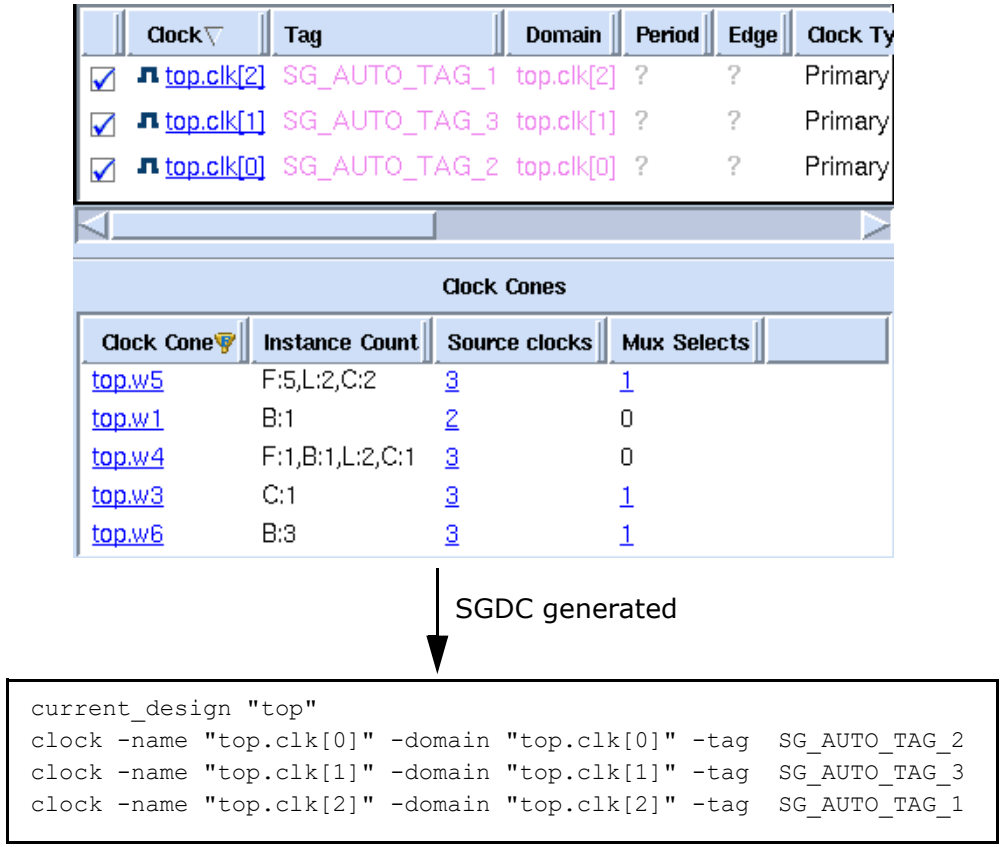


FIGURE 2. SGDC Generated from the Clock Setup Window

Filtering Information in the Clock Setup Window

You can set filters on each column of *The Clock Setup Window* to view selective information.

To filter information in a column, right-click on the column header. A shortcut menu appears showing the list of valid values in that column. By default, *All* is selected. Select the option you want to display.

In addition, you can also set custom filters using the *Custom* option from the shortcut menu. The following figure shows the *Custom Filter* dialog for the column clock:

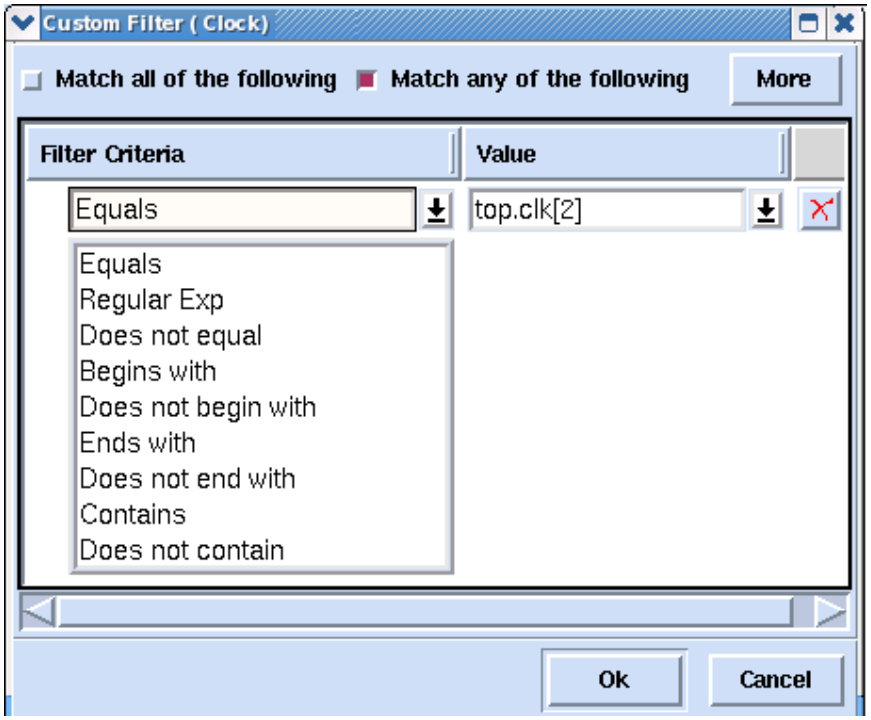


FIGURE 3. Specifying Custom Filters in the Clock Setup Window

Viewing Clock Details in HDL Window and Schematic

When you click on a clock in the *Clock Sources Section of the Clock Setup Window* view of *The Clock Setup Window*, the following occurs:

- The line of the source code containing the clock is highlighted in the HDL window.

  For virtual clocks, the line number in the corresponding SGDC file is highlighted in the HDL window.

- The clock information is highlighted in the *Incremental Schematic* window.

  The schematic shows abstracted groups, that contains combinational-logic (excluding MUX) in the clock path. The following figure shows abstract groups:

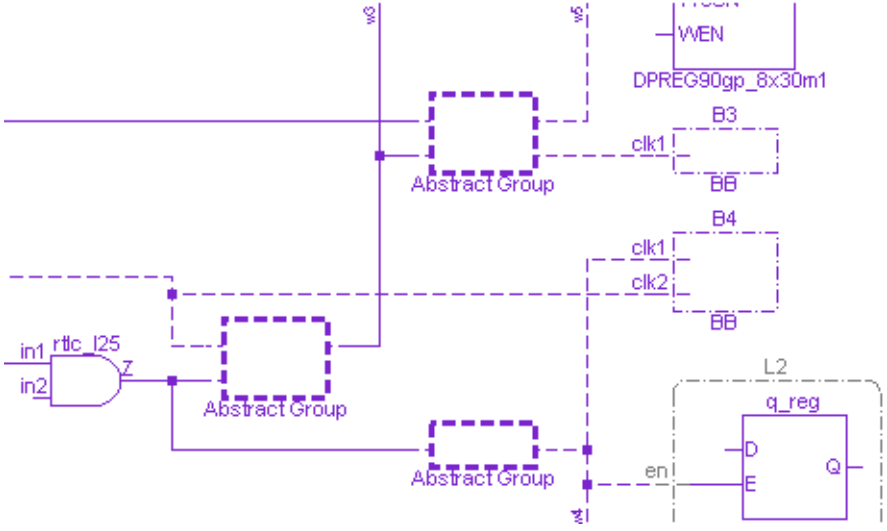


FIGURE 4. Abstract Groups in Schematic

Double-click on a group to show/hide the logic. The following figure shows some expanded abstract groups:

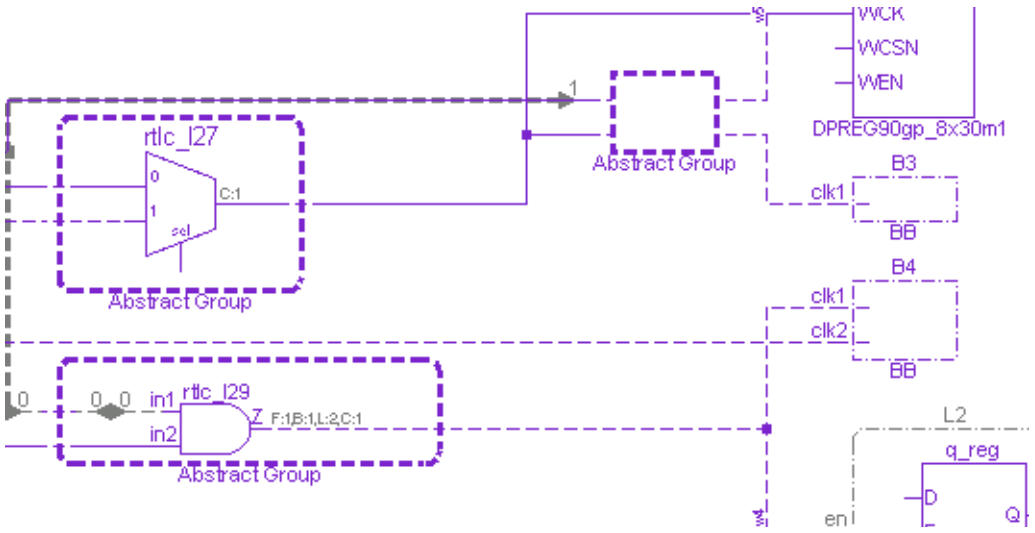


FIGURE 5. Expanded Abstract Groups

Viewing Schematic for Multiple Clocks

To view the schematic of multiple clocks appearing in the *Clock Sources Section of the Clock Setup Window* view of *The Clock Setup Window*, perform the following steps:

1. Select multiple clocks by keeping the *Ctrl* key pressed.
2. Open the *Incremental Schematic* window.
3. Select the *Edit -> Set Display Mode -> Complete* option in the *Incremental Schematic* window.

   This step sets the *incremental schematic* mode to enable the view of complete schematic for all the selected clocks. If you do not set this mode, an overlapping schematic appears.

Saving Changes in the Clock Setup Window

On closing *The Clock Setup Window* after modifications, the Sp*yGlass Warning* dialog appears that prompts you to save the changes.

On clicking *Yes* in this dialog, the CSV file is updated with the changes. This ensures that the changes are visible even when you open the *SpyGlass Clock Setup* window the next time. However, on making the changes, the original content of the CSV file is overwritten and cannot be retrieved.

Working With the Ac_sync_group Rules

The Ac_sync_group Rules report different types of synchronized and unsynchronized crossings in a design.

This section covers the following topics:

The Ac_sync_group Rules

The following rules belong to the `Ac_sync_group`:

Rule Name	Description
Ac_sync01	Reports asynchronous clock domain crossings for scalar signals that have all the sources synchronized
Ac_sync02	Reports asynchronous clock domain crossings for vector signals that have all sources synchronized
Ac_unsync01	Reports asynchronous clock domain crossings for scalar signals that have at least one unsynchronized source
Ac_unsync02	Reports asynchronous clock domain crossings for vector signals having at least one source that is unsynchronized

NOTE: *You can waive the message of the above rules by using the waive -ip constraint, only if both source and destination are the part of the specified IP.*

Objects in the Crossings Reported by Ac_sync_group Rules

The following objects are involved in the clock-domain crossings reported by *The Ac_sync_group Rules*:

- *Source*
- *Destination*
- *Qualifier*
- *Potential Qualifier*

Source

It is a source instance or a port of a clock-domain crossing having different clock domains from that of destination.

As multiple sources can converge on a combinational logic before reaching the destination, the number of sources per destination can be more than one.

Destination

It is a destination instance or a port of a clock-domain crossing.

Qualifier

It is a signal that synchronizes a data crossing.

Typically, this will be a conventional multi-flop, synchronizing cell, user-defined qualifiers, or *abstract_port* synchronized crossing.

To qualify or synchronize a data crossing, a qualifier signal converges with the source signal before reaching the destination or synchronizes the destination through enable or clock pin. This is applicable only for data crossings.

The following figure shows a qualifier signal:

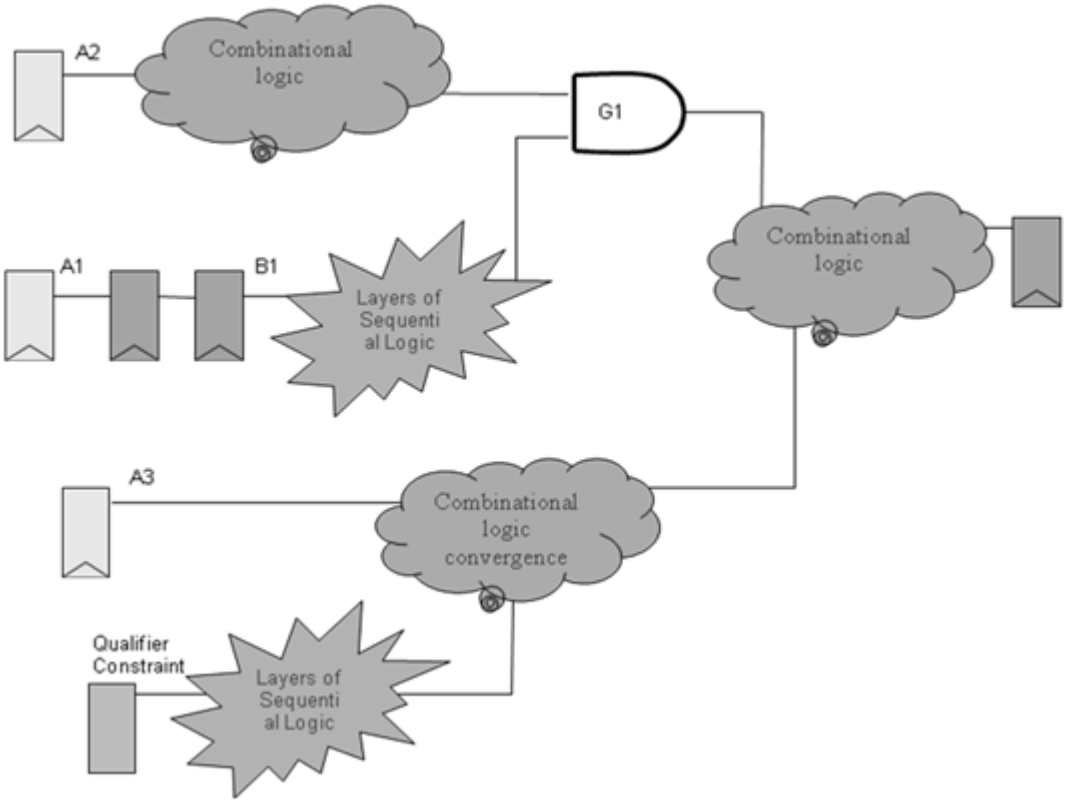


FIGURE 1. Advanced Usage

Types of Qualifiers

There are two types of qualifiers, *Detected Qualifier* and *User-Defined Qualifier*.

Detected Qualifier

It refers to a SpyGlass-inferred signal that is synchronized by using *Conventional Multi-Flop Synchronization Scheme* (`B1` in *Figure 1*) or *Synchronizing Cell Synchronization Scheme*.

This signal can qualify or synchronize a data crossing if it satisfies the following conditions:

- It has source signals (A1 in *Figure 1*) from a single clock domain.
- It is converging with the source of a data crossing (A2 in *Figure 1*) at a valid point, which can be one of the following:
  - Enable pin of destination
  - Clock pin of destination

    In this case, the qualifier is present in the enable pin of a clock-gating cell that is driving a destination clock.
  - A valid gate

    A gate is considered as valid based on the strict_gate and soft_gate options of the *ac_sync_mode* parameter.
- It is converging with the source of a data crossing (A2 in *Figure 1*) having the same clock domain as that of its source (A1 in *Figure 1*).

User-Defined Qualifier

Refers to a signal that is provided in any of the following ways:

- A user-defined qualifier specified by using the *qualifier* constraint
- A user defined *abstract_port* constraint with the -sync active argument

  In this constraint, the -from argument should have the same source-domain clock list as that of a source signal, and the -to argument should have the same destination-domain clock list as that of a destination flip-flop.

Potential Qualifier

Signal that fails to synchronize the source signal of data crossing due to the presence of invalid logic at the point of convergence with source, or the signal is not synchronized itself.

For example, the following figure illustrates the case of invalid gate convergence:

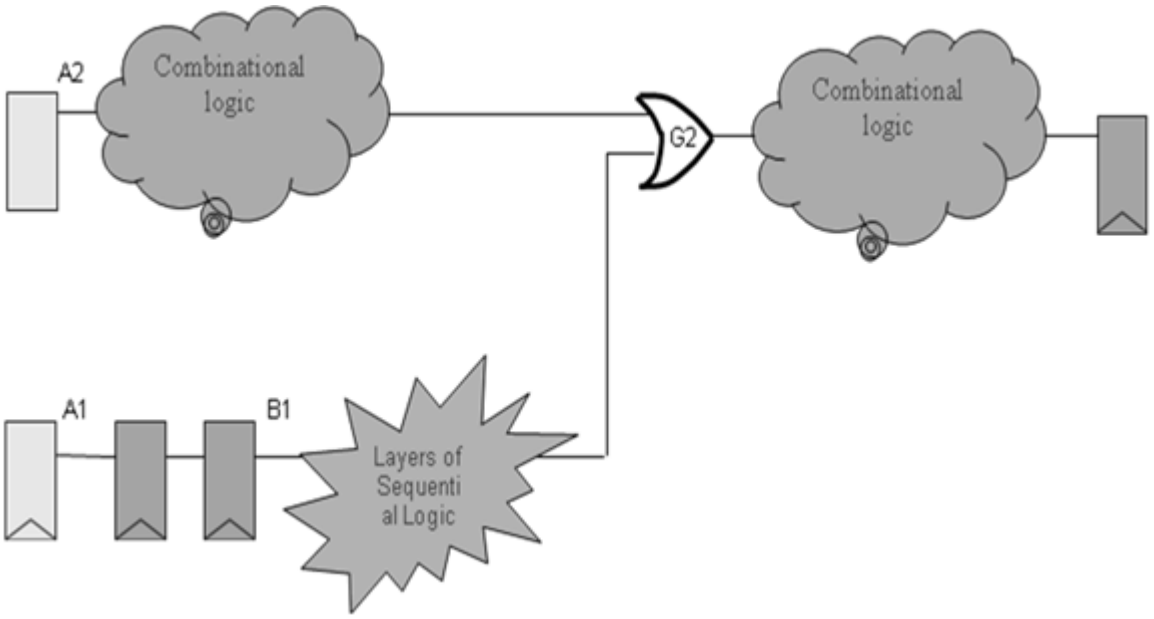


FIGURE 2. Invalid Gate Convergence

The following figure illustrates the scenario in which a synchronizer is missing:

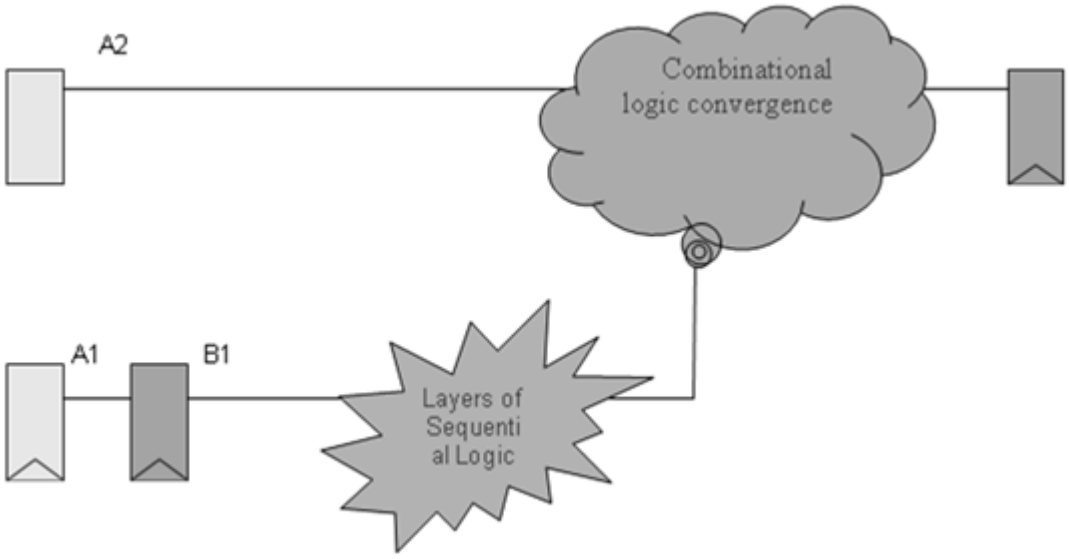


FIGURE 3. Missing Synchronizer

Potential qualifier inferred by the SpyGlass is the one whose destination satisfies the following conditions:

- Destination has source signals (`A1` in *Figure 2* / *Figure 3*) from the same domain.
- Destination is synchronized but converges with source at an invalid gate (`G2` in *Figure 2*).
- Destination is an unsynchronized scalar signal (`B1` in *Figure 3*).

Potential qualifier helps in identifying the cause of failure. If the failure is rectified, it may turn into a valid qualifier.

Special Cases of Crossings Containing Qualifiers

The following points describe special cases of synchronized and unsynchronized crossings through qualifiers:

- **Crossing contains multiple qualifiers existing for a source**

  Consider the following figure in which multiple qualifiers exist for a

source:

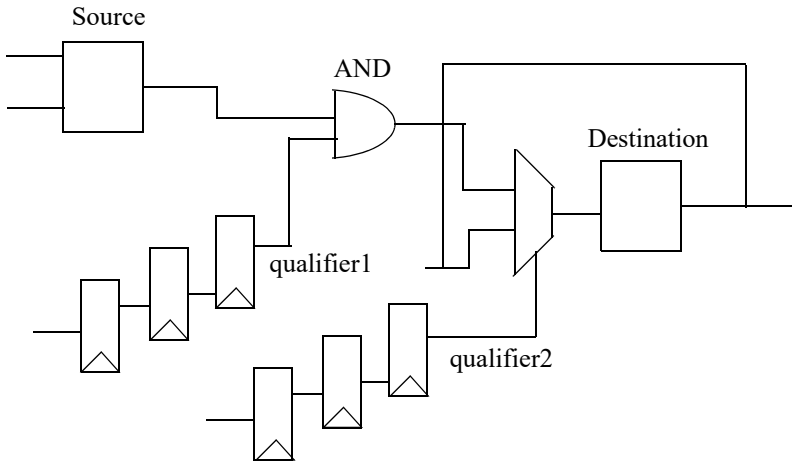


FIGURE 4. Missing Synchronizer

In the above case, first convergence of the source with a qualifier is occurring at the AND gate. By default, this gate is not considered as valid, and synchronization is reported with the second qualifier at the recirculation-mux.

However, if the *ac_sync_mode* parameter is set to `strict_gate` with the *enable_and_sync* parameter set to `yes`, or the *ac_sync_mode* parameter is set to `soft_gate`, the AND gate is considered as a valid gate. In this situation, synchronization of the source is reported at the first valid convergence, that is, with the first qualifier at the AND gate.

- **Valid qualifier converges with an unsynchronized crossing before merging with a source**

  Consider the following figure showing such a scenario:

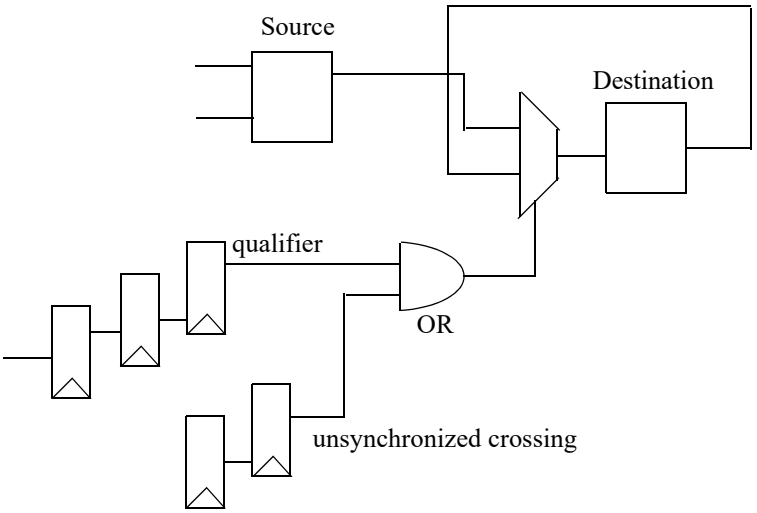


FIGURE 5. Missing Synchronizer

In the above case, the source is reported as a valid synchronized crossing by the *Ac_sync01* and *Ac_sync02* rules.

Spreadsheet Support in Ac_sync_group Rules

The `Ac_sync_group` rules generate the following types of spreadsheets (in CSV format):

- *Rule-Based Spreadsheet*
- *Message-Based Spreadsheet*

Rule-Based Spreadsheet

A rule-based spreadsheet is a global spreadsheet that displays all violations for a rule in separate rows.

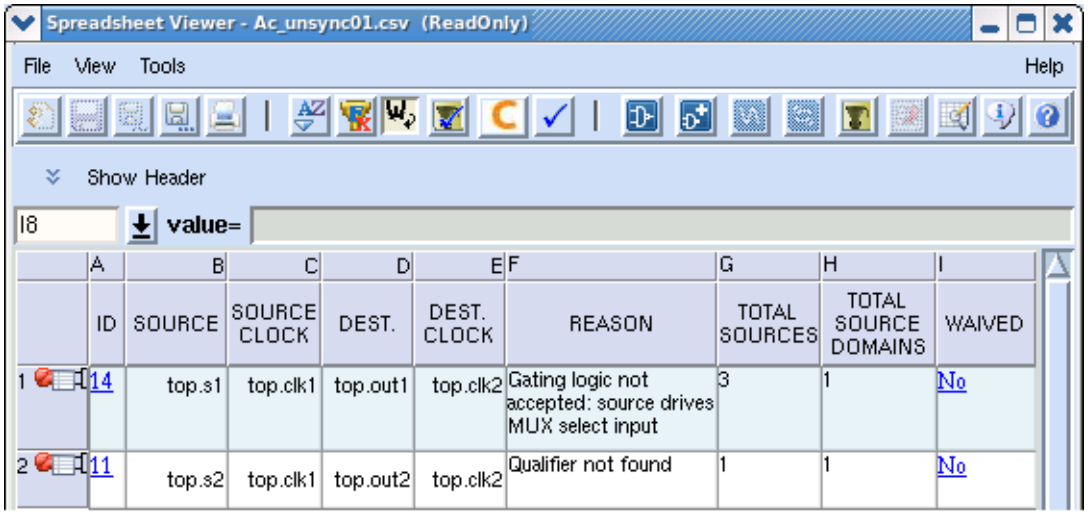
This spreadsheet displays one source per destination for a clock-domain crossing. To view details of all sources, view the *Message-Based Spreadsheet*.

Opening the Rule-Based Spreadsheet

To open the rule-based spreadsheet, right-click on the rule header in the *Results* pane, and select the *Open Spreadsheet* option from the shortcut menu.

Sample Rule-Based Spreadsheet

The following figure shows the rule-based spreadsheet of the *Ac_unsync01* rule:



	A	B	C	D	E	F	G	H	I
	ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	REASON	TOTAL SOURCES	TOTAL SOURCE DOMAINS	WAIVED
1	14	top.s1	top.clk1	top.out1	top.clk2	Gating logic not accepted: source drives MUX select input	3	1	No
2	11	top.s2	top.clk1	top.out2	top.clk2	Qualifier not found	1	1	No

FIGURE 6. Rule-Based Spreadsheet

Details of each column of the rule-based spreadsheet are described in the following table:

Column Name	Description
ID	Specifies a unique ID for a violation
SOURCE	Specifies the name of a source net or source instance
SOURCE CLOCK	Specifies the name of the clock reaching the source
DEST.	Specifies the name of a destination net or destination instance
DEST. CLOCK	Specifies the name of the clock reaching the destination

Column Name	Description
REASON / METHOD	For the *Ac_unsync01* and *Ac_unsync02* rules, the name of the column is REASON. This column displays the reason for unsynchronized crossing. For the *Ac_sync01* and *Ac_sync02* rules, the name of the column is METHOD. This column displays the synchronization method for synchronizing clock domain crossings.
TOTAL SOURCES	Specifies the total number of sources involved in a crossing
TOTAL SOURCE DOMAINS	Specifies the total number of source domains involved in a crossing
WAIVED	Specifies if the reported violation is waived

NOTE: *If you run The Ac_sync_group Rules in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the corresponding Message-Based Spreadsheet. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.*

Message-Based Spreadsheet

A message-based spreadsheet displays details of a violation selected from the *Rule-Based Spreadsheet*.

By default, the violation message of *The Ac_sync_group Rules* reports one source per destination for a clock-domain crossing. To view details of all sources per destination, view the message-based spreadsheet. These details appear in the following order in this spreadsheet:

1. Details of a destination (bus-merged destinations, if applicable)
2. Details of all sources (bus-merged sources, if applicable)
3. Details of all user-defined qualifiers (if any exists for the crossing)
4. Details of all detected qualifiers (if any exists for the crossing)
5. Details of all potential qualifiers if the data crossing is unsynchronized

Each of the above details appears in a separate row.

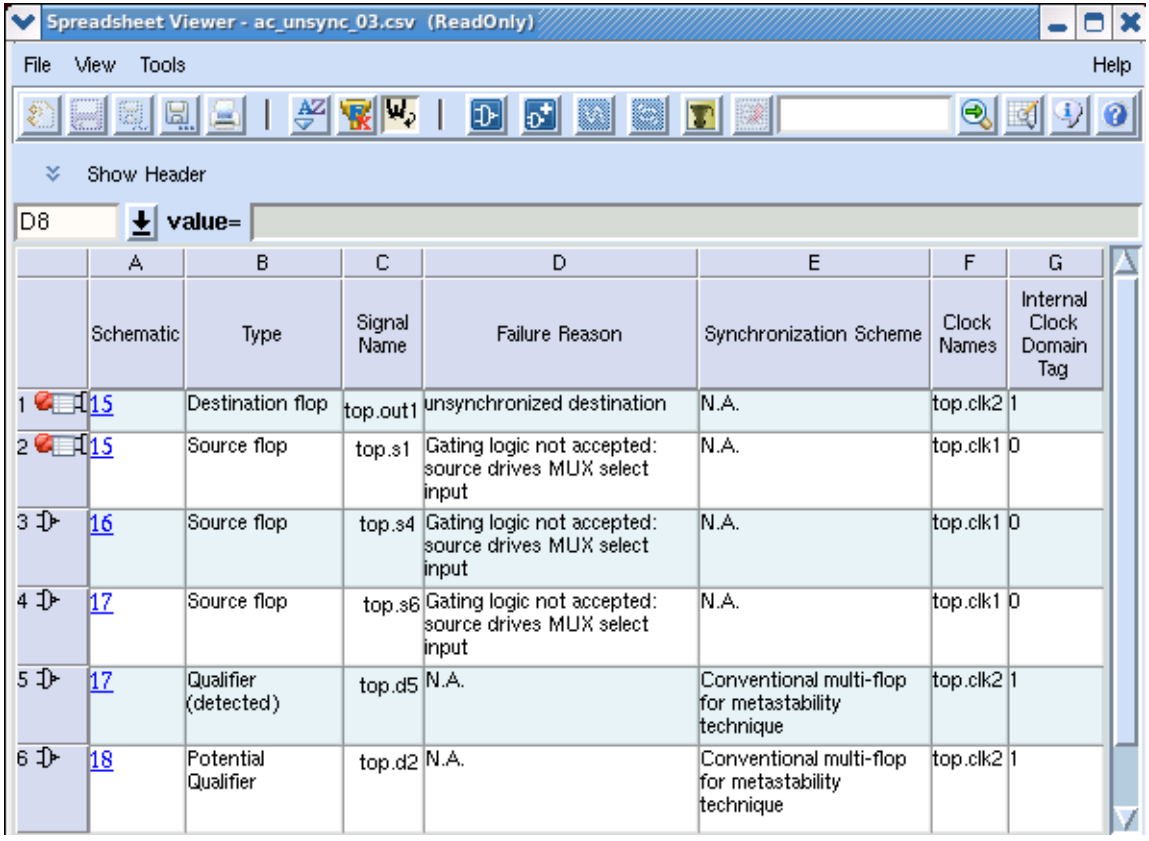
Opening the Message-Based Spreadsheet

To view this spreadsheet, perform any of the following actions:

- Click the link in the *ID* column of the *Rule-Based Spreadsheet* to view details of the violation corresponding to that ID.
- Double-click on the violation message in the *Results* pane.

Sample Message-Based Spreadsheet

The following figure shows the message-based spreadsheet that appears when you click the link *14* from the *ID* column of the rule-based spreadsheet shown in *Figure 6*:



	A	B	C	D	E	F	G
	Schematic	Type	Signal Name	Failure Reason	Synchronization Scheme	Clock Names	Internal Clock Domain Tag
1	15	Destination flop	top.out1	unsynchronized destination	N.A.	top.clk2	1
2	15	Source flop	top.s1	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1	0
3	16	Source flop	top.s4	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1	0
4	17	Source flop	top.s6	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1	0
5	17	Qualifier (detected)	top.d5	N.A.	Conventional multi-flop for metastability technique	top.clk2	1
6	18	Potential Qualifier	top.d2	N.A.	Conventional multi-flop for metastability technique	top.clk2	1

FIGURE 7. Message-Based Spreadsheet

Details of each column of the above spreadsheet are described in the following table:

Column Name	Description
Schematic	Displays a link to the schematic of the violation
Type	Specifies any of the following object types: • Source signal, such as flip-flop, latch, black box, or primary input • Destination signal, such as flip-flop, latch, black box, or primary input • Type of qualifier, such as detected qualifier, user-defined qualifier, or potential qualifier. In case of a vector qualifier, the text `vector` appears before the qualifier type.
Signal Name	Specifies the name of destination signal, source signal, or qualifier net
Failure Reason	Specifies the reason for unsynchronized crossing. This column appears in the spreadsheet of the *Ac_unsync01* and *Ac_unsync02* rules only. For synchronized sources reported by the *Ac_unsync01* and *Ac_unsync02* rules, this column shows N.A.
Synchronization Scheme	Specifies the name of the synchronization method used to synchronize a source crossing. For unsynchronized sources reported by the *Ac_unsync01* and *Ac_unsync02* rules, this column shows N.A.
Qualifier Name	See *Qualifier Name and Qualifier Depth in a Message-Based Spreadsheet*. This column appears only if the *enable_ac_sync_qualdepth* parameter is set to yes.
Qualifier Depth	See *Qualifier Name and Qualifier Depth in a Message-Based Spreadsheet*. This column appears only if the *enable_ac_sync_qualdepth* parameter is set to yes.
Clock Names	Specifies the name of clocks reaching the destination signal, source signal, or qualifier

Column Name	Description
Internal Clock Domain Tag	Specifies a unique tag number generated for a clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.
User defined qualifiers	Specifies the nets defined by the qualifier constraint.

Qualifier Name and Qualifier Depth in a Message-Based Spreadsheet

In the message-based spreadsheet of *The Ac_sync_group Rules*, qualifier name and qualifier depth are reported based on the object type in the *Type* column of this spreadsheet.

This is described in the following table:

Type Column	Qualifier Name Column	Qualifier Depth Column
Destination	-	-
Synchronized source	Name of a qualifier	Sequential depth of qualifier for the synchronized source
Unsynchronized source	-	-
Qualifier/Potential Qualifier	-	Maximum Sequential depth of a qualifier used by any source

Using the Clock Domain Tag

This is an internal tag that is computed while performing clock propagation in a design. It is useful in case multiple clocks are converging and then reaching to a sequential instance. For example, consider the following figure:

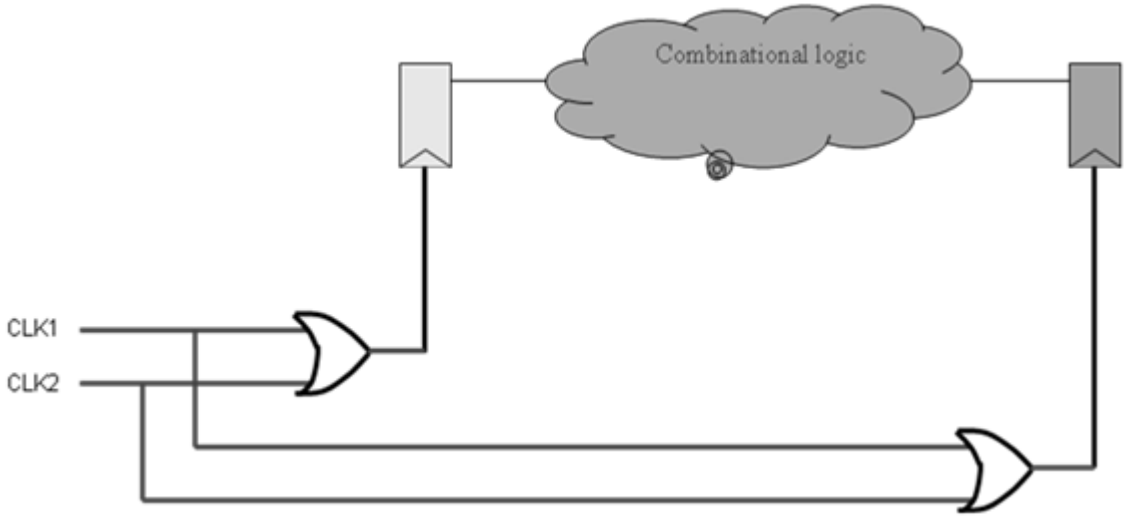


FIGURE 8. Missing Synchronizer

In the above figure, source and destinations are receiving both `CLK1` and `CLK2` but through different paths. Therefore, there is a clock-domain crossing. This would not be clear from the clock names columns in the spreadsheet because both source and destination would show `CLK1` and `CLK2`. In this case, the internal domain tag would be useful. It would be different for both source and destination because clocks are converging on different paths.

Viewing Schematic Through the Spreadsheet

The view the schematic of a crossing, click the link in the *Schematic* column of the spreadsheet and then click  in the spreadsheet tool bar.

Message-Based Spreadsheet for the Enable Condition Based Flow

The message-based spreadsheet generated for *The Enable Expression-Based*

Synchronization Analysis is similar to the *Message-Based Spreadsheet* generated in the normal flow of *The Ac_sync_group Rules*. The only difference is that in the enable condition based flow, the following additional columns are generated:

- *Enable Candidate*

  It shows the enable expression computed at the point where the source is synchronized. This expression appears only when the crossing is synchronized.

- *Enable Count*

  It shows the possible points between source and destination where a crossing could be synchronized. To control the maximum count, use the *sync_point_report_limit* parameter.

The following figure shows the example of the message-based spreadsheet generated in the enable condition based flow:

A	B	C	D	E	F	G	H	I
Schematic	Type	Signal Name	Failure Reason	Synchronization Scheme	Enable Candidate	Enable Count	Clock Names	Internal Clock Domain Tag
1	Destination flop	top.des.q	unsynchronized destination	N.A.	-	-	top.clk2	1
1	Source flop	top.src.q	No enable condition selected	N.A.	-	2	top.clk1	0
1	Qualifier (detected)	top.en1.des.q	N.A.	Conventional multi-flop for metastability technique	-	-	top.clk2	1
2	Potential Qualifier	top.en2.des.q	N.A.	Conventional multi-flop for metastability technique	-	-	top.clk2	1

FIGURE 9.

In the above spreadsheet, when you select 2 in the *Enable Count* column,

the *Spreadsheet Showing Enable Expressions* appears.

Spreadsheet Showing Enable Expressions

The following figure shows this spreadsheet that appears when you click the number in the `Enable Count` column of the *Message-Based Spreadsheet for the Enable Condition Based Flow*:

A	B	C	D
Schematic ID	Enable Analysis Point	Enable Condition	Type
1	top.rtlc_l3.in1	(top.en1_out)	Qualifier
2	top.rtlc_l5.in1	(top.en2_out)	Qualifier

FIGURE 10.

The above spreadsheet shows the enable conditions and the terminals on which these conditions are computed.

The Enable Expression-Based Synchronization Analysis

Data synchronization analysis by SpyGlass requires an enable expression that blocks the transfer of source data when the source data is changing.

To enable SpyGlass compute such expressions, set the *sync_check_type* parameter to `enable_with_qual` or `enable_with_des_dom`.

You can also generate SVA for the computed enable expressions that you can validate in other simulator tools. For details, see *Generating SVA for Enable Expressions*.

The following figure shows the flow of enable expression-based synchronization analysis:

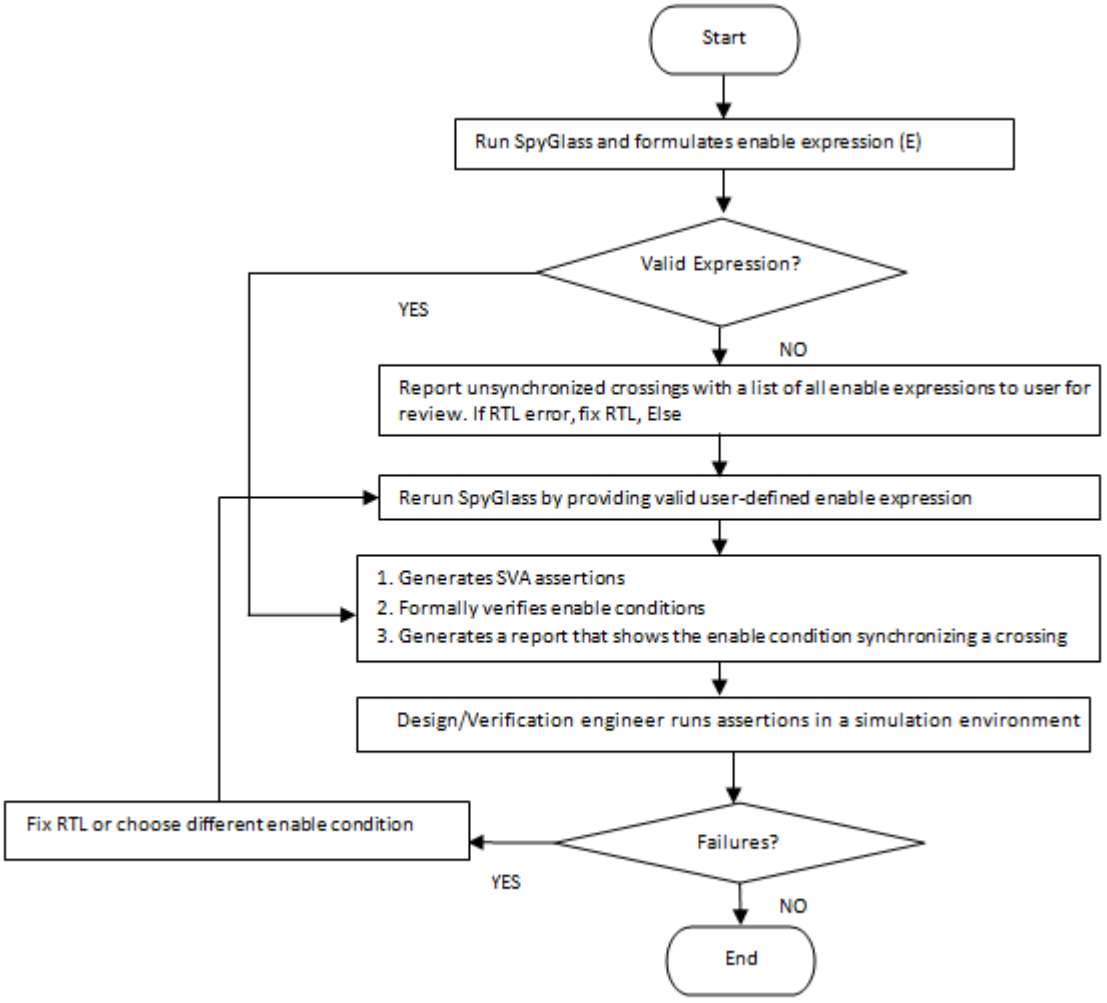


FIGURE 11.

The following figures show different enable expressions (E) computed for the AND gates and the mux:

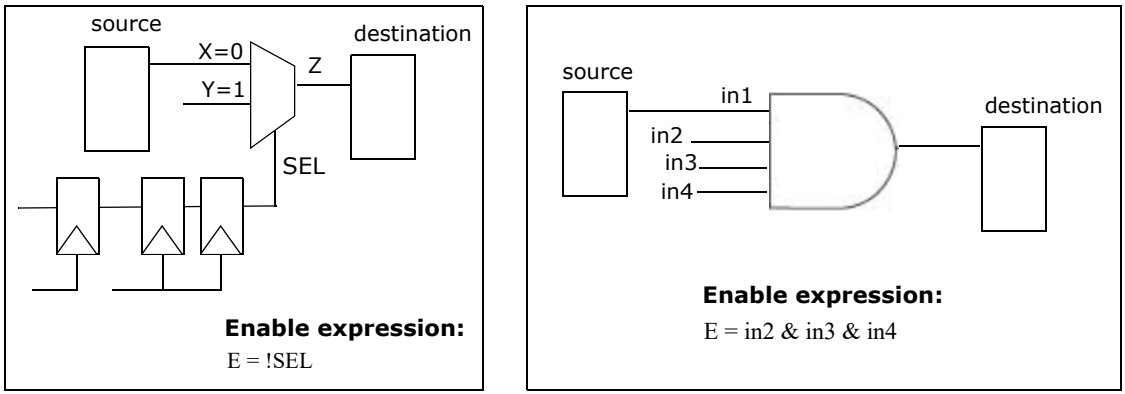


FIGURE 12.

For details on this feature, see the following topics:

- *Synchronization Requirements to Compute Enable Expressions*
- *Generating SVA for Enable Expressions*
- *Spreadsheet Generated for Enable Expression-Based Synchronization Analysis*

Synchronization Requirements to Compute Enable Expressions

Following are the requirements for the enable expression-based synchronization analysis:

- *Domain requirement*
- *Synchronizer/qualifier requirement*
- *Depth requirement*
- *Enable selection*

The following figure captures the above requirements:

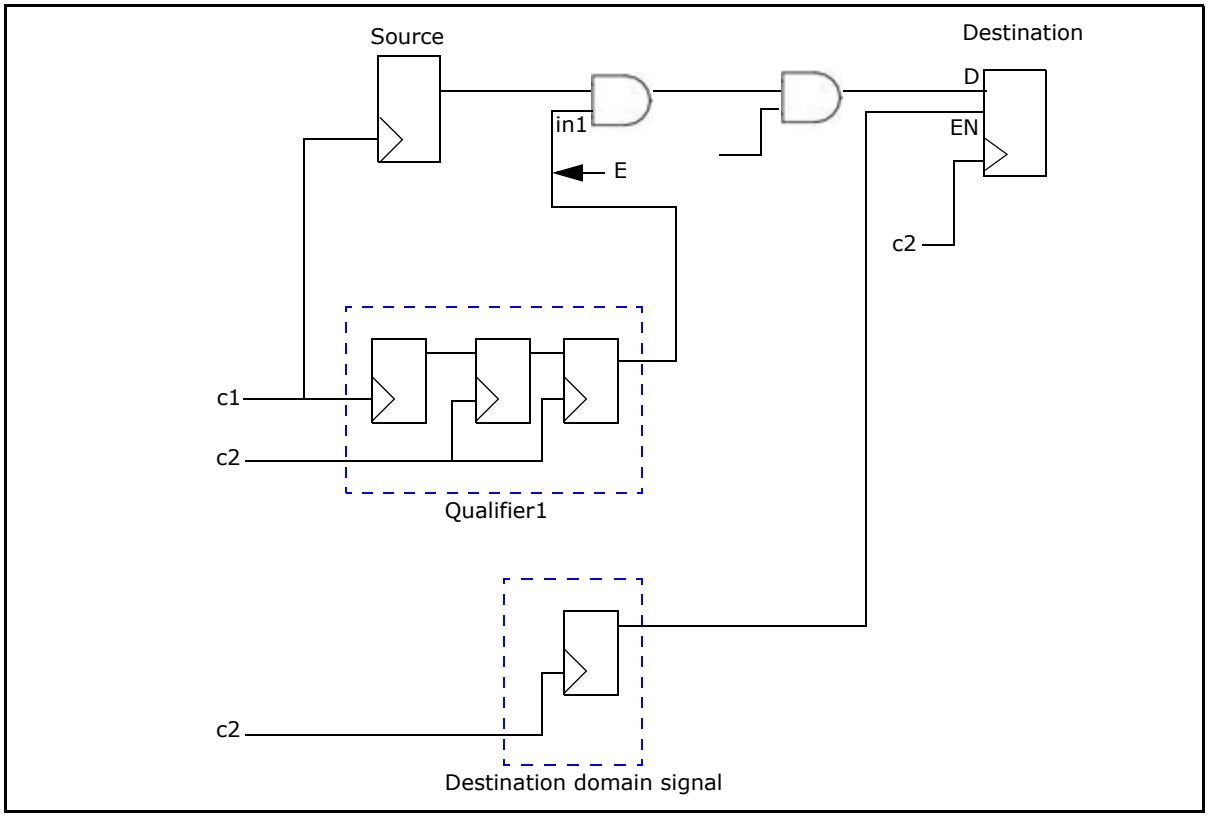


FIGURE 13. Synchronization Requirements to Compute Enable Expressions

Domain requirement

The enable signal (`E`) should be in the destination domain. That is, all the sequential elements driving `E` should be in the destination domain, as shown in *Figure 13*.

Synchronizer/qualifier requirement

A multi-flop synchronizer (`Qualifier1` in *Figure 13*) must be present in the transitive fan-in cone of the crossing to indicate that the signal for

transferring a source or destination is from the source domain and getting synchronized to the destination.

You can relax this approach by replacing the qualifier requirement with a destination domain signal (destination domain signal in *Figure 13*) in the fan-in. Enable this approach by setting the *sync_check_type* parameter to `enable_with_des_dom`.

Depth requirement

At least one path from the enable signal (`E`) to destination (`D`) does not have any sequential element, as shown in *Figure 13*.

Enable selection

You can set a valid synchronization point by specifying the gate with a valid enable.

Set the *sync_point_selection* parameter to `first`, `last`, or `none` to set the synchronization point. If you specify the value as `none`, set the limit by using the *sync_point_report_limit* parameter.

If you do not set any limit, SpyGlass reports the first five enables starting from destination, and reports the enable conditions of these enables in the *Spreadsheet Showing Enable Expressions*.

Generating SVA for Enable Expressions

SpyGlass generates SVA for the crossings having some valid expressions in the crossing path. It also generates SVA for the user-specified expressions in the *qualifier* constraint.

To enable SVA generation, set the *sync_point_selection* parameter to `none`.

Example of Generated SVA for Enable Expressions

Consider the following files specified for SpyGlass analysis:

//top.v

```
module top(input in1, in2, clk1, clk2, output out);
FLOP src1(in1, clk1, src1_out);
FLOP src2(in2, clk1, src2_out);
FLOP sync(in1, clk2, en_out);

wire w1;
assign w1 = en_out ? src1_out : src2_out;
FLOP des(w1, clk2, out);
endmodule

module FLOP(input d, clk, output reg q);
always@(posedge clk)
    q <= d;
endmodule
```

//top.sgdc

```
current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d2
```

Based on the above files, SpyGlass generates SVA for the *VCS* simulator:

VCS

```
module Assertion_mod();
wire spyglass_assert;
`include "spyglass_dynamic_setup.v"
`include "spyglass_expr.v"

DATAHOLD_Check_mod  #("SpyGlass - DataHold failure for user
specified qualifier enable (top.v:13) - Source:top.src1.q,
Destination:top.des.q") dataholdcheckinst1
(.data((top.en_out)), .enable(top.src1_out),
.src_clk(top.clk1), .des_clk(top.clk2), .rst(1'b0) );


DATAHOLD_Check_mod  #("SpyGlass - DataHold failure for user
specified qualifier enable (top.v:13) - Source:top.src2.q,
Destination:top.des.q") dataholdcheckinst2
(.data((!(top.en_out))), .enable(top.src2_out),
.src_clk(top.clk1), .des_clk(top.clk2), .rst(1'b0) );
```

Spreadsheet Generated for Enable Expression-Based Synchronization Analysis

The following spreadsheets are generated for the enable expression-based synchronization analysis:

- *Message-Based Spreadsheet for the Enable Condition Based Flow*
- *Spreadsheet Showing Enable Expressions*

Grouping Messages of the Ac_sync_group Rules

Many violations are related in such a way that by analyzing and fixing one or a few violations, a large number of other related violations may get automatically fixed. You can group such violations so that they appear under a single group in the GUI and reports.

You can group or merge messages by using any of the following approaches:

- *Instance-Based Grouping*
- *User-Specified String-Based Grouping*
- *Netlist Bus Merging*

Instance-Based Grouping

In this type of grouping, messages are grouped based on instance names of source and destination signals.

Instance-based grouping occurs if the following conditions hold true:

- Destination signals of different crossings are mapped to the same net of a module.
- Each source in different crossings is mapped to the same net of a module.

Consider an example of the following two types of crossings:

Crossing 1:	***Crossing 2:***
Destination: top.m1.n1.p1.net1	Destination: top.m1.n1.p2.net1
Source: top.m1.n1.SRC1	Source: top.n2.SRC1

Now consider that `m1` is an instance of the `M` module, `n1` and `n2` are instances of the `N` module, and `p1` and `p2` are the instances of the `P` module.

In this case, the same destination net, `net1`, and same source, `SRC1`, are reported through two leaf-level instances of the same module. Therefore, grouping will happen in this case.

To disable this type of grouping, set the *disable_inst_grouping* parameter to `yes`.

The following figure shows an example of an instance-based grouping:

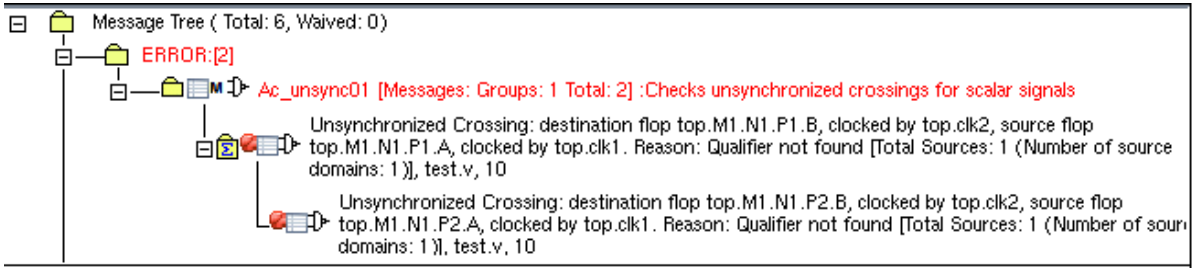


FIGURE 14. Instance-Based Grouping

NOTE: *In addition, messages are grouped for collection objects, such as VHDL record type. For example, violations reported on all the members of a VHDL record are grouped together.*

User-Specified String-Based Grouping

In this case, messages in which source or destination signals match the string specified by the *user_group_str* parameter are grouped together.

If there are multiple sources, all sources should match the string before they are considered for grouping.

The following figure shows a grouping when the `user_group_str` parameter is set to `cfg`:

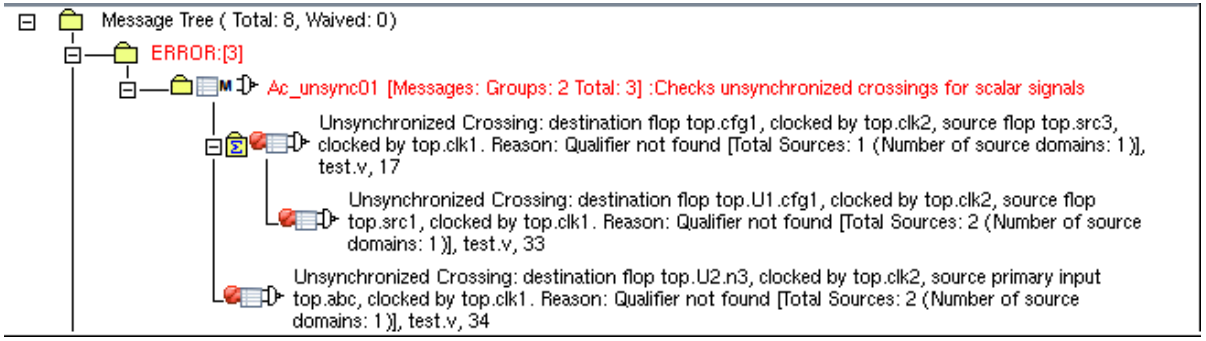


FIGURE 15. User-Specified String-Based Grouping

Note the following points about this type of grouping:

- It is given preference over instance-based grouping.
- It is case-sensitive.

Viewing Grouped Messages in a Spreadsheet

Grouped messages in a spreadsheet appear as shown in the following figure:

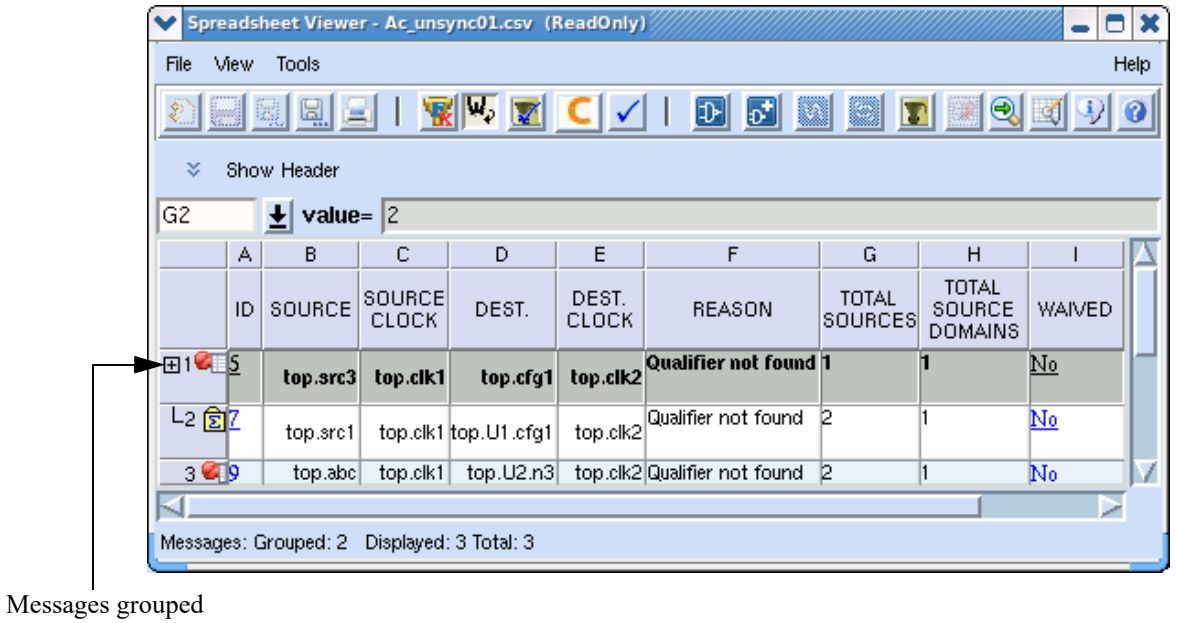


FIGURE 16. Grouped Messages in a Spreadsheet

In the above figure, click the + sign to view all the grouped messages.

Netlist Bus Merging

In this case, destination signal names matching the same suffix specified by the *netlist_name_convention* parameter are merged together and reported as a single message.

In addition, source names matching the same suffix are also merged in the spreadsheet viewer.

For example, consider that the value of the `netlist_name_convention` parameter is set to `x`. In this case, destination signals `top.u1.sig_1x`, `top.u1.sig_2x`, and `top.u1.sig_3x` are merged together and only one message is displayed that shows the name as `top.u1.sig_[1:3]x`.

The following figure shows this type of merging:

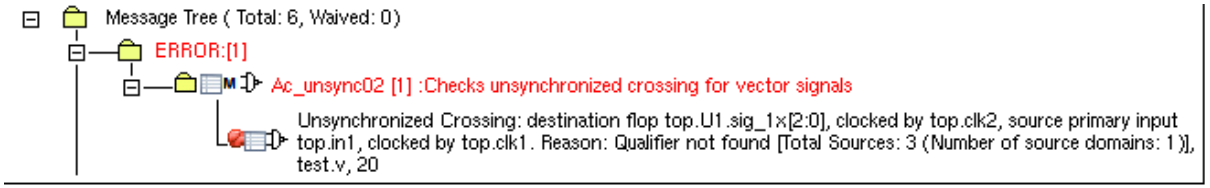


FIGURE 17. Netlist Bus Merging

If you set the *netlist_name_convention* parameter to `" "`, the following signals are merged together:

`top.u1.sig_1`, `top.u2.sig_2`, and `top.u3.sig_3`

As a result of netlist bus merging, the *Ac_sync01* rule violations are reported under the *Ac_sync02* rule violations. Similarly, the *Ac_unsync01* rule violations are reported under the *Ac_unsync02* rule violations.

Handling of Hanging Nets From Combinational Logic by the Ac_sync_group Rules

Consider the scenario shown in the following figure:

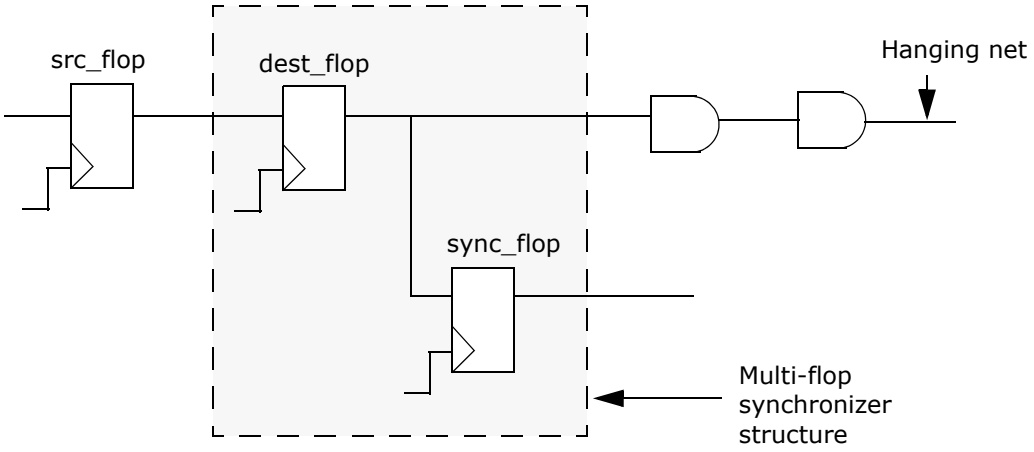


FIGURE 18. Hanging net after combinational logic

In the above scenario, by default, *The Ac_sync_group Rules* do not ignore the hanging net coming from a combinational logic and do not report the multi-flop structure as synchronized.

When you set the *cdc_reduce_pessimism* parameter to `skip_unused_paths`, *The Ac_sync_group Rules* ignore the hanging net and report the multi-flop structure as synchronized with the following synchronization reason:

```
Destination instance is driving multiple paths
```

Reasons for Synchronized Crossings Reported by Ac_sync_group Rules

A source is considered as synchronized if it is synchronized through any of the synchronization schemes mentioned in the following table:

Synchronization Scheme	Synchronization Method Reported in the Method Column of the Rule-Based Spreadsheet
Conventional Multi-Flop Synchronization Scheme	**Methods reported:** • `Conventional multi-flop for metastability technique` • `Conventional multi-flop (library-cell) for metastability technique` • `Conventional multi-flop synchronizer is hanging` For details, see *Conventional multi-flop Method*.
Synchronizing Cell Synchronization Scheme	**Method reported:** `Synchronizing cell (cell name : '<cellname>')` For details, see *Synchronizing Cell Method*.
Synchronized Abstract Port Synchronization Scheme	**Method reported:** `Synchronized Abstract Port` For details, see *Synchronized Abstract Port Method*.
Qualifier Synchronization Scheme Using qualifier -crossing	**Method reported:** `qualifier '<net-name>' defined on destination` For details, see *Qualifier Defined on Destination Method*.
Synchronized Enable Synchronization Scheme	**Methods reported:** • `Enable Based Synchronizer` • `Enable Based User-Defined Qualifier` For details, see *Enable Based Method*.
Clock-Gating Cell Synchronization Scheme	**Methods reported:** • `Clock Gate Synchronization (library clock-gating cell)` • `Clock Gate Synchronization (auto-detected clock-gating)` • `Clock Gate Synchronization (user-defined clock-gating cell)` • `Clock Based User-defined Qualifier` For details, see *Clock Gate Synchronization Method*.

Synchronization Scheme	Synchronization Method Reported in the Method Column of the Rule-Based Spreadsheet
Recirculation MUX Synchronization Scheme	**Methods reported:** • `Recirculation flop` • `Recirculation flop (user-defined qualifier)` For details, see *Recirculation Flop Method*.
MUX-Select Sync (Without Recirculation) Synchronization Scheme	**Methods reported:** • `Mux-select sync` • `Mux-select sync (user-defined qualifier)` For details, see *Mux-Select Sync Method*.
AND Gate Synchronization Scheme	**Methods reported:** • `Synchronization at AND gate` • `Synchronization at And gate (user-defined qualifier)` For details, see *Synchronization at AND Gate Method*.
Glitch Protection Cell Synchronization Scheme	**Methods reported:** • `Synchronization at Glitch Protection Cell` • `Synchronization at Glitch Protection Cell (user-defined qualifier)` For details, see *Synchronization at Glitch Protection Cell Method*.
Synchronizer present in data path of a destination	**Methods reported:** • `Merges with valid inferred qualifier` • `Merges with valid user-defined qualifier` For details, see *Merging with a Valid Inferred Qualifier Method*.
Delay Signals Synchronization Scheme	**Method reported:** `does not require synchronization (long-delay/quasi-static)` For details, see *No Synchronization (long-delay/quasi-static) Method*.
Constant Source Flop Synchronization Scheme	**Method reported:** `Source is constant` For details, see *Constant Source Method*.
-	**Method reported:** `User-defined enable expression` For details, see *User-Defined Enable Expression Method*.
-	**Method reported:** `Valid enable condition found` For details, see *Finding Valid Enable Condition Method*.

Conventional multi-flop Method

The following methods appear in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules when all the sources reaching to the destination belong to the same domain and are synchronized by *Conventional Multi-Flop Synchronization Scheme*:

```
Conventional multi-flop for metastability technique
```

The following method appears if there is a multi-flop chain inside a library cell:

```
Conventional multi-flop (library-cell) for
metastability technique
```

However, if the output of the synchronizer is hanging, the following method is reported:

```
Conventional multiflop synchronizer is hanging
```

Synchronizing Cell Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules when all the sources reaching to the destination belong to the same domain and the sources are synchronized by *Synchronizing Cell Synchronization Scheme*:

```
Synchronizing cell (cell name : '<cellname>')
```

The destination in this case is an instance of a synchronizing cell specified by using the *synchronize_cells* parameter, the *synchronize_data_cells* parameter, or the *sync_cell* constraint.

Synchronized Abstract Port Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules when a source, synchronized by the *Synchronized Abstract Port Synchronization Scheme*, is coming from a black box or a port and it is defined as synchronized by using the *abstract_port* `-sync active` constraint.

```
Synchronized Abstract Port
```

The following figure shows the scenario in which this method is reported:

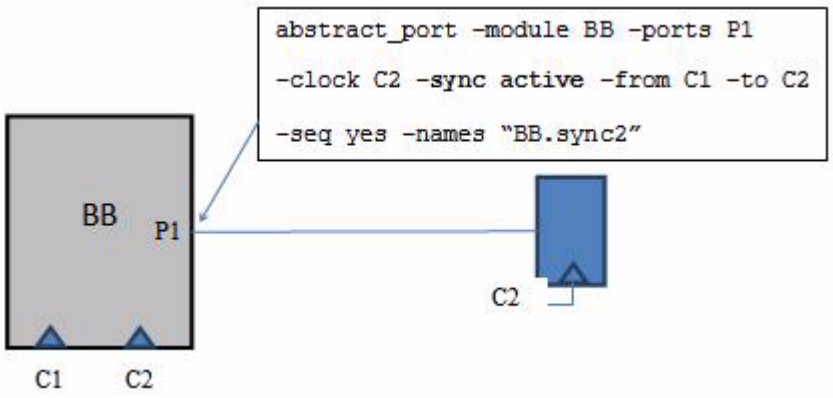


FIGURE 19. Synchronized Abstract Port Synchronization Scheme

Qualifier Defined on Destination Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if a qualifier crossing is defined by the *qualifier* constraint with the `-crossing` argument (*Qualifier Synchronization Scheme Using qualifier -crossing*):

```
qualifier '<net-name>' defined on destination
```

Enable Based Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules when all sources, synchronized by the *Synchronized Enable Synchronization Scheme*, reach the destination at the data pin, and the qualifier resides at the enable pin path of the destination instance:

- `Enable Based Synchronizer`
- `Enable Based User-Defined Qualifier`

In this case, all the sources must belong to the same clock domain.

NOTE: *By default, the Synchronized Enable Synchronization Scheme is enabled. Set the enable_sync parameter to* `no` *to disable this scheme.*

Clock Gate Synchronization Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules when all sources, synchronized by the *Clock-Gating Cell Synchronization Scheme*, reach the destination at the data pin or enable pins, and the qualifier resides at the enable path of a clock-gating cell:

- `Clock Gate Synchronization (library clock-gating cell)`
- `Clock Gate Synchronization (auto-detected clock-gating)`
- `Clock Gate Synchronization (user-defined clock-gating cell)`
- `Clock Based User-defined Qualifier`

In such cases, all the sources must belong to the same clock domain.

NOTE: *By default, the Clock-Gating Cell Synchronization Scheme is enabled. Set the enable_clock_gate_sync parameter to* `no` *to disable this scheme.*

Recirculation Flop Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if a qualifier resides at the mux-select path and converges with the source at the MUX before reaching the destination:

- `Recirculation flop`
- `Recirculation flop (user-defined qualifier)`

In this case, a clock domain crossing occurs through one of the MUX input pins, and the other MUX input pin is driven by the same destination flip-flop output. This is known as the *Recirculation MUX Synchronization Scheme*.

NOTE: *By default, this scheme is enabled. Set the enable_mux_sync parameter to* `none` *to disable this scheme.*

Mux-Select Sync Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if a qualifier resides at the MUX-select path and converges with the source at the MUX before reaching the destination:

- `Mux-select sync`
- `Mux-select sync (user-defined qualifier)`

In this case, the MUX is without recirculation. Clock domain crossing occurs through one of the MUX input pins, and the other MUX input pin is driven by a signal in the same destination domain or a constant signal. This is known as the *MUX-Select Sync (Without Recirculation) Synchronization Scheme*.

NOTE: *By default, this scheme is disabled. Set the enable_mux_sync parameter to* `mux_select` *or* `all` *to enable this scheme. Alternatively, set the ac_sync_mode parameter to* `soft_gate`.

Synchronization at AND Gate Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules when a qualifier, synchronized by the *AND Gate Synchronization Scheme*, converges with the source at an AND/NAND gate before reaching to the destination:

- `Synchronization at AND gate`
- `Synchronization at And gate (user-defined qualifier)`

NOTE: *By default, this scheme is disabled. Set the enable_and_sync parameter to* `yes` *to enable this scheme. Alternatively, set the ac_sync_mode parameter to* `soft_gate`.

Synchronization at Glitch Protection Cell Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if a qualifier, synchronized by the *Glitch Protection Cell Synchronization Scheme*, converges with a source at a glitch gating cell before reaching to the destination:

- `Synchronization at Glitch Protection Cell`

- `Synchronization at Glitch Protection Cell (user-defined qualifier)`

You can specify names of glitch protection cells for this scheme by using the *glitch_protect_cell* parameter.

Merging with a Valid Inferred Qualifier Method

One of the following methods appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if a qualifier, `Synchronizer present in data path of a destination`, converges with the source at a valid gate before reaching the destination:

- `Merges with valid inferred qualifier`
- `Merges with valid user-defined qualifier`

A gate is considered as valid if any of the following conditions hold true:

- The gate is an OR/NOR gate, and the *enable_and_sync* parameter to yes.
- The gate is any combinational gate, and the *ac_sync_mode* parameter is set to `soft_gate`.

The following figure shows the scenario in which this method is reported:

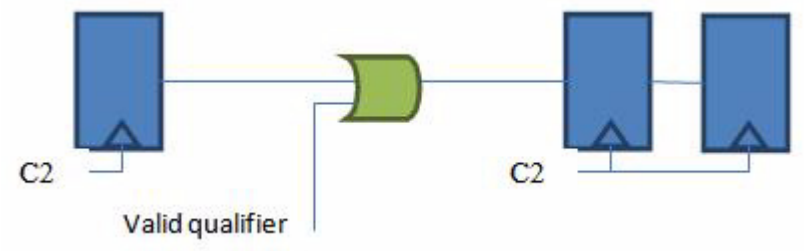


FIGURE 20. Synchronizer present in data path of a destination

No Synchronization (long-delay/quasi-static) Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if the path from a source to destination has a net on which you have specified the *quasi_static*

constraint:

```
does not require synchronization (long-delay/quasi-static)
```

In this case, the crossing is synchronized by the *Delay Signals Synchronization Scheme*.

Constant Source Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules if the data pin of a sequential element (source) in a crossing is tied to a constant value:

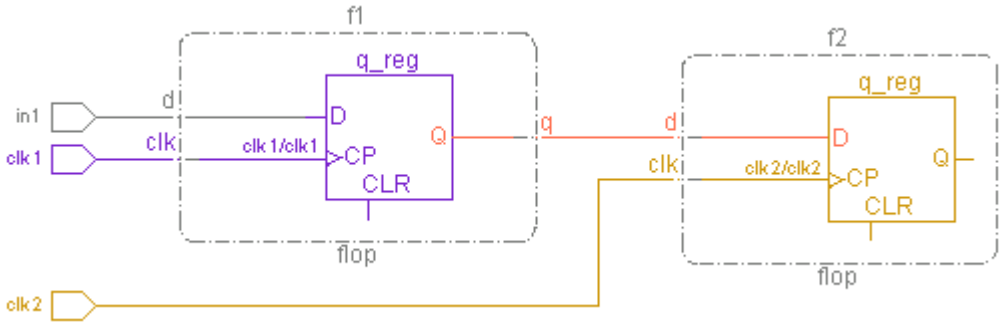
```
Source is constant
```

SpyGlass CDC analysis for such sequential elements is performed only when the *cdc_reduce_pessimism* parameter is set to *const_source*. This scheme is called the *Constant Source Flop Synchronization Scheme*.

Constant Source Flop Synchronization Scheme

In this scheme, SpyGlass CDC analysis for sequential elements (whose data pin is tied to a constant) is performed only when the *cdc_reduce_pessimism* parameter is set to *const_source*.

Consider the following scenario:




```
// SGDC File
   set_case_analysis -name in1 -value 0

// Project File
   set parameter cdc_reduce_pessimism const_source
```

FIGURE 21.

In the above scenario, the data pin of the `f1` flip-flop is tied to a constant. Therefore, SpyGlass reports the method "`Source is constant`" in the violation message and spreadsheet.

User-Defined Enable Expression Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules during *The Enable Expression-Based Synchronization Analysis* when you specify a qualifier by using the qualifier `-enable` *<enable-expr>* constraint:

`User-defined enable expression`

For example, this method appears in the following scenario where the `qual` net is specified as the qualifier:

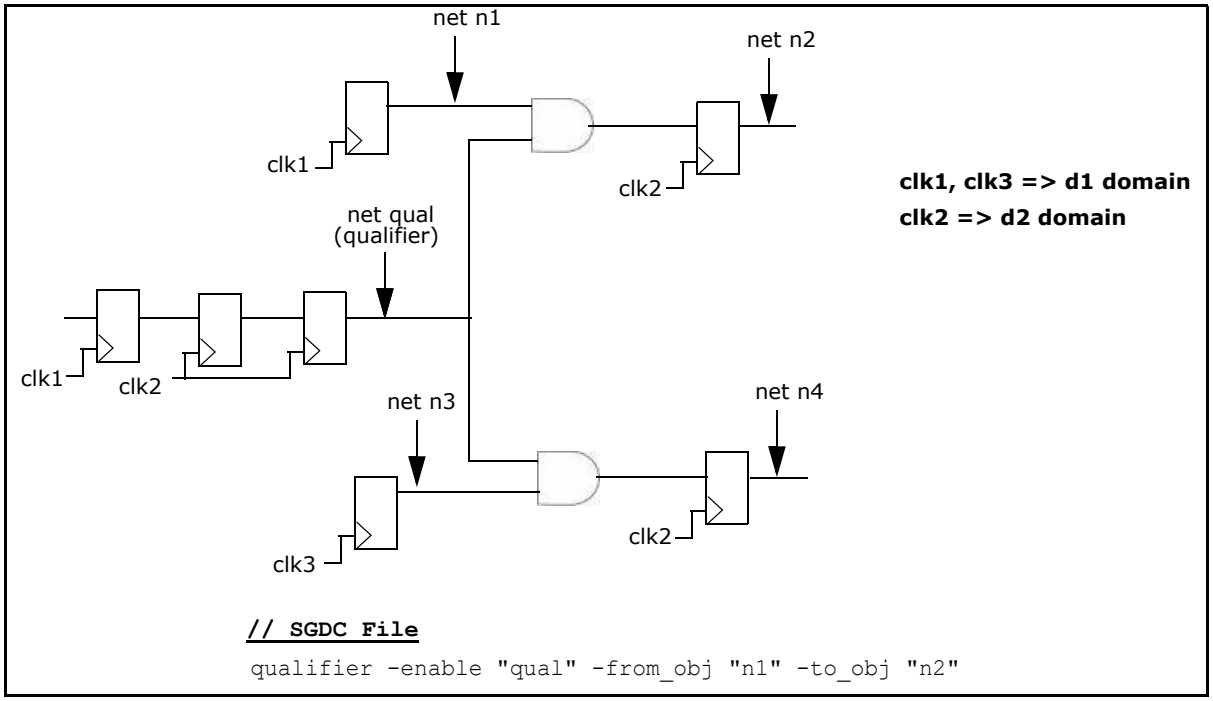


FIGURE 22.

Finding Valid Enable Condition Method

The following method appears in the *METHOD* column of the *Rule-Based Spreadsheet* of the *Ac_sync01* and *Ac_sync02* rules during *The Enable Expression-Based Synchronization Analysis* when a crossing is synchronized and a valid enable condition is found for that crossing:

```
Valid enable condition found
```

Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules

The following are the reasons (starting from highest priority) for unsynchronized crossings reported by the *Ac_unsync01* and *Ac_unsync02* rules:

- *Reason - Sources from different domains converge before being synchronized*
- *Reason - Qualifier not found*
- *Reason - Conventional multi-flop synchronizer disallowed*
- *Reason - Clock phase difference between destination instance and synchronizer flop*
- *Reason - Clock domains of destination instance and synchronizer flop do not match*
- *Reason - Synchronizer flop is the destination flop for another crossing*
- *Reason - Number of inverters/buffers between sync flops exceeds limit*
- *Reason - Sync reset used in multi-flop synchronizer*
- *Reason - Destination instance is driving multiple paths*
- *Reason - Combinational logic used between crossing*
- *Reason - Specify 'synchronize_data_cells', not 'synchronize_cells' for bus signals*
- *Reason - Specify 'synchronize_cells', not 'synchronize_data_cells' for single bit signals*
- *Reason - Invalid RTL flop/cell used in synchronizer chain*
- *Reason - Invalid synchronizer module/cell <name>*
- *Reason - Unsynchronized synchronous reset*
- *Reason - [User-defined qualifier/ Qualifier] merges with another source before gating logic*
- *Reason - [User-defined qualifier/Qualifier] merges with the same source before gating logic*
- *Reason - Gating logic not accepted*
- *Reason - Qualifier not accepted: crossing source is the same as source of qualifier*
- *Reason - Combinational loops on crossing*

- *Reason - No Enable Condition Selected*
- *Reason - Enable Criteria not satisfied: No destination domain*
- *Reason - Enable Criteria not satisfied: No Qualifier found*
- *Reason - Enable Criteria not satisfied: gating-type not accepted*
- *Reason - Enable Criteria not satisfied: Source reach mux select*
- *Reason - Domain Criteria not satisfied: No domain*
- *Reason - Domain Criteria not satisfied: Source domain*

Reason - Sources from different domains converge before being synchronized

This reason is reported when sources from multiple domains reach the destination without being synchronized by valid qualifiers.

Sometimes this problem could be an attempt to qualify with an incorrectly synchronized qualifier (or a qualifier that is correctly synchronized but is not recognized by SpyGlass CDC - in which case you should use the *qualifier* constraint to mark that signal).

The following figure shows the scenario in which this method is reported:

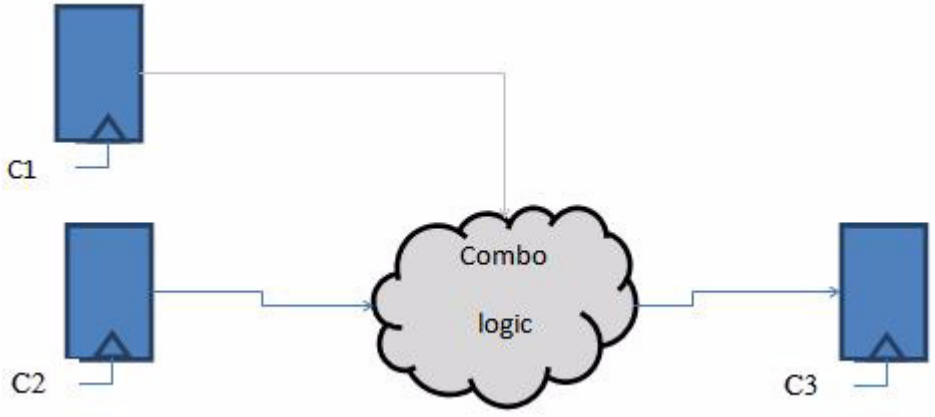


FIGURE 23. Sources from different domains converge before being synchronized

The crossing may have a conventional multi-flop structure, synchronizing cell structure (specified by the *synchronize_cells* parameter, the *synchronize_data_cells* parameter, or the *sync_cell* constraint), handshake, FIFO structures, or any other synchronization structure.

Reason - Qualifier not found

This reason is reported in any of the following cases:

- When SpyGlass is not able to find any synchronizing structure at the destination of a crossing, there is a possibility that a qualifier should have been present but either the designer forgot to add it or connect it properly, or SpyGlass is not able to find that qualifier.

  If you know that the crossing is controlled by a valid qualifier, mark the qualifying signal by using the *qualifier* constraint.

  The following figure shows the scenario in which this method is reported:

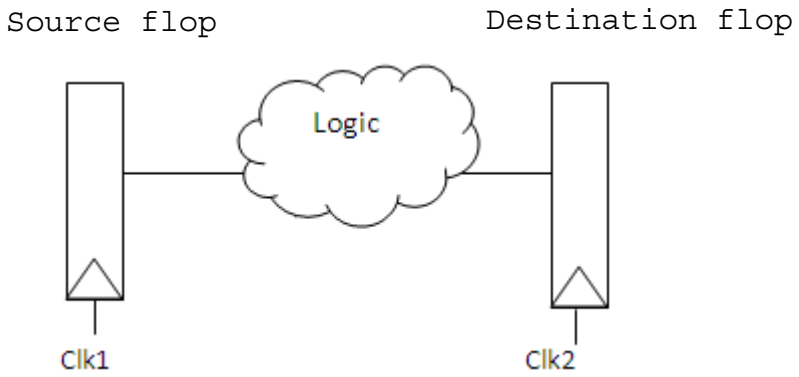


FIGURE 24. Qualifier not found

  In the above scenario, no synchronous pattern is found for the crossing and SpyGlass can also not find any possibility of a control line.

- When a source converges with an invalid qualifier at some gate.
- When a source converges with an unsynchronized qualifier crossing defined by the *qualifier* `-crossing` constraint.

Reason - Conventional multi-flop synchronizer disallowed

This reason appears when the *Conventional Multi-Flop Synchronization Scheme* is disabled by setting the *enable_multiflop_sync* parameter to `no`. In this case, all the crossings that were reported as synchronized by using the *Conventional Multi-Flop Synchronization Scheme* (unless those crossings are synchronized further using any other synchronization scheme) are considered as unsynchronized.

This scheme considers all those clock crossings where the destination domain flip-flop is followed by a chain of flip-flops in the same domain.

Reason - Clock phase difference between destination instance and synchronizer flop

This reason is reported when the destination is synchronized by *Conventional Multi-Flop Synchronization Scheme*, but there is a clock phase difference between the destination instance and the synchronized flip-flop. Such cases are checked only when the *allow_half_sync* parameter is set to `no`.

The following figure illustrates the scenario in which SpyGlass reports this reason:

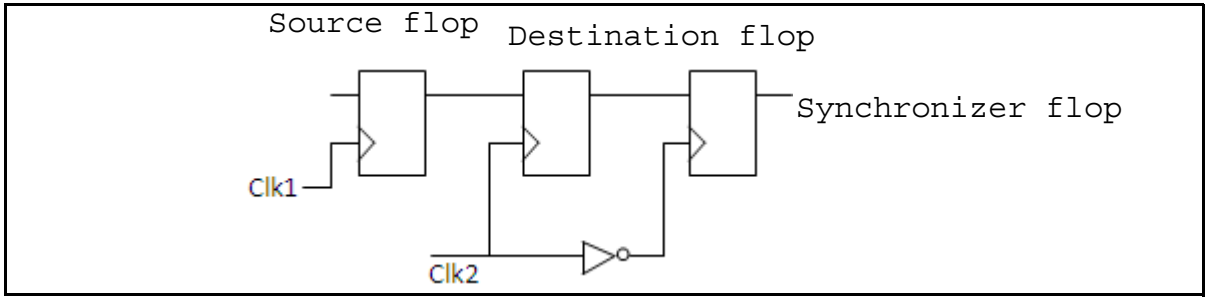


FIGURE 25. Clock phase difference between destination instance and synchronizer flip-flop

Reason - Clock domains of destination instance and synchronizer flop do not match

This reason is reported when the destination is synchronized by *Conventional Multi-Flop Synchronization Scheme*, but clock domains of the destination instance and the synchronizer flip-flop do not match.

The following figure illustrates the scenario in which SpyGlass reports this reason:

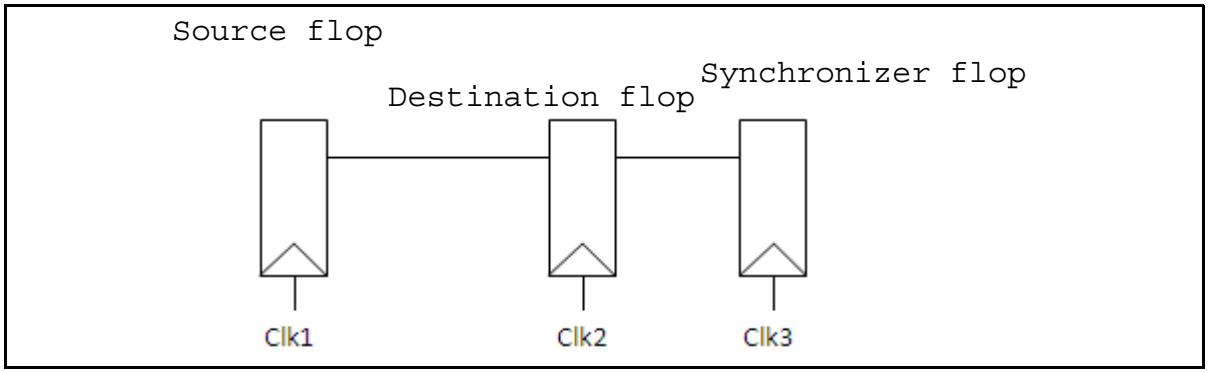


FIGURE 26. Clock domains of destination instance and synchronizer flip-flop do not match

Reason - Synchronizer flop is the destination flop for another crossing

This reason is reported when the destination is synchronized by *Conventional Multi-Flop Synchronization Scheme*, but the synchronizer flip-flop is also the destination flip-flop for another crossing.

Such cases occur when another source is feeding the synchronizer flip-flop through an enable line or through a logic at the data input.

The following figure illustrates the scenario in which SpyGlass reports this reason:

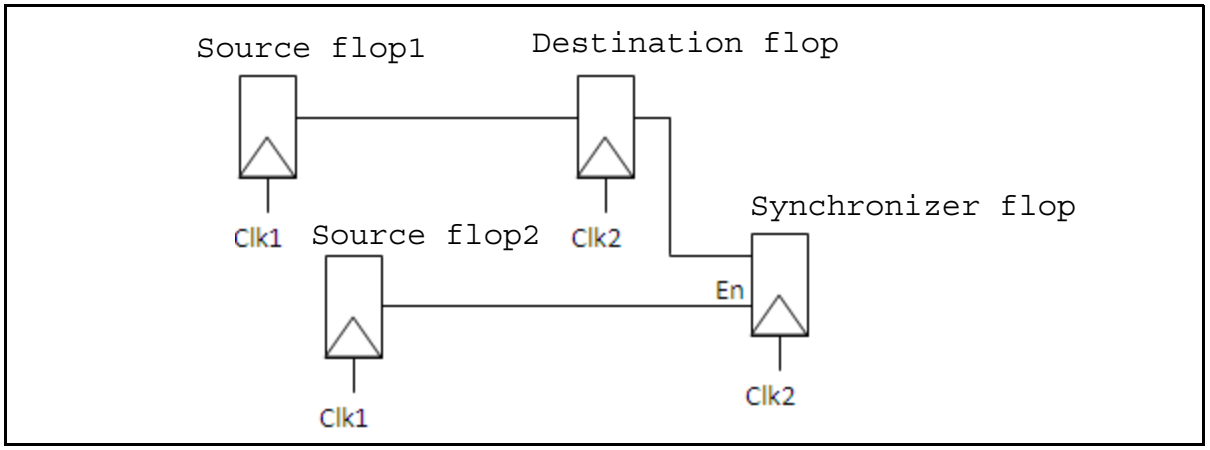


FIGURE 27. Synchronizer flip-flop is the destination flip-flop for another crossing

In the above figure, `Synchronizer flop` is the destination flip-flop for `Source flop2`.

Reason - Number of inverters/buffers between sync flops exceeds limit

This reason is reported when the destination is synchronized by *Conventional Multi-Flop Synchronization Scheme*, but the number of buffers/inverters between synchronous flip-flops exceeds the limit specified by the *ignore_num_rtl_buf_invs* parameter.

For example, in the scenario shown in the following figure, if the *ignore_num_rtl_buf_invs* parameter is set to `1`, a violation will be reported:

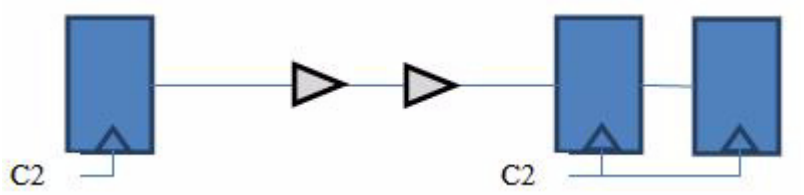


FIGURE 28. Number of inverters/buffers between sync flip-flops exceeds limit

Reason - Sync reset used in multi-flop synchronizer

This reason is reported when the destination is synchronized by *Conventional Multi-Flop Synchronization Scheme* but a synchronous reset is present between synchronizer flip-flops, and either the *reset* `-sync` constraint is not specified or the *sync_reset* parameter is not set to `yes`.

In such cases, a single AND/OR gate is present between the synchronizing flip-flops, and they may be present because of a synchronous reset that is used but not declared.

The following figure illustrates the scenario in which SpyGlass reports this reason:

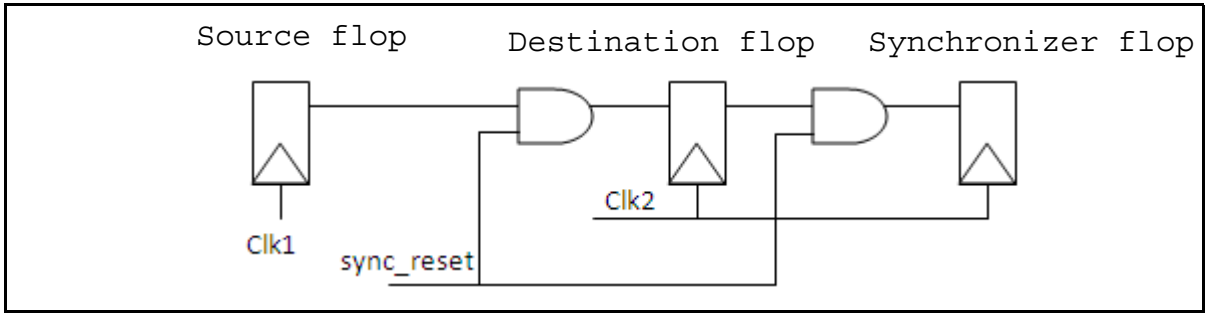


FIGURE 29. Sync reset used in multi-flop synchronizer

Reason - Destination instance is driving multiple paths

This reason is reported when multiple fan-outs are present in the

destination instance output, and each of these paths may or may not have a synchronization structure.

The following figure illustrates a scenario in which SpyGlass reports this reason:

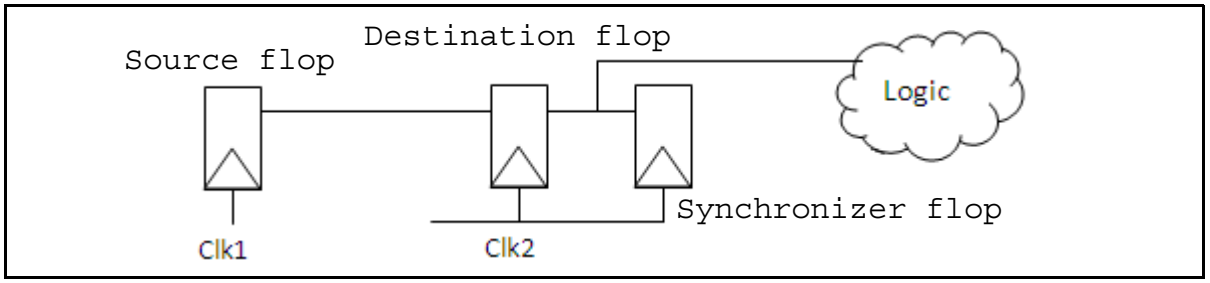


FIGURE 30. Destination instance is driving multiple paths

In the above figure, any logic using the output of the first flip-flop will be using metastable value, thereby causing crossing as unsynchronized.

Reason - Combinational logic used between crossing

This reason is reported when the destination is synchronized by *Conventional Multi-Flop Synchronization Scheme*, *Synchronizing Cell Synchronization Scheme*, or defined by the *qualifier* `-crossing` constraint, but there is a combinational logic present between the source and destination and the *allow_combo_logic* constraint or the *allow_combo_logic* parameter is not specified for this crossing.

The following figure illustrates the scenario in which SpyGlass reports this reason:

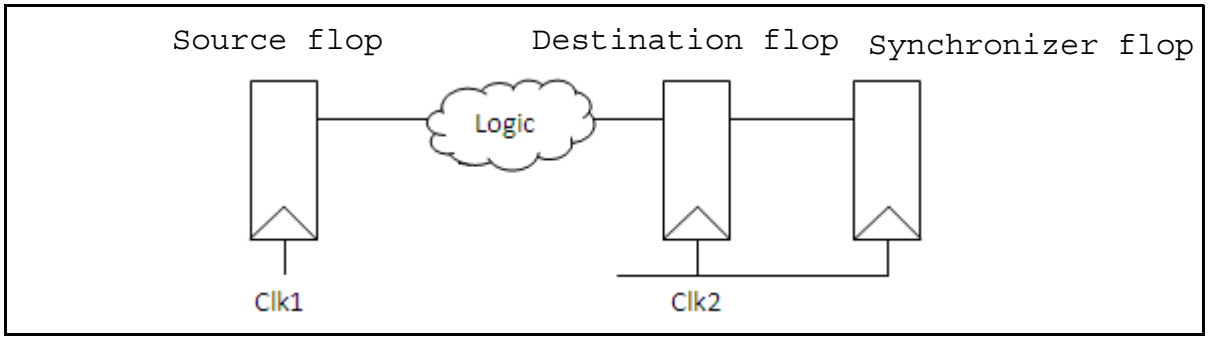


FIGURE 31. Combinational logic used between crossing

Such logic can cause a glitch to be captured by destination.

Reason - Specify 'synchronize_data_cells', not 'synchronize_cells' for bus signals

This reason is reported if you specify a cell used to synchronize a single-bit signal, but the cell is found connected to a multi-bit signal. This is either a design issue or a setup problem.

Reason - Specify 'synchronize_cells', not 'synchronize_data_cells' for single bit signals

This reason is reported if you specify a cell used to synchronize a multi-bit signal, but the cell is found connected to a single bit signal. This is either a design error or a setup problem.

Reason - Invalid RTL flop/cell used in synchronizer chain

A destination may be synchronized by *Conventional Multi-Flop Synchronization Scheme*, but any of the synchronizer flip-flops used in the synchronizer chain is not among the allowed cells specified by the *num_flops* constraint.

SpyGlass reports such cases with one of the following failure reasons in the

message and spreadsheet:

- `Invalid RTL flop used in synchronizer chain`
- `Invalid cell <cell-name> used in synchronizer chain`

For example, consider that you specify the following constraint:

```
num_flops -to_clk clk2 -value 3 -cells FD2
```

In the above example, the allowed cell in the synchronizer chain for the `clk2` clock is `FD2`.

Now consider that the synchronizer chain contain also contains another cell `FD1` in addition to `FD2`, as shown in the following figure:

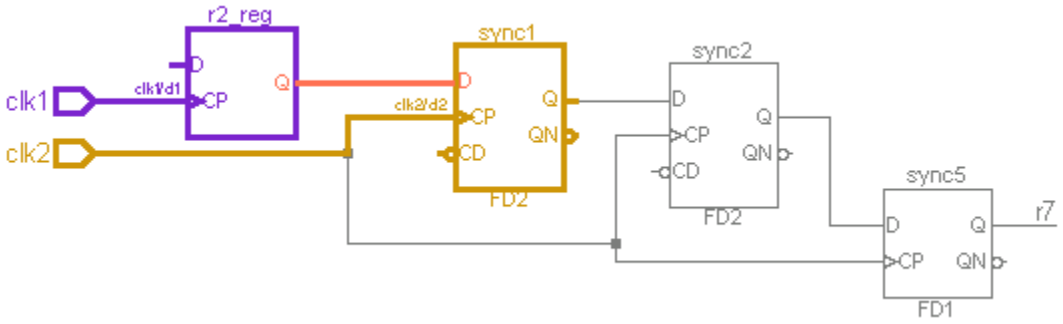


FIGURE 32. Synchronizer Chain

In the above case, SpyGlass reports the following reason:

`Invalid cell FD1 used in synchronizer chain`

Reason - Invalid synchronizer module/cell <name>

SpyGlass reports this reason when a destination matches with the synchronized cell modules specified by the *sync_cell* constraint, but the clock, domain, or period does not match.

For example, consider the following constraints:

```
sync_cell -name sync_latch_2 -to_clk clk2
sync_cell -name sync_latch_1 -to_clk clk3
```

Now consider the following figure:

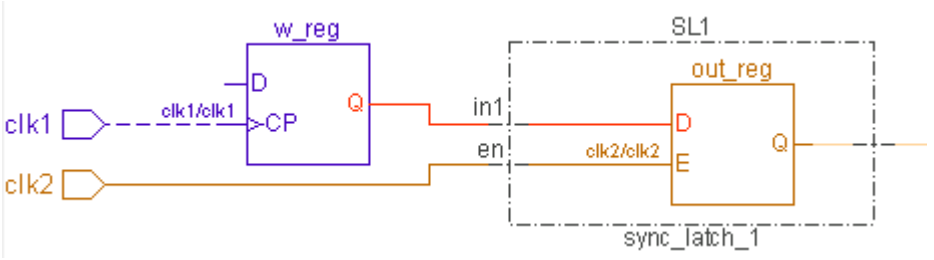


FIGURE 33. Invalid synchronizer module/cell

In the above figure, the destination latch matches with the synchronized cell module, `sync_latch_1`, specified by the *sync_cell* constraint. However, this destination is clocked by the `clk2` clock that does not match with the `clk3` clock specified for `sync_latch_1` in the *sync_cell* constraint. Therefore, SpyGlass reports the following reason in this case:

```
Invalid synchronizer module/cell sync_latch_1
```

Reason - Unsynchronized synchronous reset

This reason is reported when a synchronous reset is not synchronized by *Conventional Multi-Flop Synchronization Scheme* or *Synchronizing Cell Synchronization Scheme*. Such reset sources are not checked for data synchronization and are reported as unsynchronized.

The following schematic shows the scenario in which this reason is reported:

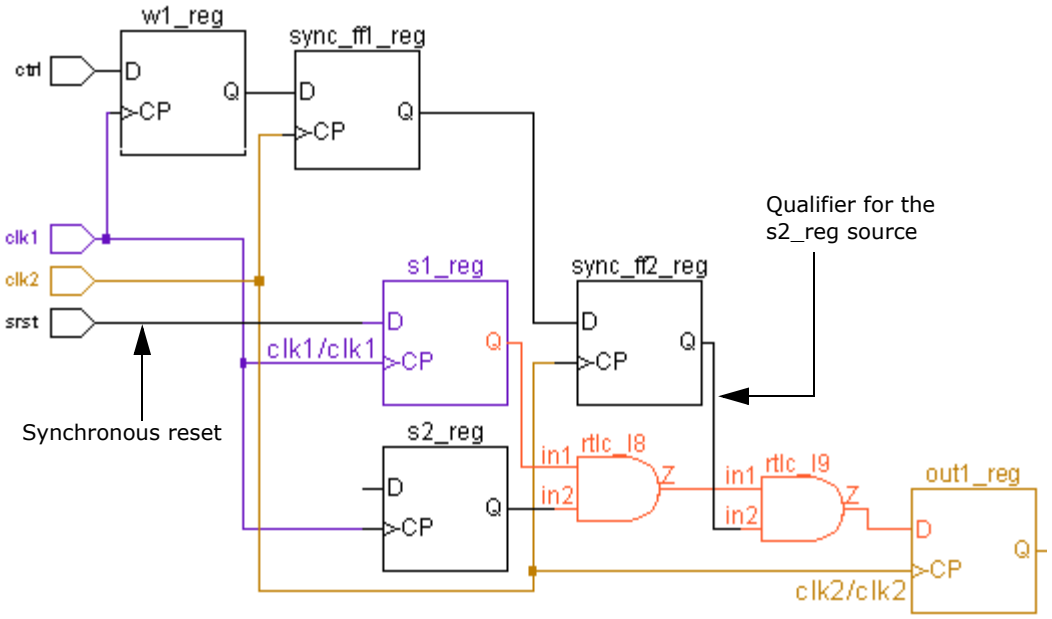


FIGURE 34. Unsynchronized synchronous reset

Reason - [User-defined qualifier/ Qualifier] merges with another source before gating logic

This reason is reported when a qualifier merges with another source before reaching the gating logic whose other input is driven by the source of the crossing.

The following figure shows the scenario in which SpyGlass reports this reason:

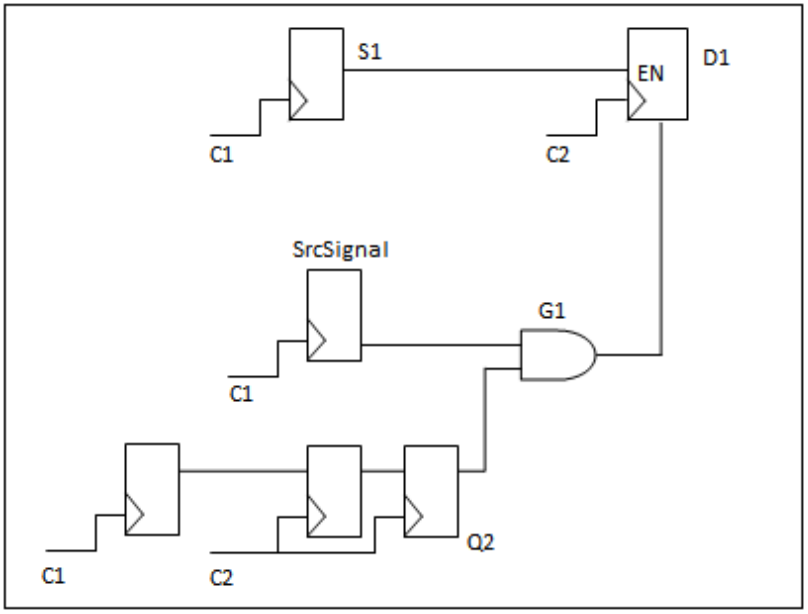


FIGURE 35. Qualifier merging with another source

In Figure 156, SpyGlass reports a crossing between S1 and D1 as unsynchronized because a qualifier is merging with the source. To debug/fix such unsynchronized crossings, consider the following scenarios:

- If SrcSignal is reported synchronized with the qualifier, SrcSignal is scalar, and the *allow_merged_qualifier* (default `yes`) parameter is specified to `yes`, SpyGlass reports the crossing between S1 and D1 as synchronized.
- If SrcSignal is reported synchronized with the qualifier, SrcSignal is vector, and the *allow_merged_qualifier* (default `yes`) parameter is specified to `yes`, and the constraint *signal_type -type* control is specified on the SrcSignal, SpyGlass reports the crossing between S1 and D1 as synchronized.
- If SrcSignal is unsynchronized with the qualifier, debug the reasons for unsynchronized crossing source SrcSignal by using its failure reasons as reported by SpyGlass.

Reason - [User-defined qualifier/ Qualifier] merges with another source with non-deterministic enable condition before gating logic

This reason is reported when a qualifier merges with another source before reaching the gating logic with non-deterministic enable condition, whose other input is driven by the source of the crossing.

The following figure shows the scenario in which SpyGlass reports this reason:

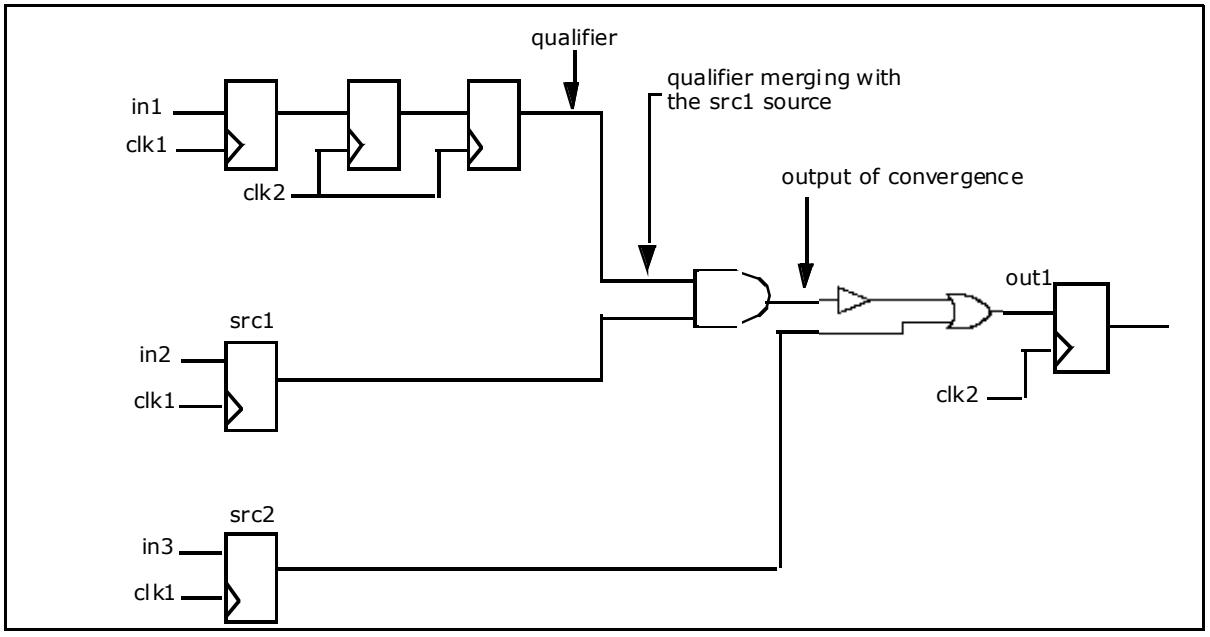


FIGURE 36. Qualifier Merging with a Source

In the above scenario, if the *allow_merged_qualifier* parameter is set to `strict`, the blocking value is 0 at the output of convergence because of the AND gate where the `src1` source and qualifier are getting merged. Therefore, the qualifier can block src1 with blocking value 0 at the AND gate. This value is propagated further through buffers and inverters and the same 0 value reaches the input of the OR gate, which cannot block

another source `src2`. Therefore, `src2` is reported as unsynchronized with the following reason:

`[User-defined qualifier/Qualifier] merges with another source with nondeterministic enable condition before gating logic`

When the buffer in the path is replaced by an inverter, the crossing is reported as synchronized because the blocking value at the input of the OR gate will be '1', which can now block the source.

Reason - [User-defined qualifier/Qualifier] merges with the same source before gating logic

This reason is reported when a source diverges and the diverged source path merges with a valid qualifier before re-converging.

For example, SpyGlass reports this reason in the following scenario where `S1` is diverging and one of the diverged path merges with a qualifier before re-converging on the AND gate.

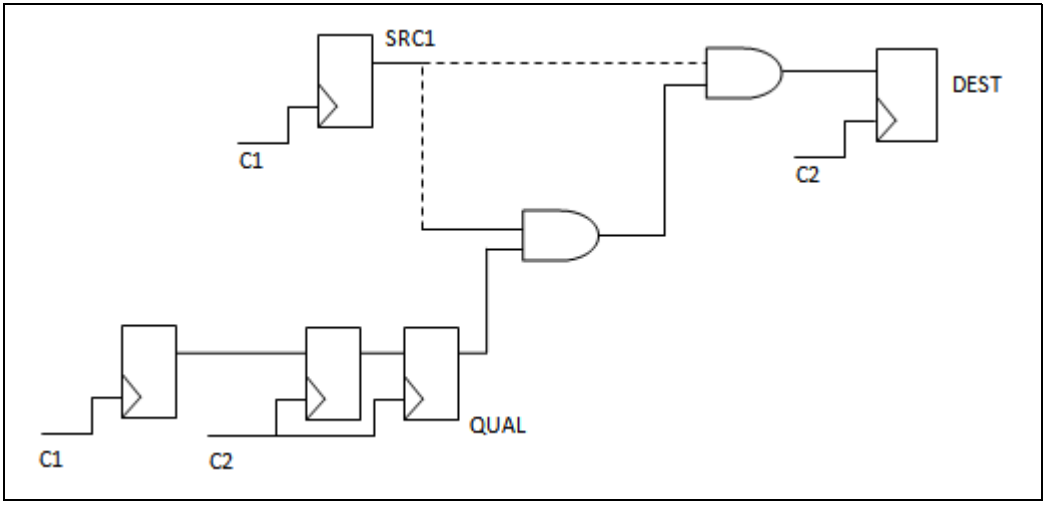


FIGURE 37. Diverging Source merging with a valid qualifier before reconverging

Reason - Gating logic not accepted

When a source converges with a valid qualifier signal at an invalid combinational gate, SpyGlass reports such cases with one of the following failure reasons in the message and spreadsheet:

- `Gating logic not accepted: gate-type invalid`

  For example, SpyGlass reports this reason in the following scenario as the qualifier merges with the XOR gate, which is considered invalid:

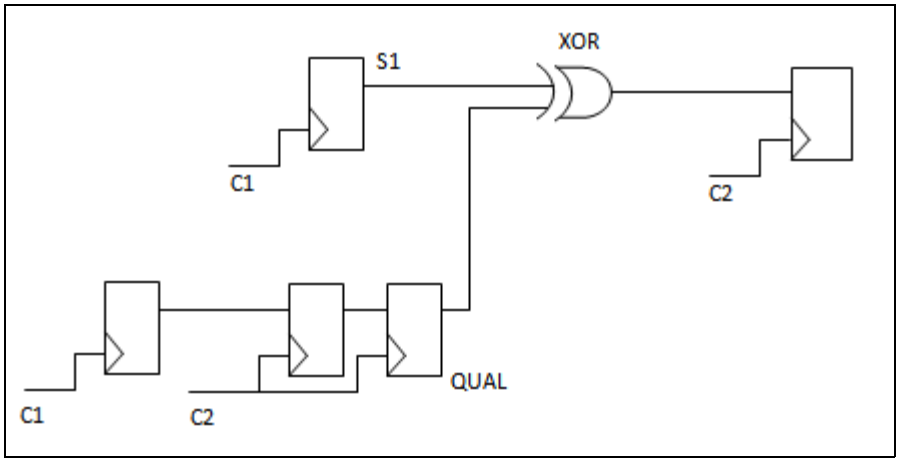


FIGURE 38. Qualifier merging with the XOR gate

- `Gating logic not accepted: source drives MUX select input`

  For example, SpyGlass reports this reason in the following scenario as the `S2` source drives the mux select input because of which the `Q1` qualifier does not synchronize the `S1` source:

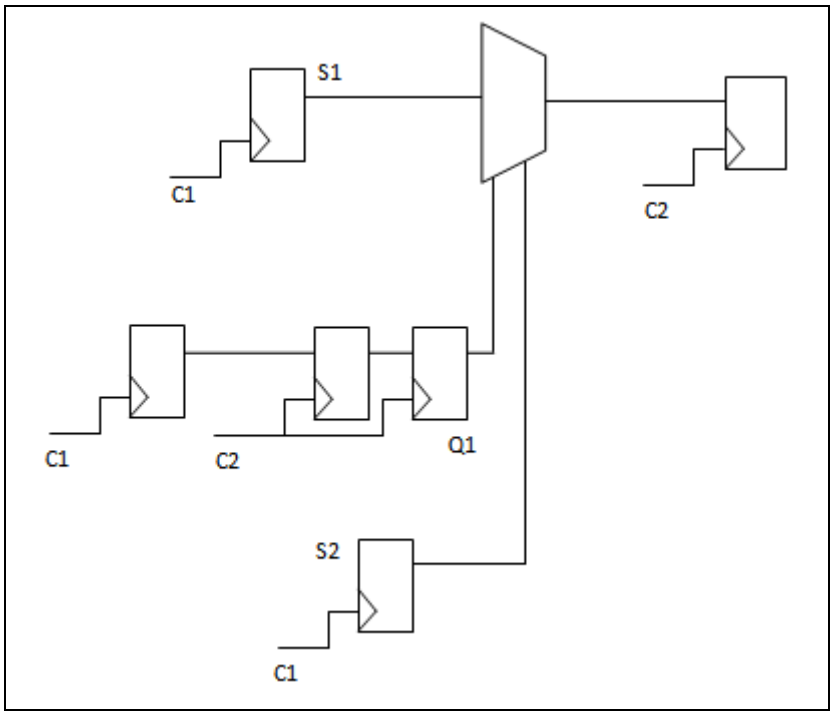


FIGURE 39. Source driving the MUX select input

- `Gating logic not accepted: source and [qualifier/ user-defined qualifier] drive MUX data inputs`

  For example, SpyGlass reports this reason in the following scenario as the `Q1` qualifier drives the mux data input rather than the mux select pin. As a result, the Q1 qualifier does not synchronize the `S1` source.

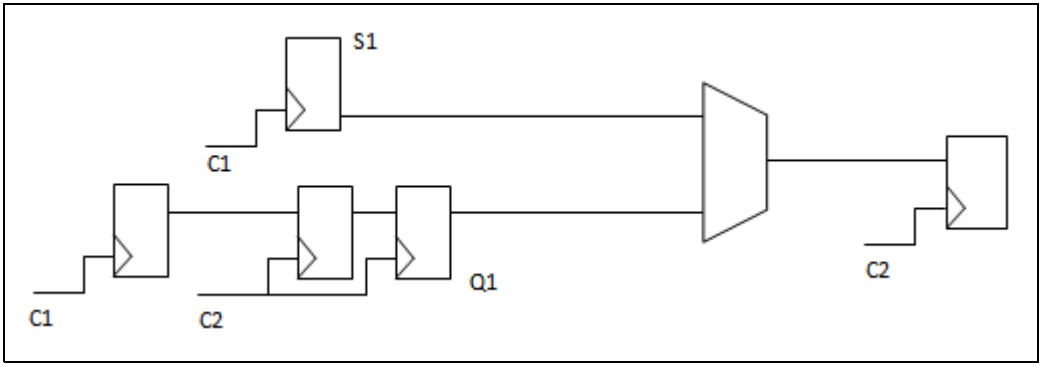


FIGURE 40. Source and qualifier driving MUX data inputs

- `Gating logic not accepted: only sources drive MUX data inputs; at least one destination domain signal should drive a MUX data input`

  For example, SpyGlass reports this reason in the following scenario as no destination domain signal is detected on the data pins of the mux.

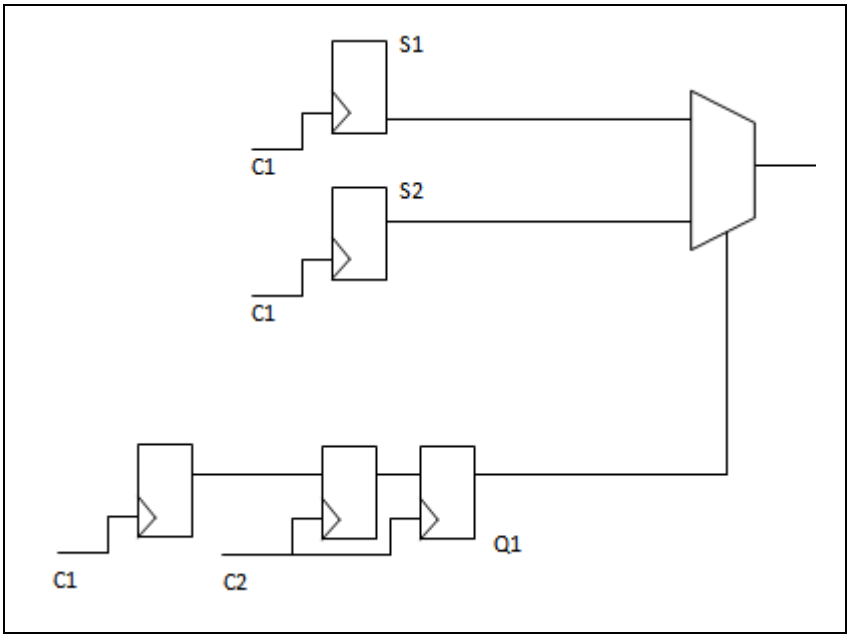


FIGURE 41. Only sources driving MUX data inputs

In the above scenario, at least one destination domain signal should drive the mux data pin for the `Q1` qualifier to synchronize the `S1` source.

Reason - Qualifier not accepted: crossing source is the same as source of qualifier

This reason appears when the source of a crossing and the source of a qualifier crossing is same. In such cases, the qualifier is not considered as a valid qualifier. This is because the source acts as both data and control signal in the data crossing.

The qualifier in such cases can be synchronized by either of the *Conventional Multi-Flop Synchronization Scheme*, *Synchronizing Cell Synchronization Scheme*, or *Qualifier Synchronization Scheme Using qualifier -crossing*.

For example, consider the following schematic:

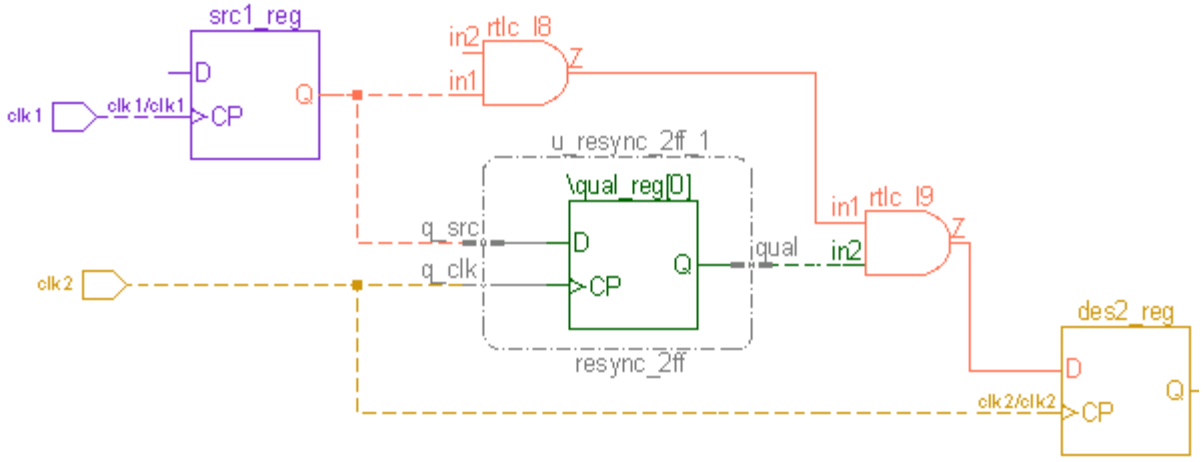


FIGURE 42.

In the above schematic, `src1` is the source signal to both the `des1` destination and the `qual1` qualifier. Therefore, qual1 is not considered as a valid qualifier.

Reason - Combinational loops on crossing

This reason is reported when a combinational loop is present in the crossing path. In this case, a qualifier drives one gate in the loop while the source drives another gate.

Reason - No Enable Condition Selected

This reason is reported during *The Enable Expression-Based Synchronization Analysis* when both the following conditions hold true:

- The *sync_point_selection* parameter is set to `user`.
- A possible point exists in the source to destination paths where valid synchronizing signal is reaching.

  This scenario is shown in the following schematic:

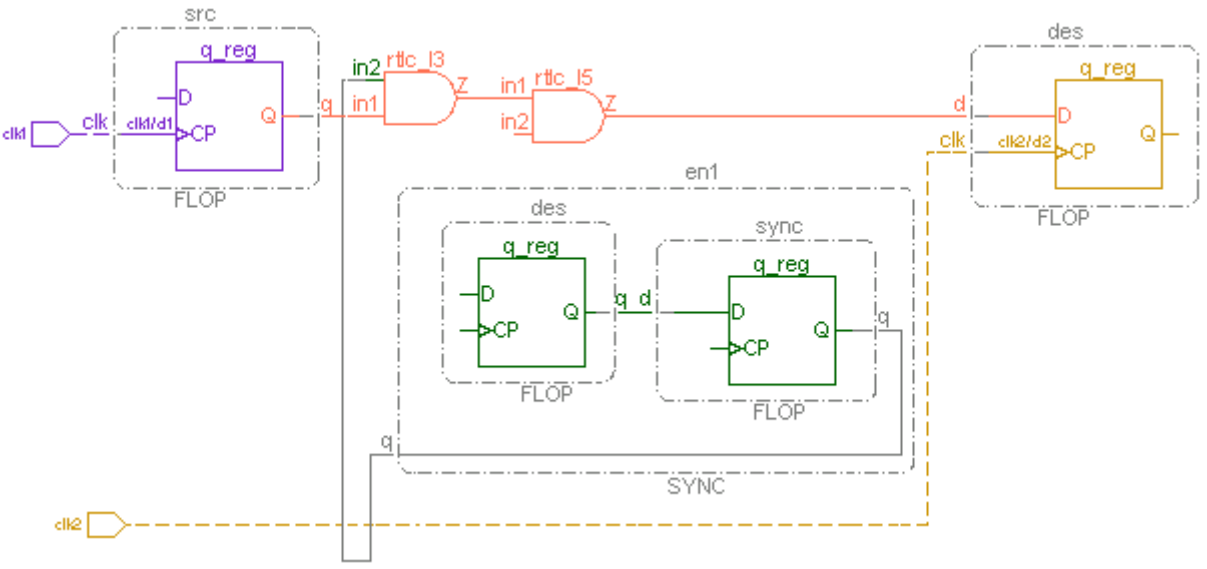


FIGURE 43.

Reason - Enable Criteria not satisfied: No destination domain

This reason is reported during *The Enable Expression-Based Synchronization Analysis* when both the following conditions hold true:

- The `sync_check_type` parameter is set to `enable_with_des_dom`.
- A crossing is unsynchronized as the source is not merging with any valid destination domain signal in its path.

This scenario is shown in *Figure 44*.

Reason - Enable Criteria not satisfied: No Qualifier found

This reason is reported during *The Enable Expression-Based Synchronization Analysis* when both the following conditions hold true:

- The `sync_check_type` parameter is set to `enable_with_qual`.
- A crossing is unsynchronized as the source is not merging with any valid qualifier in its path.

This scenario is shown in the following schematic:

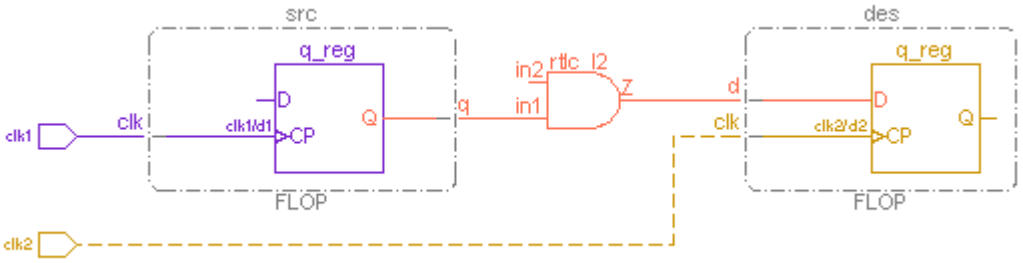


FIGURE 44.

Reason - Enable Criteria not satisfied: gating-type not accepted

This reason is reported during *The Enable Expression-Based Synchronization Analysis* when the requirement of an enable is satisfied, but one of the following conditions is true:

- The enable is reaching an unacceptable gate, such as XOR, OR, or AND gate.
- The *enable_and_sync* parameter is not set to `yes`.

The following schematic shows the scenario in which this reason is reported:

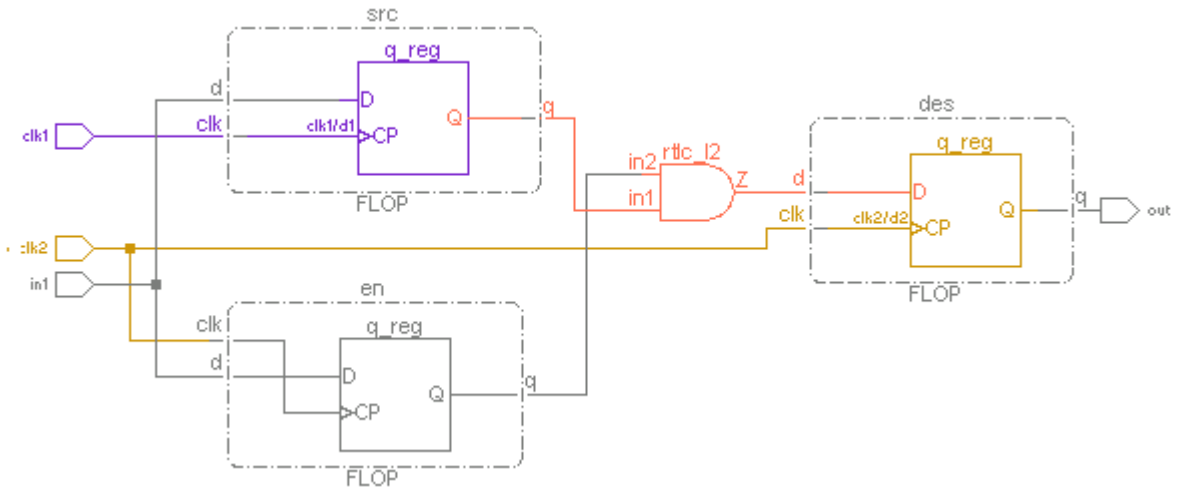


FIGURE 45.

Reason - Enable Criteria not satisfied: Source reach mux select

This reason is reported during *The Enable Expression-Based Synchronization Analysis* when one the mux select line is receiving along with the valid enable signal.

This scenario is shown in the following schematic:

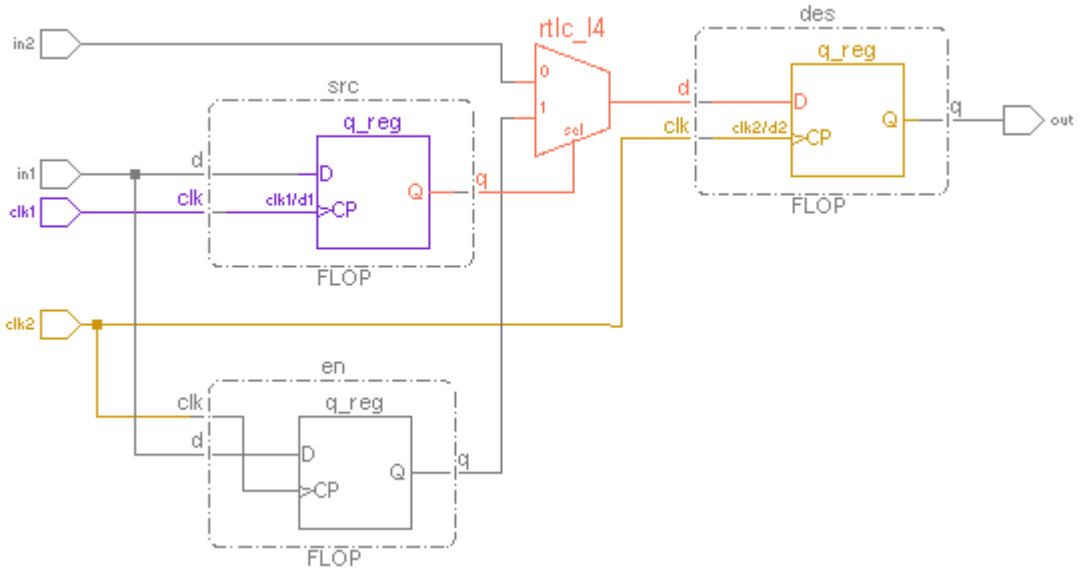


FIGURE 46.

Reason - Domain Criteria not satisfied: No domain

This reason appears if one of the enable terminals of a synchronizing gate is not receiving any domain.

For example, in the following figure, the C pin of the AND gate does not receive any domain:

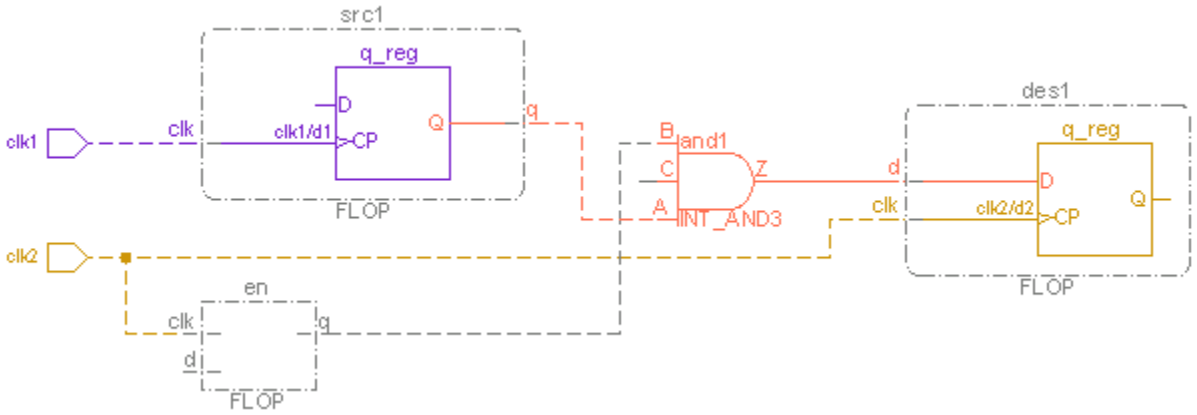


FIGURE 47.

Reason - Domain Criteria not satisfied: Source domain

This reason appears when one of the enable terminals of a synchronizing gate is receiving the source domain.

For example, in the following figure, the `src1` and `src2` sources have the same domain:

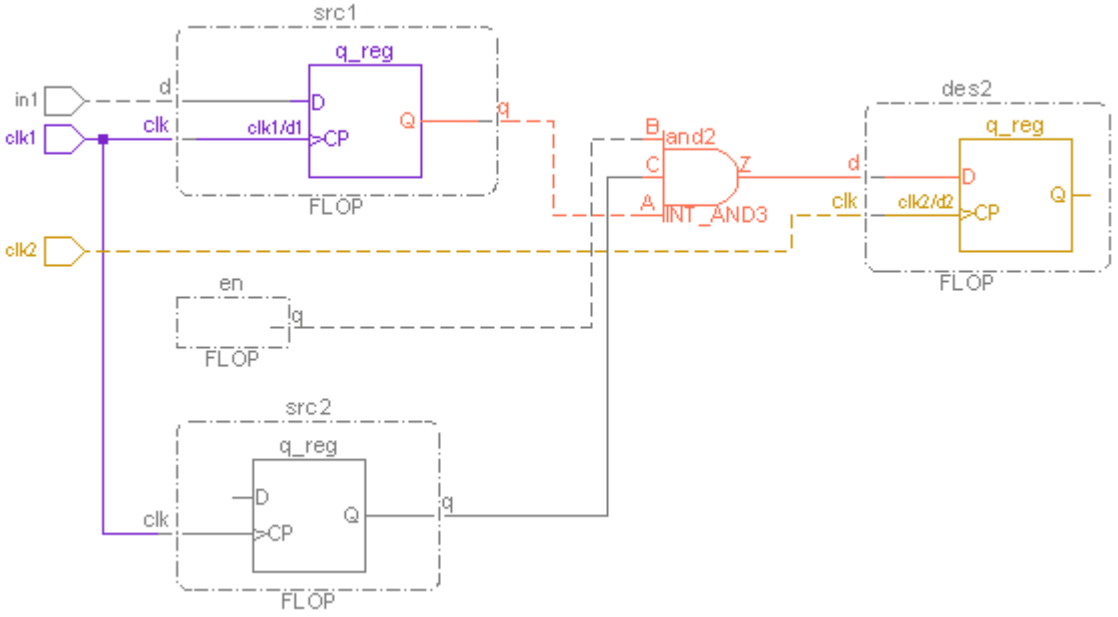

src1
q_reg
in1
d
D
clk1
clk
clk1/d1
CP
Q
q
FLOP
B
and2
C
Z
A
INT_AND3
des2
q_reg
d
clk
clk2/d2
D
CP
Q
FLOP
clk2
en
q
FLOP
src2
q_reg
clk
D
CP
Q
q
FLOP

FIGURE 48.

Parameters of the Ac_sync_group Rules

The following are the common parameters used by the `Ac_sync_group` rules:

- *ac_sync_mode*: Default value is `strict_gate`, `strict_qual_logic`. Set this parameter to specify a mode in which rule should run to check data crossings. Other possible values are `soft_gate`, `soft_qual_logic`.
- *all_potential_qual*: Default value is `no`. Set this parameter to `yes` to show all potential qualifiers in the spreadsheet of this rule.
- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *allow_enabled_multiflop*: Default value is `no`. Set this parameter to `yes` to consider enabled flip-flops as destination or synchronizer flip-flops in conventional multi-flop synchronization scheme. Other possible value is `same_enable`.
- *allow_half_sync*: Default value is `yes`. Set this parameter to `no` to not treat half synchronizers as valid synchronizers.
- *allow_merged_qualifier*: Default is `yes`. When a qualifier merges with a source, the output of the convergence is not considered as a valid qualifier to qualify other sources. However, if the source is a control signal, the convergence output may still act as a qualifier.

  By default, such convergence outputs are considered as valid qualifiers. Set this parameter to no to disallow them from being considered as qualifiers.
- *cdc_bus_compress*: Default value is `Ac_glitch03`. Set this parameter to `DeltaDelay02` to check all the bits of the source bus by the *DeltaDelay02* rule. For information on the other possible values, see *Possible values of the cdc_bus_compress parameter*.
- *cdc_qualifier_depth*: Default value is -1. Specify a positive integer value indicating the depth of sequential logic till which a qualifier should be searched.
- *cdc_qualifier_depth_start*: Default value is `num_flop`. Set this parameter to `sync_chain` so that the last flip-flop of the synchronization chain is

the starting point beyond which a qualifier should be searched. Other possible value is `dest`.

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *clock_gate_cell*: Default value is `NULL`. Specify a comma or space-separated list of clock-gating cell names.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *delayed_ptr_fifo*: Default value is `no`. Set this parameter to `yes` when the read/write pointers are delayed and the multiplexer inside the memory is one-hot or implemented using gates.
- *disable_inst_grouping*: Default value is `no`. Set this parameter to `yes` to disable *Instance-Based Grouping* of messages of this rule.
- *disable_seq_clock_prop*: Default value is `no`. Set this parameter to `yes` to disable propagation of clocks beyond flip-flops.
- *dump_inst_type*: Default value is `all`. Set this parameter to `flop` to generate destinations and synchronizers that are flip-flops in *The SynchInfo Report* and *The CrossingInfo Report*.
- *dump_sync_info*: Default value is `no`. Set this parameter to `yes` to generate *The SynchInfo Report* and *The CrossingInfo Report* reports in the default format. Other possible values are `no`, `detailed`, and `detailed_mod`.

- *enable_ac_sync_qualdepth*: Default value is `no`. Set this parameter to `yes` to report qualifier name and qualifier depth in the *Message-Based Spreadsheet*.
- *enable_and_sync*: Default value is `no`. Set this parameter to `yes` to enable the *AND Gate Synchronization Scheme*.
- *enable_clock_gate_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Clock-Gating Cell Synchronization Scheme*.
- *enable_clock_path_crossings*: Default value is `no`. Set this parameter to `yes` to identify potential sources in a clock path of flip-flops.
- *enable_delayed_qualifier*: Default value is `yes`. Set this parameter to `no` to disable searching for qualifiers and potential qualifiers through a combinational logic. Other possible value is `strict`.
- *enable_glitchfreecell_detection*: Default value is `no`. Set this parameter to `yes` to report glitch-free multiplexers in a design.
- *false_path_enable_hier_view*: Default value is `no`. Set this parameter to `yes` to correctly support hierarchical terminals where the specified *cdc_false_path* constraint is retained on the hierarchical terminal itself.
- *sync_point_report_limit*: Default value is `5`. Set this parameter to an integer (greater or equal to 2) to specify the maximum number of enable points (to be shown in the *Message-Based Spreadsheet for the Enable Condition Based Flow*) at which a crossing could be synchronized.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Synchronized Enable Synchronization Scheme*.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *format_report*: Default value is `no`. Set this parameter to `yes` to wrap text in the reports generated by the SpyGlass CDC solution.
- *glitch_protect_cell*: Default value is `NULL`. Specify a comma or space-separated list of glitch protection cell names for the *Glitch Protection Cell Synchronization Scheme*.

- *ignore_num_rtl_buf_invs*: Default value is `many`. Set this parameter to `one` to allow one buffer and inverter. Other possible values are `two` and `none`.
- *netlist_name_convention*: By default, this parameter is not set to any value. Set this parameter to a string value to specify a naming convention for a generated net in netlist designs.
- *num_flops*: Default value is 2. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.
- *prefer_abstract_port*: Default value is `no`. Set this parameter to `yes` to consider a black box as the source on which the *assume_path* and *abstract_port* constraints are applied.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *report_instance_pin*: Default value is `no`. Set this parameter to `yes` to report the name of instance pin of a netlist design. Other possible values are `flop`, `latch`, `bbox`, `seqCell`, and `all`.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *show_module_in_spreadsheet*: Default value is `no`. Set this parameter to `yes` to generate module and instance data in the generated spreadsheet.
- `sta_based_clock_relationship`: Default value is `no`. Set this parameter to `yes` to compute domains based on the specification of the *sg_clock_group* constraint.
- *strict_double_flop*: Default value is `no`. Set this parameter to yes to mark clock crossings as synchronized.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *sync_check_type*: Default value is `qual_only`. Set this parameter to `enable_with_qual` to allow a qualifier as a valid enable for *The*

Enable Expression-Based Synchronization Analysis. Other possible value is `enable_with_des_dom`.

- *sync_point_selection*: Default value is `first`. Set this parameter to `last` to consider the last possible enable signal (from source) where a crossing could be synchronized. Other possible values are `none` and `gp_sync`.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *user_group_str*: By default, this parameter is not set to any value. Set this parameter to a comma or space-separated list of strings based on which messages of the `Ac_sync_group` rules are grouped.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *cdc_ignore_multi_domain*: Default value is `none`. Set this parameter to `data_path` to enable synchronization analysis of a data path clock domain crossing involving multiple source domains.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraints of the Ac_sync_group Rules

The following are the common constraints used by the `Ac_sync_group` rules:

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.

NOTE: *The Ac_sync_group rules report violations on blackbox input/output ports if the abstract_port constraint is defined with the* `-end` *argument.*

- *allow_combo_logic* (Optional): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *fifo* (Optional): Use this constraint to specify FIFO information.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *ip_block* (Optional): Use this constraint to specify IP blocks in your design.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *output* (Optional): Use this constraint to specify clock domain at output ports.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *signal_type* (Optional): Use this constraint to specify the signal type (control or data).

- *sync_cell* (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain the specified frequencies, source/destination clocks, or domains.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Important Information Regarding the Ac_sync_group Rules

Note the following points for the `Ac_sync_group` rules:

- If a destination black box is not modeled by using either *abstract_port* or *sync_cell* constraint or the *synchronize_cells* parameter and the black box has multiple domain clocks reaching to it, such black boxes are not checked for crossing detection. However, a crossing check is performed if the black box receives a single clock.
- If you have specified a destination black box pin by using the *abstract_port* constraint, this black box is considered as an abstracted model and no crossing checks are performed on it as a destination.
- If a destination black box pin is a vector terminal, a violation is reported on the nets connected to these black box vector pins that will be bus-merged.
- If a destination signal is specified as a control signal by using the *signal_type* constraint, it is checked only for the control synchronization schemes.
- If a destination signal is specified as a data signal by using the *signal_type* constraint, it is checked only for the data synchronization schemes.

Limitations of the Ac_Sync_Group Rules

The `Ac_sync_group` rules have following limitations:

- These rules do not detect crossings if a sequential library cell with a combinational arc for a pin is present between a source and destination.
- These rules do not consider the following constraint specification:

  ```
  qualifier -type src
  ```

- These rules may report false crossings with an IO pad cell present between the source and destination.
- These rules analyze inside library cells for accurate synchronization checking. It is not supported for a library cell that has a pin, which diverges and converges back inside the library cell.

Performing Functional Analysis in SpyGlass CDC

While performing structural analysis of a design, SpyGlass may not detect functional bugs in the design. For example, structural analysis may report about a missing synchronizer in a crossing. However, even after inserting the synchronizer, the design may not function as expected. In such cases, structural analysis cannot identify the issues causing the unexpected design behavior. To identify such issues, you should perform functional analysis.

Functional analysis (also known as *The Functional Validation Methodology*) checks the design functionality and detects design bugs. For example, functional analysis can be:

- Checking for gray encoding of the vector signal that is crossing a clock domain.
- Checking for correct functionality of a handshake scheme.
- Checking for FIFO overflow and underflow conditions.

The Functional Validation Methodology

Functional validation methodology is the sequence of steps that enables you to check the functional correctness of a design.

To check the functional correctness of a design, you begin by identifying the design functionality to be tested. Such design functionality is known as a **property**. Once you identify *Properties*, you write directives to the formal verification tool to verify the given property. Such directives are known as *Assertions*. You then verify the assertions and debug the reported violations that indicate the *Passed*, *Failed*, and *Partially Proved* properties.

For large designs, this methodology enables you to converge to the best possible outcome (highest number of *Passed* properties and all the bugs detected) in short time.

The functional validation methodology consists of the following stages:

- *Stage 1: Running SpyGlass in the Audit Mode*
- *Stage 2: Analyzing Design Setup*
- *Stage 3: Running SpyGlass in the Default Mode*
- *Stage 4: Diagnosing and Fixing Design Bugs*
- *Stage 5: Running SpyGlass with a Higher CPU Time*
- *Stage 6: Concluding Partially-Proved Assertions*

The following diagram is the recommended methodology for functional validation using SpyGlass:

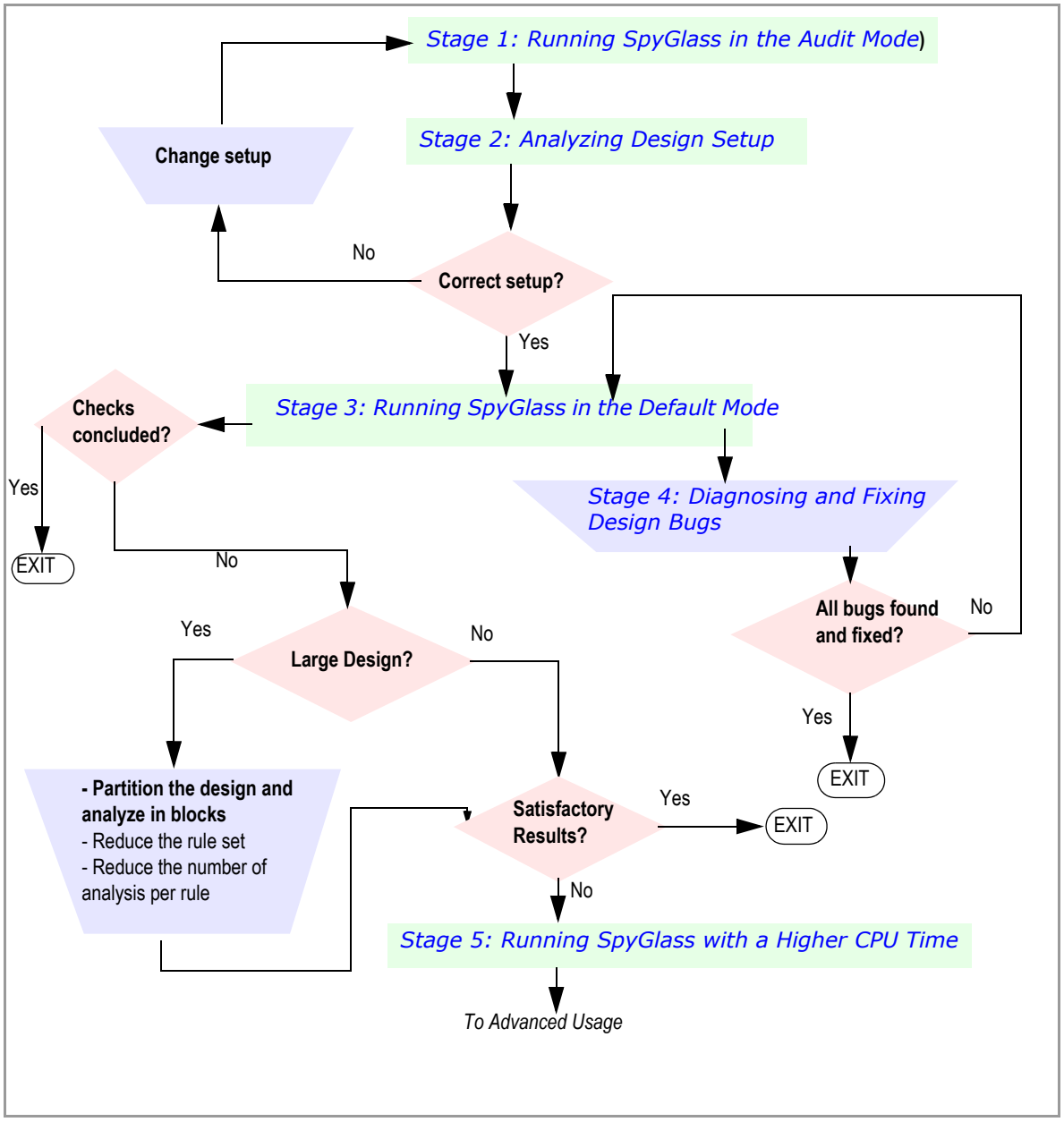


FIGURE 1. Functional Validation Methodology - Basic Usage

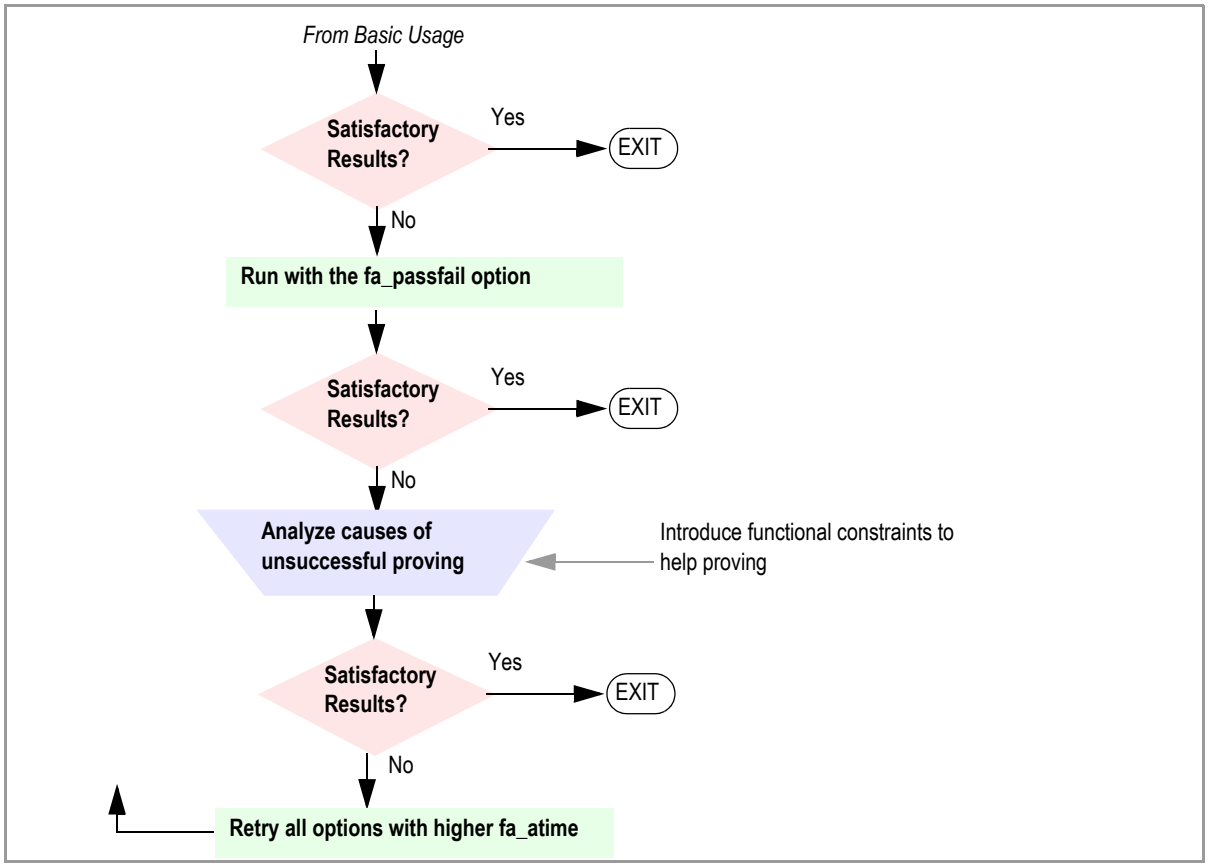


FIGURE 2. Functional Validation Methodology - Advanced Usage

NOTE: *All incremental runs should use the Property file created in the previous run.*

Stage 1: Running SpyGlass in the Audit Mode

In the audit mode, you generate a list of *Assertions* of a design without performing functional analysis on these assertions.

The following figure shows the flow of this stage:

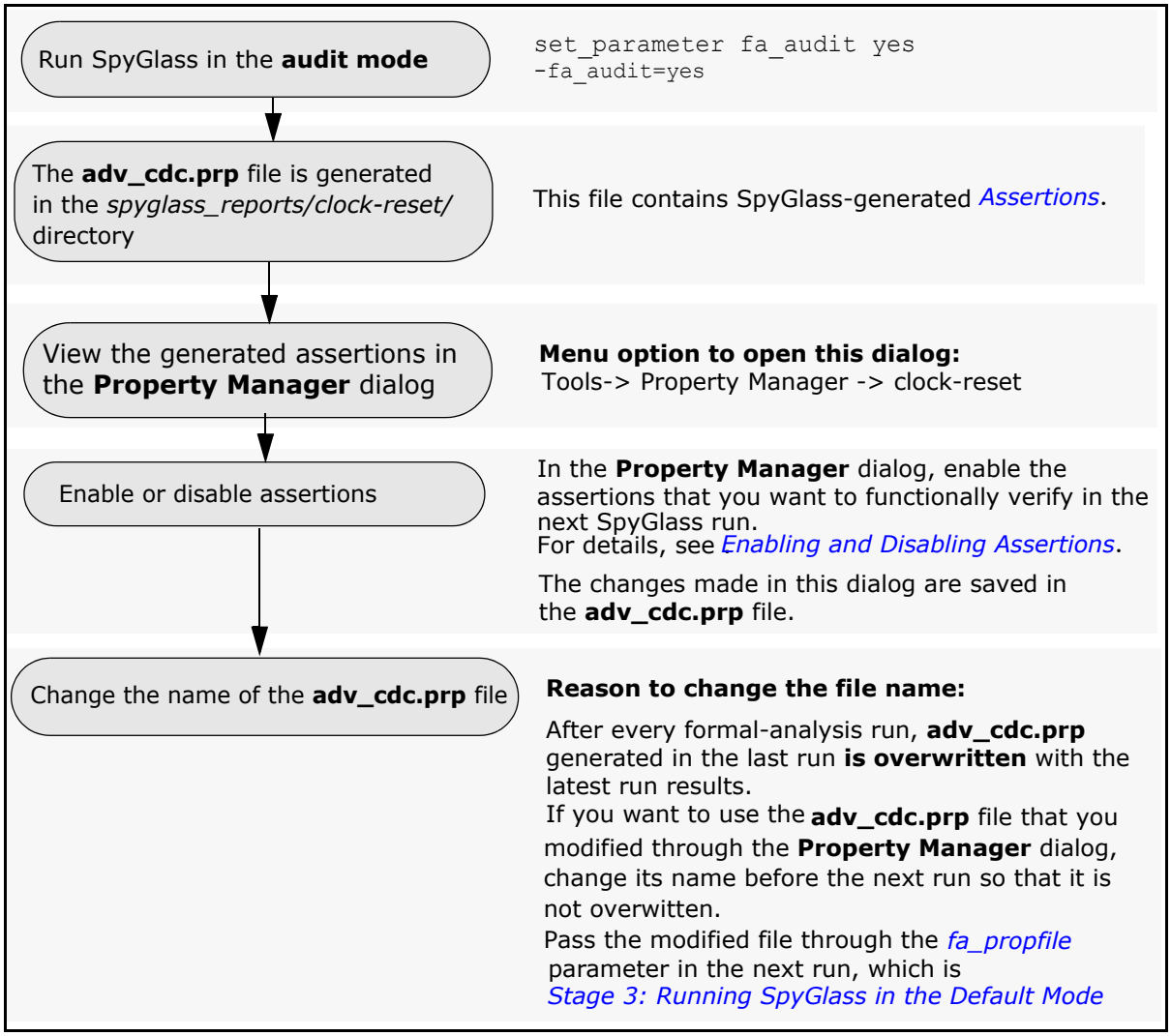


FIGURE 3. Flow during the Audit Mode

Stage 2: Analyzing Design Setup

Analyze the design information described in the following table:

Design Information	Analysis
Clocks	Perform the following analysis of the clocks in the design: • Check clock definitions in the design. If a clock definition is not reported correctly, define the clock in an SGDC file. • Check for the automatically generated clocks. For a clock source that is not user-defined, SpyGlass generates a default clock running at 100MHz with a rising edge at 0 and a falling edge at 5ns. Look at such automatically-generated clocks and correct and/or add attributes such as clock frequencies. Specify this information in an SGDC file to be specified to the next SpyGlass run. • Refer to the schematic and RTL back annotations can be used to visualize all clock sources and destination registers. • After constraint or design modifications, SpyGlass can be launched again, with the fa_audit parameter, to explore and further tune the clock definition until a correct clock definition is obtained.
Resets	Explore the reset signals of your design. Make sure that all reset signals are listed and the attributes are correct (active high or active low, hard reset or soft reset).
Initial state	Look for registers that are not initialized (registers at "x"). Non-initialized registers can cause false failure because they will be assigned to the convenient value to make a property fail. Provide SpyGlass Design Constraints (reset signals, reset vector, initial state) to help initializing the registers. You may also want to validate the correctness of the initial state.
Properties	Explore the extracted properties of your design from adv_cdc.prp file. Statistics after audit can be used to determine a functional exploration strategy for a design. You may decide to validate each rule or set of rules separately if the number of rules is too high. You can estimate the complexity, coverage, run time memory requirements, etc.

Stage 3: Running SpyGlass in the Default Mode

In this stage, SpyGlass verifies the generated *Assertions*.

The following figure shows the flow of this stage:

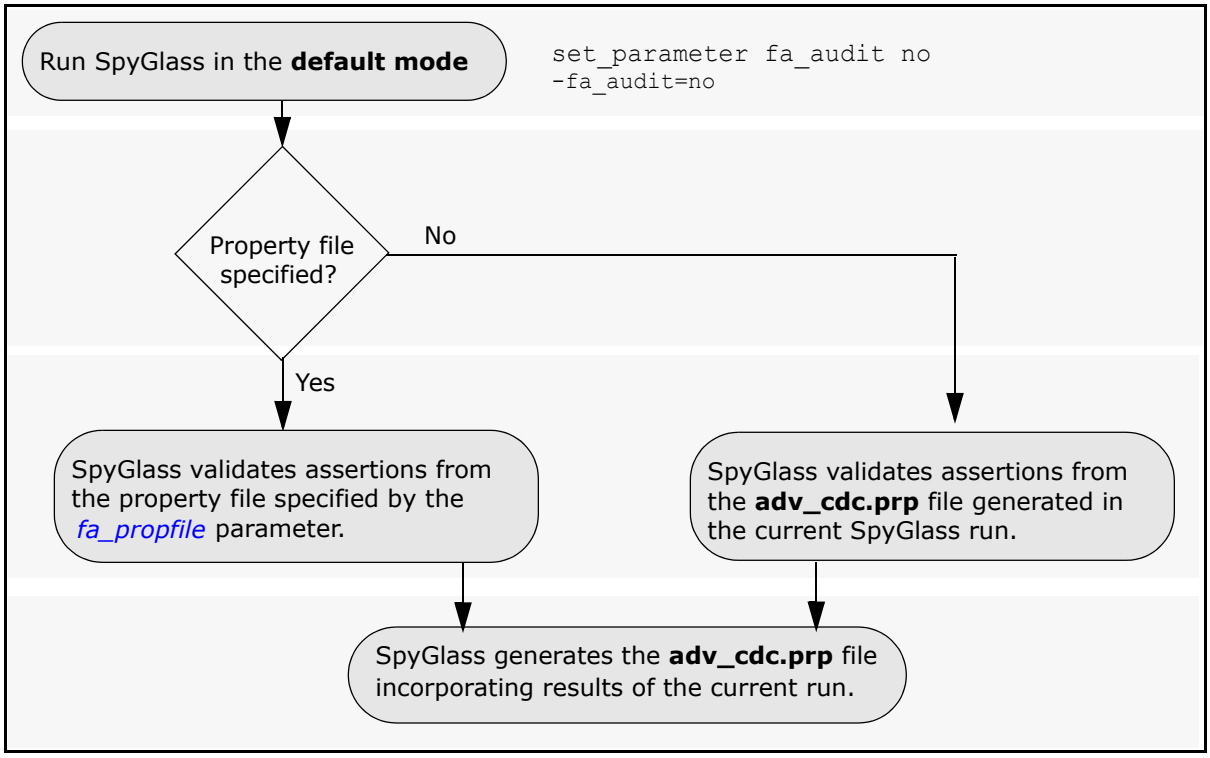


FIGURE 4. Flow during the Default Mode of functional analysis

Stage 4: Diagnosing and Fixing Design Bugs

Based on *Assertions* verification during *Stage 3: Running SpyGlass in the Default Mode*, SpyGlass reports appropriate violations of different status, such as *Passed*, *Failed*, and *Partially Proved*.

Assuming the setup is correct (*Stage 2: Analyzing Design Setup*), diagnose a violation in the following order:

- *Reading the Violation Message*
- *Examining RTL and Corresponding Schematic Diagram*
- *Examining Waveform Showing a Concise Trace of the Violation*

Reading the Violation Message

Each violation message provides the following types of information that help you identify the cause of a bug:

- Type of problem

  For example, if the problem is specific to clock-crossing data loss issue or a gray-code failure.

- Involved signals and their values at the time of failure

  This is the list of signals that enable you to diagnose the problem. For example, the signal may be the bus on which gray-coding is checked.

Examining RTL and Corresponding Schematic Diagram

When you double click on a violation message, the RTL code containing the problem is highlighted in the HDL Viewer. Explore the RTL code surrounding the reported signals to find the exact cause of the violation.

You can cross-probe to the schematic from the RTL to view the graphical format of the design area containing the problem.

Examining Waveform Showing a Concise Trace of the Violation

For complex bugs involving many signals and rooted deep in the sequential space, the message and RTL/schematic viewer may not suffice to find the exact cause of a bug.

In such cases, view the waveform to know the cause of functional bug. For details on using the waveform, see *Using Waveform during Functional Analysis*.

Stage 5: Running SpyGlass with a Higher CPU Time

If during *Stage 4: Diagnosing and Fixing Design Bugs*, you are not able to detect the bug, increase the CPU time that SpyGlass uses to analyze *Assertions*.

With a higher CPU time (also known as **atime**), chances for proving an assertion are high.

Specify a higher CPU time by using the *fa_atime* parameter.

Stage 6: Concluding Partially-Proved Assertions

Even after *Stage 5: Running SpyGlass with a Higher CPU Time*, some *Assertions* remain in the *Partially Proved* state. The reason for this can be *Complex Assertions* and/or *Large and Complex Design*.

Complex Assertions

Complex assertions are the assertions that require monitoring of signals for hundreds of cycles.

One way to conclude such assertions is to explore the design starting from various initial states. In this case, chose an initial state to take the design closer to failing conditions. For example, if an assertion is set to check for FIFO overflow, push data into the FIFO so the FIFO is almost full before the overflow check is launched.

Large and Complex Design

A check may not be completely analyzed because of *High Design Cycle* and/or *Large Design*.

High Design Cycle

In case of multiple clocks in a design, hundreds or thousands of cycles may be required before all clocks are again lined the same way as the starting cycle. This number defines the design cycle. If the design cycle is high, it becomes difficult to conclude *Assertions*.

Refer to *Design Period* for more information on design cycle.

For the cases in which clock periods are slightly misaligned resulting in a high design cycle, slight modification in clock periods can drastically bring down the design cycle, thereby increasing the chances of formal results to be concluded.

For example, consider the following clock specifications:

```
clock -name clk1 -period 10.5
clock -name clk2 -period 19.5
```

The above clock specifications result in a high design period and design cycle, that is, 136.5 and 37, respectively.

To reduce the design period and design cycle in this case, change the clock periods, as shown below:

```
clock -name clk1 -period 10
clock -name clk2 -period 20
```

Now the design period is reduced to 20 and the design cycle is reduced to 4.

Large Design

Large design does not necessarily imply difficult assertion check. The complexity of a check depends on the complexity of the cone of influence of combined assertion and logic involved in the assertion.

The complexity cannot be measured by gate count or register count only; it also depends on the implicit complexity of the transition logic.

Consequently, a perceived complex check in a large design may be solved faster than a perceived simple check in a smaller design. Only extensive exploration of the assertion and the design (including SpyGlass performance limitations can help in concluding that a check/design is complex and the check cannot be fully analyzed.

Enabling and Disabling Assertions

While debugging CDC issues, you may want to focus on the violations of specific assertions, such as *Passed*, *Failed*, or *Partially Proved*.

In such cases, use *The Property Manager dialog* to select and/or deselect assertions so that these changes are saved in the property file. In the next SpyGlass run, pass that property file to run the selected assertions.

To open the *Property Manager* dialog, select *Tools -> Property Manager -> Clock-Reset* menu option. the following figure shows the *Property Manager* window:

FIGURE 5. The Property Manager dialog

The above dialog shows the names of rules whose violations contain assertion status.

Click on the *Edit* button adjacent to a rule name to enable and/or disable assertions of that rule. For example, the following figure shows the assertion details of the violations reported by the *Ac_cdc01a* rule:

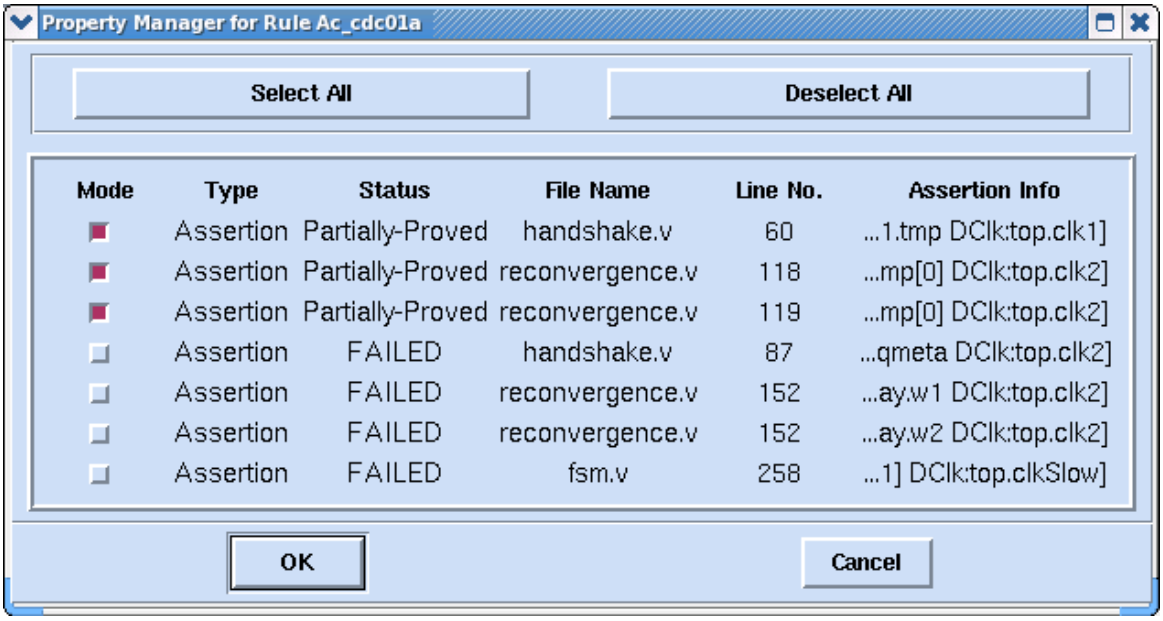


FIGURE 6. Selecting and deselecting assertions

In the above dialog, select and/or deselect assertions and click *OK*. The changes are saved to the property file.

Next time when you run SpyGlass and pass that property file, the violations of the selected assertions are reported.

Property Status Reported during Functional Analysis

The functional analysis rules may report the status of *Properties* as any of the following based on whether a property file could be generated for those properties:

- *Passed*
- *Failed*
- *Partially Proved*
- *Others (Internal-Error)*
- *Others (Constraints-Conflict)*
- *Not-Analyzed*

Passed

A property is considered as passed when SpyGlass is able to generate a property file for that property. For details, refer to *Specifying Properties in a Property File*.

For example, for a property to pass, at least one design state should be reachable in which the property is valid. In all such cases, a sequence of input vectors can be generated (known as witness), which will lead to that particular design state. If it is possible to generate a witness for a property, the property or assertion holds true (or in other words it passes).

Failed

A property is considered as failed when the property file for that property cannot be generated under any circumstances. For details, refer to *Specifying Properties in a Property File*.

Witness

A witness is the input sequence that eventually makes Assertions true while satisfying the given constraints throughout the path.

For example, for a property to pass, at least one design state should be reachable in which the property is valid. In all such cases, a sequence of input vectors can be generated, which will lead to that particular design state. This sequence is known as “Witness”. Therefore, if it is possible to

generate a witness for a property, the property or assertion holds true, that it is passed.

However, if it is not possible to generate a single witness under the given constraints, the property or the assertion fails.

Partially Proved

A property is considered as partially-proved when SpyGlass cannot find the property file for that property and also cannot guarantee that a *Specifying Properties in a Property File* is not possible.

Others (Internal-Error)

A property is considered as Others (Internal-Error) when the value of the design cycle for the clocks in that property cone exceeds the threshold value of 65535.

Others (Constraints-Conflict)

A property is reported as Others (Constraints-Conflict) when constraints in the property cone are not satisfiable.

Not-Analyzed

A property is reported as Not-Analyzed when it is switched off in the property file and not analyzed during functional verification.

Specifying Properties in a Property File

Specify *Properties* in a property file (.prp), and pass this file to SpyGlass by using the *fa_propfile* parameter.

A property file is used for incremental validation purpose. You modify the property file generated in the current run by *Enabling and Disabling Assertions*. For example, you can disable the previously validated assertions and then launch an incremental run in which you pass the modified property file so that the previously validated assertions are not reanalyzed.

Property File Format

A property file has assertion information based on rules in the following format:

```
RuleName: <rule-name>
<selection> <type> <status> <file-name> <line-num> <hier>
   [ <info> ]
...
RuleName: <rule-name>
...
```

Where:

<rule-name>

The name of the SpyGlass rule.

<selection>

Set to `on` when the assertion/constraint is enabled or `off` when the assertion/constraint is disabled.

<type>

The property type is `Assertion` in SpyGlass CDC.

<status>

Indicates the assertion status. The status values are given below:

<status> Value	Indicates that the assertion
PROVED	Proved in the current run
FAILED	Failed in the current run
Partially-Proved	Partially-proved in the current run
Others (Constraints-Conflict)	Constraints conflict in the current run
Others (Internal-Error)	Internal error in the current run
Not-Analyzed	Not analyzed in the current run or a previous run
[PROVED]	Proved in a previous run
[FAILED]	Failed in a previous run
[Partially-Proved]	Partially-proved in a previous run
[Others (Constraints-Conflict)]	Constraints conflict in a previous run
[Others (Internal-Error)]	Internal error in a previous run

NOTE: *The status for constraints is indicated as* `NA`*.*

NOTE: *Internal errors occur in case a design cycle reaches a value higher than 65535.*

NOTE: *The properties with status Partially Proved are reported with selection* `on` *and the properties with all other status are reported with selection* `off`*. You can modify the selection as required for the next run.*

NOTE: *The Others (Constraints-Conflict) status is reported in case of over-constraining found while evaluating any particular assertion.*

<file-name> and <line-num>

The location of the assertion.

<hier>

The design hierarchy where the assertion/constraint was checked.

<info>

Printed for selected rules only. The details are described under the respective rules.

Property File Example

The contents of a typical property file are shown in the following example:

```
RuleName: Ac_cdc01a
off Assertion   FAILED ../src/test.v 559 top.U1
[SFlop:top.U1.Q1 SClk:top.clk1 DFlop:top.U1.Q2
DClk:top.clk2]
...
on Assertion   Partially-Proved ../src/test.v 563 top.U21
[SFlop:top.U21.Q12 SClk:top.clk21 DFlop:top.U21.Q22
DClk:top.clk2]


off Assertion   PROVED ../src/test.v 586 top.U45
[SFlop:top.U45.Q_45 SClk:top.clk_45 DFlop:top.U45.Q_46
DClk:top.clk_46]
...
```

Property File Processing

SpyGlass processes a property file in the following ways:

1. SpyGlass skips checking for a rule when a related assertion is not found in the design. The *Ac_sanity01* rule message is reported for the rule. No information is printed in the property file for such rules. The remaining rules are still processed as applicable.
2. The attributes of assertions in the property file override the attributes of assertions in the HDL instances. For instance, if a design assertion (functional constraint) is specified as a functional constraint (assertion) in the property file, SpyGlass processed it as a functional constraint (assertion).
3. Any assertion/constraint instantiated in a design that is not present in the property file or is explicitly disabled in the property file is ignored for functional analysis.
4. If SpyGlass is run with a property file provided, the new property file generated is based on the existing property file incorporating the results of the current run. Therefore, design properties that are not in the original property file do not appear in the new property file.

Specifying OVL Constraints

The *CDC Verification Rules* support OVL constraints.

Refer to Accellera Standard Open Verification Library (Version Oct 2002) of properties that can be instantiated in a design as a functional constraint. Any OVL constraint inserted in the RTL code by the user is considered as a user-provided functional constraint. Example: `assert_range()`.

You can specify OVL constraints in any of the following ways:

- By embedding OVL constraints directly in an RTL file.
- By specifying OVL constraints in a separate file. For details, refer to the *Separate File OVL Support* section of the *SpyGlass Auto Verify Rules Reference Guide*.

NOTE: *Please note the following points:*

- *SpyGlass CDC solution uses OVL as constraints only. Therefore, from SpyGlass CDC solution perspective, you need to always set the value of* `property_type` *to* `1` *so that OVL constraints are honored by SpyGlass CDC solution.*
- `options` *in OVL 1.0 has been renamed to* `property_type` *in OVL 2.0.*
- *SpyGlass CDC solution honors both OVL 1.0 and OVL 2.0.*

Prerequisites for Using OVL Constraints

If you are using OVL constraints, then while creating custom goals for *CDC Verification Rules*, you should append the following lines in the custom goal file:

```
-sgdc $SPYGLASS_HOME/auxi/policy_data/Clock-Reset/verif.sgdc
-lib accellera_vlog $SPYGLASS_HOME/auxi/ovl/precompile/
$SPYGLASS_PLATFORM/accellera_verilog
-lib accellera $SPYGLASS_HOME/auxi/ovl/precompile/
$SPYGLASS_PLATFORM/accellera_vhdl
```

Why Use OVL Constraints?

For formal analysis, it is recommended that you use a block to chip level approach, that is, you should clean up block-level violations first, and then run formal checks at chip level.

While resolving block-level violations, you might notice some dependencies between certain blocks.

For example, some logic may be generated in the `B1` block and the output of this block feeds the `B2` block. However, when you run formal checks on the `B2` block by treating this block as the top module, ports of this block would become primary inputs. This is incorrect because, as per the design characteristics, it is made sure that these ports were always being fed from output of the B1 block, and therefore, these ports were expected to behave in a particular fashion and were not purely primary inputs.

Therefore, you need to constraint the inputs of this block or else the formal tool would report false failures.

Using set_case_analysis constraint is one of the simplest ways in SpyGlass by which you can constraint your logic.

However, using this constraint has the following limitations:

- You can use this constraint to set a net to either 0 or 1 for the entire run.
- This constraint does take any clock or reset into account.

You can overcome the above limitations by using OVL constraints. OVL, with its set of standard predefined constraints, provides you the flexibility to constraint your design.

The assert_gray Constraint

Use the `assert_gray` constraint to ensure that the value of the specified expression is gray-encoded.

The usage of the `assert_gray` constraint is as follows:

```
// Usage: assert_gray #(severity, width, options, loop)
```

The parameters of the above constraint are described in the following

table:

Parameter	Description
severity	Specifies the severity, such as ERROR and WARNING. This parameter is not used by the SpyGlass CDC solution.
width	Specifies the width of the bus on which gray-encoding check is to be performed.
options	This parameter is used as a constraint or assertion. For SpyGlass CDC solution, it is used as a constraint and, therefore, this parameter should be set to 1.
loop	If this parameter is set to 0 then 00->01->01->11->10 is not considered as gray-encoded. However, if it is set to 1 then the above sequence of transitions is considered as gray-encoded.

The arguments of the `assert_gray` constraint are described in the following table:

Parameter	Description
clock	Specifies the clock on which to the gray-encoding check should be performed
reset	Specifies a reset net
register	Specifies the register on which the gray-encoding check should be performed

The following example shows the usage of the `assert_gray` constraint:

```
assert_gray #(0, 2, 1, 1) gray_inst(clk1,1'b1, reg1);
```

Examples of Using OVL Constraints

The following are some examples of using the OVL constraints:

- Consider that you specify the following OVL constraint on a design on which you want to run the Ac_cdc08 rule:

```
// Usage: assert_gray #(severity, width, options, loop)
assert_gray #(0, 2, 1, 1) gray_inst(clk1,1'b1, reg1); //
```

```
3rd argument -> OVL constraint
```

The above command constrains the `reg1` register so that is gray-encoded. As a result, the *Ac_cdc08* rule reports the status as `PASSED`.

For details on the `assert_gray` constraint, see *The assert_gray Constraint*.

- Consider that you specify the following OVL constraint on a design on which you want to run the Ac_cdc01 rule:

```
//Syntax :      assert_cycle_sequence #(severity, num_cks,
necessary_condition, options)

assert_cycle_sequence #(0, 6, 1, 1)
assumeSequenceZeroOne(clk1, rst, {D == `ZERO, D == `ZERO,
D == `ZERO, D == `ZERO, D == `ZERO, D == `ONE});
```

The above command constrains `D` so that `D` is 0 for five cycles and is 1 for one cycle.

Limitations of Using OVL Constraints

Writing OVL constraints in VHDL is currently not supported.

The workaround for this problem is to specify the required OVL constraints in Verilog in a separate file, and pass that file to the tool by using the following command in a project file:

```
set_option ovl_verilog { <OVL-file> }
```

The following example shows the OVL file used for a VHDL design:

VHDL File (test.vhd)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.Numeric_Std.ALL;
library accellera;
use accellera.ovl_assert.all;

entity gray is
port (
  clk1, clk2, rst : in std_logic;
  in1 : in std_logic_vector(1 downto 0);
  out1 : out std_logic_vector(1 downto 0));
end entity;

architecture gray_arc of gray is
  signal reg1,reg2 : std_logic_vector(1 downto 0);
   :::
   :::
end architecture;
```

OVL File (verilog_ovl_file.ovl)

```
attach_properties gray:gray_arc
begin_ovl
 assert_gray #(0, 2, 1, 1) gray_inst(clk1, 1'b1, reg1);
end_ovl
```

Project File Command

```
read_file -type vhdl test.vhd
set_option ovl_verilog { verilog_ovl_file.ovl }
```

Using Waveform during Functional Analysis

Waveform Viewer is the SpyGlass analysis tool used to perform root cause analysis of a functional bug in a design.

When a property fails (status reported as *Failed*), SpyGlass generates a *Specifying Properties in a Property File* for the failure. This witness is the sequence of events from the initial state of a design/sub-design to the time when a bug appears. This witness can be generated as the set of simulation vectors in the VCD format, *<advance-cdc-rule-name>.<counter>.vcd*. For example, Ac_fifo01.2.vcd (See *Viewing VCD Files*).

Each event or time frame in VCD corresponds to an edge of a clock relevant to the violation (for more detailed description of clock cycles and clock edge count, see *Clock Cycle Count and Sequential Depth*).

When you select a violation, then besides RTL back annotation and schematic highlight, a waveform viewer is launched displaying the VCD content. The presence of a waveform display is indicated by the waveform icon in front of a violation.

Viewing VCD Files

The details of a VCD file and its corresponding display in the *Waveform Viewer* window are described below:

- A VCD file contains values of all the signals lying in the fan-in cone of an assertion, starting from time 0 till the failure depth.
- The exact time stamp of failure is highlighted by a red vertical marker in the *Waveform Viewer* window. (See *Figure 7*)
- By default, only the most relevant signals in the failing assertion are shown in the *Waveform Viewer* window to avoid cluttering of signals.

  You can view other signals in the fan-in cone by performing any of the following actions in the *Waveform Viewer* window:

  - Click on any net, and select its *fan-in / fan-in cone*.
  - Select the *View->Show and Hide Signals* menu option.

    This displays the *Show or Hide Signals* dialog in which you can select the signals you want to view.

The following figure displays the details of a sample VCD file in the *Waveform Viewer* window:

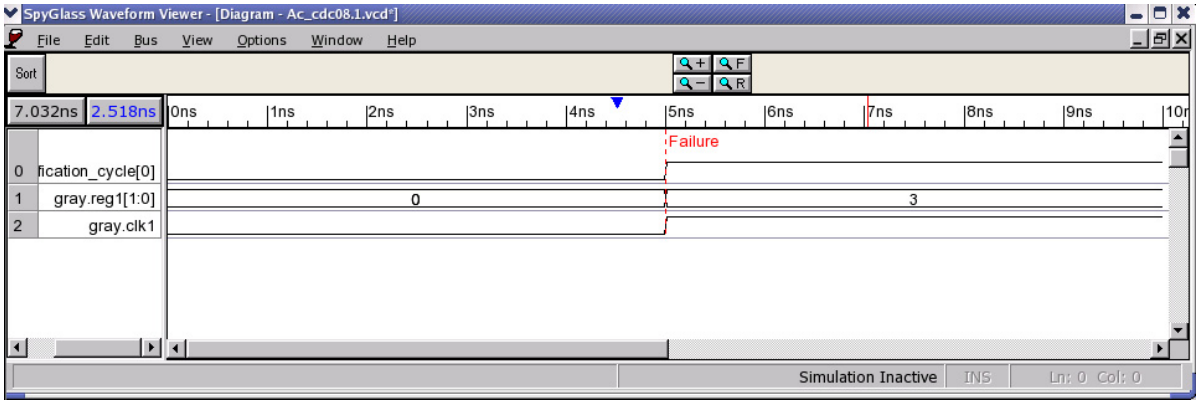


FIGURE 7. Waveform Viewer Window Displaying the Details of a Sample VCD File

In addition to the VCD files generated for a rule, the *adv_cdc_init_seq.vcd* file is also generated corresponding to the *Ac_initstate01* rule, which reports the initial state of a design.

Cross-Probing a Net in Waveform through Schematic

While debugging the waveform of a failed property (violation witness), perform the following steps to cross-probe a net in the waveform from schematic:

1. Open the Waveform Viewer window.
2. Open the incremental schematic.
3. <Ctrl>+click on a net in the schematic while tracing the fan-in of the violating logic.

After performing the above steps, the selected net is cross-probed in waveform viewer if it exists in the VCD generated by the tool. If that net does not impact the violation, SpyGlass reports a message indicating that the net does not exist in VCD.

Handling generated_clock Constructs on Library Pins

Consider the following example of a library pin declaration:

```
generate_clock(o1) {
  clock_pin : o1
  master_pin : in1;
  divide_by : 2
}
```

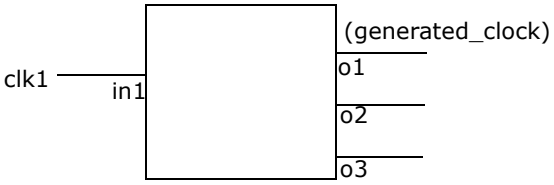
In the above example, the `o1` pin is considered as a generated clock when its source pin `in1` receives a clock.

The domain assignment for such generated clocks happen as per the following rules:

- *One Clock Reaches the Source of Generated Clock*
- *Multiple Clocks Reach the Source of Generated Clock*

One Clock Reaches the Source of Generated Clock

Consider the following figure:



Library-pin definition on which generated clock construct is assigned

```
generated_clock(o1) {
        clock_pin : o1;
        master_pin : in1;
        multiplied_by : 1;
        duty_cycle : 50.000000;
      }
```

FIGURE 1.

In the above figure, one clock (`clk1`) reaches the `in1` pin (master pin of `o1` on which `generated_clock` construct is defined). Therefore, the domain of the `clk1` clock is assigned to the `o1` clock.

Multiple Clocks Reach the Source of Generated Clock

Consider the following figure:

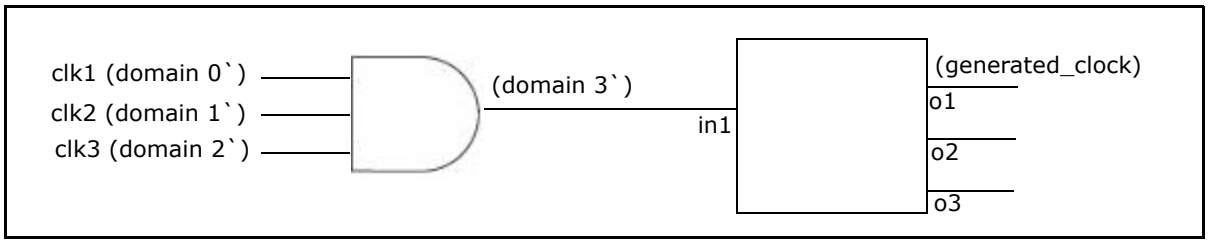


FIGURE 2.

In the above figure, different domain clocks reach to source pin `in1` of the generated clock `o1`.

By default, SpyGlass picks the domain of any one source clock and assigns that domain to the generated clock.

To assign the merged domain (`domain 3‘`) to the generated clock `o1`, set the value of the *clock_reduce_pessimism* parameter to `all_master_clocks`.

Constraints Generated on the Library Pins Defined With generated_clock

The *Setup_clock01* rule generates the following constraints for the library pins on which the `generated_clock` construct is defined:

- *clock*

  This constraint is generated when the *enable_generated_clocks* parameter is set to `no`.

  This constraint is saved in the *cdc_setup_clocks.sgdc* file in the *spyglass_reports/clock-reset* directory.

- *generated_clock*

  This constraint is generated when the *enable_generated_clocks* parameter is set to `yes`.

  This constraint is saved in the *cdc_setup_generated_clocks.sgdc* file in the *spyglass_reports/clock-reset* directory.

Reports and Other Files in SpyGlass CDC

The SpyGlass CDC solution generates the following reports:

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: ● Synchronized and unsynchronized clock domain crossings ● Crossings filtered by using the *cdc_false_path* constraint ● Flip-flops that have their data pin tied to a constant ● Synchronization techniques
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.
SyncRstTree	Shows synchronous set/reset trees in a design.
PortClockMatrix	Provides information on constraints coverage of ports.

Report Name	Description
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.
CDC-report	Provides summary on design, design setup, and verification results.
CDC-summary	Provides a concise summary of the design, design setup, and verification results. It is similar to The CDC Report with some extra information.
CDC-detailed	Provides a concise summary of the design, design setup, and verification results. It is similar to The CDC Report with some extra information.
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.
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.

Report Name	Description
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.
Module_Topology	Shows the dependency of modules in a design.
distributed_time	Shows run time details of SpyGlass CDC rules that are run in parallel on same or different machines.

NOTE: *By default, the width of each column in the reports is dependent on the length of information. You can format the display of information by wrapping up the text by setting the format_report parameter to* `yes`.

Viewing Reports in GUI

The SpyGlass CDC reports are saved in the *spyglass_reports/clock-reset/* directory under the current working directory.

To view a report, perform any of the following actions:

- Select the required report from the *Tools -> Reports* menu option.
- Click  in the *Results* pane under the Analyze Results tab, and select the required report from the *clock-reset* menu.
- Specify the report to be generated by using the `report` command in a project file.
- Specify the report to be generated by using the *Reports Name* option under the *Set Read Options* tab.

Specifying the Report to be Generated through a Project File

Specify the report to be generated by using any the following commands in a project file:

- `set_option report <report-name>`
- `set_goal_option report <report-name>`

where, `<report_name>` can be any of the following report names depending on the report to be generated:

Ac_sync_group_detail	Ac_sync_qualifier	CDC-detailed-report
CDC-report	CDC-summary-report	CKCondensedTree
CKPathInfo	CKSGDCInfo	module_topology
CKTree	Clock-Reset-Detail	Clock-Reset-Summary
CrossingInfo	DeltaDelay-Concise	DeltaDelay-Detailed
DeltaDelay-Summary	DeltaDelay02-Detailed	NoClockCell-Summary
PortClockMatrix	RSTree	SyncRstTree
SynchInfo	cdc_matrix	

For more information on generating reports, refer to *Generating Reports* section in the *Atrenta Console Reference Guide*.

The Clock-Reset-Summary Report

The `Clock-Reset-Summary` report provides an overview of primary setup issues in constraints as well as signals causing most of the clock domain crossing issues.

This report contains the following sections:

Section A: Case Analysis Settings Section	*Section B: Propagated Control Signals Section*
Section C: Top 5 Domain Crossing Sources Section	*Section D: Cases not checked for clock domain crossings Section*
Section E: Inferred Control Signals Section	*Section F: Clock-Reset Matrix Section*
Section G: Black Boxes in Clock Path Section	

Section A: Case Analysis Settings Section

This section of *The Clock-Reset-Summary Report* lists the names and value settings for case analysis-related signals specified by using the *set_case_analysis* constraint.

Section B: Propagated Control Signals Section

This section of *The Clock-Reset-Summary Report* lists information of propagated clock and reset signals in different sections, as described below.

The Propagated Clock Signals Section

This section is available in any of the following conditions:

- You have specified the *clock* constraint.
- You have run the *Clock_info01* rule with the *use_inferred_clocks* parameter set to `yes`).

This section provides the following information:

- Clock name (hierarchical)
- Number of flip-flops driven on positive, negative, and unknown edges
- Whether there is gating in the clock tree
- Total number of signals crossing to other clock domains (if the `Ac_sync_group` group rules are run)
- Number of signals crossing to other clock domains involving black boxes (if the `Ac_sync_group` group rules are run)

The Propagated Reset Signals Section

This section is available in any of the following conditions:

- You have specified the *reset* constraints.
- You run the *Reset_info01* rule with the *use_inferred_resets* parameter set to `yes`)

This section provides the following information:

- Reset name (hierarchical), if the reset is propagated
- Number of synchronous reset used, asynchronous set used and asynchronous reset used, only if *Reset_check04* rule is run.

Section C: Top 5 Domain Crossing Sources Section

This section of *The Clock-Reset-Summary Report* lists top five source signals in a domain crossing.

These signals are arranged in the descending order based on the number of crossings driven by these signals.

This section displays the source signal name, total number of crossings driven by this source signal, and the number of crossings driven by this source signal involving black boxes.

In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, the source column displays the source instance name. Otherwise, the details of the section are the same as in case of RTL designs.

Section D: Cases not checked for clock domain crossings Section

This section of *The Clock-Reset-Summary Report* lists the names of clock nets of unconstrained flip-flops and flip-flops with constant inputs, the number of flip-flops connected to such clock nets, and the name and type of source clock, if any, in the design after the *Clock_info03* rule has run.

See *Section B: Flops with Data pin set to constant value Section* of *The Clock-Reset-Detail Report* for details on flip-flops with constant inputs.

A clock net is assumed to be not participating in clock domain crossing checks under any of the following conditions:

- The clock has not been specified using the *clock* keyword in a SpyGlass Design Constraints file (reported by the *Clock_info03a* rule).
- A flip-flop triggered by the clock net has a constant input (reported by the *Clock_info03b* rule).
- A *set_case_analysis* specification is overriding the clock (reported by the *Clock_info03c* rule).

You should examine this section carefully to ensure all possible problems are resolved. Otherwise, you may overlook potential domain crossing issues.

Section E: Inferred Control Signals Section

This section of *The Clock-Reset-Summary Report* shows the following information:

- Clocks that are found in forward propagation of the clocks specified in the SGDC file or inferred using the *use_inferred_clocks* parameter.

  The inferred clock signal sub section contains the following:

  - Clock name (hierarchical)
  - Clock type (Primary, Derived, Black box, Gated, Undriven, or Virtual)
  - Black box master module and clock pin name (for black box type)
  - Clock drivers (for Derived or Gated types)

- Resets that are found in forward propagation of the resets specified in the SGDC file or inferred using the *use_inferred_resets* parameter.

  The inferred reset signal sub section contains the following:

  - Reset name (hierarchical)
  - Reset type (Primary, Derived, black box, Gated, or Undriven)
  - Reset drivers (for Derived or Gated types)

In addition, the clock_sig.csv and reset_sig.csv files are also generated in the current working directory. These files can be imported into a spreadsheet application, such as Microsoft Excel.

Section F: Clock-Reset Matrix Section

This section of *The Clock-Reset-Summary Report* lists the clock domain-to-reset use matrix as generated by the *Clock_Reset_info01* rule.

Section G: Black Boxes in Clock Path Section

This section of *The Clock-Reset-Summary Report* lists the clock, the black box design unit, and the black box terminal that lies in the path of the clock.

Use this information to set the *assume_path* constraint.

The Clock-Reset-Detail Report

The `Clock-Reset-Detail` report shows the following information:

- Details of synchronized and unsynchronized clock domain crossings based on each source-destination clock pair
- Details of clock domain crossings filtered by using the *cdc_false_path* constraint
- Number of times different synchronization schemes are used in the design

This report contains the following sections:

Section A: Clock Crossings Section	*Section B: Flops with Data pin set to constant value Section*
Section C: Filtered/False Clock Crossings Section	*Section D: Summary of Synchronization Techniques Section*

When the *The Ac_sync_group Rules* are run, data for this report is fetched from the Ac_sync_group rules and all sources per destination are considered by default because the Ac_sync_group rules report all the sources.

Section A: Clock Crossings Section

This section of *The Clock-Reset-Detail Report* shows the following information for each pair of clocks in a design:

- List of unsynchronized crossings reported by the *Ac_unsync01* or *Ac_unsync02* rule
- List of synchronized crossings (along with synchronization scheme) reported by the *Ac_sync01* or *Ac_sync02* rule

For RTL designs, this section shows the output net of flip-flops. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this section shows the flip-flop names.

Section B: Flops with Data pin set to constant value Section

This section of *The Clock-Reset-Detail Report* shows the list of output nets of flip-flops that do not take part in synchronization checks because their data pin is tied to a constant value when the *Clock_info03b* rule is run.

For RTL designs, this section shows the output net of flip-flops. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this section shows flip-flop names.

Section C: Filtered/False Clock Crossings Section

This section of *The Clock-Reset-Detail Report* shows a list of clock-domain crossings that were ignored because the *cdc_false_path* or *cdc_filter_path* constraint was set on these paths.

It also shows the SGDC file name and its line number where these constraints are specified.

Section D: Summary of Synchronization Techniques Section

This section of *The Clock-Reset-Detail Report* shows a list of synchronization schemes, their status (enabled/disabled), and the number of clock crossings detected by each scheme in the design.

This section also lists the total number of synchronized and unsynchronized clock crossings.

NOTE: *The number of clock crossings is calculated after exploring the bus-bits.*

The CKTree Report

The `CKTree` report is generated when you run the *Clock_info02* rule and shows clock trees in a design. It considers any case analysis conditions, if specified.

This report is generated if you specify the *clock* constraints or enable SpyGlass to use the clocks inferred by the *Clock_info01* rule.

The `CKTree` report shows the clock hierarchy in a tree-like format. The leaves in the clock tree can be one of the following:

- A register inferred from the RTL

  The hierarchical name of the registered net is shown along with the polarity of the clock at that point.

NOTE: *In RTL designs, output net of flip-flops is reported. In case of netlist designs, if the report_inst_for_netlist parameter is set to* `yes`*, the flip-flop names are reported. Otherwise, the details of the section are the same as in the case of RTL designs.*

- A black box

  The hierarchical name of the black box instance is annotated with its master name, the name of the pin connected to the clock, the polarity of the clock, and `(black box)` suffix to indicate that this is a black box instance.

- A latch inferred from the RTL

  The hierarchical name of the latched net is shown along with the polarity of the clock and `(Latch)` suffix to indicate that this is latched net. For a latch to be a leaf, the enable pin of the latch should be connected to the clock.

NOTE: *In RTL designs, output net of flip-flops is reported. In case of netlist designs, if the report_inst_for_netlist parameter is set to* `yes`*, the flip-flop names are reported. Otherwise, the details of the section are the same as in the case of RTL designs.*

- A sequential leaf cell

  The hierarchical name of the sequential instance, which is neither known to be a flip-flop nor a latch, is annotated with its master name, the name of the pin connected to the clock, the polarity of the clock and `(Sequential Logic Cell)` suffix to indicate that this is a

sequential leaf cell.

The intermediate nodes of a clock tree can be one of the following:

- A net

  The hierarchical name of the net found while walking along a path from the root of the clock tree to a leaf is shown.

- A register inferred from the RTL

  This is possible when the register is a divide-by-two counter used to divide the clock frequency by two. The hierarchical name of the registered net is shown along with the polarity of the clock at that point.

If the same intermediate node is observed more than once on different paths from root to leaves, then the sub-tree beyond this node is shown only after its first occurrence in the tree. For the other occurrences, a reference to the sub-tree is shown. This rule is recursively followed for the sub-trees also.

An example of a clock tree shown in the `CKTree` report is as follows:

```
ROOT CLOCK: top.in1
'-- (1)top.w5
    '-- (2)top.w3
        |-- (3)top.out2 (-)
        `-- (3)top.w2
            `-- (4)top.out (+)
```

In the report, the Root Clock name is shown in the <Instance.Object_Name> format. Therefore, in designs with multiple levels of pin hierarchies, the Root Clock name reported is very long. To reduce the length of the clock name in such cases, use the *report_clock_tag_names* parameter to report the clock tag names instead of the clock object names in the CKTree.rpt report.

For example, if the `report_clock_tag_names` parameter is not used, a Root Clock is reported as:

```
ROOT CLOCK: top.clk
```

When you use the `report_clock_tag_names` parameter, the Root Clock is reported as:

```
ROOT CLOCK: t1
```

The (+)/(-) suffix to a flip-flop output net name indicates the clock polarity reaching the flip-flop.

When present, the (x) suffix indicates that clock polarity could not be uniquely determined.

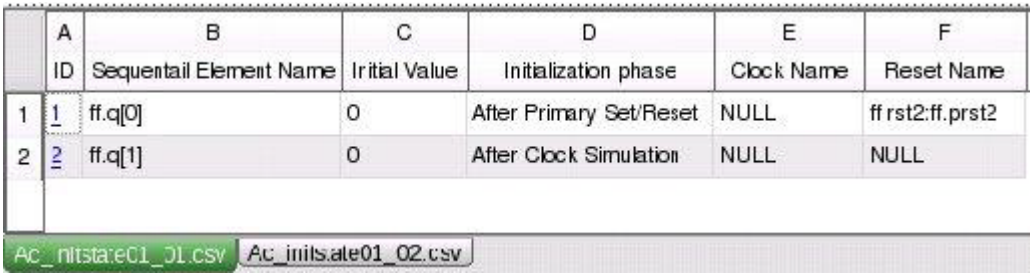
Ac_initstate01 Spreadsheet Report

The *Ac_initstate01* rule generates the Ac_initistate01.csv file that contains details about the initialized and uninitialized sequential elements. This file also displays a non-default value for each pin.

This spreadsheet contains two tabs that show details of initialized and uninitialized sequential elements. By default, the tab for uninitialized sequential elements is selected.

To open this spreadsheet, double-click on the violation of the *Ac_initstate01* rule.

The following figure shows the Ac_initistate01.csv for initialized sequential elements:



	A ID	B Sequentail Element Name	C Initial Value	D Initialization phase	E Clock Name	F Reset Name
1	1	ff.q[0]	0	After Primary Set/Reset	NULL	ff rst2:ff.prst2
2	2	ff.q[1]	0	After Clock Simulation	NULL	NULL

Ac_nitsta:eC1_01.csv | Ac_initsate01_02.csv

FIGURE 1. Spreadsheet report listing initialized sequential elements

The following figure shows the Ac_initistate01.csv for uninitialized sequential elements:

	A	B	C	D	E	F	G	H	I	J	K
	ID	NAME	MODULE	ASYNC RESET	ASYNC SET	CLOCK	ENABLE	DATA	OTHERS	CLOCK NAME	RESET NAME
1	1	ff.c[4.2]	ff	0	0-	0	X	no	NULL	NULL	

Ac_initstate01_01.csv | Ac_initstate01_02.csv

FIGURE 2. Spreadsheet report listing uninitialized sequential elements

If the reset or set value of a pin is 0 or 1, the corresponding schematic shows the path of that value. For example, if the reset and set value is 0, the schematic will display the path.

To open the schematic, click 1 in the *ID* column in the spreadsheet and then click the button. The following figure shows the schematic highlighting the path:

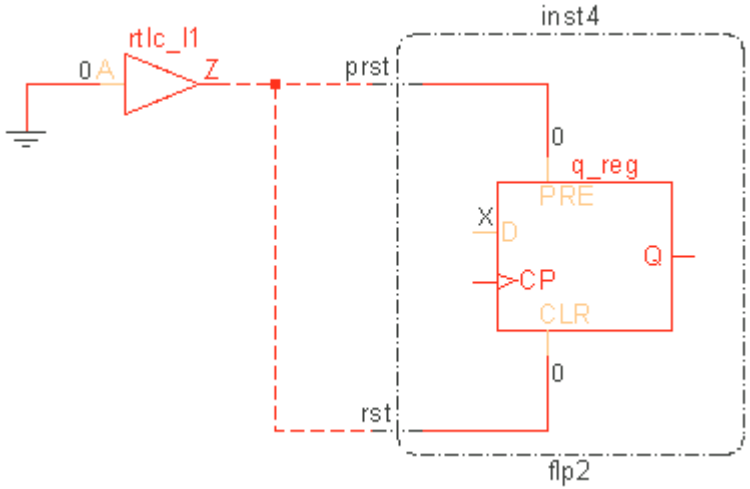


FIGURE 3. Ac_initstate01 Example

Details of the Ac_initstate01 Spreadsheet

Details of various columns of the Ac_initstate01 Spreadsheet are described in the following table:

Column Name	Description
NAME	Specifies a sequential element name. If a sequential cell is a flip-flop, latch, or clock-gating cell, this column displays bus-merged output net name. Else, library instance pins are displayed. This column supports cross-probing to incremental schematic, which highlights a sequential instance along with terminals/pins that have a non-default value in the remaining columns.
MODULE	Specifies the name of a leaf-level parent module of a sequential element.
ASYNC RESET	Specifies an asynchronous reset value. This value can be any of the following: • 1: This value is displayed when the last value on a reset pin of a sequential instance with active low reset is 1 during initial state detection. • 0: This value is displayed when the last value on a reset pin of a sequential instance with an active high reset is 0 during initial state detection. • X: This value is displayed when the last value on a reset pin of a sequential instance is X during initial state detection. • -: This is a default value, and it is displayed when a reset pin does not exist or does not have an unexpected value during initial state detection.
ASYNC SET	This is similar to the ASYNC RESET column with the difference that in this case, the *Ac_initstate01* rule checks asynchronous set pin instead of asynchronous reset pin.
CLOCK	This column displays any of the following values based on different conditions: • -: This value is displayed if there has been toggling from 0 to 1 or from 1 to 0 during initial state detection. • 1: This value is displayed when a clock pin is stuck at 1. • 0: This value is displayed when a clock pin is stuck at 0. • X: This value is displayed when a clock pin is stuck at X. • Comma separated list of *<clkPinName>:<clkTermValue>* is displayed when a sequential element contains multiple clock pins.

Column Name	Description
ENABLE	This column displays any of the following values based on different conditions: ● 1: This value is displayed when the last value on an enable pin of a sequential instance with active low enable is 1 during initial state detection. ● 0: This value is displayed when the last value on an enable pin of a sequential instance with active high enable is 0 during initial state detection. ● X: This value is displayed when the last value on an enable pin of a sequential instance is X during initial state detection. ● -: (Default value): This value is displayed whenever an enable pin does not exist or does not have an unexpected value during initial state detection. The *Ac_initstate01* rule checks for enable pins of a flip-flop and scan enable pins of sequential library cells.
DATA	This column displays any of the following values based on different conditions: ● X: This value is displayed when the last value on a data pin of a sequential instance is X during initial state detection. ● -: (Default value) This value is displayed whenever a data pin does not have an unexpected value during initial state detection. The *Ac_initstate01* rule checks for data and scan data pins.
OTHERS	This column displays any of the following values based on different conditions: ● YES: This value is displayed when a sequential cell is uninitialized and the *Ac_initstate01* rule could not find a possible reason in an asynchronous reset, set, clock, load or data field. ● NO: This value is displayed when any one of the field is a non-default.
CLOCK NAME	Name of the clock that is driving the clock pin of the corresponding sequential element
RESET NAME	Name of the reset that is driving the clock pin of the corresponding sequential element

NOTE: *The value* U *instead of* X *is used for hanging pins in the spreadsheet. The* X *value is a default value for simulation through a design is initialized for simulation in the Ac_initstate01 rule.*

The CKCondensedTree Report

The `CKCondensedTree` report shows condensed clock trees in a design.

This report is similar to *The CKTree Report* except that the `CKCondensedTree` report shows the number of leaves instead of actually listing the leaves.

This report is generated when you run the *Clock_info02* rule and when any of the following conditions hold true:

- If you have specified the *clock* constraints
- If you have enabled SpyGlass to use the clocks inferred by the *Clock_info01* rule.

This report also considers any case analysis conditions, if specified.

The equivalent `CKCondensedTree` report for the example shown in *The CKTree Report* is as follows:

```
ROOT CLOCK: top.in1
`-- (1)top.w5
    `-- (2)top.w3
        |-- (3)Leaf Cells : 1
        `-- (3)top.w2
            `-- (4)Leaf Cells : 1
```

The RSTree Report

The `RSTree` report shows the set/reset trees in a design. It also considers any case analysis conditions, if specified.

This report is generated when you run the *Reset_info02* rule and when any of the following conditions hold true:

- If you have specified the *reset* constraints
- If you have enabled SpyGlass to infer resets by using the *use_inferred_resets* parameter

Types of Leaves in the Reset Tree

The `RSTree` report has the set/reset hierarchy shown in a tree-like format. The leaves on the tree can be one of the following:

Register Inferred from RTL

The hierarchical name of the registered net is shown along with the polarity of the reset at that point.

NOTE: *In RTL designs, output net of flip-flops is reported. In case of netlist designs, if the report_inst_for_netlist parameter is set to* `yes`*, the flip-flop names are reported. Otherwise, the details of the section are the same as in the case of RTL designs.*

Black Box

The hierarchical name of the black box instance is annotated with its master name, the name of the pin connected to the reset, the polarity of the reset, and (BB) suffix to indicate that this is a black box instance.

Latch Inferred from RTL

The hierarchical name of the latched net is shown along with the polarity of the reset and (L) suffix to indicate that this is latched net. For a latch to be a leaf, the data pin of the latch should be connected to the reset.

NOTE: *In RTL designs, output net of flip-flops is reported. In case of netlist designs, if the report_inst_for_netlist parameter is set to* `yes`*, the flip-flop names are reported. Otherwise, the details of the section are the same as in the case of RTL designs.*

Sequential Leaf Cell

The hierarchical name of the sequential instance, which is neither known to be a flip-flop nor a latch, is annotated with its master name, the name of the pin connected to the reset, the polarity of the reset and (SLC) suffix to indicate that this is a sequential leaf cell.

Nodes in the Reset Tree

The intermediate nodes of a set/reset tree can be the hierarchical name of the net that is found while walking along a path from the root of the tree to a leaf.

If the same intermediate node is observed more than once on different paths from root to leaves, then the sub-tree beyond this node is shown only after its first occurrence in the tree. For the other occurrences, a reference to the sub-tree is shown. This rule is recursively followed for the sub-trees also.

Sample RSTree Report

An example of a set/reset tree shown in the `RSTree` report is as follows:

```
=============================================
ROOT RESET: top.rst1
`-- (1)top.instance1.out (+)
=============================================
ROOT RESET: top.rst2
|-- (1)top.instance2.out (+)
`-- (1)top.instance3.out (+)
```

The SyncRstTree Report

The `SyncRstTree` report shows the synchronous set/reset trees in a design. It also considers any case analysis conditions, if specified.

This report is generated when you run the *Ar_syncrstTree* rule and specify the *reset* `-sync` constraints.

The nodes and type of leaves generated in this report are similar to *The RSTree Report*. For details, see *Types of Leaves in the Reset Tree* and *Nodes in the Reset Tree*.

Sample SyncRstTree Report

Following is the sample `SyncRstTree` report:

```
==========================================
ROOT SRESET: top.in3
`-- (1)top.bbRst
    `-- (2)top.t2 (+)
==========================================
ROOT SRESET: top.in4
`-- (1)top.in
    `-- (2)top.inn
        `-- (3)top.temp4
            `-- (4)top.t4 (-)
```

The PortClockMatrix Report

The `PortClockMatrix` report lists primary port names, port attributes, and clock names of flip-flops connected to them (directly or indirectly).

This report is generated by the *Clock_info15* rule, and it displays the following information:

- Input ports with the clocks of connected flip-flops
- Output ports with the clocks of flip-flops generating them
- Inout ports with clocks of connected flip-flops driving or driven by these ports
- Input or output ports related to pure combinational paths are shown without any direct clocks. However, if the *report_indirect_port_clock* parameter is set to `yes`, the following types of ports with indirect clocks (if any) are also reported:
  - Input port that is going to an output port through pure combinational path and is also going to a registered output.
  - Output port that is driven by an input port through pure combinational path and is also coming from a registered input.

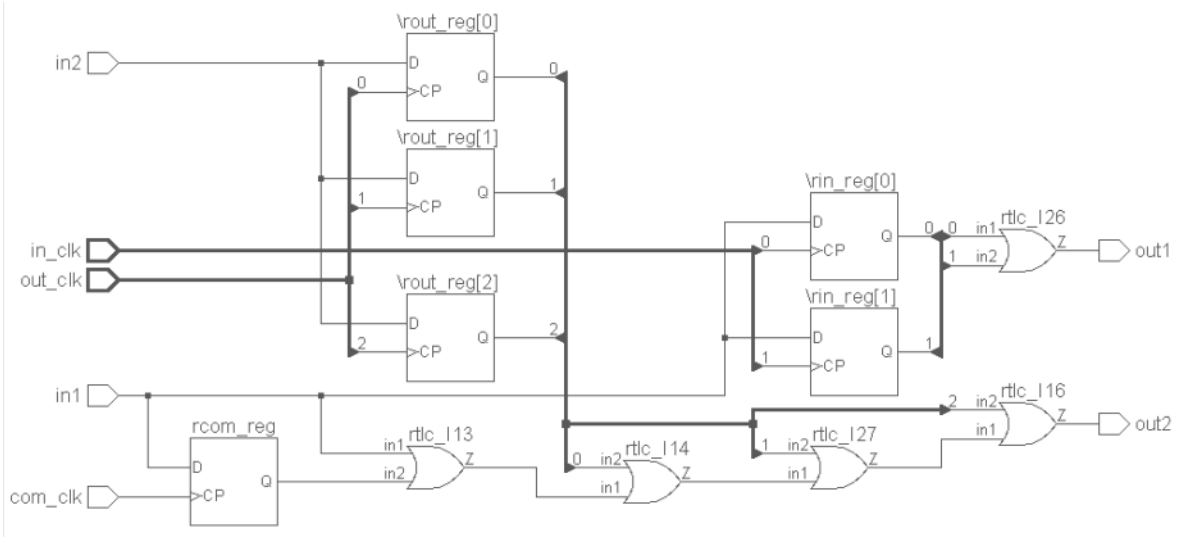


FIGURE 4. Output port driven by an input port through combinational path

These ports are reported with additional clock domains of the registered output/input port, respectively.

- Ports with special attributes, such as unconnected ports, or ports that are constrained by the *set_case_analysis* constraint, clock ports, synchronous or asynchronous reset ports, or blocked ports, are reported without any clocks.

The following table describes various types of attributes reported by `PortClockMatrix` report and their interpretations:

Attribute Type	Port Type	Situation and Interpretation
unconnected	input, output, inout	The port is hanging
const_value:<*x*>	input, output, inout	The port is constrained with set_case_analysis, where *x* is the value of set_case_analysis
clock	input	The port is a clock port. The clock can be user-defined or automatically-inferred.

Attribute Type	Port Type	Situation and Interpretation
sync_reset	input	The port is a synchronous reset port. The reset can be user-defined.
reset	input	The port is an asynchronous reset port. The reset can be user-defined or automatically-inferred.
blocked	input	All the paths from the port are blocked
	input, output, inout	a) Only direct clocks are present: There are no pure combinational paths from this port to any other port. b) Only indirect clocks are present:- There exists only pure combinational from this port to other ports. c) Both are present: Some pure combinational paths + some direct clocks. d) None are present: For output port, all paths driving this port are blocked or have no direct clocks. For input and inout ports, they are either driving a pure combinational circuit or not reaching flip-flop data or load pin.

NOTE: *Only one attribute is reported for a port even though it can satisfy multiple scenarios. The attribute precedence is from top to bottom with respect to port type.*

Sample PortClockMatrix Report

The sample `PortClockMatrix` report is shown below:

```
*****************************************************
*                 Input Port - Clock Matrix         *
*****************************************************
-----------------------------------------------------
S. No. Input Attribute Direct Clocks Indirect Clocks
-----------------------------------------------------
1.     d[5]   unconnected     -                    -
2.     clk1   clock           -                    -
3.     clk2   clock           -                    -
```

```
4.      clk3    clock           -                       -
5.      clk4    clock           -                       -
6.      rst1    reset           -                       -
7.      rst2    sync_reset      -                       -
8.      sr      unconnected     -                       -
9.      d[0]    -               top.clk1, top.clk4      -
10.     d[1]    -               top.clk2                -
11.     d[2]    -               top.clk2, top.clk3      -
12.     d[3]    -               top.clk4                -
13.     d[4]    -               top.clk3                -
14.     tm      -               -                       top.clk3
-------------------------------------------------------------
```

NOTE: *The sample report shown above is generated with the report_indirect_port_clock parameter set to yes.*

The SynchInfo Report

By default, the `SynchInfo` report shows the destinations and synchronizers that are flip-flops, latches, or sequential cells. Set the value of the *dump_inst_type* parameter to `flop` to show only destinations and synchronizers that are flip-flops.

NOTE: *This report is generated when the dump_sync_info parameter is set to* `yes`*,* `detailed`*, or* `detailed_mod`*.*

This `SynchInfo` report consists of the following sections:

Section 1: Synchronized Crossings by 'Conventional Multi-Flop' synchronization	*Section 2: Synchronized Crossings by Synchronizing Cell Techniques*
Section 3: Synchronized Resets by Multi-Flop Synchronization	*Section 4: Synchronized Resets by Reset Synchronizing Cell Technique*
Section 5: Clock domain crossings for quasi-static signals	*Section 6: Synchronized Reset Domain Crossings by Conventional Multi-Flop technique*
Section 7: Synchronized Reset Domain Crossings by Synchronize cell technique	*Section 8: Synchronized Crossings on Reset Path by 'Conventional Multi-Flop' synchronization technique*
Section 9: Synchronized Crossings on Reset Path by 'synchronize cell' technique	

Section 1: Synchronized Crossings by 'Conventional Multi-Flop' synchronization

This section of *The SynchInfo Report* shows the following information:

- Hierarchical instance names of source and destination instances
- The last synchronizer flip-flop for all clock crossings synchronized by the *Conventional Multi-Flop Synchronization Scheme* if the *dump_sync_info* parameter is set to `yes`.

- All the flip-flops in the synchronization chain if the *dump_sync_info* parameter is set to `detailed`.

  The following figure shows the flip-flops in the synchronization chain:

```
*******************************************************************************
        Section: 1 Crossings Synchronized by 'Conventional Multi-Flop' technique
*******************************************************************************
-------------------------------------------------------------------------------
       Source        Destination Flop   Synchronizer Flop(s)  Synchronizer Flop Count
-------------------------------------------------------------------------------
top1.tmp1_reg      top1.tmp2_reg        top1.tmp3_reg,          2
                                        top1.q_reg
top2.tmp1_reg      top2.tmp2_reg        top2.tmp3_reg,          2
                                        top2.q_reg
-------------------------------------------------------------------------------
```

FIGURE 5.

- All the flip-flops in the synchronization chain with the module name appended to each flip-flop name if the *dump_sync_info* parameter is set to `detailed_mod`.

  The following figure shows the flip-flop names appended with their module name:

```
*******************************************************************************
        Section: 1 Crossings Synchronized by 'Conventional Multi-Flop' technique
*******************************************************************************
-------------------------------------------------------------------------------
       Source    Destination Flop     Synchronizer Flop(s)   Synchronizer Flop Count
-------------------------------------------------------------------------------
top1.tmp1_reg  top1.tmp2_reg(RTL_FD)  top1.tmp3_reg(RTL_FD),  2
                                      top1.q_reg(RTL_FD)
top2.tmp1_reg  top2.tmp2_reg(RTL_FD)  top2.tmp3_reg(RTL_FD),  2
                                      top2.q_reg(RTL_FD)
-------------------------------------------------------------------------------
```

FIGURE 6. Module name RTL_FD printed with the flip-flop name

- Number of flip-flops in the synchronizer chain

This section is generated when *The Ac_sync_group Rules* are run, and at least one violation is reported by either of these rules.

Section 2: Synchronized Crossings by Synchronizing Cell Techniques

This section of *The SynchInfo Report* lists all the synchronized crossings synchronized by the synchronized cell technique. These details are generated when the *dump_sync_info* parameter is set to `yes`, `detailed`, or `detailed_mod`.

This section is generated when *The Ac_sync_group Rules* are run and at least one violation is reported by either of these rules.

NOTE: *The dump_inst_type parameter does not affect this section.*

Section 3: Synchronized Resets by Multi-Flop Synchronization

This section of *The SynchInfo Report* shows the number of flip-flops in a synchronizer chain and the name of the synchronizing clock.

In addition, based on the following values specified to the *dump_sync_info* parameter, this section generates the following information:

- `detailed`

  List of all the flip-flops in the synchronization chain in the *Synchronizer Flop(s)* column.

- `detailed_mod`

  - List of all the flip-flops in the synchronization chain in the *Synchronizer Flop(s)* column.
  - The module name appended to the flip-flop name in the *Synchronizer Flop(s)* column.

- `yes`

  Name of the first synchronizer flip-flop.

This section is generated when at least one of the following rules is run and at least one violation is reported by these rules:

Reset_sync02	*Reset_sync03*	*Reset_sync04*	*Reset_check04*
Reset_check10	*Reset_check07*	*Clock_info15*	*Ar_sync01*
Ar_unsync01	*Ar_asyncdeassert01*	*Ar_syncdeassert01*	

Section 4: Synchronized Resets by Reset Synchronizing Cell Technique

This section of *The SynchInfo Report* lists all the reset signals synchronized by reset synchronizing cell technique. These details are generated when the *dump_sync_info* parameter is set to `yes`, `detailed`, or `detailed_mod`.

This section is generated when at least one of the following rules is run and at least one violation is reported by these rules:

Reset_sync02	*Reset_sync03*	*Reset_sync04*	*Reset_check04*
Reset_check10	*Reset_check07*	*Clock_info15*	*Ar_sync01*
Ar_unsync01	*Ar_asyncdeassert01*	*Ar_syncdeassert01*	

NOTE: *The dump_inst_type parameter does not affect this section.*

Section 5: Clock domain crossings for quasi-static signals

This section of *The SynchInfo Report* lists source and destination for all quasi-static crossings.

This section is generated when *The Ac_sync_group Rules* are run and at least one violation is reported by either of these rules.

Section 6: Synchronized Reset Domain Crossings by Conventional Multi-Flop technique

This section of *The SynchInfo Report* is generated only when the *Ar_resetcross01* rule is run. It shows the following information:

- Hierarchical instance names of the source, source reset, destination, destination reset, and synchronizer flip-flops for all the reset crossings synchronized by *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)*.
- The last synchronizer flip-flop for all clock crossings synchronized by the *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)* if the `dump_sync_info` parameter is set to `yes`.
- All the flip-flops in the synchronization chain if the `dump_sync_info` parameter is set to `detailed`.
- Details of the crossings containing source and destination flip-flops and the module names of the flip-flops in the synchronizer chain if the `dump_sync_info` parameter is set to `detailed_mod`.

The example of this section is shown below:

```
*************************************************************************************************
*  Section: 6 Reset Domain Crossings Synchronized by 'Conventional Multi-Flop' technique  *
*************************************************************************************************
-------------------------------------------------------------------------------------------------

S.No. Source        Source Reset  Destination    Destination   Synchronizer     Synchronizer
                                  Flop           Reset         Flop(s)          Flop Count
-------------------------------------------------------------------------------------------------
1.    top.F1.q_reg  "top.r1",     top.F2.q_reg   "top.r2",     "top.F3.q_reg",  4
                    "top.r3",                    "top.r6"      "top.F4.q_reg",
                    "top.r4",                                  "top.F5.q_reg",
                    "top.r5",                                  "top.F6.q_reg"
                    "top.r7",
                    "top.r8"
2.    top.F1.q_reg  "top.r1",     top.F7.q_reg    "top.r2",    "top.F8.q_reg",  2
                    "top.r3",                     "top.r6"     "top.F9.q_reg"
                    "top.r4",
                    "top.r5",
                    "top.r7",
                    "top.r8"
-------------------------------------------------------------------------------------------------
```

FIGURE 7. Section 6 of the SynchInfo Report

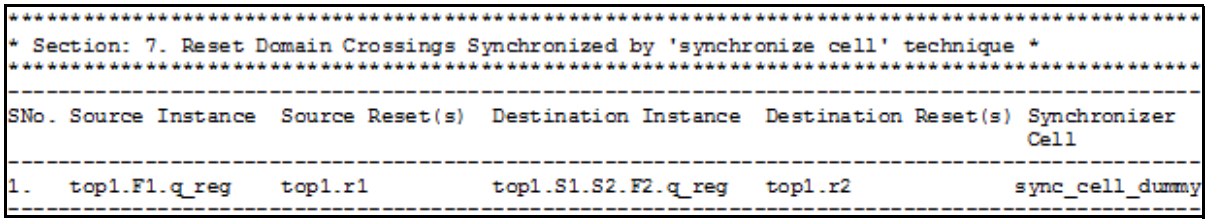
Section 7: Synchronized Reset Domain Crossings by Synchronize cell technique

This section of *The SynchInfo Report* is generated only when the *Ar_resetcross01* rule is run.

It shows the hierarchical instance names of the source, source reset, destination, destination reset, and synchronizer flip-flops for all the reset crossings synchronized by the *For Control path crossings: Synchronizing Cell Synchronization Scheme (for RDC)*.

These details are generated when the `dump_sync_info` parameter is set to `yes`, `detailed`, or `detailed_mod`.

The example of this section is shown below:



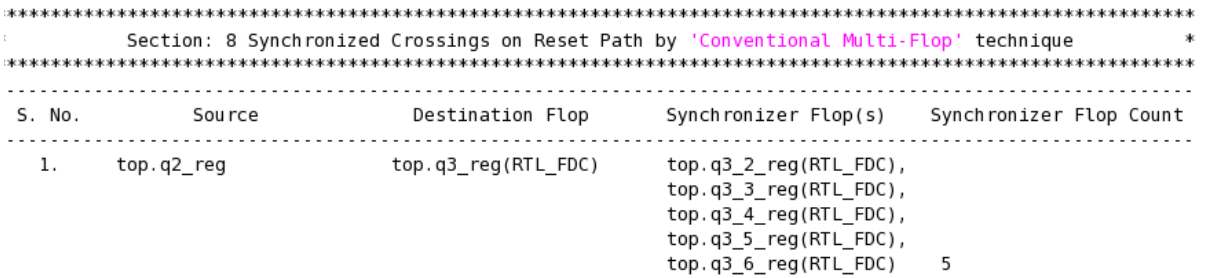
```
*************************************************************************************************************
* Section: 7. Reset Domain Crossings Synchronized by 'synchronize cell' technique *
*************************************************************************************************************
-------------------------------------------------------------------------------------------------------------
SNo. Source Instance  Source Reset(s)  Destination Instance  Destination Reset(s)  Synchronizer
                                                                                   Cell
-------------------------------------------------------------------------------------------------------------
1.   top1.F1.q_reg    top1.r1          top1.S1.S2.F2.q_reg   top1.r2               sync_cell_dummy
-------------------------------------------------------------------------------------------------------------
```

FIGURE 8. Section 7 of the SynchInfo Report

Section 8: Synchronized Crossings on Reset Path by 'Conventional Multi-Flop' synchronization technique

This section of *The SynchInfo Report* shows the following information:

- Hierarchical instance names of source and destination instances
- The last synchronizer flip-flop for all the crossings in reset path synchronized by the Conventional Multi-Flop Synchronization Scheme if the *dump_sync_info* parameter is set to `yes`.
- All the flip-flops in the synchronization chain if the *dump_sync_info* parameter is set to `detailed`. The following figure shows the flip-flop names appended with their module name:



```
*************************************************************************************************************
          Section: 8 Synchronized Crossings on Reset Path by 'Conventional Multi-Flop' technique          *
*************************************************************************************************************
-------------------------------------------------------------------------------------------------------------
 S. No.          Source              Destination Flop       Synchronizer Flop(s)    Synchronizer Flop Count
-------------------------------------------------------------------------------------------------------------
   1.     top.q2_reg              top.q3_reg(RTL_FDC)      top.q3_2_reg(RTL_FDC),
                                                           top.q3_3_reg(RTL_FDC),
                                                           top.q3_4_reg(RTL_FDC),
                                                           top.q3_5_reg(RTL_FDC),
                                                           top.q3_6_reg(RTL_FDC)    5
```

FIGURE 9. Section 8 of the SynchInfo Report

- All the flip-flops in the synchronization chain with the module name appended to each flip-flop name if the *dump_sync_info* parameter is set to `detailed_mod`.

This section is generated when the *Ar_cross_analysis01* rule is run, and at

least one violation is reported by the rule.

Section 9: Synchronized Crossings on Reset Path by 'synchronize cell' technique

This section of *The SynchInfo Report* lists all the synchronized crossings in reset path synchronized by the synchronized cell technique. These details, as shown in the following figure, are generated when the *dump_sync_info* parameter is set to `yes`, `detailed`, or `detailed_mod`.

```
*****************************************************************************************************
*              Section: 9. Synchronized Crossings on Reset Path by 'synchronize cell' technique
*****************************************************************************************************
-----------------------------------------------------------------------------------------------------
  S. No.              Source                     Destination Flop                Synchronizer Cell
-----------------------------------------------------------------------------------------------------
    1.     test.src1_reg                  test.D.out_reg                  DES
    2.     test.src2_reg                  test.D.out_reg                  DES
-----------------------------------------------------------------------------------------------------
```

FIGURE 10. Section 9 of the SynchInfo Report

This section is generated when *Ar_cross_analysis01* rule is run, and at least one violation is reported by the rule.

The CrossingInfo Report

The `CrossingInfo` report lists the source and destination flip-flops for all synchronized and unsynchronized clock crossings.

By default, the `CrossingInfo` report generates destinations and synchronizers that are flip-flops, latches, or sequential cells. Set the value of the *dump_inst_type* parameter to `flop` to dump only those destinations and synchronizers that are flip-flops.

NOTE: *This report is generated when the dump_sync_info parameter is set to* `yes` *or* `detailed`*.*

NOTE: *Refer to The Ac_sync_group_detail Report to view all the synchronized and unsynchronized crossings reported by The Ac_sync_group Rules.*

This report is divided into the following sections:

Section 1: Synchronized Crossings	*Section 2: Unsynchronized Crossings due to Destination Instance Driving Multiple Paths*
Section 3: Unsynchronized Crossings due to Mismatch of Destination and Synchronizer Instance Clock Domains	*Section 4: Unsynchronized Crossings due to Other Reasons*

All the above sections are generated when the `Ac_sync_group` group rules are run, and at least one violation is reported by either of these rules.

Section 1: Synchronized Crossings

This section of *The CrossingInfo Report* shows the following information:

- A list of source and destination instances in synchronized crossings.
- The name of a synchronizing technique.

  These details are generated when the *dump_sync_info* parameter is set to `yes` or `detailed`.

Section 2: Unsynchronized Crossings due to Destination Instance Driving Multiple Paths

This section of *The CrossingInfo Report* shows the following information:

- A list of hierarchical instance names of source and destination instances for the crossings that are unsynchronized because the destination instance drives multiple paths.

  These details are generated when the *dump_sync_info* parameter is set to `yes`.
- A list of potential synchronizer flip-flops when the *dump_sync_info* parameter is set to `detailed`.

Section 3: Unsynchronized Crossings due to Mismatch of Destination and Synchronizer Instance Clock Domains

This section of *The CrossingInfo Report* shows the following information:

- A list of hierarchical instance names of source and destination instances for crossings that are unsynchronized due to the mismatch of clock domains of destination instance and synchronizer.

  These details are generated when the *dump_sync_info* parameter is set to `yes`.
- A list of potential synchronizer flip-flops when the *dump_sync_info* parameter is set to `detailed`.

Section 4: Unsynchronized Crossings due to Other Reasons

This section of *The CrossingInfo Report* lists unsynchronized crossings due to reasons other than those specified in sections 2 and 3.

These details are generated when the *dump_sync_info* parameter is set to `yes` or `detailed`.

The CKPathInfo Report

The `CKPathInfo` report shows a list of cell names instantiated in clock paths.

It lists cell names and the number of instances found for each clock in the design.

The *CKPathInfo* report contains the following sections:

- RTL Cells

  This section lists the RTL cells that are instantiated in the clock paths.

- Other Cells

  This section lists all other cells that are instantiated in the clock paths.

If either of RTL or other type of cells are not present in the clock paths, the corresponding section is displayed in the report. For example, if RTL cells are not instantiated, the CKPathInfo report will display only the Other Cells section.

NOTE: *The CKPathInfo Report is enabled by the run_cells_in_cktree_rules parameter.*

The CKSGDCInfo Report

The CKSGDCInfo Report lists the user-specified SpyGlass design constraints information in the following sections:

- *Section AA: Signals Specified by the cdc_filter_coherency Constraint*
- *Section BB: Signals Specified by the generated_clock Constraint*
- *Section CC: Modules Specified using meta_module Constraint*
- *Section DD: Hierarchical Instances Specified by the meta_inst Constraint*
- *Section EE: Crossings Specified by the reset_filter_path Constraint*
- *Section FF: Signals Specified by the cdc_attribute Constraint*
- *Section HH: Values of the quasi_static_style Constraint*

NOTE: *This report shows single dimensional names for multi-dimensional arrays.*

Section A: Names of Clocks Specified By the clock Constraint

This section of *The CKSGDCInfo Report* shows the list of hierarchical clock names specified by using the *clock* constraint.

Section B: Names of Resets Specified By the reset Constraint

This section of *The CKSGDCInfo Report* shows the list of hierarchical names of all the synchronous and asynchronous reset signals specified by using the *reset* constraint. It also reports its type and active value.

Section C: Port Names on which set_case_analysis Constraint is Set

This section of *The CKSGDCInfo Report* shows the list of ports, hierarchical terminals, and their corresponding constant values specified by using the *set_case_analysis* constraint.

Section D: Valid Reset Ordering Specified by the define_reset_order Constraint

This section of *The CKSGDCInfo Report* shows the list of reset order that determines the flow of data from one reset to another reset as specified by using the *define_reset_order* constraint.

This section is generated only when the `Ar_resetcross01` rule is enabled.

Section E: Modules Specified by the allow_combo_logic Constraint

This section of *The CKSGDCInfo Report* shows the list of modules specified by the *allow_combo_logic* constraint.

If a wildcard is used, SpyGlass reports all the matching modules one by one referring to the same line in the SGDC file.

Section F: Signals Specified by the quasi_static Constraint

This section of *The CKSGDCInfo Report* shows the list of hierarchical signal names specified by the *quasi_static* constraint.

If a wildcard is used, SpyGlass reports all the matching nets one by one referring to the same line in the SGDC file.

Section G: Output Ports Specified by the output_not_used Constraint

This section of *The CKSGDCInfo Report* shows the list of output ports specified by using the *output_not_used* constraint.

Section H: Conventional Multi-Flop Synchronizer Data by the num_flops Constraint

This section of *The CKSGDCInfo Report* shows the list of hierarchical clock names specified by using the *num_flops* constraint.

If only the default value is specified, that value is generated with "default"

in the bracket.

Clock and domain names (whichever is specified in the constraint) appear in the same column.

Section I: Cells Specified by the network_allowed_cells Constraint

This section of *The CKSGDCInfo Report* shows the list of cell names specified by the *network_allowed_cells* constraint.

If a wildcard is used, all the matching cell names, which exist in the specified clock/reset network, are generated in the report. If you have not specified the `-type` argument, all the matching cell names from the entire design are listed.

This section is generated only when the *Clock_Reset_check01* rule is enabled.

Section J: Signals Specified by the qualifier Constraint

This section of *The CKSGDCInfo Report* shows the list of hierarchical names of nets that are inferred by using the *qualifier* constraint.

If wildcard is used, SpyGlass reports all the matching nets one by one with the same from/to clock.

If the *qualifier* constraint is specified as a hierarchical terminal or through module/port format, its connected net is displayed.

Clock and domain names (whichever is specified in the constraint) are generated in the same column.

Section K: Modules Specified by the ip_block Constraint

This section of *The CKSGDCInfo Report* shows the list of all modules and their corresponding instances that are inferred by using the *ip_block* constraint.

This section also shows the total count of crossings waived in each instance. Each crossing is counted separately, when multiple sources converge on the same destination. In addition, vectored signals are also

counted based on the total number of bits involved in the crossings.

Section L: FIFO Specified by the fifo Constraint

This section of *The CKSGDCInfo Report* shows the list of memory and hierarchical read/write pointers that are inferred by the using the *fifo* constraint.

If a memory is specified, its inferred read/write pointers also are shown and vice-versa. In case of partial FIFOs, only inferred memory names are reported even though user has specified read/write pointers in the constraint.

Section M: False Path Specified by the cdc_false_path Constraint

This section of *The CKSGDCInfo Report* shows the list of signals inferred by using the *cdc_false_path* constraint.

If wildcard is used, this section shows all the matching signals.

Section N: Top-Level Ports Specified by the abstract_port Constraint

This section of *The CKSGDCInfo Report* shows the list of top-level ports specified by using the *abstract_port* constraint.

Section O: Top-Level Input Ports Specified by the input Constraint

This section of *The CKSGDCInfo Report* shows the list of all the top-level input ports specified by using the *input* constraint.

Section P: Top-Level Output Ports Specified by the output Constraint

This section of *The CKSGDCInfo Report* shows the list of all the top-level output ports specified by using the *output* constraint.

Section Q: Top-Level Ports Not Specified by Any Constraint

This section of *The CKSGDCInfo Report* shows the list of top-level ports that are not specified by any of the *input*, *output*, *clock*, *reset* (asynchronous reset), *set_case_analysis*, or *abstract_port* constraints.

Section R: Black Box Data Ports Specified by the abstract_port Constraint

This section of *The CKSGDCInfo Report* shows the list of black box ports specified by the *abstract_port* constraint.

Section S: Black Box Ports Specified by the assume_path Constraint

This section of *The CKSGDCInfo Report* shows the list of black box ports specified by the *assume_path* constraint.

Section T: Black Box Ports Specified by the signal_in_domain Constraint

This section of *The CKSGDCInfo Report* shows the list of black box ports specified by the *signal_in_domain* constraint.

Section U: Black Box Data Ports Not Specified by Any Constraint

This section of *The CKSGDCInfo Report* shows the list of black box ports that not specified by any of *assume_path*, *clock*, *reset*, *set_case_analysis*, or *abstract_port* constraints.

Section V: Synchronizer Module/Cell Data Specified by the sync_cell Constraint

This section of *The CKSGDCInfo Report* shows the list of all the *sync_cell* constraints.

Section W: Reset Synchronizers Specified by the reset_synchronizer Constraint

This section of *The CKSGDCInfo Report* shows information, such as synchronizer name, reset source, and synchronizer clock of valid *reset_synchronizer* constraints.

Section X: Isolation Enables Specified by the power_data Constraint

This section of *The CKSGDCInfo Report* shows information about isolation enable signals specified in a UPF file. Information under this section is generated when you run the `Ac_psetup01` rules.

Section Y: Valid Signals Specified by the gray_signals Constraint

This section of *The CKSGDCInfo Report* shows information about valid signals specified using the *gray_signals* constraint.

Section Z: Valid Stop Point for Clocks by the clock_sense Constraint

This section of *The CKSGDCInfo Report* shows valid stop points for clock specified by using the *clock_sense* constraint.

Section AA: Signals Specified by the cdc_filter_coherency Constraint

This section of *The CKSGDCInfo Report* shows signals specified by using the *cdc_filter_coherency* constraint.

Section BB: Signals Specified by the generated_clock Constraint

This section of *The CKSGDCInfo Report* shows signals specified by using the *generated_clock* constraint.

Section CC: Modules Specified using meta_module Constraint

This section of *The CKSGDCInfo Report* shows modules specified by the *meta_module* constraint.

Section DD: Hierarchical Instances Specified by the meta_inst Constraint

This section of *The CKSGDCInfo Report* shows the hierarchical instances specified by the *meta_inst* constraint.

Section EE: Crossings Specified by the reset_filter_path Constraint

This section of *The CKSGDCInfo Report* shows the filtered-out reset crossings

specified *reset_filter_path* constraint.

Section FF: Signals Specified by the cdc_attribute Constraint

This section of *The CKSGDCInfo Report* shows the mutually-exclusive and unrelated signals specified by using the *cdc_attribute* constraint. The following example shows this section of the report:

```
Number of cdc_waive constraints                                          :  2

-------------------------------------------------------------------------------------------
S.No. From Object  To Object  User-Specified   Constraint       Comments   SGDC File:Line
                              Type             Used by

-------------------------------------------------------------------------------------------
1.    test.src1    test.dest  crossing,        crossing,glitch     -       test.sgdc:4
                              validation,
                              glitch


2.    test.src2    test.dest  crossing,        crossing,glitch     -       test.sgdc:4
                              validation,
                              glitch
```

FIGURE 11.

Section HH: Values of the quasi_static_style Constraint

This section of *The CKSGDCInfo Report* shows the values of the various options of the *quasi_static_style* constraint. The following example shows this section of the report:

HH. Values for the options of quasi_static_style constraint

S. No.	root name	min_seq_ fanouts	min_domain_ fanouts	check_all_ signals	report_full_ count	names
1.	test	10	1	false	false	-

FIGURE 12. Values of the quasi_static_style Constraint

The CDC Report

The CDC Report provides a concise summary of the design, design setup, and verification results in the following sections:

Section A	*Section B*	*Section C*	*Section D*	*Section E*
Section F	*Section G*	*Section H*	*Section I*	

Along with this report, SpyGlass generates some additional files. For details, see *Files Generated with the CDC Report*.

Section A

This section of *The CDC Report* shows the run information, such as:

- Performance information, such as total run time and peak memory of SpyGlass run.
- User-defined parameter values.

Section B

This section of *The CDC Report* lists the design information, such as:

- Count of flat instances.
- Total number of black boxes.

Section C

This section of *The CDC Report* displays the design setup information, such as:

- Count of different types of clocks, such as user-defined clocks, user-defined generated clocks, virtual clocks, black box clocks, and inferred clocks.
- Count of different types of resets, such as user-defined resets and black box resets.

- Count of different types of user-specified constraints.

Section D

This section of *The CDC Report* shows the setup audit information that you should review to sign off SpyGlass CDC setup.

This section shows the count of messages generated by rules that ensure the following:

- Registers are properly clocked and reset
- No clocks are defined in the fan-out of other clocks.
- All input ports are associated with clocks or these ports are constrained.

You should not have any messages reported for the above situations. Therefore, all the counts reported in this section should ideally be zero to ensure a setup sign off.

Section E

This section of *The CDC Report* shows the information related to structural SpyGlass-CDC violations. For example, it shows the following:

- Count of messages generated by the rules that ensure that there are no metastability problems across clock domains.
- Count of synchronized and unsynchronized clock crossings in a design.

For signing off SpyGlass CDC verification, all the counts (other than the count of synchronized clock crossings) reported in this section should ideally be zero.

Section F

This section of *The CDC Report* shows the count of different types of qualifiers used to synchronize data crossings.

The count of qualifiers appear regardless of the data they synchronize.

Section G

This section of *The CDC Report* shows the count of message of the rules that ensure the following:

- There are no issues with correlated signals crossing clock domains.
- There is no data loss for signals crossing a faster clock domain to a slower clock domain.
- Complex synchronizers, such as FIFO and Handshake are working properly.

Section H

This section of *The CDC Report* shows the count of messages of the rules that ensure the following:

- Resets are properly synchronized before they assert any register in the design.
- Resets are deasserted properly.

Section I

This section of *The CDC Report* shows the count of violations of different types of mismatches, such as *Clock Domain Mismatch* and *Reset Mismatch* reported during abstract-view validation.

For information on these violations, see *Rule-based spreadsheet - Ac_abstract_validation01.csv* generated by the *Ac_abstract_validation01* rule.

Files Generated with the CDC Report

Along with *The CDC Report*, SpyGlass also generates comma-separated data files (*<rule-name>.csv)* for the following rules:

Clock_info03a	*Clock_info05*	*Clock_info05a*	*Clock_info05b*
Clock_check03	*Clock_check04*	*Clock_check05*	*Clock_check07*
Ac_unsync01	*Ac_unsync02*	*Clock_sync03*	*Clock_sync05*
Clock_sync06	*Ac_cdc08*	*Clock_sync09*	*Ac_sanity04*

Ac_sync02	*Ac_sync01*	*Ac_clockperiod01*	*Ac_clockperiod02*
Ac_clockperiod03	*Ac_resetvalue01*		

The CDC-Summary-Report

This report provides a concise summary of the design, design setup, and verification results. It is similar to *The CDC Report* with some extra information.

The following sections describe the extra information displayed with respect to *The CDC Report*:

Section A	*Section B*	*Section C*	*Section D*
Section E	*Section F*	*Section G*	*Section H*
Section I			

Section A

This section of *The CDC-Summary-Report* shows the following information in addition to the information in *Section A* of *The CDC Report*:

- Goals run
- The run type, such as save-restore
- The machine used

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section A. Run Information:
=========================================================================
Goal(s)                       : Setup, Struct, Functional
Run Type                      : save-restore=no
Total Time (in sec)           : 14
Memory used (in KB)           : 1030980
Machine used                  : ugo
```

Section B

This section of *The CDC-Summary-Report* shows the following information in

addition to the information in *Section B* of *The CDC Report*:

- Number of registers
- Number of primary inputs
- Number of primary outputs
- Number of IO pad cells

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section B. Design Information:
=========================================================================
Number of flat Instances in design     : 34
Number of registers in design          : 18
Number of primary inputs               : 17
Number of primary outputs              : 11
Number of black-boxes (AnalyzeBBox)     : 3
Black-boxes without definition or synthesis error (ErrorAnalyzeBBox) : 1
Number of IO Pad cells                                               : 1
```

Section C

This section of *The CDC-Summary-Report* shows the following information in addition to the information in *Section C* of *The CDC Report*:

- Number of abstract views
- Number of user-defined synchronous resets
- Number of each constraint reported by *The CKSGDCInfo Report*

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section C. Setup Information:
=========================================================================
User defined Parameter values                              : 5
Number of abstract models                                  : 3
Number of total clocks                                     : 5
```

```
Number of user defined clocks                                    : 4
Number of user defined generated clocks                          : 0
Number of user defined virtual clocks                            : 1
Number of inferred clocks                                        : 0
```

Section D

This section of *The CDC-Summary-Report* shows the following information in addition to the information in *Section D* of *The CDC Report*:

- Number of library cells that are not fully constrained

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section D. Setup Errors:
=========================================================================
Number of registers missing clock definition (Clock_info03a)           : 0
Number of clock convergences on unselected MUX(Clock_info05)           : 0
Number of clock convergences at non-mux gate (Clock_info05b)         : 0
Number of multi-definition on clock (Clock_check07)                     : 0
Number of registers missing asynchronous reset definition(Reset_info09a): 1

Number of Constant functional flops in synchronous reset de-assert
mode(Ar_syncrst_setupcheck01)                                          : 0
Number of primary ports not associated with clocks (Setup_port01)
                                                                 : 3(17.65 %)
Number of black-boxes not fully constrained (Setup_blackbox01)         : 1
Number of library-cells not fully constrained (Setup_library01)        : 0

For details, refer to following files:
--------------------------------------
 - Setup_blackbox01.csv
```

Section E

This section of *The CDC-Summary-Report* shows the following information in addition to the information in *Section E* of *The CDC Report*:

Total number of clock domain crossings (synchronized and unsynchronized).

The following example shows this section of *The CDC-Summary-Report*:

```
================================================================================
Section E. CDC Analysis and Verification:
================================================================================
Number of Clock Domain Crossings                                        : 3
Number of Unsynchronized Crossings                                      : 2
Number of scalar (Single-bit) crossings (Ac_unsync01)                  : 2
Number of vector (Multi-bit) crossings (Ac_unsync02)                   : 0
Number of Synchronized Crossings                                        : 1
Number of Scalar (Single-bit) Crossings (Ac_sync01)                    : 1
Number of Vector (Multi-bit) Crossings (Ac_sync02)                     : 0
Glitches in synchronized control crossing paths (Ac_glitch03)           : 0

For more details, refer to following files:
-------------------------------------
   - Ac_sync01.csv
   - Ac_unsync01.csv
```

Section F

This section of *The CDC-Summary-Report* is same as *Section F* of *The CDC Report*.

The following example shows this section of *The CDC-Summary-Report*:

```
================================================================================
Section F. Asynchronous Reset Analysis and Verification:
================================================================================
Asynchronous resets without reset synchronizers (Ar_unsync01)           : 1
Asynchronous resets with reset synchronizers (Ar_sync01)                : 0
Asynchronous resets with asynchronous deassertion (Ar_asyncdeassert01: 1
Asynchronous resets with synchronous deassertion (Ar_syncdeassert01) : 0
Asynchronous resets synchronized multiple times in same domain
(Reset_sync04)                                                          : 0
Asynchronous resets generated from a different domain (Reset_sync02) : 0
Invalid reset ordering between same domain registers (Ac_resetcross01)
```

```
                                                                        : 0
For more details, refer to following files:
  ------------------------------------
  - Ar_unsync01.csv
  - Ar_asyncdeassert01.csv
```

Section G

This section of *The CDC-Summary-Report* is same as *Section G* of *The CDC Report*.

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section G. Hierarchical Verification : Abstraction Validation:
=========================================================================
Abstract model mismatch with top (Ac_abstract_validation01)              : 4
Case Analysis Mismatch                                                   : 4
No of blocks(block instances) with abstract model mismatches             : 1(3)
Clock mapping of an abstracted instance (SGDC_abstract_mapping01)        : 0
For more details, refer to following files:
  ------------------------------------
   - Ac_abstract_validation01.csv
```

Section H

This section of *The CDC-Summary-Report* shows the following information in addition to the information in *Section H* of *The CDC Report*:

- Number of clocks with the delta delay mismatches
- Number of clocks with the shoot-through problem

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section H. DeltaDelay Verification:
=========================================================================
Number of clocks with delta delay mismatches (DeltaDelay01)              : 0
Number of clocks with shoot-through problem (DeltaDelay02)               : 0
=========================================================================
```

Section I

This section of *The CDC-Summary-Report* shows information about the glitch-free multiplexers in a design.

The following example shows this section of *The CDC-Summary-Report*:

```
=========================================================================
Section I. Glitch Free Cell:
=========================================================================
-------------------------------------------------------------------------
S. No.     Glitch Free Type              Constraint on Mux     Mux Name
                                         Select Pin Specified
-------------------------------------------------------------------------
1.         Recirculating                 N.A.                top.rtlc_I15
2.         Clock Switching               No                  top.rtlc_I13
3.         Reset synchronization bypass  Yes                 top.rtlc_I17
4.        Reset synchronization bypass   No                  top.rtlc_I19
-------------------------------------------------------------------------
=========================================================================
```

The CDC-Detailed-Report

This report provides a concise summary of the design, design setup, and verification results. It is similar to *The CDC Report* with some extra information.

SpyGlass CDC provides the *RTL Results Difference Utility* that you can use to identify the differences in the results generated by two SpyGlass CDC runs. This utility uses the *The CDC-Detailed-Report* to show the difference between the two runs. For details on this utility, refer the RTL Results Difference Utility section.

The following sections describe the extra information displayed with respect to *The CDC Report*:

Section A	*Section B*	*Section C*	*Section D*
Section E	*Section F*	*Section G*	*Section H*
Section J	*Section K*		

Section A

This section of *The CDC-Detailed-Report* shows the run information, such as:

- Types of goals run
- The type of run, such as save-restore
- Total time taken to run the design
- Total peak memory
- User-specified parameters
- Name of the machine on which goals are run

The following example shows this section of *The CDC-Detailed-Report*:

```
============================================================
Section A. Run Information:
============================================================
Goal(s)                    : Setup, Struct, Functional
Run Type                   : save-restore=no
Total Time (in sec)        : 14
```

```
Memory used (in KB)          : 1030980
Machine used                 : ugo
```

Section B

This section of *The CDC-Detailed-Report* shows the design information, such as:

- Detailed information on flat instances, flip-flops, primary inputs and outputs in the design
- Detailed information on black boxes and IO pad cells

The following example shows this section of *The CDC-Detailed-Report*:

```
============================================================
Section B. Design Information:
============================================================
Number of black-boxes             : 5
------------------------------------------------
S. No.   Black-box name   Present in Clock Path
------------------------------------------------
1.      test.b1           Yes
2.      test.b2           Yes
3.      test.absxx        No
4.      test.abs2         No
5.      test.abs3         No
------------------------------------------------
```

Section C

This section of *The CDC-Detailed-Report* shows the setup information, such as:

- Total clocks and resets in the design
- User-defined parameters and constraints

Section D

This section of *The CDC-Detailed-Report* shows the setup errors, such as clock, reset, and constraint-related design violations.

The following example shows this section of *The CDC-Detailed-Report*:

```
============================================================
Section D. Setup Errors:
============================================================
Number of primary port constraints data associated with clocks
(Setup_port01) : 10
-------------------------------------------------------------------------
S. No. Port Name  Port Direction  Status                    Constraint(s)
-------------------------------------------------------------------------
1.     "reset"    input           partially constrained     "reset"
                                  (suggested constraint
                                  abstract_port)

2.     "in1"      input           fully constrained         "input"
-------------------------------------------------------------------------
Number of black-boxes not fully constrained (Setup_blackbox01): 1
-------------------------------------------------------------------------
S. No. ID   BLACK-BOX   NUMBER OF   PERCENTAGE   UNCONSTRAINED    WAIVED
            NAME        PINS        OF PINS
-------------------------------------------------------------------------
1.     1C     BBOX        6           0.00                          no
-------------------------------------------------------------------------
```

Section E

This section of *The CDC-Detailed-Report* shows the information related to SpyGlass CDC analysis and verification.

This information includes the results of the rules, which ensure that:

- There is no metastability problem across clock domains.
- All the crossings involving quasi-static signals have been analyzed.
- There is no issue with the correlated signals crossing clock domains.

- There is no data loss for the signal crossing a fast clock domain to a slow clock domain.

The following example shows this section of *The CDC-Detailed-Report*:

```
============================================================
Section E. CDC Analysis and Verification:
============================================================
Number of Clock Domain Crossings              : 3
a. Number of unsynchronized crossings         : 2
   i. Number of scalar(Single-Bit) crossings (Ac_unsync01 : 2
------------------------------------------------------------

S.No. Dest.    Dest.         Source  Source     Failure       Sync.Scheme
      Name     Clock Names   Name    Clock      Reason
                                     Names

-------------------------------------------------------------------------
1.    t.out3   "t.clk1"      t.dstFF  "t.clk2"  Qualifier not found    N.A.
2.    t.FF2    "t.clk2"      t.FF1    "t.clk1"  Qualifier not found    N.A.
-------------------------------------------------------------------------
```

The report also includes the waived violations of the rules in the `Waived Msg` column.

Section F

This section of *The CDC-Detailed-Report* shows the information related to SpyGlass asynchronous reset analysis and verification.

This information includes the results of the rules, which ensure that:

- Asynchronous resets have been properly synchronized in each domain.
- Asynchronous resets have been de-asserted properly.
- All the registers have been properly reset.

The following example shows this section of *The CDC-Detailed-Report*:

```
=====================================================================
Section F. Asynchronous Reset Analysis and Verification:
=====================================================================
Asynchronous resets without reset synchronizers (Ar_unsync01)    : 1
---------------------------------------------------------------------
```

```
S.No. ID RESET  PINTYPE  INSTANCE CLOCK(s)  REASON     WAIVED
-------------------------------------------------------------------------
1.    18 t.rst  clear    t.rstF2  "t.clk2"  "Missing synchronizer"   no
-------------------------------------------------------------------------
...
...
 Asynchronous resets with asynchronous deassertion (Ar_asyncdeassert01)
: 1
-------------------------------------------------------------------------
S.No. ID  RESET   PINTYPE  INSTANCE  CLOCK(s)  REASON              WAIVED
-------------------------------------------------------------------------
1.    19  t.rst   clear    t.rstF2   "t.clk2"  "Improper deassertion"   no
-------------------------------------------------------------------------
```

Section G

This section of *The CDC-Detailed-Report* shows the information related to SpyGlass CDC verification (abstract validation).

This information includes the results of the rules, which ensure that all the abstracted blocks are in sync with the top-level constraints.

The following example shows this section of *The CDC-Detailed-Report*:

```
=========================================================================
Section G. Hierarchical Verification : Abstraction Validation:
=========================================================================
Number of abstract model mismatch with top (Ac_abstract_validation01 : 3
-------------------------------------------------------------------------
-------------------------------------------------------------------------
S.No. ID  Block-Name  Block     Problem       Count  WAIVED   CSV FILE
                      Instance  Type
                      Name
-------------------------------------------------------------------------
-------------------------------------------------------------------------
1.     26   ABS       test.abs  Case Analysis  2      no      abs/eg.csv
                                 Mismatch
-------------------------------------------------------------------------
```

Section H

This section of *The CDC-Detailed-Report* shows the information related to delta-delay verification.

This information includes the results of the rules, which verify delta-clock delays on flip-flops and latches.

The following example shows this section of *The CDC-Detailed-Report*:

```
=========================================================================
Section H. DeltaDelay Verification:
=========================================================================
Delta Delay Concise Report(DeltaDelay01)
Clock --> test.clk1:
*************************************************************************
*     Delay        Number of Flops             Number of Latches         *
*************************************************************************
         0           4                             0
*************************************************************************|

Clock --> test.clk2:
*************************************************************************
*     Delay        Number of Flops             Number of Latches         *
*************************************************************************
         0           6                             0
*************************************************************************
```

Section J

This section of *The CDC-Detailed-Report* shows the signals involved in convergence issues (reported by *Ac_conv01*, *Ac_conv02*, and *Ac_conv03*) but suppressed by the *cdc_attribute* constraint.

The following example shows this section of *The CDC-Detailed-Report*:

```
=========================================================================
Section J. Filtered signal convergence:
=========================================================================
-------------------------------------------------------------------------
S. No.     Unrelated/Exclusive Signals      Type          SGDC File:Line
```

```
-------------------------------------------------------------------------
1.          top.f5                              Ac_conv02     test.sgdc:6
            top.f3
2.          top.dest[1:0]                       Ac_conv02     test.sgdc:7
3.          top.f8                              Ac_conv01     test.sgdc:6
            top.f5
            top.f3
4.          top.dest[1:0]                       Ac_conv01     test.sgdc:7
            top.f8
-------------------------------------------------------------------------
```

Section K

This section of *The CDC-Detailed-Report* shows the glitch-free multiplexers found in a design.

The following example shows this section of *The CDC-Detailed-Report*:

```
=========================================================================
Section K. Glitch Free Cell:
=========================================================================
-------------------------------------------------------------------------
S. No.     Glitch Free Type              Constraint on Mux     Mux Name
                                         Select Pin Specified
-------------------------------------------------------------------------
1.         Recirculating                 N.A.                top.rtlc_I15
2.         Clock Switching               No                  top.rtlc_I13
3.         Reset synchronization bypass  Yes                 top.rtlc_I17
4.     Reset synchronization bypass   No                     top.rtlc_I19
-------------------------------------------------------------------------
=========================================================================
```

The Advanced CDC Report

The Advanced CDC report, adv_cdc.rpt, provides information to help you analyze the cause of a bug or to gather functional analysis statistics.

This report is automatically generated when you run any of the *CDC Verification Rules*.

The Advanced CDC report contains the following sections:

Section A: Clock Information	*Section B: Reset Information*
Section C: Set Case Analysis Settings	*Section D: Initial State of the Design*
Section E: Results Summary (Current)	*Section F: Results Summary (Cumulative)*
Section G: Assertion Details	

Section A: Clock Information

This section of *The Advanced CDC Report* shows a summary of clock definitions as reported by the *Propagate_Clocks* rule. The *Design Period* is also reported in this section.

Each clock is reported in the following format:

```
<clk-name>: <clk-period>; <clk-source>; <clk-edge>;<edge-list>; <num-flops-posedge>; <num-flops-negedge>;
```

Where:

<clk-name>

The clock name.

<clk-period>

The clock period (specified using the `-period` argument of the `clock` constraint).

<clk-source>

SGDC is printed for clocks that have been specified in a SpyGlass Design

Constraints file using the `clock` constraint. For automatically-inferred clocks, `Auto-Inferred` is printed.

<clk-edge>

The starting clock edge (`Rising` or `Falling`).

<edge-list>

The clock edge list (specified using the `-edge` argument of the `clock` constraint).

<num-flops-posedge>

The number of flip-flops triggered by the clock on the positive edge.

<num-flops-negedge>

The number of flip-flops triggered by the clock on the negative edge.

User-specified clocks and automatically-inferred clocks are reported under separate headings. A separate file named *adv_cdc.reg* reports controlling clocks for individual registers. See *The Register Info Report* for details of the *adv_cdc.reg* file.

This section also contains the *Design Period* in terms of number of fastest clock cycles and non-overlapping clock edges.

Section B: Reset Information

This section of *The Advanced CDC Report* shows resets that are used in initial state detection and for functional analysis.

User-specified resets and automatically-inferred resets are reported under separate headings.

Each reset is reported in the following format:

```
<reset-name> ; Active High | Active Low : [soft reset]
```

All the resets are assumed to be hard resets unless marked as soft resets. All hard resets are deactivated during functional analysis. The soft resets are used only in initial state search and are not deactivated during functional analysis. A separate file named adv_cdc.reg reports controlling resets for individual registers. See *The Register Info Report* for details of the

adv_cdc.reg file.

Section C: Set Case Analysis Settings

This section of *The Advanced CDC Report* shows a summary of *set_case_analysis* constraints that have been applied on the design through the SGDC file.

Each *set_case_analysis* constraint is reported in the following format:

```
<net-name> ; <net-value>
```

Where:

<net-name>

The net name specified by the *set_case_analysis* constraint in an SGDC file. For scalar nets, the exact name as specified in the SGDC is shown. For vector nets, the hierarchical name of the net is shown.

<net-value>

The specified value of the net.

Section D: Initial State of the Design

This section of *The Advanced CDC Report* lists the initial state statistics of the design along with the reset percentage.

This section reports a summary of initial state as reported in the *Ac_initstate01* rule.

A separate file named *adv_cdc.reg* reports initial state assignments for individual registers. See *The Register Info Report* for details of the *adv_cdc.reg* file.

Section E: Results Summary (Current)

This section of *The Advanced CDC Report* lists the statistics of the assertions formed for each rule.

The analysis statistics are reported as listed below:

- General analysis statistics, such as:
  - Number of properties passed
  - Number of properties failed
  - Number of properties partially proved
  - Number of properties that are not analyzed

    On running SpyGlass with the *fa_audit* parameter set to `yes`, functional analysis is skipped and all properties are marked as *Not-Analyzed*.
  - Number of the following types of assertions:
    - Number of assertions that encountered conflicting constraints in its fan-in cone.
    - Number of assertions that were not analyzed because design cycle was huge or an NTP conversion did not happen properly.
  - Total number of properties in the design
- Rules table

  A table of the following format is reported for current run:

Rule Name	Passed	Failed	Partially Proved (Average Depth)	Not Analyzed	Others	Total
Ac_cdc01a	4	3	0	0	0	7
Ac_cdc01b	0	1	0	0	0	1
Ac_cdc01c	0	0	3(65)	4	0	7
Ac_cdc08	2	1	0	0	1	4
Ac_fifo01	1	2	1	0	0	4
Ac_handshake01	0	0	0	1	0	1
Ac_handshake02	0	0	1	0	0	1
TOTAL	7	7	5	5	1	25

The *Passed* column shows the number of properties that are proven TRUE. The *Failed* column shows the number of properties that are proven FALSE. The *Partially Proved* column shows the number of properties that are not concluded. The *Average Depth* is the average

depth of all the properties that are not concluded.

Section F: Results Summary (Cumulative)

This section of *The Advanced CDC Report* lists the summary of the cumulative set of assertions formed in the current run and the information of earlier runs in the property file. This section is generated when you specify a property file using the *fa_propfile* parameter.

A rules table is also printed in this section. This table has the cumulative results for the current run and the earlier runs' information in the property file.

Section G: Assertion Details

This section of *The Advanced CDC Report* lists the assertion details for each rule. Based on the type of assertion, a detailed report is generated as follows:

- For FAILED assertion:

```
RuleName: <rule-name>
(<module-name>), <assertion-details>, <file-name>,
<line-number> (<VCD-file-name>):
FAILED through depth <cycle-depth> (<depth>)
```

- For Partially-Proved assertion:

```
RuleName: <rule-name>
(<module-name>), <assertion-details>, <file-name>,
<line-number> (<VCD-file-name>):
Partially-Proved through depth <cycle-depth> (<depth>)
[Flop-Count:<flop-count>, Design-Cycle:<design-cycle>]
```

- For PROVED assertion:

```
RuleName: <rule-name>
(<module-name>), <assertion-details>, <file-name>,
<line-number>: PROVED
```

- For Internal-Error assertion:

```
RuleName: <rule-name>
```

```
(<module-name>), <assertion-details>, <file-name>,
<line-number>: Others(Internal-Error)
```

NOTE: *Internal errors are reported if the design cycle reaches a value greater than 65535.*

NOTE: *For the Ac_datahold01a rule, the Internal-Error status also appears if the design contains a black box and the assume_path constraint is specified between the source and destination of a clock-domain crossing.*

- For assertions that found conflicting constraints:

  ```
  RuleName: <rule-name>
  (<module-name>), <assertion-details>, <file-name>,
  <line-number>: Others(Constraints-Conflict)
  ```

  When you double-click on a violation message that displays the `Others(Constraints-Conflict)` assertion, a dialog appears displaying details of constraints conflict, as shown in the following figure:

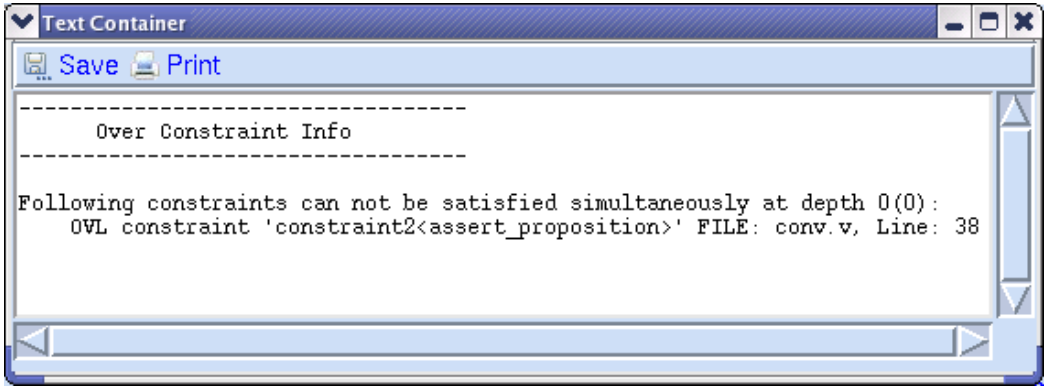


FIGURE 13. Details of constraints conflict

For rules reporting such kinds of assertions, a separate file is generated that contains details of constraints conflict shown in the above dialog. The name of this file is in the following format:

```
<rule-name>.<ID>.OverConstrainInfo
```

The *<cycle-depth>* argument specifies the number of cycles of the fastest clock till the assertion was evaluated.

The *<depth>* argument specifies the verification depth (number of cycles

of the cumulative clocks) involved in verification till the assertion was evaluated.

NOTE: *The depth information reported for a Partially Proved property indicates that the property was still being analyzed at the reported depth when the analysis of the property was stopped. Therefore, it is possible that a property is reported as Partially Proved at a certain depth in a SpyGlass run and is reported as FAILED at the same depth in another run with a different set of SpyGlass options.*

The *<flop-count>* argument specifies the number of flops present in the input cone of the property. In general, the complexity of solving a property is proportional to number of flops in the input cone. The number reported for the flop count is dependent on the engine and may vary based on the engine being run. It is an estimate of the property size.

The number of flops considered for solving a property can be limited by the *fa_flopcount* parameter. It is reported for a partially-proved property in the adv_cdc.rpt report.

NOTE: *The value of parameter* `fa_flopcount` *is not honored on control paths and all the flops present in such paths are considered.*

The <design-cycle> argument indicates the number of clock cycles completed during verification for an unsolved property. The design-cycle is proportional to the complexity of solving a property. It is reported for a partially-proved property in the adv_cdc.rpt report.

Difference Between Advanced CDC and SpyGlass TXV Initialization Report

Information generated in the Advanced CDC report and SpyGlass TXV solution initialization report (/spyglass_reports/txv/0.init) has the following differences:

- Difference in information related with clocks
  - SpyGlass CDC solution report does not contain generated clocks. However, this information is present in the report of SpyGlass TXV solution.
  - SpyGlass CDC solution report shows clocks that have not been used on any flip-flop. However, SpyGlass TXV solution report does not show such clocks.

- Difference in information related with latches
  - For SpyGlass TXV solution, latches are counted as sequential elements while calculating an initialization percentage. This can result in a different *sequential element count* and a *different initialization percentage*, as shown below:

    Initialization percentage in SpyGlass TXV = (Number of initialized sequential elements) /(Total no of sequential elements including latches) * 100

    Initialization percentage in SpyGlass CDC = (Number of initialized flip-flops) /(Total no of flops) * 100
- Difference in information related with resets
  - If a reset is specified twice in an SGDC file, SpyGlass CDC solution report shows that reset twice, while SpyGlass TXV solution report shows it once.
  - SpyGlass CDC solution report shows resets that have not been used on any flip-flop, while SpyGlass TXV solution report does not show such resets.
- Difference in sequential element initialization list

  For a design containing complex clocks, different precision levels of clock periods in SpyGlass TXV solution and SpyGlass CDC solution may result in different clocks edges. This can cause a difference in the initialization of sequential elements in the two products.

  For example, clock period for a clock with its period as 6.66667 could be treated differently in the SpyGlass TXV solution and SpyGlass CDC solution due to different precision levels in the two products, which can cause a difference in initialization results.

The Register Info Report

The Register Info report, adv_cdc.reg, provides information on clocks, resets and registers in a design.

It is automatically generated when you run any of the *CDC Verification Rules*.

It contains the following sections:

Section A: Clocks in the design	*Section B: Resets in the design*
Section C: Uninitialized Registers (after primary sets/resets are applied)	*Section D: Register Information*

Section A: Clocks in the design

This section of *The Register Info Report* lists all the clocks in a design.

An integer ID number is assigned to each clock signal. *Section D: Register Information* shows Clock ID numbers of such clocks instead of the actual clock names.

Section B: Resets in the design

This section of *The Register Info Report* lists all the synchronous and asynchronous resets in a design.

An integer ID number is assigned to each reset signal. Section D reports these Reset ID numbers instead of actual reset names.

Section C: Uninitialized Registers (after primary sets/resets are applied)

This section of *The Register Info Report* lists the uninitialized registers after applying primary sets/resets.

Section D: Register Information

This section of *The Register Info Report* lists register information used for functional analysis.

Initial values are as inferred after applying primary sets/resets and/or after random simulation.

The NoClockCell-Summary Report

The NoClockCell-Summary report shows the following information:

- Details of nets specified as start points using the *noclockcell_start* constraints
- Details of ports/pins/nets specified as stop points using the *noclockcell_stop_signal* constraints
- Details of modules specified as stop points using the *noclockcell_stop_module* constraints
- Details of instances specified as stop points using the *noclockcell_stop_instance* constraints

The DeltaDelay-Concise Report

The DeltaDelay-Concise report displays the following information:

- Different delta delay values for each clock in a design.
- Number of flip-flops and latches for each delta delay value.

This report is generated when the *DeltaDelay01* rule is run.

Following is the example showing the list of delays in the DeltaDelay-Concise report:

```
================================================
Clock --> top.clk:
************************************************
Delay       Number of Flops      Number of Latches
************************************************
  24                8                    0
  26                8                    0
  29                8                    0
  31                8                    0
  37                8                    0
  39                8                    0

************************************************
```

The DeltaDelay-Detailed Report

The DeltaDelay-Detailed report shows the following information:

- List of different delta delays for each clock
- Net names of flip-flops for each delay value
- Net names of latches for each delay value

For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, the instance names of flip-flops/latches are shown. Otherwise, net names are shown.

An example of a list of delays shown in the DeltaDelay-Detailed report is as follows:

```
=================================================
Clock --> top.clk:
   Delay --> 24:
      Flop: top.I2.r_ff1[0]
      Flop: top.I2.r_ff1[1]
      Flop: top.I2.r_ff1[2]
      Flop: top.I2.r_ff1[3]
      Flop: top.I2.r_ff1[4]
      Flop: top.I2.r_ff1[5]
      Flop: top.I2.r_ff1[6]
      Flop: top.I2.r_ff1[7]
   Delay --> 26:
      Flop: top.I2.r_ff2[0]
      Flop: top.I2.r_ff2[1]
      Flop: top.I2.r_ff2[2]
      Flop: top.I2.r_ff2[3]
      Flop: top.I2.r_ff2[4]
      Flop: top.I2.r_ff2[5]
      Flop: top.I2.r_ff2[6]
      Flop: top.I2.r_ff2[7]

=================================================
```

The DeltaDelay02-Detailed Report

The DeltaDelay02-Detailed report lists flip-flops that can cause simulation problems due to delta delay issues. The report contains the following sections:

- *Section A*
- *Section B*

Section A

This section of *The DeltaDelay02-Detailed Report* lists flip-flops that do not have explicit physical delay to avoid simulation problems.

Section B

This section of *The DeltaDelay02-Detailed Report* lists derived clocks that have an explicit `delay` statement.

Sample Report

A sample DeltaDelay02-Detailed report is as follows:

```
=====================================================
A. Flops which do not have explicit physical delay to avoid
simulation problems
************************************************************
************************************************************
------------------------------------------------------------
  S. No.       Clock Name                   Flop Name
------------------------------------------------------------
    1.     test.clk1              test.src1
                                  test.src2
------------------------------------------------------------
```

```
B. Derived clocks which have explicit delay statement
************************************************************
************************************************************
------------------------------------------------------------
  S. No.       Clock Name                   Flop Name
------------------------------------------------------------
    1.     test.clk1              test.der_clk
------------------------------------------------------------
```

The DeltaDelay-Summary Report

The DeltaDelay-Summary report provides the following details:

- Details of nets specified as start points using the *deltacheck_start* constraints or the default start nets if *deltacheck_start* constraint is not used
- Details of ports/pins/nets specified as stop points using the *deltacheck_stop_signal* constraints
- Details of modules specified as stop points using the *deltacheck_stop_module* constraints
- Details of instances specified as stop points using the *deltacheck_stop_instance* constraints
- Maximum and minimum delta clock delay values
- Total number and checked number of flip-flops and latches

The Ac_sync_group_detail Report

The Ac_sync_group_detail report contains the details of all the violations reported by the *Ac_sync02*, *Ac_sync01*, *Ac_unsync02*, and *Ac_unsync01* rules.

This report contains the following sections:

- *A. Unsynchronized Vector Signal Crossings*

  This section lists the unsynchronized crossings reported by the *Ac_unsync02* rule.

- *B. Unsynchronized Scalar Signal Crossings*

  This section lists the unsynchronized crossings reported by the *Ac_unsync01* rule.

- *C. Synchronized Vector Signal Crossings*

  This section lists the synchronized crossings reported by the *Ac_sync02* rule.

- *D. Synchronized Scalar Signal Crossings*

  This section lists synchronized crossings reported by the *Ac_sync01* rule.

The Ac_sync_qualifier Report

The Ac_sync_qualifier report is generated by the *Ac_sync01* and *Ac_sync02* rules.

This report categorizes *Qualifier* into the following sections:

- *Section A* that reports qualifiers that synchronize data crossings.
- *Section B* that reports qualifiers that do not synchronize data crossings.

If a user-defined qualifier or *abstract_port* constraint is specified, the following information appears in this report:

- The qualifier name appears as a destination name and the qualifier clock name appears as a destination clock.
- The "-" symbol appears for a source name and clock names.
- The reported synchronization scheme appears in the last column.

This report shows all the qualifiers reaching to a synchronizing gate even though the spreadsheet of the *Ac_sync01* and *Ac_sync02* rules shows only one qualifier.

NOTE: *Please note the following points:*

- *Bus-merged qualifiers are reported in Section A when any of the bits of the qualifier are used to synchronize data crossings.*
- *Unused qualifiers specified by the abstract_port constraint are not reported in Section B.*

The sample report is shown below:

```
=============================================================================================
A. Qualifiers used to synchronize data crossings
=============================================================================================
---------------------------------------------------------------------------------------------
S.   Dest. Name            Dest.      Source Name    Source    Sync.
No.                        Clock                     Clock     Scheme
                           Names                     Names
---------------------------------------------------------------------------------------------
1.  top.dest               "top.c2"  top.src       "top.cl"  Conventional multi-flop
                                                               for metastability
                                                               technique
2.  top.a_sync_cell_used   "top.c2"  top.syncCell  "top.cl"  synchronizing cell
    (sync_cell/SpyPortl)             _src                     (cell name : 'sync_cell')
3.  top.a_pp               "top.c2"  -                 -      Abstract Port Qualifier
    (abstract_port/outl)
4.  top.apsyncPort         "top.c2"  -                 -      Abstract Port Qualifier
5.  top.qualOut            "c2"      -                 -      User Defined Qualifier
---------------------------------------------------------------------------------------------

=============================================================================================
B. Qualifiers not used to synchronize any data crossings
=============================================================================================
---------------------------------------------------------------------------------------------
S.  Dest. Name         Dest.       Source           Source  Sync.
No.                    Clock       Name             Clock   Scheme
                       Names                        Names
---------------------------------------------------------------------------------------------
1.  top.dest_unused    "top.c2"  top.src_unused   "top.cl"  Conventional multi-flop
                                                            for metastability technique

2.  top.a_sync_cell    "top.c2"  top.syncCell     "top.cl"  synchronizing cell
    _unused(sync_cell              _unused_src               (cell name : 'sync_cell')
    /SpyPortl)

3. top.qualOut_unused  "c2"      -                -         User Defined Qualifier
---------------------------------------------------------------------------------------------
```

FIGURE 14. The Ac_sync_qualifier Report

You can also view this report in a spreadsheet (.csv) format. This spreadsheet file (Ac_sync_qualifier.csv) is generated under the /spyglass_reports/clock-reset/ directory.

The Glitch_detailed Report

The Glitch_detailed report is generated by the *Ac_glitch03* rule.

This report contains a consolidated summary of all the sources that are crossing destinations and contain glitch-related issues.

The sample Glitch_detailed report is shown below:

```
*******************************************************************************
                          Glitch_detailed Text Report
*******************************************************************************
-------------------------------------------------------------------------------
Dest.    Dest.    Source  Source Source  Multiple  Synchronous  Source     Gray
Name     Clocks   Name    Type   Clocks  source    signals      reconverge -encoding
                                        Domains   convergence

-------------------------------------------------------------------------------
t.des    "t.ck4"  t.in2   Sync   "t.ck4" yes       yes          NA         DISABLED
t.des    "t.ck4"  t.in1   Async  "vk"    yes       yes          no         DISABLED
t.des    "t.ck4"  t.in3   Async  "t.ck3" yes       yes          yes        DISABLED
t.des1   "t.ck3"  t.src1  Async  "t.ck1" no        no           no         FAILED
t.des1   "t.ck3"  t.src2  Async  "t.ck1" no        no           no         FAILED
t.dest2  "t.ck3"  t.src3  Async  "t.ck1" no        no           no         PASSED
t.dest2  "t.ck3"  t.src4  Async  "t.ck1" no        no           no         PASSED
-------------------------------------------------------------------------------
```

FIGURE 15. The Glitch_detailed Report

You can also view this report in a spreadsheet (.csv) format. This spreadsheet file (Glitch_detailed.csv) is generated under the <current-working-directory>/spyglass_reports/clock-reset/ directory.

The Module Topology Report

The *module_topology* report generated by the *Ac_topology01* rule shows dependency between instances in a design.

Figure 16 and *Figure 19* show the sample *module_topology* reports.

Use this report to perform top-down constraint migration in a correct order based on the dependency of instances shown in this report. For details, see *Example Code and/or Schematic*.

This report shows the following information:

- Name of instances in a design
- Name of the modules of instances
- *Topological order of instances*

  It refers to a numeric value depicting dependency between instances.

  If there is **no cyclic dependency** between instances, a unique topological order is assigned to each instance based on which you can determine the sequence in which top-down constraint migration should be done for instances. For details, see *Example 1*.

  If there is **cyclic dependency** between instances, same topological order is assigned to such instances. For details, see *Example 2*. In such cases, check the verification order to determine the sequence in which top-down constraint migration should be done for instances.
- *Verification order of instances*

  It refers to a numeric value depicting dependency between instances.

  Based on this value, you can determine the sequence in which top-down constraint migration should be done for instances.

  The difference between topological order and verification order is that in case of cyclic dependency between instances, topological order for instances is same whereas verification order for each instance is unique.

Overconstrain Info File

SpyGlass CDC solution consolidates all user-specified and generated constraints and applies them together.

However, if any conflicting constraints are found during advanced SpyGlass CDC solution rule-checking, an overconstrain info file is generated that contains details of conflicting constraints.

The name of this file is in the following format:

<rule-name>.<ID>.OverConstrainInfo

NOTE: *If you run the Ac_sanity04 rule, the file name is OverConstrain.info, and not Ac_sanity04.<ID>.OverConstrainInfo.*

Messages Reported in the Overconstrain Info File

This file contains the following messages:

- Comb-loop involving net '<net-name>' is unstable
- Reset constraint on net '<net-name>'

  The reset can be an asynchronous reset or a synchronous reset.
- set_case_analysis constraint on net '<net-name>'
- OVL constraint '<name>' FILE: <file-name>, Line: <line-num>
- Some constraints cannot be satisfied simultaneously at depth *<cycledepth>* (*<depth>*)

  This message appears if the time taken during message generation for the Overconstrain Info file exceeds the time-out limit.

Sample Overconstrain Info File

Consider the following design on which *Ac_cdc01c* rule-checking is done:

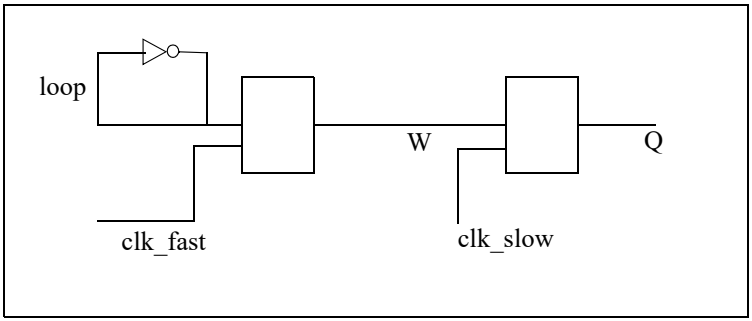


FIGURE 16. Design for which Overconstrain Info File is Being Generated

For the above design, the following Ac_cdc01c.1.OverConstrainInfo file is generated:

```
Following constraints cannot be satisfied simultaneously at
depth 0(0):

    Comb-loop involving net 'top.loop' is unstable
```

The CDC Matrix Report

The CDC Matrix report (*cdc_matrix.rpt*) shows details of SpyGlass-CDC attributes and whether they follow the limits set by the *cdc_matrix_attributes* constraint. This report is generated by the *Setup_req01* rule.

NOTE: *You must specify the cdc_matrix_attributes constraint to generate this report.*

The purpose of viewing this report is to check the health of SpyGlass-CDC setup before proceeding with SpyGlass CDC verification.

If the limit of the CDC attributes exceeds the prescribed limit, fix the issues that are 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.

For information on generating this report and its path, see *Viewing Reports in GUI* and *Specifying the Report to be Generated through a Project File*.

This report contains the following sections:

Section A	*Section B*	*Section C*

Section A

This section of *The CDC Matrix Report* shows a summary of SpyGlass-CDC attributes and whether they follow the limit set by the *cdc_matrix_attributes* constraint.

Refer to *Figure 15* to view the sample report.

Section B

This section of *The CDC Matrix Report* shows the name of the destination to which sources more than the specified limit are reaching.

This section is not generated if you set the *-src_per_dest_limit* argument of the *cdc_matrix_attributes* constraint to -1.

Refer to *Figure 15* to view the sample report.

Section C

This section of *The CDC Matrix Report* shows details of top five clock pairs for which the number of crossings between them exceeds the limit specified by the *-crossing_per_clock_pair_limit* argument of the *cdc_matrix_attributes* constraint.

This section is not generated if you set this argument to -1.

Refer to *Figure 15* to view the sample report.

The Distributed Time Report

The distributed time report (*distributed_time.rpt*) shows run time details of SpyGlass CDC rules that are run in parallel on same or different machines.

The run time details include the information, such as:

- Pack ID

  Assertions reported by SpyGlass CDC rules are grouped into chunks called packs. Each pack is assigned a unique pack ID.
- Number of assertions in a pack
- CPU time and wall clock time consumed by each pack
- Machine name for a pack

The sample distributed time report is shown below:

```
RuleName    PackID  Machine     Assertions    CPU Time  Wall Clk    Overall
                                                         Time       CPU Time
-------------------------------------------------------------------------
Ac_cdc01a   1       machine_1     4             21        22
            1       machine_3     4             30        36
            2       machine_2     2             16        18
                                                                      67
-------------------------------------------------------------------------
Ac_cdc01b   1       machine_1     5             44        44
            1       machine_3     5             45        54
            2       machine_1     5             44        9
            3       machine_4     1             11        14
                                                                     144
-------------------------------------------------------------------------
Ac_cdc01c   1       machine_1     6             34        34
            1       machine_2     6             34        34
            2       machine_4     1             7         7
                                                                      75
-------------------------------------------------------------------------
```

Input Port Constraints File

When you run the *Clock_info15* or the *Setup_port01* rule, SpyGlass CDC generates the following SGDC file containing *abstract_port* constraints generated for input ports of a block:

<wdir>/<project-name>/<goal-path>/spyglass_reports/clock-reset/<top-level>_input_abstract.sgdc

For example, consider the scenario in which the `IN1` input port is sampled into the `CK1` clock domain inside the `BLK` block. In this case, SpyGlass CDC solution generates the following constraint:

```
abstract_port -scope cdc -module BLK -ports IN1 -clock CK1
```

Note the following points:

- The *abstract_port* `-ignore` constraints is generated if all the fan-out of an input port are hanging or blocking.
- The *abstract_port* constraint is not generated in the following cases:
  - If an input port is connected to an unconstrained output port through pure combinational logic only.
  - If an input port is defined by using the *quasi_static* constraint.
- If an input port is connected to the data or enable pin of a sequential element that is constrained by the *clock* constraint, the domain of this sequential element is assigned in the generated *abstract_port* constraint.
- Input ports that are defined by user-defined constraints, such as *clock*, *quasi_static*, or *set_case_analysis,* are not included in the input port constraints file.

SpyGlass CDC solution generates different *abstract_port constraint specifications based on different scenarios, as described in the following topics:*

- *abstract_port Constraints for Ports Connected with Multiple Sequential Elements*
- *abstract_port Constraints for Ports Connected with Sequential Elements*
- *abstract_port Constraints for Multiple Ports Reaching Same Sequential Element*
- *abstract_port Constraints for Ports Connected to Data Pin of a Multi-Flop Structure*

abstract_port Constraints for Ports Connected with Multiple Sequential Elements

If an input port is connected to multiple flip-flops sampled in multiple clock domains, SpyGlass CDC solution generates *abstract_port* constraint specifications for each clock domain.

However, if different clocks are in the same domain, only one *abstract_port* constraint specification is generated with any such clocks.

For example, consider the scenario shown in the following figure:

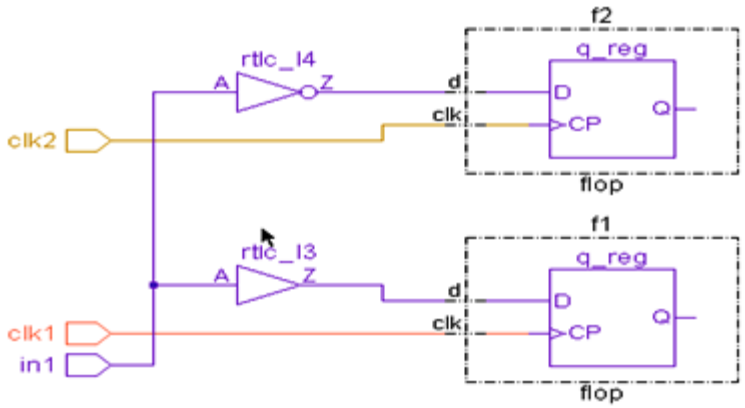


FIGURE 17. Design for Which the abstract_port Constraint is Being Generated

For the above case, the following constraints are generated:

```
abstract_port -scope cdc -module BLK -ports in1 -clock clk1
abstract_port -scope cdc -module BLK -ports in1 -clock clk2
```

abstract_port Constraints for Ports Connected with Sequential Elements

If an input port is connected to a sequential element clocked by multiple clocks, SpyGlass CDC solution generates one *abstract_port* constraint specification containing multiple clocks.

For example, consider the scenario shown in the following figure:

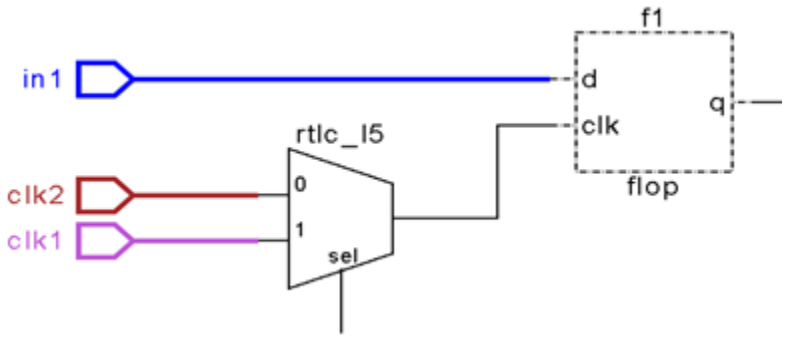


FIGURE 18. Design for Which the abstract_port Constraint is Being Generated

For the above case, the following constraint is generated:

```
abstract_port -scope cdc -module BLK -ports in1 -clock clk1
clk2
```

abstract_port Constraints for Multiple Ports Reaching Same Sequential Element

If multiple input ports reach to the same sequential element, SpyGlass CDC solution generates *abstract_port* constraint specifications in which the domain of the sequential element is assigned to such input ports:

For example, consider the scenario shown in the following figure:

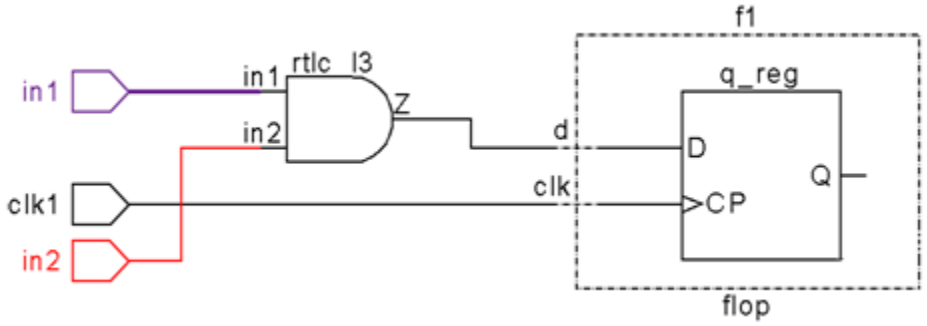


FIGURE 19. Design for Which the abstract_port Constraint is Being Generated

For the above case, the following constraints are generated:

```
abstract_port -scope cdc -module BLK -ports in1 -clock clk1
abstract_port -scope cdc -module BLK -ports in2 -clock clk1
```

abstract_port Constraints for Ports Connected to Data Pin of a Multi-Flop Structure

If an input port is connected to the data pin of a control synchronizer (see *Conventional Multi-Flop Synchronization Scheme* and *Synchronizing Cell Synchronization Scheme*), SpyGlass CDC solution assigns clock domains as a virtual clock to the generated *abstract_port* constraint specifications and set the `-combo` argument of these specifications to `no` and the `-combo_ifn` argument to the clock associated with the synchronizer.

NOTE: *During abstract_port constraint generation, SpyGlass CDC solution considers the following parameters and constraints while detecting a multi-flop structure:*

Parameters	*sync_reset*, *allow_combo_logic*, *strict_sync_check*, *ignore_num_rtl_buf_invs*, *num_flops*, *synchronize_cells*, and *synchronize_data_cells*
Constraints	*reset -sync*, *allow_combo_logic*, *num_flops, sync_cell*

The following points cover different cases in which an input port is connected to the data pin of a multi-flop structure:

- If multiple inputs drive the same control synchronizer and the *allow_combo_logic* parameter is set to `yes`, all ports are assigned to the same virtual clock in the generated *abstract_port* constraint specifications.

  For example, consider the scenario shown in the following figure:

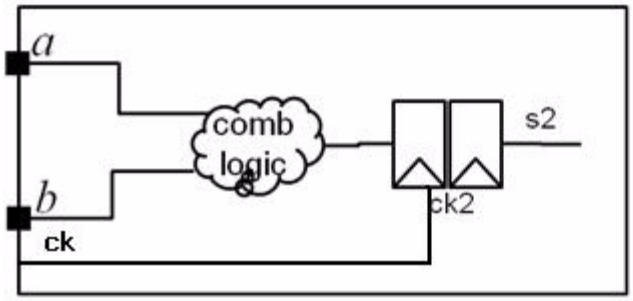


FIGURE 20. Design for Which the abstract_port Constraint is Being Generated

  For the above example, the following constraints are generated:

```
abstract_port -scope cdc -module BLK -ports a -clock VCK1 -
combo_ifn ck2 -combo no
```

```
abstract_port -scope cdc -module BLK -ports b -clock VCK1 -
combo_ifn ck2 -combo no
```

- If the P1 port, as shown in the following figure, reaches the F1 control synchronizer in one path and a single flip-flop F2 in another path, the *abstract_port* constraint is generated only for single flip-flops:

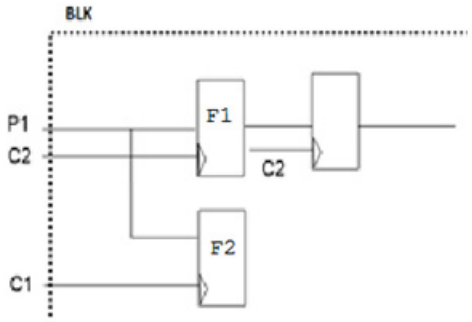


FIGURE 21. Design for Which the abstract_port Constraint is Being Generated

For the above example, the following constraint is generated:

```
abstract_port -scope cdc -module BLK -ports P1 -clock C1
-combo_ifn C2 -combo no
```

- If the P2 port, as shown in the following figure, reaches to the F1 control synchronizer in one path and to multiple single flip-flops [F2, F3] in other paths, *abstract_port* constraint specifications are generated for each clock domain on single flip-flops and the `-combo` argument is set to `no`:

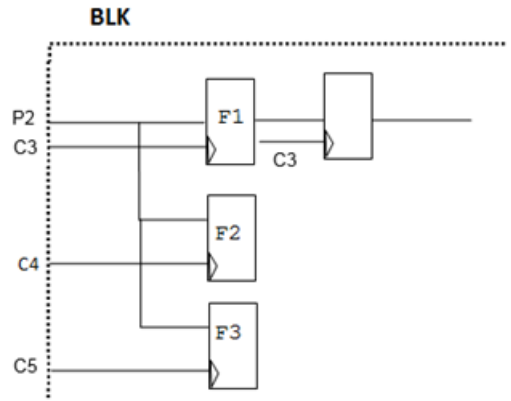


FIGURE 22. Design for Which the abstract_port Constraint is Being Generated

For the above example, the following constraints are generated:

```
abstract_port -scope cdc -module BLK -ports P2 -clock C4 -
combo no
abstract_port -scope cdc -module BLK -ports P2 -clock C5 -
combo no
```

- If the P1 input port, as shown in the following figure, reaches to the F3 single flip-flop in one path and to multiple control synchronizers [F1, F2] in other paths, a single *abstract_port* constraint specification is generated containing a virtual clock and the `-combo` argument is set to `no`.

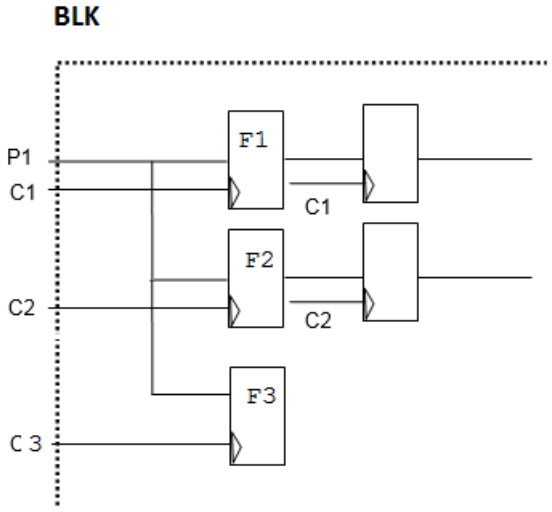


FIGURE 23. Design for Which the abstract_port Constraint is Being Generated

For the above example, the following constraint is generated:

```
abstract_port -scope cdc -module BLK -ports P1 -clock vclk
-combo no
```

The adv_cdc Spreadsheet

This spreadsheet displays results of formal SpyGlass CDC analysis. It provides a quick summary of formal results under different tabs so that you do not need to explicitly open the *adv_cdc.prp* file and *The Advanced CDC Report* to view formal results.

This spreadsheet contains the following tabs:

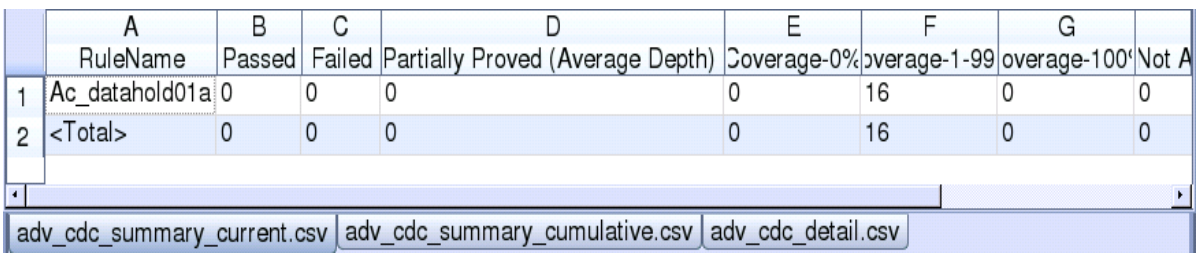
- *adv_cdc_summary_current*
- *adv_cdc_summary_cumulative*
- *adv_cdc_summary_detail*

adv_cdc_summary_current

This spreadsheet shows the statistics of different assertions for each formal rule that is run. This data is same as the data displayed in *Section E: Results Summary (Current)* of *The Advanced CDC Report*.

In the coverage-driven flow, this spreadsheet shows coverage details under additional columns, such as *Coverage-0%*, *Coverage-1-99%*, and *Coverage-100%*.

The following figure shows the example of this spreadsheet:



	A RuleName	B Passed	C Failed	D Partially Proved (Average Depth)	E Coverage-0%	F overage-1-99	G overage-100	 Not A
1	Ac_datahold01a	0	0	0	0	16	0	0
2	<Total>	0	0	0	0	16	0	0

adv_cdc_summary_current.csv | adv_cdc_summary_cumulative.csv | adv_cdc_detail.csv

FIGURE 24. Example of the adv_cdc_summary_current spreadsheet

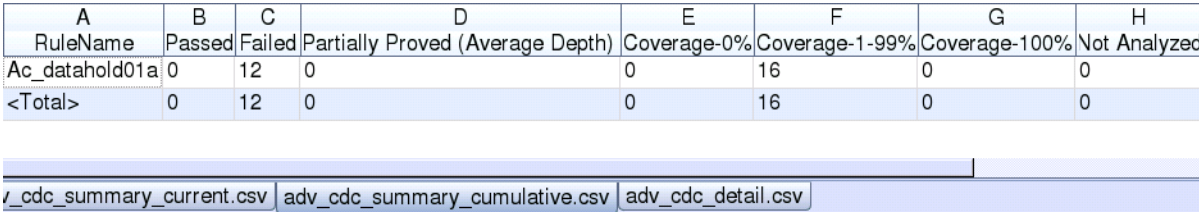
adv_cdc_summary_cumulative

This spreadsheet shows the summary of the cumulative set of assertions formed in the current run and the information of earlier runs based on a property file. This data is same as the data displayed in *Section F: Results*

Summary (Cumulative) of *The Advanced CDC Report*.

In the coverage-driven flow, this spreadsheet shows coverage details under additional columns, such as *Coverage-0%*, *Coverage-1-99%*, and *Coverage-100%*.

The following figure shows the example of this spreadsheet:



A RuleName	B Passed	C Failed	D Partially Proved (Average Depth)	E Coverage-0%	F Coverage-1-99%	G Coverage-100%	H Not Analyzed
Ac_datahold01a	0	12	0	0	16	0	0
<Total>	0	12	0	0	16	0	0

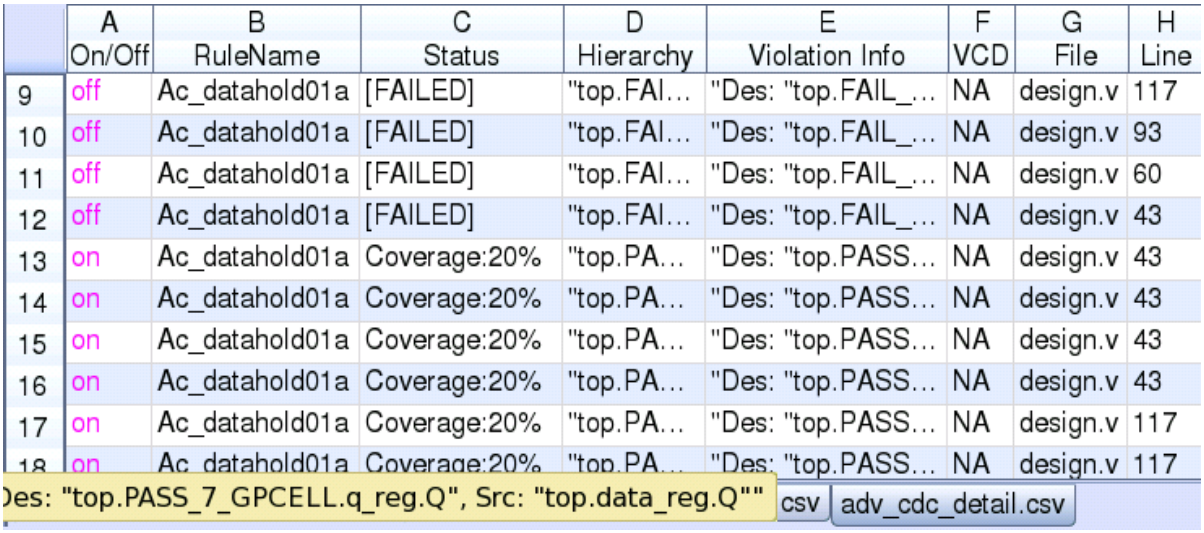
FIGURE 25. Example of the adv_cdc_summary_cumulative spreadsheet

adv_cdc_summary_detail

This spreadsheet shows detailed information on the assertions for each formal rule that is run. This data is same as the data displayed in *Section G: Assertion Details* of *The Advanced CDC Report*.

In the coverage-driven flow, this spreadsheet also shows the percentage of coverage.

The following figure shows the example of this spreadsheet:



	A On/Off	B RuleName	C Status	D Hierarchy	E Violation Info	F VCD	G File	H Line
9	off	Ac_datahold01a	[FAILED]	"top.FAI...	"Des: "top.FAIL_...	NA	design.v	117
10	off	Ac_datahold01a	[FAILED]	"top.FAI...	"Des: "top.FAIL_...	NA	design.v	93
11	off	Ac_datahold01a	[FAILED]	"top.FAI...	"Des: "top.FAIL_...	NA	design.v	60
12	off	Ac_datahold01a	[FAILED]	"top.FAI...	"Des: "top.FAIL_...	NA	design.v	43
13	on	Ac_datahold01a	Coverage:20%	"top.PA...	"Des: "top.PASS...	NA	design.v	43
14	on	Ac_datahold01a	Coverage:20%	"top.PA...	"Des: "top.PASS...	NA	design.v	43
15	on	Ac_datahold01a	Coverage:20%	"top.PA...	"Des: "top.PASS...	NA	design.v	43
16	on	Ac_datahold01a	Coverage:20%	"top.PA...	"Des: "top.PASS...	NA	design.v	43
17	on	Ac_datahold01a	Coverage:20%	"top.PA...	"Des: "top.PASS...	NA	design.v	117
18	on	Ac_datahold01a	Coverage:20%	"top.PA...	"Des: "top.PASS...	NA	design.v	117

FIGURE 26. Example of the adv_cdc_summary_detail spreadsheet

The CrossingMatrix Spreadsheet

The *CrossingMatrix* spreadsheet shows the summary of crossings per clock pair with their clock domains.

View this spreadsheet to check the overall SpyGlass CDC risk on a design, crossings congestion area, and setup issues causing unusual crossing distribution.

This spreadsheet is generated in the *spyglass_reports/clock-reset/Ac_crossing* directory.

For details on this spreadsheet, see the following topics:

- *Data Fetched in the CrossingMatrix.csv Spreadsheet*
- *Information Format in the CrossingMatrix.csv Spreadsheet*
- *Example of the CrossingMatrix.csv Spreadsheet*

Data Fetched in the CrossingMatrix.csv Spreadsheet

When the *The Ac_sync_group Rules* are run, data for this report is fetched from the Ac_sync_group rules and all sources per destination are considered by default because the Ac_sync_group rules report all the sources.

Information Format in the CrossingMatrix.csv Spreadsheet

Information per clock crossing pairs appears in the following format in the spreadsheet:

<a>/<b>, <c>/<d>

where,

- *<a>* is the count of *Ac_unsync01*/*Ac_unsync02* messages.
- *<b>* is the count of total unsynchronized crossings that exist for the clock crossing pair.
- *<c>* is the count of *Ac_sync01*/*Ac_sync02* messages.
- *<d>* is the count of total synchronized crossings that exist for a clock crossing pair.

NOTE: *<b> and <d> may be different from <a> and <c> in case multiple clocks are reaching to crossing instances.*

For details, see *Sample Spreadsheet generated by Ac_sync_group rules*.

Example of the CrossingMatrix.csv Spreadsheet

Consider the following design:

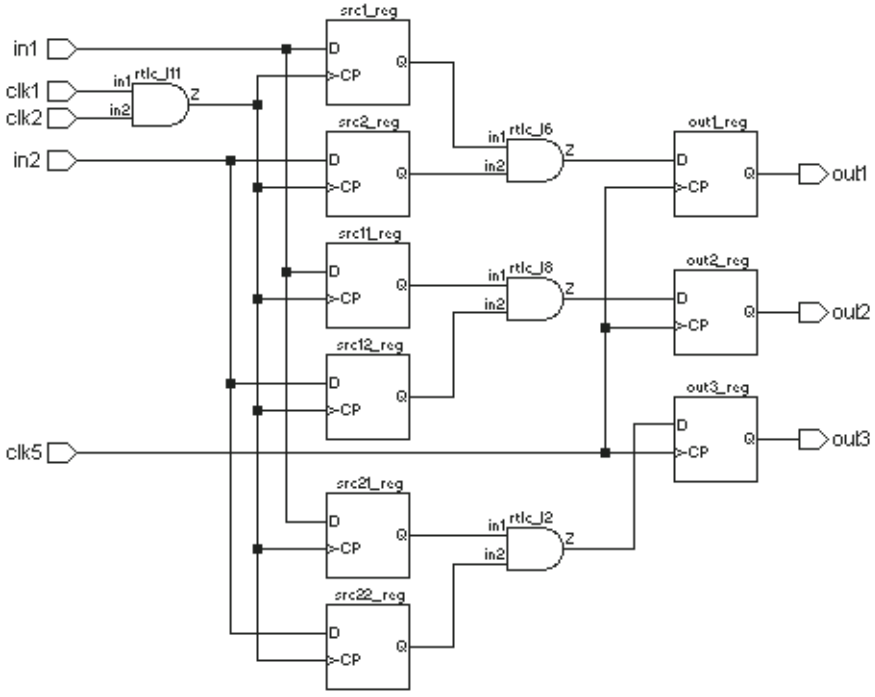


FIGURE 27. Design on which the Ac_crossing01 Rule is being Run

When you run the *Ac_crossing01* rule on the above design, *The CrossingMatrix Spreadsheet* is generated. Now based on the rules enabled, such as `Ac_sync_group` rules, different clock pair information is shown in the spreadsheet.

Sample Spreadsheet generated by Ac_sync_group rules

The following figure shows the crossing matrix spreadsheet generated when the `Ac_sync_group` is enabled:

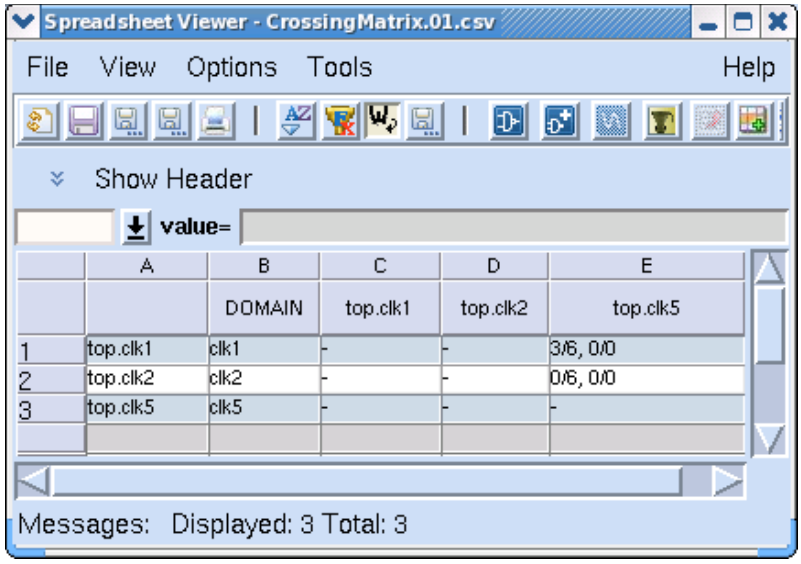


FIGURE 28. Crossing Matrix Spreadsheet

The above spreadsheet shows information per clock crossing pairs in the following format:

<a>/<b>, <c>/<d>

Where:

- *<a>* is the total number of *Ac_unsync01* and *Ac_unsync02* messages for the given clock pair.
- *<b>* is the total number of crossings for this clock pair for all the sources involved in the crossing.
- *<c>* is the total number of *Ac_sync01* and *Ac_sync02* violation messages for the given clock pair.
- *<d>* is the total number of crossings for this clock pair for all the sources involved in the crossing.

Description of the above format for each clock pair in this example is described below:

- For the clock pair `clk1-clk5`

- ❒ Three violations of the *Ac_unsync01* rule are reported in this case, and all sources are considered between the `clk1-clk5` clock pair.

  Therefore, the value of *<a>* and *<b>* is `3` and `6`, respectively.

- ❒ No *Ac_sync01*/*Ac_sync02* violation is reported in this case.

  Therefore, the value of *<c>* and *<d>* is `0`.

- For the `clk2-clk5` clock-pair

No violation is reported for this clock pair and all sources are considered. Therefore, the value of *<a>* and *<b>* is `0` and `6`, respectively.

The Ar_cross_analysis01 Spreadsheet

The Ar_cross_analysis01 spreadsheet contains information of all the asynchronous clock domain crossings on the reset path in a design.

To view this spreadsheet, double-click on the violation of the Ar_cross_analysis01 rule.

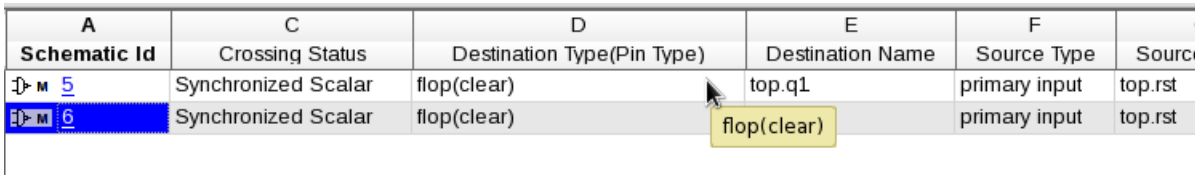


FIGURE 29. Sample Ar_cross_analysis01 Spreadsheet

Details of the Ar_cross_analysis01 Spreadsheet

The following table describes the column details of the Ac_cross_analysis spreadsheet:

Column Name	Description
Schematic ID	Specifies the unique ID for a violation.
Crossing Status	Specifies the crossing status, such as synchronized scalar or unsynchronized vector.
Destination Type(Pin Type)	Specifies the type of the destination object. The object type can be: • Source signal, such as flip-flop, latch, black box, or primary input • Destination signal, such as flip-flop, latch, black box, or primary input • Type of qualifier, such as detected qualifier, user-defined qualifier, or potential qualifier. In case of a vector qualifier, the text vector appears before the qualifier type. The 'Pin Type' can be 'clear' or 'preset' to distinguish the crossings of clear and preset pins of the same instance.
Destination Name	Specifies the name of the destination.

Column Name	Description
Source Type	Specifies the type of the source object. The object type can be: ● Source signal, such as flip-flop, latch, black box, or primary input ● Destination signal, such as flip-flop, latch, black box, or primary input ● Type of qualifier, such as detected qualifier, user-defined qualifier, or potential qualifier. In case of a vector qualifier, the text vector appears before the qualifier type.
Source Name	Specifies the name of the source.
Destination Clock Name	Specifies the name of the clocks reaching the destination signal.
Destination Domain	Specifies the destination domain.
Source Clock Names	Specifies the name of the clocks reaching the source signal.
Source Domain	Specifies the source domain.
Failure Reason	Specifies the reason for unsynchronized crossing.
Sync Scheme	Specifies the name of the synchronization method used to synchronize a crossing.
Qualifier Type	Specifies whether the qualifier is user defined or internally detected.
Qualifier Name	Specifies the name of the qualifier in the crossing.
Qualifier Depth	Specifies the qualifier depth.
Waived	Specifies if the violation is waived
Comments	Specifies user comments, if any.

The Spreadsheets of the Ac_abstract_validation01 Rule

The *Ac_abstract_validation01* rule generates the following spreadsheets:

Clock Mismatch Spreadsheet	*Clock Domain Mismatch Spreadsheet*
Case Analysis Mismatch Spreadsheet	*Quasi static Mismatch Spreadsheet*
Data Path Domain Mismatch Spreadsheet	*Combo Check Mismatch Spreadsheet*
Qualifier Mismatch Spreadsheet	*Virtual Clocks Mismatch Spreadsheet*
Reset Mismatch Spreadsheet	

NOTE: *If you run Ac_abstract_validation01 in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the corresponding message-based spreadsheet. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.*

Clock Mismatch Spreadsheet

Based on *Example - Clock Mismatch*, the following spreadsheet appears showing *Clocks Mismatch*:

A	B	C	D
ID	Block Port Name	Block Clock Defined	Top Clock
12	clk2	no	top.clk2,top.clk5
13	clk3	no	top.clk3

FIGURE 30. The Clock Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Block Port Name	Specifies the clock port of a block
Block Clock Defined	Specifies if the clock constraint is specified for the clock port of the clock.
Top Clock	Specifies the name of the top-level clock

Clock Domain Mismatch Spreadsheet

For *Clock Domain Mismatch*, the *Ac_abstract_validation01* rule generates two types of spreadsheets based on the following cases:

- *Same domain block level clocks connected to different domain top-level clocks*
- *Different domain block level clocks connected to same domain top-level clocks*

Same domain block level clocks connected to different domain top-level clocks

Based on *Example 1 - Clock Domain Mismatch*, the following spreadsheet appears showing *Clock Domain Mismatch*:

A	B	C	D	
ID	Block Domain	Block Clocks	Top Clocks	
7	d1	vck1 ck1	test.clk1 test.clk2	
8	d1	vck1 ck3	test.clk3 test.clk2	

FIGURE 31. The Clock Domain Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Block Domain	Specifies the domain of the abstract view

Column Name	Description
Block Clocks	Specifies the clock on the abstract view
Top Clocks	Specifies the name of the top-level clock

Different domain block level clocks connected to same domain top-level clocks

Based on *Example 2 - Clock Domain Mismatch*, the following spreadsheet appears showing *Clock Domain Mismatch*:

A	B	C	D
ID	Block Clocks	Top Clocks	Top internal domain Tag
3	clk1 clk2	top.clk2 top.clk2	1

FIGURE 32. The Clock Domain Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Block Clocks	Specifies the clock on the abstract view
Top Clocks	Specifies the name of the top-level clock
Top Internal Domain Tag	Specifies a unique tag number generated for the top-level clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.

Case Analysis Mismatch Spreadsheet

Based on *Example - Case Analysis Mismatch*, the following spreadsheet appears showing information related to *Case Analysis Mismatch*:

A	B	C	D
ID	Port Name	Block Value	Top Value
1	in3	0	-
2	in2	0	-

FIGURE 33. The Case Analysis Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Port Name	Specifies the name of the port on the abstract view
Block Value	Specifies the set_case_analysis value on the port of the abstract view
Top Value	Specifies the case_analysis_value propagated at the top-level

Quasi static Mismatch Spreadsheet

Based on *Example - Quasi Static Mismatch*, the following spreadsheet appears showing information related to *Quasi Static Mismatch*:

A	B	C	D
ID	Port Name	Block port quasi_static	Top quasi_static
1	a[1:0]	no	yes

FIGURE 34. The Quasi static Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Port Name	Specifies the name of the port on the abstract view
Block port quasi_static	Specifies if the quasi_static constraint is defined on the port of the abstract view
Top quasi_static	Specifies if a top-level quasi_static signal is reaching the port of the abstract view

Data Path Domain Mismatch Spreadsheet

Based on *Example - Data Path Domain Mismatch*, the following spreadsheet appears showing information related to *Data Path Domain Mismatch*:

A	B	C	D
ID	Port Name	Block Domain(s)	Top Domain(s)
6	in3	-	top.clk1
7	in3	-	top.clk2
8	in3	top.clk1,top.clk2(clk1)	-

FIGURE 35. The Data Path Domain Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Port Name	Specifies the name of the port on the abstract view
Block Domain(s)	Specifies the unmatched clock domain of the abstract view
Top Domain(s)	Specifies the unmatched top-level clock domain

Combo Check Mismatch Spreadsheet

Based on *Example - Combo Check Mismatch*, the following spreadsheet appears showing information related to *Combo Check Mismatch*:

A	B	C	D
ID	Port Name	Block Combo	Top Combo
7	in3	no	yes

FIGURE 36. The Combo Check Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Port Name	Specifies the name of the port on the abstract view
Block Combo	Specifies if a block-level combinational logic exists
Top Combo	Specifies if a top-level combinational logic exists that reaches the block port

Qualifier Mismatch Spreadsheet

Based on *Example - Qualifier Mismatch*, the following spreadsheet appears showing information related to *Qualifier Mismatch*:

A	B	C	D	E	F	G	H
ID	ort Nam	ock Source Clo	Block Destination Clock	p Source Clo	op Destination Cloc	Block Sync Type	op Sync Typ
4	d1	-	-	test.clk1	test.clk4	-	active
5	d1	-	-	test.clk4	test.clk3	-	active
6	d1	test.B1_V1	test.clk2	-	-	active	-
7	d1	test.clk1	test.clk3	-	-	active	-

FIGURE 37. The Qualifier Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Port Name	Specifies the name of the port on the abstract view
Block Source Clock	Specifies the clock name associated with the pin (-from argument) of the unmatched abstract_port constraint
Block Destination Clock	Specifies the clock names associated with the pin (-to argument) of the unmatched abstract_port constraint
Top Source Clock	Specifies the source clock of the unmatched synchronizer at the top level
Top Destination Clock	Specifies the destination clock of the unmatched synchronizer at the top level
Block Sync Type	Specifies the -sync argument of the abstract_port constraint
Top Sync Type	Specifies the sync type of the top-level synchronizer. If it is control type synchronizer, `active` appears in the spreadsheet. Else, `inactive` appears.

Virtual Clocks Mismatch Spreadsheet

For *Virtual Clocks Mismatch*, the *Ac_abstract_validation01* rule generates two types of spreadsheets based on the following cases:

- *Mismatch due to no top-level domain available*
- *Mismatch due to conflict top-level domains*

Mismatch due to no top-level domain available

Based on *Example 1 - Virtual Clocks Mismatch*, the following spreadsheet appears showing information related to *Virtual Clocks Mismatch*:

A	B	C	D	E
ID	Virtual Clock Name	Port Name	Top Clocks	Top Internal Domain Tag
1	V1	d1	<unconstrained>	-
2	V2	d2	<unconstrained>	-

FIGURE 38. The Virtual Clocks Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Virtual Clock Name	Specifies the name of virtual clock on the abstract view
Port Name	Specifies the name of the port of the abstract view
Top Clocks	Specifies the name of the top-level clock
Top Internal Clock Domain Tag	Specifies a unique tag number generated for the top-level clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.

Mismatch due to conflict top-level domains

Based on *Example 2 - Virtual Clocks Mismatch*, the following spreadsheet appears showing information related to *Virtual Clocks Mismatch*:

A ID	B Virtual Clock Name	C Port Name	D Top Clocks	E Top Internal Domain Tag
1	V1	d2	test.clk1	0
2	V1	d1	test.clk1,test.clk2	2

FIGURE 39. The Virtual Clocks Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Virtual Clock Name	Specifies the name of virtual clock on the abstract view
Port Name	Specifies the name of the port of the abstract view
Top Clocks	Specifies the name of the top-level clock
Top Internal Clock Domain Tag	Specifies a unique tag number generated for the top-level clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.

Reset Mismatch Spreadsheet

Based on *Example - Reset Mismatch*, the following spreadsheet appears showing information related to *Reset Mismatch*:

A	B	C	D	E	F	G
ID	Port Name	Top Reset	Block Reset Type	Top Reset Type	Block Reset Value	Top Reset Value
1	rst1	top.rst1	Sync	Sync	1	-
2	rst2	top.rst2	Sync	Sync	0	-
3	rst3	top.rst3	Sync	Sync	0	-
4	rst4	top.rst4	Sync	Sync	1	-
5	rst5	top.rst5	Sync	Sync	-	1
6	rst6	top.rst6	Sync	Sync	-	1
7	rst7	top.rst7	Sync	Sync	-	0
8	rst8	top.rst8	Sync	Sync	-	0

FIGURE 40. The Reset Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Port Name	Specifies the port of the abstract view
Top Reset	Specifies the name of the top-level reset
Block Reset Type	Specifies if the reset on the abstract view is synchronous reset or asynchronous reset
Top Reset Type	Specifies if the top-level reset is synchronous reset or asynchronous reset
Block Reset Value	Specifies the value of the reset on the abstract view
Top Reset Value	Specifies the value of the top-level reset

num_flops Mismatch Spreadsheet

Based on *Example - num_flops Mismatch*, the following spreadsheet appears showing information related to *num_flops Mismatch*:

A	B	C	D	E
ID	num_flops Source	num_flops Destination	Block Value	Top Value
1	clk1	clk2	2	4

FIGURE 41. The num_flops Mismatch Spreadsheet

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
ID	Specifies a unique ID for a violation
Block num_flops Source Clock	Specifies the top-level clock port on which the *num_flops* constraint is defined
Block num_flops Destination Clock	Specifies the abstract block's clock port on which the *num_flops* constraint is defined
Block Value	Specifies the value of the *num_flops* constraint defined on the abstract block's clock port
Top Value	Specifies the value of the *num_flops* constraint defined on the top-level clock port

The Ac_abstract_validation02 Spreadsheet

This spreadsheet shows all the types of mismatches reported by the *Ac_abstract_validation02* rule. To view this spreadsheet, double-click on the violation of the *Ac_abstract_validation02* rule.

Figure 42 shows the example of this spreadsheet:

ID	Block Instance Name	Block Name	Block Port Name(s)	MisMatch Type	Unmatched Top Attribute	Unmatched Block Attribute	Matched Attribute	Waived	User Comments
96	test.B1	bbox	ck6	Clock	test.clk6	no	-	No	-
99	test.B1	bbox	ck7	Clock		ck7	-	No	-
85	test.B1	bbox	ck1	Same Block Clock Domain	test.clk4(clk4)	ck1(d1)	-	No	-
90	test.B1	bbox	ck4	Same Top Clock Domain	test.clk2,test.clk3,test.clk4(d1,clk2,clk4)	ck4(d3)	-	No	-
37	test.B1	bbox	i1	Case Analysis	-	1	-	No	-
3D	test.B1	bbox	rst1	Case Analysis	-	0	-	No	-
4C	test.B1	bbox	nqs	Quasi Static	yes	no	-	No	-
45	test.B1	bbox	rst2	Reset		Active low synchronous reset rst2	-	No	-
13	test.B1	bbox	i3	Virtual Clock		vck	-	No	-
50	test.B1	bbox	nsyncActiveMerge	Data Path Domain	test.clk2	-	-	No	-
54	test.B1	bbox	i2	Data Path Domain	M test.clk4 test.clk5	M test.clk2(ck2) test.clk2,test.clk3,test.clk4(ck5)	test.clk3(ck3)	No	-

FIGURE 42.

See *Column Details of the Ac_abstract_validation02 Spreadsheet* for a description of the columns included in the Ac_abstract_validation02 Spreadsheet.

In the above spreadsheet, the violations appear block wise. First, all the violations of one block appear followed by the violations of another block.

Column Details of the Ac_abstract_validation02 Spreadsheet

The following table describes the column details of the *Ac_abstract_validation02* spreadsheet:

Column Name	Description
ID	Specifies the unique ID for a violation.
Block Instance Name	Specifies the instance name of the abstract block.
Block Name	Specifies the name of the abstract block.
Block Port Name(s)	Specifies the name of the port on the abstract block.
Mismatch Type	Specifies the type of mismatch between the abstract block and the top-level design. There can be different types of mismatches, such as clocks mismatch, clock domain mismatch, and reset mismatch. For information on all the types of mismatches, refer to the documentation of the *Ac_abstract_validation01* rule in the *CDC Rules Reference Guide*.
Unmatched Top Attribute	Specifies the name of the top-level attribute, such as clock or reset that does not match with the block-level attribute.
Unmatched Block Attribute	Specifies the name of the block-level attribute, such as clock or reset that does not match with the top-level attribute.
Matched Attribute	Specifies the matching attributes, such as clocks or resets in the case of domain level mismatch.
Waived	Specifies if the violation is waived
Comments	Specifies user comments, if any.

Simulator File in SpyGlass CDC

A simulator file (used by the *DeltaDelay01* and *DeltaDelay02* rules) contains simulator-specific delta-delay information for RTL constructs.

It is an ASCII file that contains statements (one statement per line) in the following format:

```
<language>:<keyword>:<deltadelayvalue>
```

Where *<language>* is the HDL language, such as VHDL or Verilog (case-insensitive) and *<keyword>* is the statement type. For details on keywords, see *Keywords Used in a Simulator File in SpyGlass CDC*.

Specifying a Simulator File

Use the *simulator_file_name* parameter to specify a simulator file.

Sample Simulator File

The *simulator_file.txt* file is the sample simulator file present in the *SPYGLASS_HOME/policies/clock* directory.

You can directly use this file in the SpyGlass run or modify it to specify different delay values for different simulators.

Keywords Used in a Simulator File in SpyGlass CDC

The following table describes the keywords used in a simulator file:

Keyword	VHDL	Verilog
signal_assignment_01	Indicates concurrent signal assignments, as in the following example: `t0 <= i0;`	Not available

Keyword	VHDL	Verilog

signal_assignment_02	Indicates sequential signal assignments, as in the following example:
	`process (clk)`
	`begin`
	`if (clk'event and clk ='1')`
	`  t0 <= i0;`
	`end if`
	`end process`
		Not available

signal_assignment_03	Indicates assignments in procedures, as in the following example:
	`procedure convert (`
	`  signal i0: in std_logic;`
	`  signal t0: out std_logic)`
	`is`
	`begin`
	`...`
	`t0 <= i0;`
	`...`
	`end convert;`
		Not available

variable_assignment_01	Indicates variable assignments, as in the following example: `t0 := i0;`	Not available

port_connection_01	Indicates direct connection of an input port to an input pin of a VHDL design unit instance, as in the following example:
	`entity entity_name is`
	`  port(`
	`  i0: in std_logic;`
	`  t0: out std_logic);`
	`end entity_name;`
	`architecture ...`
	`...`
	`U1: module_name`
	`PORT MAP (`
	`  i0_in => i0,`
	`  t0_out => t0 );`
	`...`
		Indicates direct connection of a register to an input pin of a VHDL design unit instance, as in the following example:
		`...`
		`reg i0;`
		`...`
		`module_name U1 (`
		`  .i0_in (i0),`
		`  .t0_out (t0)`
		`);`
		`...`

Keyword	VHDL	Verilog

port_connection_02	Indicates direct connection of an output port to an output pin of a VHDL design unit instance, as in the following example:
	`entity entity_name is`
	`  port(`
	`  i0: in std_logic;`
	`  **t0: out std_logic**);`
	`end entity_name;`
	`architecture ...`
	`...`
	`U1: module_name`
	`PORT MAP (`
	`  i0_in => i0,`
	`  **t0_out => t0** );`
	`...`
		Indicates direct connection of a wire to an input pin of a VHDL design unit instance, as in the following example:
		`...`
		`**wire i0;**`
		`...`
		`module_name U1 (`
		`  **.i0_in (i0),**`
		`  .t0_out (t0)`
		`);`
		`...`

port_connection_03	Indicates a signal connected to an input pin of a VHDL design unit instance, as in the following example:
	`...`
	`**signal clock : std_logic;**`
	`...`
	`U1: module_name`
	`PORT MAP (`
	`  **clock => clock,**`
	`  reset => reset,`
	`  i0_in => i0,`
	`  t0_out => t0);`
	`...`
		Indicates direct connection of a wire to an output pin of a VHDL design unit instance, as in the following example:
		`...`
		`**wire t0;**`
		`...`
		`module_name U1 (`
		`  .i0_in (i0),`
		`  **.t0_out (t0)**`
		`);`
		`...`

Keyword	VHDL	Verilog

port_connection_04	Indicates a signal connected to an output pin of a VHDL design unit instance, as in the following example:
	`...`
	`signal t0 : std_logic;`
	`...`
	`U1: module_name`
	`PORT MAP (`
	`clock => clock,`
	`reset => reset,`
	`i0_in => i0,`
	`t0_out => t0);`
	`...`
		Indicates direct connection of a Verilog input port to an input pin of a VHDL design unit instance, as in the following example:
		`module top(i0, t0);`
		`input i0;`
		`output t0;`
		`...`
		`vhdl_du_name U1 (`
		`.i0_in (i0),`
		`.t0_out (t0));`
		`...`

port_connection_05	Indicates direct connection of an input port to an input pin of a Verilog design unit instance, as in the following example:
	`entity entity_name is`
	`port(`
	`i0: in std_logic;`
	`t0: out std_logic);`
	`end entity_name;`
	`architecture ...`
	`...`
	`U1: verilog_module_name`
	`PORT MAP (`
	`i0_in => i0,`
	`t0_out => t0 );`
		Indicates direct connection of a Verilog output port to an output pin of a VHDL design unit instance, as in the following example:
		`module top(i0, t0);`
		`input i0;`
		`output t0;`
		`...`
		`vhdl_du_name U1 (`
		`.i0_in (i0),`
		`.t0_out (t0));`
		`...`

Keyword	VHDL	Verilog

port_connection_06	Indicates direct connection of an output port to an output pin of a Verilog design unit instance, as in the following example:
	`entity entity_name is`
	`port(`
	`i0: in std_logic;`
	`t0: out std_logic);`
	`end entity_name;`
	`architecture ...`
	`...`
	`U1: verilog_module_name`
	`PORT MAP (`
	`i0_in => i0,`
	`t0_out => t0 );`
		Not available

port_connection_07	Indicates a signal connected to an input pin of a Verilog design unit instance, as in the following example:
	`…`
	`signal clock : std_logic;`
	`...`
	`U1: verilog_module_name`
	`PORT MAP (`
	`clock => clock,`
	`reset => reset,`
	`i0_in => i0,`
	`t0_out => t0)`
		Not available

Keyword	VHDL	Verilog

port_connection_08	Indicates a signal connected to an output pin of a Verilog design unit instance, as in the following example:
	`...`
	`signal t0 : std_logic;`
	`...`
	`U1: verilog_module_name`
	`PORT MAP (`
	`clock => clock,`
	`reset => reset,`
	`i0_in => i0,`
	`t0_out => t0);`
		Not available

procedure_01	Indicates the connection of a signal to an input of a procedure, as in the following example:
	`...`
	`signal i1: std_logic;`
	`signal t1: std_logic;`
	`...`
	`convert (`**`i1`**`,t1);`
	`...`
	`procedure convert (`
	`signal i0: in std_logic;`
	`signal t0: out std_logic;`
	`) is`
	`...`
		Not available

procedure_02	Indicates the connection of a signal to an output of a procedure, as in the following example:
	`...`
	`signal i1: std_logic;`
	`signal t1: std_logic;`
	`...`
	`convert (i1,`**`t1`**`);`
	`...`
	`procedure convert`
	`(signal i0: in std_logic;`
	`signal t0: out std_logic;``)`
	`is ...`
		Not available

Keyword	VHDL	Verilog

procedure_03	Indicates the connection of an input port to an input of a procedure, as in the following example:
	`entity entity_name is`
	`  port(`
	`  i1: in std_logic;`
	`  t1: out std_logic);`
	`end entity_name;`
	`architecture ...`
	`...`
	`convert (i1,t1);`
	`...`
	`procedure convert (`
	`  signal i0: in std_logic;`
	`  signal t0: out std_logic;`
	`) is ...`
		Not available

ve_continous_assignment_01	Not Available	Indicates the continuous assignment, as in following example:
		`wire buf_clk;`
		`assign buf_clk = clk;`

ve_blocking_assignment_01	Not Available	Indicates the blocking assignment as in following example:
		`reg div_clk;`
		`always @(posedge clk)`
		`div_clk = clk;`

ve_nonblocking_assignment_01	Not Available	Indicates the non blocking assignment as in following example:
		`reg div_clk;`
		`always @(posedge clk)`
		`div_clk <= clk;`

Keyword	VHDL	Verilog

procedure_04	Indicates the connection of an output port to an output of a procedure, as in the following example:
	`entity entity_name is`
	`  port(`
	`  i1: in std_logic;`
	`  **t1: out std_logic**);`
	`end entity_name;`
	`architecture ...`
	`...`
	`convert (i1,**t1**);`
	`...`
	`procedure convert (`
	`  signal i0: in std_logic;`
	`  **signal t0: out std_logic**;`
	`) is ...`
		Not available

CSV Files Generated On Running SpyGlass CDC Goals

The following CSV files are generated when you run a SpyGlass CDC goal:

- *resets.csv*

  Contains details of all the user-defined resets. The following example shows the data in this spreadsheet:

  ```
  RESET, VALUE
  conv.rst1, 1
  conv.rst2, 0
  ```

- *clocks.csv*

  Contains details of all the user-defined clocks. The following example shows the data in this spreadsheet:

  ```
  CLOCK,TAG,DOMAIN,PERIOD
  conv1.c1,,domain1,3
  conv1.c2,,domain2,5
  conv1.c3,,domain3,5
  ```

- *set_case_analysis.csv*

  Contains details of all the user-defined *set_case_analysis* constraints. The following example shows the data in this spreadsheet:

  ```
  SIGNAL,VALUE
  in2,0
  ```

- *bbClocks.csv*

  Contains details of all the user-defined black box clocks. The following example shows the data in this spreadsheet:

  ```
  CLK-NAME,BB-NAME
  top.q2,bbox
  top.d1,bbox
  ```

- *cdc_false_path.csv*

  Contains details of all the user-defined *cdc_false_path* constraints. The following example shows the data in this spreadsheet:

  ```
  FROM,THROUGH,TO,FROM-TYPE,TO-TYPE
  clk1,,clk2,data,data
  ```

- *assume_path.csv*

  Contains details of all the user-defined *assume_path* constraints. The following example shows the data in this spreadsheet:

  ```
  BB NAME,INPUT,OUTPUT
  BB,A,Z
  ```

- *reset_sig.csv*

  Contains reset-net details, such as reset name, its type, and its driver reset if it is gated or a generated reset. The following example shows the data in this spreadsheet:

  ```
  Reset,Type,Drivers
  conv1.rst,PRIMARY
  ```

- *clock_sig.csv*

  Contains details of all clock net, its name, its type and its driver clocks in case it is gated or generated clock (summary of clock sources in a design)

  ```
  Clock,Type,Drivers
  top.clk1,PRIMARY
  top.clk2,PRIMARY
  top.clk_1,GATED,"top.clk1,top.clk2"
  ```

- *fileList.csv*

  Specifies the path of the SGDC file containing details of all the user-defined constraints.

- *parameterList.csv*

  Specifies the list of user-defined parameters and their values. The following example shows the data in this spreadsheet:

  ```
  PARAMETER, VALUE
  -conv03_report_seq_conv, yes
  ```

- *Block_Summary.csv*

  Shows the summary of the block on which SpyGlass is run. The following example shows the data in this spreadsheet:

  ```
  Set-Case-Analysis,0
  Assume-path,0
  CDC-False-Path,0
  ```

```
Total Time,26
Total flat instances,0
Clocks,2
Domains,2
BB Clocks,0
Resets,0
BB Resets,0
Clock re-definitions(Clock_check07),0
Abstract model mismatch with
top(Ac_abstract_validation01),2
```

RTL Results Difference Utility

The RTL Results Difference utility enables you to identify the differences between two SpyGlass CDC runs. The sg_results_diff.pl file uses the *The CDC-Detailed-Report* generated by the two SpyGlass runs and presents the differences in HTML format.

To run this utility, use the sg_results_diff.pl command. The following syntax shows the usage:

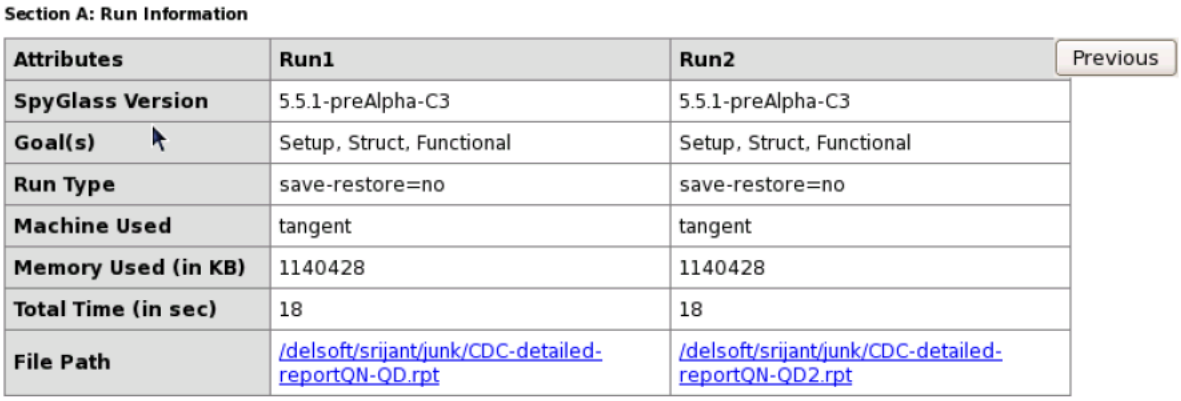
```
%sg_results_diff.pl
-version1_report < CDC-detailed-report of run1>
-version2_report < CDC-detailed-report of run2>
-outfile <file-name>
-alldata
```

Table 1 describes the arguments in the above syntax:

Argument	Description
-version1_report	Specifies the path to the CDC-detailed-report.rpt file generated in the first SpyGlass CDC run
-version2_report	Specifies the path to the CDC-detailed-report.rpt file generated in the second SpyGlass CDC run
-outfile (Optional)	Specifies the path of the HTML file generated by the `sg_results_diff.pl` utility. By default, the `diff.html` file is generated in the current working directory.
-alldata (Optional)	Specifies if the attributes that did not change between the two SPyGlass runs should be included in the HTML report. By default, only the attributes that have changed between the two runs are listed in the HTML report.

TABLE 1 Arguments of the sg_results_diff.pl Utility

The utility generates the `diff.html` file in the current working directory. The following figure shows a sample report.



Section A: Run Information

Previous

Attributes	Run1	Run2
SpyGlass Version	5.5.1-preAlpha-C3	5.5.1-preAlpha-C3
Goal(s)	Setup, Struct, Functional	Setup, Struct, Functional
Run Type	save-restore=no	save-restore=no
Machine Used	tangent	tangent
Memory Used (in KB)	1140428	1140428
Total Time (in sec)	18	18
File Path	/delsoft/srijant/junk/CDC-detailed-reportQN-QD.rpt	/delsoft/srijant/junk/CDC-detailed-reportQN-QD2.rpt

FIGURE 43. Sample diff.html file

The generated report consists of the following sections:

- *Run Information*
- *Top-level Overview of the Result Differences*
- *Summary Table for Differences in each CDC-detailed-report sections*
- *Detailed Difference Report*

Run Information

This section lists the data in Section A of the CDC-Detailed report generated in the two SpyGlass runs as shown in Figure 255.

Top-level Overview of the Result Differences

This section gives an summarizes the count of differences between the CDC-Detailed reports generated in the two SpyGlass runs. The following figure shows the `Top-level Overview of the Result Differences` section of the diff.html file:

1) Top-level Overview of the result differences:

Previous

Section	Category	Run1 (Total Count)	Run2 (Total Count)	Run1 Only	Run2 Only	Run1 and Run2(with different attributes)	Run1 and Run2(with same attributes)
Section C: Setup Information	Number of User defined Parameter Values	4	4	0	0	0	4
Section C: Setup Information	Number of Clocks	2	2	0	0	0	2
Section C: Setup Information	Number of Clock Domains	2	2	0	0	0	2
Section C: Setup Information	Primary Inputs Association With Clock/Domain	2	2	0	0	0	2
Section C: Setup Information	Number of top-level ports not specified using any of 'input', 'output', 'clock', 'reset'(asynchronous reset), 'set_case_analysis', 'quasi_static' or 'abstract_port' constraints	2	2	0	0	0	2
Section E: CDC Analysis and Verification	Ac_unsync01	1	1	0	0	1	0
Section E: CDC Analysis and Verification	Ac_sync01	2	2	0	0	2	0

FIGURE 44. Top-level Overview of the Result Differences sections

Summary Table for Differences in each CDC-detailed-report sections

This section shows a summary table for each CDC-detailed-report section with an additional column for the following:

- Run 1 only
- Run 2 only
- Both in Run 1 and Run 2 (with same attributes)
- Both in Run 1 and Run 2 (with different attributes)

The following figure shows `Summary Table for Differences in each CDC-detailed-report sections`:

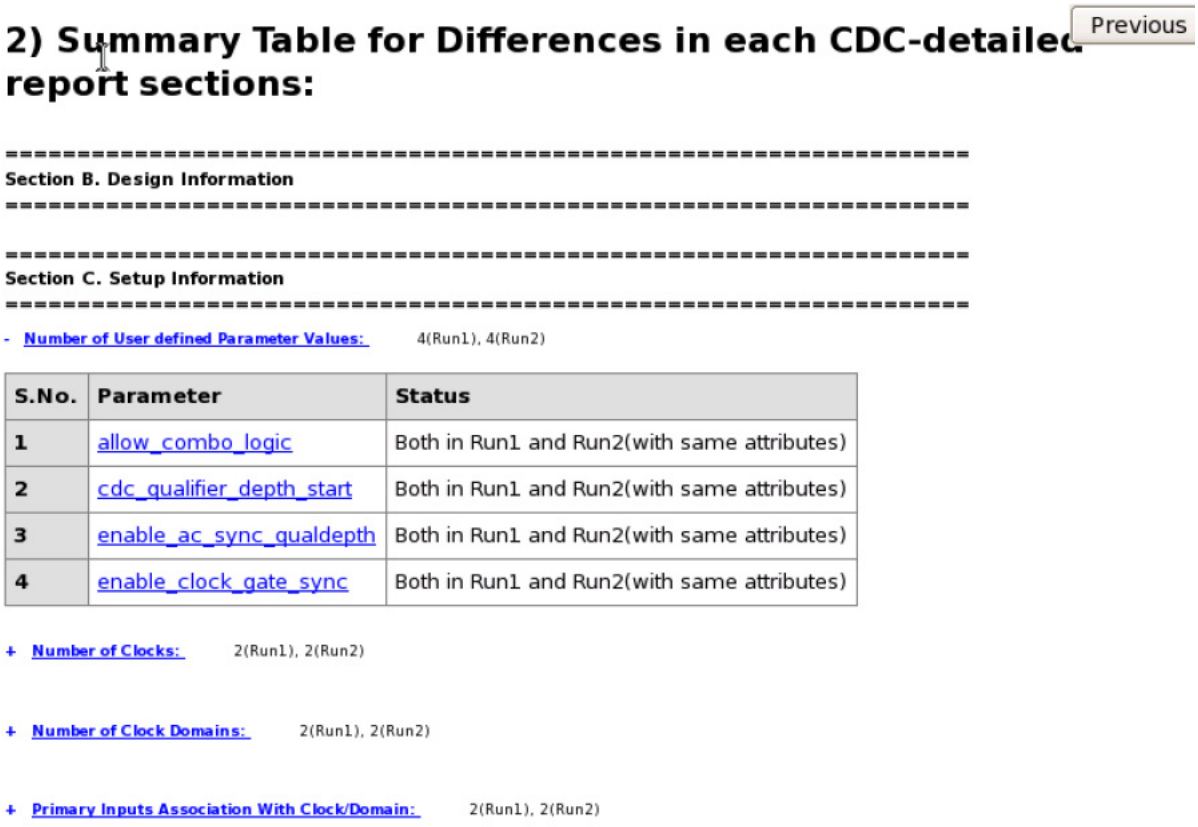
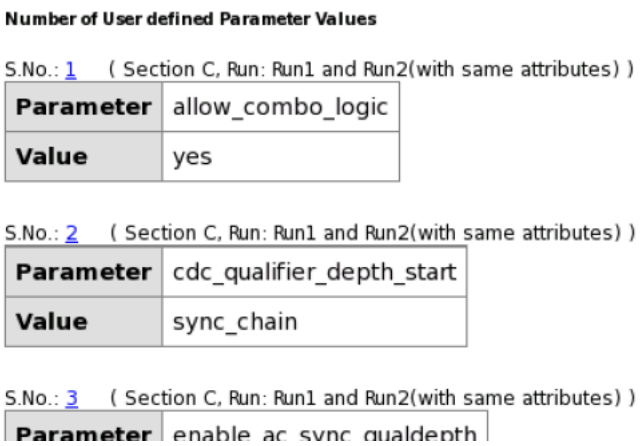


FIGURE 45. Summary Table for Differences in each CDC-detailed-report sections

You can click the links for additional details about the information contained in the report. For example, clicking the `Number of User defined Parameter Values` link in the report lists the parameters used for running SpyGlass.

You can click each parameter to get information on the parameter values used for each SpyGlass run in a tabular format as shown in the following

figure:



Number of User defined Parameter Values

S.No.: 1 (Section C, Run: Run1 and Run2(with same attributes))

Parameter	allow_combo_logic
Value	yes

S.No.: 2 (Section C, Run: Run1 and Run2(with same attributes))

Parameter	cdc_qualifier_depth_start
Value	sync_chain

S.No.: 3 (Section C, Run: Run1 and Run2(with same attributes))

Parameter	enable_ac_sync_qualdepth

FIGURE 46. Parameter Values

Detailed Difference Report

This section consists of the following four sub-sections:

- In Run1 only
- In Run2 only
- Both in Run1 and Run2(with different attributes)
- Both in Run1 and Run2(with same attributes)

The following figures shows the `Both in Run1 and Run2(with different attributes)` section of the `diff.html` file:

c) Both in Run1 and Run2(with different attributes)

==========

Section B. Design Information

==========

==========

Section C. Setup Information

==========

==========

Section D. Setup Errors

==========

==========

Section E. CDC Analysis and Verification

==========

Ac_unsync01

S.No.: 1 (Section E, Rule: Ac_unsync01, Run: Run1 and Run2(with different attributes))

Attributes	Run1	Run2
Dest. Name	top.dest	top.dest
Dest. Clock Names	top.clk2	top.clk2
Source Name	top.source	top.source
Source Clock Names	top.clk1	top.clk1
Failure Reason	Qualifier not found	Qualifier not found
Sync. Scheme	N.A.	N.A.
Qualifier Name	-	-
Qualifier Depth	-	2
File:Line	top.v:19	top.v:19

FIGURE 47. Both in Run1 and Run2(with different attributes) section

This section contains detailed information about the differences between the two SpyGlass runs. The `sg_results_diff.pl` utility highlights the differences in red. For example, the above figure shows that the `Qualifier Depth` attribute of the `Ac_unsync01` rule was set to `0` and `2` in the first and second SpyGlass runs respectively.

Internal Rules in SpyGlass CDC

There are certain internal rules that run by default. The following table describes such rules.

Prerequisite Rule	Description
_cdc_save_license01	This rule performs license related operations for advanced SpyGlass CDC rules/features that require additional license in the *enable_save_restore* mode.
Ac_setOvlDataInSynthesis	This rule populates the required OVL data in the synthesis engine. It is applicable when OVL constraints are used in the SpyGlass run.
Clock_exit01	This rule performs internal memory clean up operations when all the SpyGlass CDC rules have performed the required functionality.
Ac_psync_init	This rule collects the metastability-related information on the source and destination signals of isolation enables from the design on which the *Ac_punsync01* and *Ac_psync01* rules are run.
Ac_upfsetup01	This rule collects the power domain and isolation enable data specified in the UPF format for the design on which the `Ac_punsync01` and `Ac_psync01` rules are run.

Prerequisite Rule	Description
Ac_upfsetup02	It is enabled when the `Ac_punsync01` and `Ac_psync01` rules are run. This rule reports a violation in the following cases: ● If an element specified to the *set_isolation*/ *set_level_shifter* commands does not belong to any domain. ● If an inout port/pin is specified to the *set_isolation*/ *set_level_shifter* command. ● If multiple isolation/level-shifter strategies are specified for the same domain element.
top_vs_block_val_prereq	This rule populates relevant data for the validation rules.
AcOvlRtl	This rule is run when OVL assumptions are specified in the design. It extracts initial values from the initial block and modify the object model for the *ovl_error* task.
_deltaDelayNom	This rule retains the port for the modules for which port delays are to be considered by the *DeltaDelay01*/ *DeltaDelay01* rules.
_deltaDelay	This rule parses the simulator file specified by the *simulator_file_name* parameter. The delay values are stored and used by the *DeltaDelay01* rule.
_constrCoverage	This rule computes the constraint coverage for black boxes in the design and also computes the constraint coverage for the top-level design unit. This information is used by the constraint-coverage reporting rules, such as *Setup_blackbox01*, *Setup_port01*, and *Clock_info18*.
Ar_sync_init	This rule collects reset synchronization and reset deassertion information for the reset synchronization rules, such as *Ar_sync01* and *Reset_sync02*.
_clock_hier_rules	This rule is the parent rule for the *Clock_hier01*, *Clock_hier02*, *Clock_hier03*, and *SGDC_clock_path_wrapper_module01* rules.
_propagate_cdcAttrib	This rule propagates data of the *cdc_attribute* constraint.
_debugData	This rule creates clock-domain information for nets in the data and control paths in a design.
CDCSet_License01	This rule checks the licensing for the CDC setup manager.

Prerequisite Rule	Description
_sync_qualifier	This rule detects whether a signal crossing clock domain has been synchronized by the *qualifier* constraint. It is used by clock synchronization rules.
_synci	This rule detects for synchronization by handshake. It is used by clock synchronization rules
_sync_and	This rule detects for synchronization at AND gates (based on the *enable_and_sync* parameter) in the data path of a clock domain crossing. It is used by clock synchronization rules.
_sync_gp	This rule detects for valid multi-flop synchronizer in the other fan-in (pin not connected to the source flip-flop) of a glitch protection cell in the data path (based on the *glitch_protect_cell* parameter).
_sync_clock	This rule detects for valid multi-flop synchronizers in the other fan-in (pin not connected to clock) of clock gating cells (based on the *clock_gate_cell* parameter). It is used by clock synchronization rules.
_syncg	This rule detects for synchronization where the following configuration exists: • The source flip-flop has a recirculation mux with the mux select pin acting as the source clock. • The destination flip-flop has a recirculation mux with the mux select pin acting as the destination clock. • There exists a flip-flop between the source flip-flop and the destination flip-flop that is clocked by the source clock with an inverted phase.
_syncd	This rule detects the following synchronization schemes: • *Synchronized Enable Synchronization Scheme* • *Recirculation MUX Synchronization Scheme* • *MUX-Select Sync (Without Recirculation) Synchronization Scheme*
_syncdw	This rule detects DesignWare FIFOs. This rule is used by the clock synchronization rules.
_syncUDfifo	This rule detects user-defined FIFOs in a design (based on the *fifo* constraint). It is used by the clock synchronization rules.
_syncfifo	This rule detects FIFO in the design. It is used by the clock synchronization rules.
_fifo01	This rule reports read and write pointers of FIFOs.

Prerequisite Rule	Description
_syncb	This rule detects whether a clock domain crossing is synchronized by the *Conventional Multi-Flop Synchronization Scheme*. This rule is used by the clock synchronization rules.
_syncc	This rule detects whether a clock domain crossing is synchronized by the user-specified synchronizing cell. This rule is used by the clock synchronization rules.
_syncreset_prop	This rule propagates synchronous resets in a design and computes the information that is used by other reset related rules.
Reset_prop	This rule propagates asynchronous resets in one go for domain computation. The resets specified by the *reset* constraint or those inferred automatically (when *use_inferred_resets* parameter is set to *yes*) are propagated across the design. The gated and derived resets are automatically detected and propagated. This information is useful for analysis of reset domain crossings and other resets related checks.
Clock_prop	This rule propagates clocks in one go for domain computation. The clocks specified by the *clock* constraint or those inferred automatically (when *use_inferred_clocks* parameter is set to *yes*) are propagated across the design. The gated and derived clocks are automatically detected and propagated. This information is useful for analysis of clock domain crossings and other clock related checks.
Pragma_setupb	This rule populates information about the top-level modules. This information is used by the *Clock_setup02* rule.
Pragma_setupa	This rule runs when the *Reset_check01* and *Clock_info16* rules are run. It populates relevant data structures for these rules.
Clock_setup01	This rule reads various parameters that are used in SpyGlass CDC and populates its internal data structures.
Clock_setup02	This rule reads clocks and resets from SGDC files and identifies if the design is a netlist design.
_abstract_port	This rule populates data for the *abstract_port* constraint.

Prerequisite Rule	Description
_syncCellDelayedQualifier	This rule runs if the *sync_cell* constraint is specified. It tags all the instances inside this constraint when *cdc_qualifier_depth* is specified as 0.
_rstSyncCellInst	This rule runs if the parameter is *reset_synchronize_cells* specified. It tags all the instances that lie inside the modules specified by this parameter.
_ipblock	This rule runs if the *ip_block* constraint is specified. It tags all the instances inside IP blocks.
_allowInst	This rule runs if the *allow_combo_logic* constraint is specified. It tags all the instances that are allowed as combo logic.
_portReten	This rule runs if the *allow_combo_logic* constraint is specified. It retains the port for the modules specified by this constraint.
_portRetenClocknReset	This rule runs if the *clock* and *reset* constraints are specified. It retains the ports for the pins specified by these constraints.

Rules in SpyGlass CDC

The SpyGlass CDC solution provides the following types of rules:

Setup Rules	*Formal Setup Rules*
Clock Information Rules	*Reset Information Rules*
Clock and Reset Information Rules	*Reset Synchronization Rules*
CDC Verification Rules	*Clock Glitch Checking Rules*
Clock Checking Rules	*Reset Checking Rules*
Clock and Reset Checking Rules	*Delta Delay Rules*
Block Constraint Generation Rules	*Block Abstraction Rules*
Block Constraint Validation Rules	*Synchronous Reset Verification Rules*
Must Rules	

Setup Rules

The SpyGlass CDC solution has the following setup rules:

Rule	Reports
Setup_clock01	Information needed for Clock Setup.
Setup_clockreset01	Setup issues related with clocks or asynchronous resets.
Setup_library01	Library cells with incomplete definitions
Setup_quasi_static01	Signals that are likely to be quasi-static signals in a design
Setup_port01	Summary of unconstrained ports for a top-level design unit
Setup_blackbox01	Summary of unconstrained pins for black boxes
Setup_check01	Reports if contradicting constraints are applied on objects
Setup_check02	The *signal_in_domain* constraint is applied on the objects on which the *abstract_port* constraint is applied.
Setup_req01	Status in the *CDC Matrix Report* to show if SpyGlass CDC setup requirements are followed as per the limits set by the *cdc_matrix_attributes* constraint
Ac_topology01	Status in *The Module Topology Report* that shows the dependency among the modules that are instantiated at the top level.
Ac_svasetup01	Setup issues in SVA

Setup_clock01

Generates information needed for clock setup

When to Use

Use this rule to view clock sources, clock enables, and clock cones information in a design.

Description

The *Setup_clock01* rule generates a clock setup view that shows information about clock sources, clock enables, and clock cones in a design.

The following figure shows the example of clock sources and clock cones:

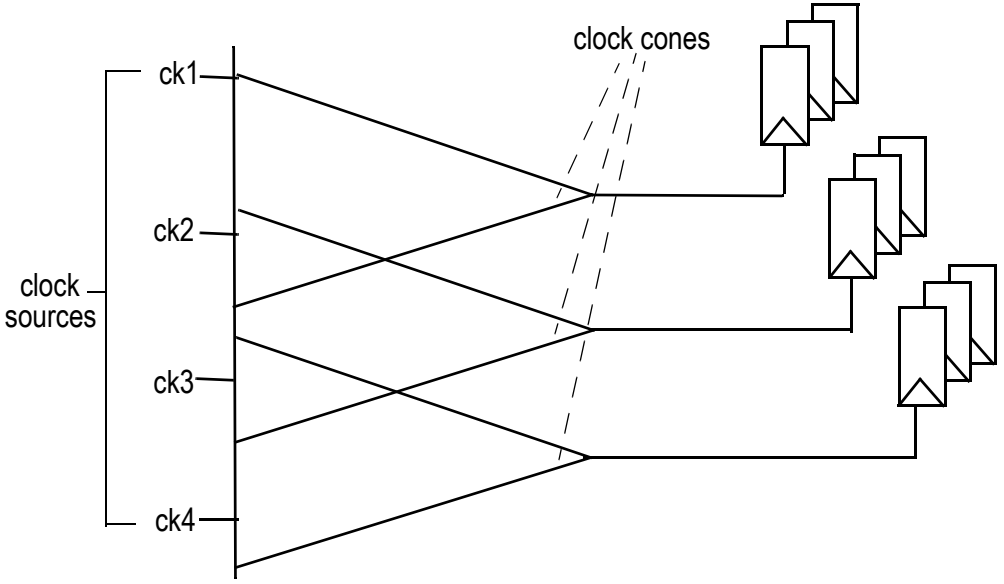


FIGURE 1. Example of Clock Sources and Clock Cones

You can view the clock setup view in the *Clock Setup* window. For details, see *Using the Clock Setup Window*.

NOTE: *By default, this rule is switched off. To enable this rule, specify the set_goal_option*

addrules {Setup_clock01} command in the project file or run the `cdc_setup` *goal available under the SpyGlass CDC solution methodology.*

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *master_clock_limit*: The default value is 1000. Set this parameter to limit the number of master clocks for which the *generated_clock* constraint should be dumped for a derived clock.
- *enable_generated_clocks*: Default value is `no`. Set this parameter to `yes` to dump *generated_clock* constraints for derived clocks.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

[SCLK01] [INFO] `For design '<design-name>', '<num-clks>' source clock(s) and '<num-clk-cones>' clock cone node(s) are identified`

The arguments of the above message are explained below:

Argument	Description
<design-name>	Top-level design unit name
<num-clks>	Number of clock sources
<num-clk-cones>	Number of clock cones

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

You can use this rule to analyze clock tree, and tune clock definitions.

Perform the following steps in the *Clock Setup* window:

1. Confirm that all the listed clocks are correct.
   - ❒ Explore the clock tree to understand the clock architecture of your design that helps you to define correct clocks.
   - ❒ Remove or add clocks, as required.
2. Confirm that all clocks are properly constrained.
   - ❒ Review all clock domains, and make sure they are correct.
   - ❒ Review and define their frequencies as needed.
3. Define case analysis to constraint the clock tree.

   In this step, the auto_case_analysis.sgdc file is generated that contains the *set_case_analysis* constraints generated for mux-select and clock enable signals in the clock path. Review this SGDC file, and specify a value in the `-value` field after un-commenting the required constraints.
4. Save the clocks constraints.

   After making all modifications, save constraints in an SGDC file by using the *Generate SGDC* as option.

Example Code and/or Schematic

Consider the following RTL code:

```
module test(input in,clk1,clk2,clk3,en1,en2,en3,output
out1,out2);

reg src, dest;

assign tt = en1;
assign qq = ~tt;
assign and1 = clk1 & qq;
assign and2 = and1 & clk3;
assign and3 = and2 & en3;
assign pp = en2;
assign rr = ~pp;
assign mux_out = (rr) ? clk2 : clk1;
```

```
always @ (posedge and3)
begin
    src <= in;
end

always @ (posedge mux_out)
begin
    dest <= in;
end
endmodule
```

When you run the Setup_clock01 rule on the above example and double-click on the message of this rule in GUI, the *Clock Setup* window is displayed. The following figure illustrates the *Clock Setup* window describing various sections in the window:

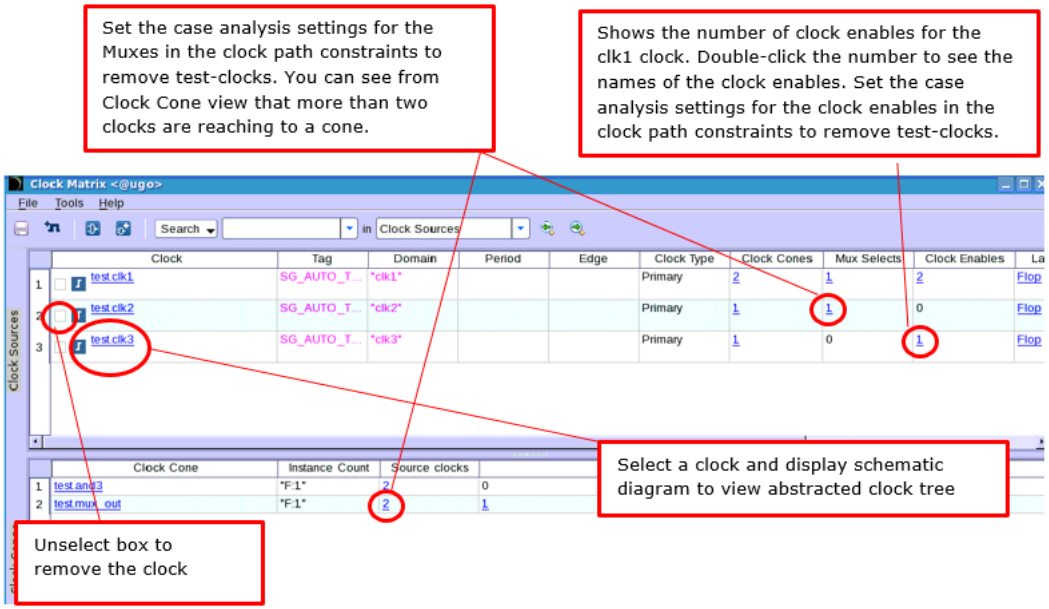


FIGURE 2. The Clock Setup Window

In the above window, you can explore the clock tree for a clock of your interest. In addition, you can add, remove, or modify clock information from this setup window.

Default Severity Label

Info

Rule Group

SETUP

Reports and Related Files

This rule generates the following files in the spyglass_reports/clock-reset/ directory:

- auto_case_analysis.sgdc

  This file contains the *set_case_analysis* constraints generated for mux-select and clock enable signals in the clock path.

  Use the *sel_case_analysis_mode* parameter to detect probable mux select and clock enable signals in the clock path where `set_case_analysis` settings are required, and save such signals in this file.

  The sample *auto_case_analysis.sgdc* file is shown below:

```
current_design "test"

############################################################
########### Dumping Mux Selects in clock path ############

############################################################

##set_case_analysis -name "test.en2" -value 0/1

############################################################

# Dumping clock enables of Combinational gates in clock
path#

############################################################

##set_case_analysis -name "test.en1" -value 0/1
##set_case_analysis -name "test.en3" -value 0/1
```

  In this file, the constraints generated are commented so that you can review them. After reviewing the generated constraints, you can uncomment the required constraints and specify the value in the -value field.

- cdc_setup_clocks.sgdc

  Contains the *clock* constraints reported by this rule.

  This file generates different information based on the following conditions:

  - If you have not provided any SGDC file, this file contains automatically-inferred clocks.
  - If you have specified both SGDC file and the *use_inferred_clocks* parameter, both SGDC clocks and clocks automatically-inferred for the rest of the sequential elements (which do not receive SGDC clocks) are dumped in this file.

- cdc_setup_generated_clocks.sgdc

  Contains the *generated_clock* constraints. This file is generated only if the *enable_generated_clocks* parameter set to `yes`.

  This file generates different information based on the following conditions:

  - If you do not specify any SGDC file, this file contains automatically-inferred generated clocks in the form of *generated_clock* constraints.
  - If you specify an SGDC file and also set the *use_inferred_clocks* parameter to `yes`, this file contains the following information:
    - *generated_clock* constraints based on the information in the specified SGDC file.
    - *generated_clock* constraints that are inferred automatically for the remaining sequential elements for which no clock information is provided in the specified SGDC file.

If you have provided only SGDC file, only those clocks are dumped in this file. In this case, there is no difference between the user-defined SGDC file and the generated SGDC file.

Setup_clockreset01

Clocks/Resets must be specified for the design

When to Use

Use this rule to check for setup issues related with clocks or asynchronous resets.

NOTE: *The Setup_clockreset01 rule is a pre-requisite rule and runs by default.*

Description

The *Setup_clockreset01* rule reports a violation if all the following conditions hold true:

For Clocks

- If no clock definition is specified either by using the *clock* constraint or by setting the *use_inferred_clocks* parameter to `yes`
- Clocks are present in the design.
- Rules that require clocks as input are enabled in the SpyGlass run.

For Asynchronous Resets

- If no reset definition is specified either by using the *reset* constraint or by setting the *use_inferred_resets* parameter to `yes`
- Asynchronous resets are present in the design.
- Rules that require asynchronous resets as input are enabled in the SpyGlass run.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears when clocks or resets are not specified for the design *<design-name>*:

[WARNING] <Clocks | Resets> have not been specified for the design '<design-name>'

Potential Issues

This violation appears if your design does not contain any defined clock or reset.

Consequences of Not Fixing

If you do not fix this violation, rules enabled in current SpyGlass run that require clocks/asynchronous reset as input are ignored and no checking is done by these rules.

How to Debug and Fix

To fix this violation, specify clock and reset signals for your design in any of the following ways:

- Clocks signals
  - By using the *clock* constraint
  - By setting the *use_inferred_clocks* parameter to `yes`
- Resets signals
  - By using the *reset* constraint
  - By setting the *use_inferred_resets* parameter to `yes`

Example Code and/or Schematic

Consider a crossing between the `clk1` and `clk2` clocks that should be reported by the *Ac_unsync01*/*Ac_unsync02* rule.

Also consider that you do not specify the *clock* constraint or set the *use_inferred_clocks* parameter to `yes`, but you enable the *Ac_unsync01*/*Ac_unsync02* rule in the current run.

In this case, the *Ac_unsync01*/*Ac_unsync02* rule does not perform any rule-

checking and the *Setup_clockreset01* rule reports a violation for missing clocks.

Default Severity Label

Warning

Rule Group

INFORMATION

Reports and Related Files

No report or related file

Setup_library01

Reports incomplete definition of library pins

When to Use

Use this rule to detect library cells of incomplete definition.

Prerequisite

Specify a library cell by using any of the following project-file commands:

```
read_file -type gateslib <library-file>
read_file -type sglib <SGLIB-file>
```

Description

The *Setup_library01* rule reports a violation for a library cell of incomplete definition.

Library Cells of Incomplete Definition

A library cell is considered of incomplete definition in any of the following cases:

- If any clock pin (specified by the `related_pin` attribute) that is associated with the input and output pins of a library cell is not driven by a top-level clock
- If a library pin has multiple timing arcs but no mode selection is provided

The summary of such pins appear in the *Rule-Based Spreadsheet of the Setup_library01 Rule*, and the details of these pins appear in the *Message-Based Spreadsheet of the Setup_library01 Rule*.

Parameter(s)

- *report_detail*: Default is `all`. The parameter supports the *Clock_check10* and the *Setup_library01* rules. Set this parameter to a supported rule to report all the violations of the specified rule and a reduced set of violations of the other supported rule. Other possible value is `none`.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears for *Library Cells of Incomplete Definition*:

```
[SLibrary1_1] [ERROR]'<num>' (<percentage>%) pin(s) for
instance '<instance-name>' of library cell '<lib-cell-name>'
have incomplete definition
```

The arguments of the above message are explained below:

Argument	Description
<num>	Number of library pins of incomplete definition. To understand the pins of incomplete definition, see *Description* of this rule.
<percentage>	Percentage of library pins of incomplete definition.
<instance-name>	Name of the instance having the library cell containing pins of incomplete definition
<lib-cell-name>	Name of the library cell containing pins of incomplete definition

Potential Issues

This violation appears if your design contains *Library Cells of Incomplete Definition*.

Consequences of Not Fixing

If you do not fix this violation, some valid clock-domain crossings may not occur in the design. That may produce undesired results.

How to Debug and Fix

Review *Message-Based Spreadsheet of the Setup_library01 Rule* and check the rows in red.

If the information reported in such rows is intentional, ignore this violation.

Else, check design connections and SGDC file to see if an incorrect value is propagating to the library pin.

For example, if a quasi static value is reaching to the clock pin of a library cell, check if that value is intentional. If it is not intentional, check if you have incorrectly specified the *quasi_static* constraint on a signal that is leading to this quasi static value on the clock pin.

Message 2

The following message appears if all pins of a library cell have complete definition:

```
[SLibrary1_2] [INFO] All the pins have complete definition for instance '<instance-name>' of library cell 'lib-cell>'
```

Potential Issues

Not applicable.

Consequences of Not Fixing

Not applicable.

How to Debug and Fix

Not applicable.

Example Code and/or Schematic

See *Rule-Based Spreadsheet of the Setup_library01 Rule* and *Message-Based Spreadsheet of the Setup_library01 Rule*.

Default Severity Label

Error

Rule Group

SETUP

Reports and Related Files

The *Setup_library01* rule generates the *Rule-Based Spreadsheet of the Setup_library01 Rule* and *Message-Based Spreadsheet of the Setup_library01 Rule*.

Rule-Based Spreadsheet of the Setup_library01 Rule

This spreadsheet shows the summary of library pins containing incomplete definition.

The following figure shows the rule-based spreadsheet of the *Setup_library01* rule:

ID	LIB-CELL	INST NAME	NUMBER OF PINS	PERCENTAGE OF PINS UNCONSTRAINED	WAIVED
D	test	"top.test_inst"	8	25	no

FIGURE 3. Rule-Based Spreadsheet of the Setup_library01 Rule

The details of the above spreadsheet are described in the following table:

Column Name	Description
LIB CELL	Name of the complex sequential library cell
INST NAME	Name of the instance containing the complex sequential library cell
NUMBER OF PINS	Total number of pins of complex sequential library cell
PERCENTAGE OF PINS UNCONSTRAINED	Percentage of library pins of incomplete definition
WAIVED	Specifies if the reported violation is waived

Message-Based Spreadsheet of the Setup_library01 Rule

This spreadsheet shows details of all the library pins of a library cell. The pins with incomplete definition appear in red, as shown in the following figure:

A ID	B Pin Name	C Type	D Clock Pins	E Top-level Clocks(domains)	F Mutiple arcs with c
1	out1	output	A(selected)	top.clk1(d0)	no
2	out2	output	C(selected)	top.clk1(d0),top.clk2(d1)	no
3	out3	output	combinational	N.A	N.A
4	A	clock	N.A	top.clk1(d0)	N.A
5	B	input	A(selected)	top.clk1(d0)	no
6	C	clock	N.A	top.clk1(d0),top.clk2(d1)	N.A
7	D	input	C(selected) A	top.clk1(d0),top.clk2(d1) top.clk1(d0)	yes
8	E	input	combinational	N.A	N.A

FIGURE 4. The Setup_library01 Spreadsheet

In the above spreadsheet, row 7 shows the details of the library pins of incomplete definition. To understand pins of incomplete definition, see *Library Cells of Incomplete Definition*.

The details of all the fields of the above spreadsheet are given below:

Field	Description
Pin Name	Specifies the name of the library pin. You can click a pin in the Pin Name column to refer to the schematic.
Type	Specifies the type, such as input and output of the library pin.
Clock Pins	Specifies the clock pin associated with the data pin of the library pin. In addition, it shows the clock pins selected in the current run as "selected". If you use - `cdc_reduce_pessimism = use_multi_arc`, it shows all arcs as "selected".
Top-level Clocks (domains)	Specifies the top-level clock reaching the clock pin. If the associated clock pin is driven by a constant, a hanging net, a quasi-static signal, or an unconstrained clock signal, that is reported in this column.
Multiple Timing Arc Without Mode Selection	Specifies *yes* or *no* to indicate if multiple arcs are selected for the library pin

The Setup_library01 rule generates the `spyglass_reports/clock-reset/Setup_library01/<libcell-name>.sgdc` file that contains the abstract_port constraints for all the library cell pins which have clocks specified by using the `related_pin` attribute.

For combinational arcs, the rule generates `abstract_port -path_type combo`. You can specify the `abstract_port` constraint for the pins that do not have the `related_pin` attribute in the description.

The Setup_library01 rule generates the `spyglass_reports/clock-reset/stop_lib_module_list` file that contains the configuration to make these library cell (for which sgdc are dumped by the rule) act as black-box in subsequent runs.

```
set_option stop "<library cell name>"
…
```

In subsequent runs, you can pass the above file to make the library cells act as black-box along with the generated SGDC files for those library cells to provide the constraints. This is required because otherwise you cannot specify the *abstract_port* constraint for library cells directly.

Setup_CGC

Reports incomplete definition of clock gating cells

When to Use

Use this rule to detect clock gating cells of incomplete definition.

Prerequisite

Specify a clock gating cell by using any of the following project-file commands:

```
read_file -type gateslib <library-file>
read_file -type sglib <SGLIB-file>
```

Description

The *Setup_CGC* rule reports a violation for a clock gating cell of incomplete definition.

Clock Gating Cells of Incomplete Definition

A CGC is considered as incompletely defined if proper attributes are not set for at least one of the pins of the cell. The must use attributes are:

- clock_gate_clock_pin
- clock_gate_enable_pin
- clock_gate_out_pin

The summary of such pins appear in the *Rule-Based Spreadsheet of the Setup_CGC Rule*, and the details of these pins appear in the *Message-Based Spreadsheet of the Setup_CGC Rule*.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears for *Clock Gating Cells of Incomplete Definition*:

```
[SCgc_error] [ERROR] '<num>' (<percentage>%) pin(s) of clock
gating cell '<clock-gate-cell-name>' have incomplete definition
```

The arguments of the above message are explained below:

Argument	Description
<num>	Number of clock gating cell pins of incomplete definition. To understand the pins of incomplete definition, see *Description* of this rule.
<percentage>	Percentage of clock gating cell pins of incomplete definition.
< clock-gate-cell-name>	Name of the clock gating cell containing pins of incomplete definition.

Potential Issues

This violation appears if your design contains *Clock Gating Cells of Incomplete Definition*.

Consequences of Not Fixing

If you do not fix this violation, some clocks may not be propagated and some valid clock-domain crossings may not occur in the design. That may produce undesired results.

How to Debug and Fix

Review *Message-Based Spreadsheet of the Setup_CGC Rule* and check the pins without any information on 'Type' column.

If the information reported in such rows is intentional, ignore this violation. Else, check the clock gating cell file and verify if all the pin attributes are properly specified or not.

Example Code and/or Schematic

See *Rule-Based Spreadsheet of the Setup_CGC Rule* and *Message-Based*

Spreadsheet of the Setup_CGC Rule.

Default Severity Label

Error

Rule Group

SETUP

Reports and Related Files

The *Setup_CGC* rule generates the *Rule-Based Spreadsheet of the Setup_CGC Rule* and *Message-Based Spreadsheet of the Setup_CGC Rule*.

Rule-Based Spreadsheet of the Setup_CGC Rule

This spreadsheet shows the summary of clock gating cell pins containing incomplete definition.

The following figure shows the rule-based spreadsheet of the *Setup_CGC* rule:

	A	B	C	D	E
	ID	CLOCK GATING CELL	NUMBER OF PINS	PERCENTAGE OF UNDETECTED PINS	WAIVED
1	3B	cglpd	4	50	No

FIGURE 5. Rule-Based Spreadsheet of the Setup_CGC Rule

The details of the above spreadsheet are described in the following table:

Column Name	Description
CLOCK GATING CELL	Name of the clock gating cell
NUMBER OF PINS	Total number of pins of clock gating cell
PERCENTAGE OF UNDETECTED PINS	Percentage of clock gating cell pins of incomplete definition
WAIVED	Specifies if the reported violation is waived

Message-Based Spreadsheet of the Setup_CGC Rule

This spreadsheet shows details of all the pins of a clock gating cell as shown in the following figure:

	A ID	B Pin Name	C Type
1	1	GCK	output
2	2	EN	-
3	3	CK	clock
4	4	TE	-

FIGURE 6. The Setup_CGC Spreadsheet

In the above spreadsheet, row 2 and 4 shows the details of the clock gating cell pins of incomplete definition. To understand pins of incomplete definition, see *Clock Gating Cells of Incomplete Definition*.

The details of all the fields of the above spreadsheet are given below:

Field	Description
Pin Name	Specifies the name of the clock gating cell pin.
Type	Specifies the type of clock gating cell pin. Possible types are, clock, enable, output or data.

Setup_quasi_static01

Reports signals that are likely to be quasi-static signals in a design

When to Use

Use this rule to identify quasi-static signals in a design.

Prerequisites

Following are the prerequisites for using this rule:

- Specify clock signals in any of the following ways:
  - By using the *clock* constraint.
  - By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals.
  - By using a combination of both the above methods.
- Use the `Advanced_CDC` license feature.

Description

The *Setup_quasi_static01* rule infers quasi-static signals in a design, and generates the *quasi_static* constraints for such signals in the *auto_quasi_static.sgdc* file.

You must review the *quasi_static* constraints in this file and use them in the subsequent SpyGlass CDC runs.

Conditions to Consider a Signal as Quasi Static

By default, the *Setup_quasi_static01* rule considers a signal as quasi static if all the following conditions hold true:

- If the signal is the source of clock-domain crossings.
- If the signal drives at least 10 sequential elements.
- If the signal has at least one clock-domain in the fan-out cone.

Use the *quasi_static_style* constraint to specify different criteria based on which SpyGlass infers quasi-static signals.

Rule Exceptions

Following are the exception of the *Setup_quasi_static01* rule:

- This rule ignores the following types of signals:
  - Signals that are specified as synchronous resets by using the *reset* `-sync` constraint.
  - Signals that are specified as quasi-static by using the *quasi_static* constraint.
- If a signal is connected to a black box pin on which no *abstract_port* constraint is defined, SpyGlass generates the *quasi_static* constraint for such signal in a commented form in the *auto_quasi_static.sgdc* file.

  Check this constraint to see if you want to define such signal as quasi-static.
- If multi-dimensional signals used inside `generate` blocks are inferred as quasi-static by this rule, the generated *auto_quasi_static.sgdc* file may contain incorrect bus-merged name strings.

  Review such signals from the generated file before declaring them as quasi-static, and append an escape character wherever required.

  To see an example of using an escape character, refer to the *Handling Nets declared in a Sequential Block* topic of *Atrenta Console User Guide*.

Parameter(s)

None

Constraint(s)

- *quasi_static_style* (Optional): Use this constraint to specify a criterion based on which SpyGlass infers quasi-static signals in a design.
- *reset* `-sync` (Optional): Use this constraint to specify reset signals in a design.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.
- *clock* (Optional): Use this constraint to specify clock signals.

Messages and Suggested Fix

The following message appears after this rule generates the *auto_quasi_static.sgdc* file containing the *quasi_static* constraints:

```
[SQs1_1] [INFO] For design '<design-name>', '<num>'
quasi_static constraint(s) inferred
```

Where, *<num>* refers to the number of *quasi_static* constraints generated.

Potential Issues

Not applicable.

Consequences of Not Fixing

If quasi-static signals are not properly constrained in a design by using the *quasi_static* constraint, SpyGlass may not analyze clock-domain crossings correctly. As a result:

- SpyGlass may report false violations resulting in noise due to missing *quasi_static* constraint.
- SpyGlass may not report a crossing if the *quasi_static* constraint is defined on a net that is not static.

How to Debug and Fix

Check the *auto_quasi_static.sgdc* and *Setup_quasi_static01_<top-level-design>.csv* files to identify quasi static signals inferred for your design.

Review the generated *quasi_static* constraints for these signals from the *auto_quasi_static.sgdc* file before using them in subsequent SpyGlass CDC runs.

Example Code and/or Schematic

Consider that you specify the following *quasi_static_style* constraint in an SGDC file for a design:

```
quasi_static_style -min_seq_fanouts 3 -min_domain_fanouts 3
-names "quasi*"
```

When you specify the above constraint, SpyGlass infers signals as quasi-static if they match all the following criteria:

- The signal has the fan-out and domain count as 3.
- The signal name starts with the string `quasi`.
- The signal is the source of a clock domain crossing (as the `-check_all_signals` argument of the *quasi_static_style* constraint is not specified).

SpyGlass generates the details of the inferred quasi-static signals in the Setup_quasi_static01_<*top-level-design*> spreadsheet and the auto_quasi_static.sgdc file, as described below:

- Setup_quasi_static01_<top-level-design>.csv

  This is a spreadsheet file, as shown in the following figure:

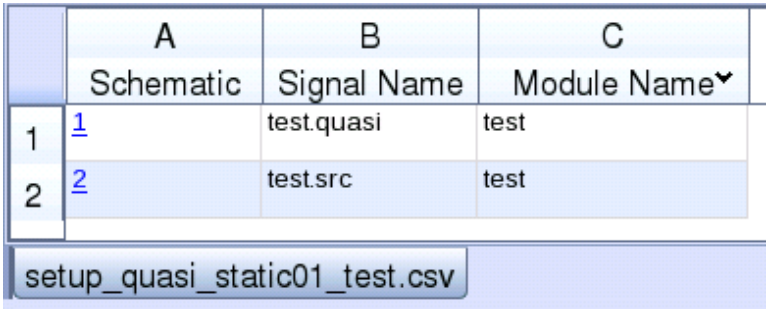


FIGURE 7. Spreadsheet Generated by the Setup_quasi_static01 Rule

The above spreadsheet lists the inferred quasi-static signals that match the criteria specified by the *quasi_static_style* constraint and the module in which these signals are found.

The spreadsheet also contains the **Sequential Fanout Count** and **Domain Count** fields when the `-report_full_count` is specified in the *quasi_static_style* constraint. When bits of quasi-static signals are reported as bus-merged, the maximum sequential count and its corresponding domain count is reported from the merged bits in the spreadsheet.

You can also view such signals in the schematic. For example, to view the `test.sb1.quasi` signal in the schematic, click *1* in the *Schematic* column of the above spreadsheet and then click . The schematic showing this signal appears, as shown in the following figure:

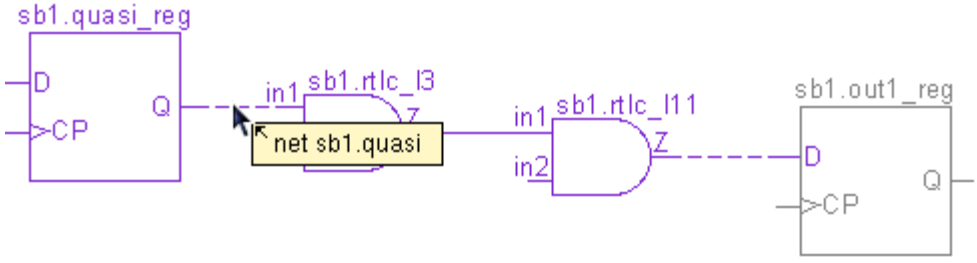


FIGURE 8. Schematic of the Setup_quasi_static01 Rule Violation

The above schematic shows the path from the quasi-static signal till one of the sequential cell.

- auto_quasi_static.sgdc

  This file contains the *quasi_static* constraints generated for the quasi-static signals inferred by the *Setup_quasi_static01* rule.

  All the *quasi_static* constraints for a module are grouped together.

  The following file is generated in this example:

```
current_design test

###Module Name :: submod
quasi_static  -name "test.sb1.quasi"
quasi_static  -name "test.sb1.quasi12"
```

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

- auto_quasi_static.sgdc

  For details, see *auto_quasi_static.sgdc*

- Setup_quasi_static01_<top-level-design>.csv
  For details, see *Setup_quasi_static01_<top-level-design>.csv*.

Setup_port01

Reports unconstrained ports summary for top-design unit

When to Use

Use this rule while performing the setup for block or SoC verification.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals.
- By using a combination of both the above methods.

Description

The *Setup_port01* rule:

- Reports the number of top-level input, inout and output ports that are not fully constrained. The details of constraint specification on each port are mentioned in *The CKSGDCInfo Report*, which is located in spyglass_reports directory.

  A port is considered constrained if the following constraints are defined:

  clock, *reset*, *set_case_analysis*, *abstract_port*, *input*, *output*, *qualifier*, or *quasi_static.*
- Generates *abstract_port* constraints on the input ports of a block. These constraints are generated in the *Input Port Constraints File*.
- Generates *abstract_port* constraints for input paths that drive the unconstrained path, blocked path, and hanging path. For example, the following constraint is generated for unconstrained path:

  ```
  abstract_port -ports i1 -ignore -comment "unconstrained clock"
  ```
- Generates *abstract_port* constraints for a path that drives a flop clocked by internal clock. For example, the following constraint is generated:

  ```
  abstract_port -ports i1 -clock SG_VIRT_CLK_1
  ```
- Generates *abstract_port* constraints for the inout ports. For example, consider the following schematic:

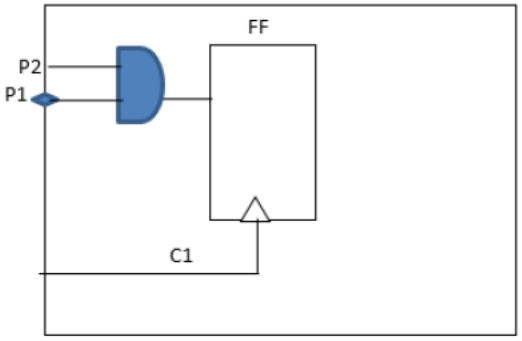


FIGURE 9.

In this case, the Setup_port01 rule generates the following constraints:

```
abstract_port -ports P2 -scope cdc -clock C1 # no change
abstract_port -ports P1 -scope cdc -clock C1 -start
-direction input # new constraint
```

Consider another example where top-level inout port P2 is connected to inout pin BP2 of BLK as shown in the following schematic:

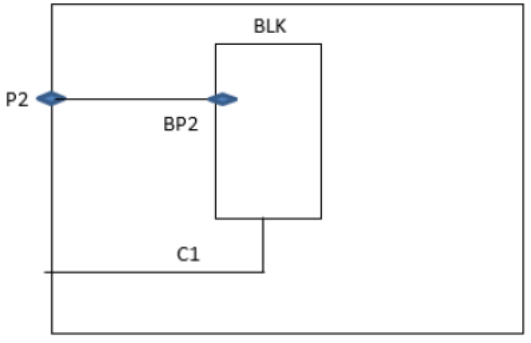


FIGURE 10.

In the above case, the following table summarizes the constraints generated by the Setup_port01 rule:

User-specified constraints	Setup_port01 generated constraints
`abstract_port -ports BP2 -clock C1 -direction input`	`abstract_port -ports P2 -clock C1 -direction input`
`abstract_port -ports BP2 -clock C1 -direction output`	No constraint generated because no fanout found for top-level port P2
`abstract_port -ports BP2 -clock C1 -direction input` `abstract_port -ports BP2 -clock C2 -direction output`	`abstract_port -ports P2 -clock C1 -direction input`
`abstract_port -ports BP2 -clock C1` `# no direction provided`	A sanity error is reported for BP2 constraint and Setup_port01 does generate any constraint
No constraint on BP2	No constraint is generated for P2

Default Rule Behavior

This rule has the following default behavior:

- *reset* or *qualifier* signals are reported as partially constrained, unless you specify *abstract_port* also.
- This rule checks only input and inout ports. To check output ports, set the *check_port_setup* parameter.
- For a bus port, all the bits of the bus are counted separately.

Parameter(s)

check_port_setup: Default is input. Set the value to output to check output ports. Set the value to all to check input, inout, and output ports. Inout ports are always checked, regardless of this parameter setting.

Constraint(s)

- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *reset* `-sync` (Optional): Use this constraint to specify reset signals in a design.

- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.
- *clock* (Optional): Use this constraint to specify clock signals.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *output* (Optional): Use this constraint to specify a clock domain at output ports.

Messages and Suggested Fix

Message 1

The following message appears after this rule checks for unconstrained ports in a top-design unit:

`[SPort1_1] [ERROR] Port coverage for top design unit <design-name> is: <num-ports> (<percentage-of-ports>) port(s) not fully constrained`

Potential Issues

The potential issues are as follows:

- When the severity of this violation message is INFO, there are no potential issues because all ports are fully constrained.
- When the severity of this violation message is ERROR, some ports are not fully constrained. Therefore, such violations should be resolved.

Consequences of Not Fixing

If some of the ports are unconstrained in the setup, the SpyGlass CDC analysis performed might not be accurate.

How to Debug and Fix

Double-click the message to view a spreadsheet that contains the

constraint information for each port. Review this spreadsheet for all ports that display **unconstrained**, **partially constrained**, **incorrect constraint**, **partially constrained (suggested -combo no)**, or **combo path** in the **Status** column.

You can click a port in the Port Name column to refer to the schematic. You can also refer the constraints by clicking a constraint in the Constraint(s) column.

A Port Name	B Port Direction	C Status	D Constraint(s)	E Inferred Data-path domain(s)
a1[0]	input	fully constrained	Multiple abstr...	-
a1[1:5]	input	unconstrained	-	top.clk2(d3) top.clk3(top.clk3)
a1[6:7]	input	fully constrained	abstract_port	-
in4[0:7]	input	fully constrained	set_case_an...	-
aexp_sel[0]	input	constraint not required (drives unconstrained path)	-	-
aexp_sel[1]	input	constraint not required (hanging)	-	-
a0[0]	input	constraint not required (hanging)	-	-
a0[1]	input	constraint not required (drives unconstrained path)	-	-
a0[2]	input	unconstrained	-	top.clk2(d3)
a0[3]	input	unconstrained	-	-

FIGURE 11. The Setup_port01 Spreadsheet

Message 2

The following message appears to indicate that the *Input Port Constraints File* is generated:

```
[SPort1_2] [INFO] Abstracted sgdc file for input ports of block
'<block-name>' is generated
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Not applicable

Default Severity Label

Error/Info

Rule Group

Setup

Reports and Related Files

- *The CKSGDCInfo Report*
- *Input Port Constraints File*

Setup_blackbox01

Reports unconstrained pins summary for black boxes

When to Use

Use this rule while performing the setup for SoC-level and when black boxes are present in the designs that have not been modeled using abstraction.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By setting the *use_inferred_clocks* parameter to `yes` to use auto generated clock signals.
- By using a combination of both the above methods.

Description

The *Setup_blackbox01* rule reports a summary of black box pins (input and output) showing the status of whether they are fully or partially constrained.

Conditions to Consider a Black-Box Input Pin as Fully Constrained

An input pin is considered as constrained if:

- The following constraints are defined on it:

  assume_path, *abstract_port*, *signal_in_domain*, *quasi_static*, *clock*, *reset*, *set_case_analysis*, or
- A top-level clock, reset, or a constant value is reaching the input pins.

Conditions to Consider a Black-Box Output Pin as Fully Constrained

An output is considered as constrained if the following constraints are defined on it:

clock, *reset*, *set_case_analysis*, *abstract_port*, *assume_path*, *signal_in_domain*, *qualifier*, or *quasi_static.*

Salient Features of the Setup_Blackbox01 Rule

This rule has the following features:

- It checks for only those black boxes that are not provided through abstract views.
- It reports the *reset* or *qualifier* signals as partially constrained, unless you specify *abstract_port* also.
- It checks for the *abstract_port* and *qualifier* constraints only on the output pins.
- Generates *abstract_port* constraints for each type of block in the design. These constraints are generated in separate files in the spyglass_reports/clock-reset/Setup_blackbox01/ directory. For example, `spyglass_reports/clock-reset/Setup_blackbox01/<block-type>_bbox_model.sgdc` file is generated for the <block-type> block.
- It reports bits of a bus port separately.

  For example, consider the following declaration of a bus port:

  ```
  output [4:0] out4;
  ```

  For the above port, this rule reports the status bits separately in the *Setup_blackbox01* rule spreadsheet, as shown in the following figure:

	A	B	C	D
	Pin Name	Pin Direction	Status	Constraint(s)
1	out4[0]	output	unconstrained	-
2	out4[1:4]	output	constraint not required (hanging)	-
3				

FIGURE 12. Reporting each bit of a bus port - The Setup_blackbox01 rule

- It checks on the first instance of a black box and ignores the remaining instances of the same black box.

Parameter(s)

report_abstract_module_coverage: Default value is `no`. Set this parameter to `yes` to enable SpyGlass CDC report the coverage of abstracted module.

Constraint(s)

- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *reset* `-sync` (Optional): Use this constraint to specify reset signals in a design.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.
- *clock* (Optional): Use this constraint to specify clock signals.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *output* (Optional): Use this constraint to specify a clock domain at output ports.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

Message 1

The following message appears after this rule checks for the black box input and output pins that are not fully constrained:

```
[SBbox1_1] [WARNING] Port coverage for black box '<black-box-name>' (instance: <instance-name>): '<num-ports>' ('<percentage-ports>') pin(s) not fully constrained.
```

Potential Issues

The following potential issues:

- When the severity of this violation message is INFO, there are no potential issues because all pins are fully constrained.

- When the severity of this violation message is WARNING, some pins are not fully constrained. Therefore, such violations should be resolved.

Consequences of Not Fixing

If some of the pins of a black box are unconstrained in the setup, SpyGlass CDC analysis performed might not be accurate.

How to Debug and Fix

Check the black box details by performing the following steps:

1. Right-click on the rule-short help in the message window, and select the *Spreadsheet Viewer* option from the shortcut menu.

   The rule-based spreadsheet appears showing all the violations of the *Setup_blackbox01* rule. The following figure shows the rule-based spreadsheet:

A	B	C	D
ID	BLACK-BOX NAME	NUMBER OF PINS	PERCENTAGE OF PINS UNCONSTRAINED
6	BlackBox1	169	97.04
7	BlackBox2	12	8.33
8	BlackBox3	12	8.33

FIGURE 13. Rule-based spreadsheet of the Setup_blackbox01 rule

   This spreadsheet displays a summary of the coverage of all the black boxes in the design.

2. Check the details, such as fully or partially constrained pins of a black box by clicking the link in the *ID* column of the black box.

   This displays the message-based spreadsheet for the selected black box.

   The following figure shows the message-based spreadsheet for *BlackBox1* in *Figure 13*. You can click a pin in the Pin Name column to refer to the schematic and a constraint in the Constraint(s) column to refer to the constraint.

A Pin Name	B Pin Direction	C Status	D Constraint(s)	E Verification clock(s)	F Validation clock(s)
out1	output	unconstrained	-	top.clk1,top.clk2,top.clk3	N.A
out2	output	constraint not required (hanging)	-	N.A	N.A
in1	input	unconstrained	-	top.clk1,top.clk2,top.clk3	N.A
in2	input	fully constrained	M abstract_port	top.clk1	N.A
clk1	input	fully constrained	M clock	N.A	N.A
clk2	input	fully constrained	M clock	N.A	N.A
clk3	input	fully constrained	clock (auto-inferred)	N.A	N.A

FIGURE 14. Message-based spreadsheet of the Setup_blackbox01 rule

You can also invoke the above spreadsheet by double-clicking on the violation of the *Setup_blackbox01* rule.

3. In the above spreadsheet, review the pins for which the status is **unconstrained** or **partially constrained**.

   For these pins, specify appropriate constraints, such as:

   - *assume_path*: If a path exists between the input and output pins of a black box, use this constraint to specify the path.
   - *signal_in_domain*: If a clock reaches the black box pin, use this constraint to specify the clock for the pin.
   - *quasi_static*: If a quasi static signal in a black box reaches the boundary of the black box, use this constraint to specify that signal.
   - *clock*/*reset*: If a clock or a reset signal in a black box reach the boundary of the black box, use these constraints.
   - *set_case_analysis*: If a constant value from a black box propagates to its boundary, specify that value using this constraint.
   - *qualifier*: If a qualifier in a black box reaches its boundary, use this constraint to specify the qualifier signal.

Message 2

The following message appears when all the pins of a black box are fully constrained:

```
[SBbox1_2] [INFO] Port coverage for black box '<black-box-
```

name>' (instance: <instance-name>): '<num-ports>' ('<percentage-ports>') pin(s) are fully constrained.

Potential Issues

This message appears when your design contains a black box with all its pins fully constrained.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Double-click on the violation of this rule to check the constraints applied on black box pins. These details appear in the message-based spreadsheet as shown in *Figure 14*.

Example Code and/or Schematic

See *How to Debug and Fix* of this rule.

Default Severity Label

Info/Warning

Rule Group

Setup

Reports and Related Files

- *The CKSGDCInfo Report*: Shows the details of constraint specification on each black box pin.
- Spreadsheet: See *How to Debug and Fix*.

Setup_check01

Reports if contradicting constraints are applied on objects

When to Use

Use this rule while performing the setup for block or SoC verification.

Description

The *Setup_check01* rule reports a violation if a combination of multiple constraints, listed as 'Yes' in the table below, is specified on a design object.

	clock/ generated _clock	reset	set_case_analysis	quasi_static	abstract_port and input/ output	signal_in_domain	qualifier
clock/ generated_clock	-	Yes	Yes	Yes	No	No	Yes
reset	Yes	-	Yes	No	No	No	Yes
set_case_analysis	Yes	Yes	-	Yes	Yes	Yes	Yes
quasi_static	Yes	No	Yes	-	No	No	Yes
abstract_port and input/output	No	No	Yes	No	-	Yes	No
signal_in_domain	No	No	Yes	No	Yes	-	No
qualifier	Yes	Yes	Yes	Yes	No	No	-

In the table above, a 'No' indicates that the *Setup_check01* rule does not report a violation if both the constraints mentioned in the corresponding row and column headings are specified on a design object.

For example, the *Setup_check01* rule reports a violation if both the *quasi_static* and the *clock* constraints are specified on a design object. Similarly, the rule does not report a violation if the *quasi_static* and the *abstract_port* constraints are specified on a design object.

Note that the *Setup_check01* rule reports a violation if the *set_case_analysis* constraint is specified together with any of the constraints mentioned in the table above.

Parameter(s)

None

Constraint(s)

- *clock*: (Optional): Use this constraint to specify clock signals.
- *generated_clock*: (Optional): Use this constraint to specify generated/ derived clocks.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *output* (Optional): Use this constraint to specify a clock domain at output ports.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The *Setup_check01* rule reports a violation message that mentions both the constraints specified on a design object. For example, the following message appears if both the *set_case_analysis* and *quasi_static* constraints are specified on a design object:

```
[WARNING] Both set_case_analysis and quasi_static constraints
have been specified on '<object-name>'
```

Consequences of Not Fixing

If you do not fix the violations reported by the *Setup_check01* rule, the

set_case_analysis constraint takes priority over the other constraint. For example in the above mentioned case, the *set_case_analysis* constraint takes priority over the *quasi_static* constraint.

How to Debug and Fix

To fix these violations, specify either of the two constraints on the violating design object.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

SETUP

Reports and Related Files

No report or related file

Setup_check02

The signal_in_domain constraint is applied on the objects on which the abstract_port constraint is already applied.

When to Use

Use this rule while performing the setup for block or SoC verification.

Description

The *Setup_check02* rule reports a violation if you apply the *signal_in_domain* constraint on the objects on which the *abstract_port* constraint is already applied.

Parameter(s)

None

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

The following message appears if the *signal_in_domain* constraint is specified on the objects on which the *abstract_port* constraint is already applied:

```
[WARNING] Both signal_in_domain and abstract_port constraint
have been specified on '<object-name>'
```

Consequences of Not Fixing

If you do not fix this violation, the *signal_in_domain* constraint on the reported object is ignored during SpyGlass analysis.

How to Debug and Fix

To fix this violation, do not specify the *signal_in_domain* constraint on the

objects on which the *abstract_port* constraint is applied.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

SETUP

Reports and Related Files

No report or related file

Setup_req01

Reports setup matrices

When To Use

Use this rule while *Creating SpyGlass CDC Setup* to check the health of the SpyGlass CDC setup.

Prerequisites

Set the limits for SpyGlass-CDC attributes by using the *cdc_matrix_attributes* constraint.

Description

The *Setup_req01* rule generates *The CDC Matrix Report* to show if SpyGlass CDC setup requirements are followed as per the limits set by the *cdc_matrix_attributes* constraint.

This rule provides an overview of the SpyGlass CDC setup of the design before *Performing CDC Verification*. The rule generates SpyGlass CDC specific statistics (in the form of CDC attributes) to check the health of the setup and reports a violation if the statistics exceed the limits specified by the *cdc_matrix_attributes* constraint.

Knowing the CDC attributes information while *Creating SpyGlass CDC Setup* enables you to fix issues that cause the limit to exceed. This way, you can ensure that the SpyGlass CDC specific statistics are good enough to proceed with *Performing CDC Verification*.

Parameter(s)

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.

- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer, same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

Constraint(s)

cdc_matrix_attributes (Mandatory): Use this constraint to set a limit for SpyGlass-CDC attributes.

Messages and Suggested Fix

Message 1

The following message appears to specify the number *<num>* of CDC attributes that exceed the prescribed limit:

```
[ACM1_2] [ERROR] <num> prescribed setup requirements violated
```

Potential Issues

This violation appears when certain CDC attributes are beyond the limit set by the *cdc_matrix_attributes* constraint.

If the *Exceeded Limit* column of *Section A* of *The CDC Matrix Report* shows the value *No*, no potential issues are considered as the CDC attributes are within the prescribed limit.

Consequences of Not Fixing

Based on the attribute that has exceeded the specified limit, the comprehensive SpyGlass CDC analysis may contain noise and may have high performance requirement.

For example, consider that you specify 10 clocks but *The CDC Matrix Report* reports 14 clocks. In this case, more clock domain crossings are reported using these extra four clocks. Therefore, analyze and verify if the reported clocks are correct.

How to Debug and Fix

Perform the following steps to fix the issues caused due to exceeded limit:

1. Review clocks or generated clocks.

   Analyze the unexpected clocks by reviewing the following:

   - Clocks specified in the given SGDC file
   - Automatically-generated clocks reported in *autoclocks.sgdc* when the *use_inferred_clocks* parameter is set to `yes`
   - *Propagate_Clocks* violation
   - *The CKTree Report*

2. Review asynchronous resets.

   Analyze the unexpected asynchronous resets by reviewing the following:

   - Resets specified in the given SGDC file
   - Automatically-generated resets reported in *autoresets.sgdc* when the *use_inferred_resets* parameter is set to `yes`
   - *Propagate_Resets* violation
   - *The RSTree Report*

3. Review synchronous resets. Analyze the unexpected synchronous resets by reviewing the following:

   - Resets specified in the given SGDC file
   - *The SyncRstTree Report*

4. Review domain limit.

   Check if you have specified correct domains for clocks.

5. Review the crossings or source per destination or crossing per clock pair limit

   If the number of crossings per destination or per clock pair is very high and exceeds the limit, there are high chances that the setup is incorrect. Therefore, check if you have specified correct constraints, such as *quasi_static* and *set_case_analysis*.

Message 2

The following message appears to specify that all the SpyGlass-CDC attributes follow the prescribed limit:

```
[AcM1_1] [INFO] All prescribed setup requirements met
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

//test.v

```
module dff (data, clk, reset, q);
 input data, clk, reset ;
 output q;
 reg q;
 always @ ( posedge clk )
  if (~reset) begin
   q <= 1'b0;
  end
  else begin
   q <= data;
  end
endmodule
module top(data,clk1,clk2,clk3,clk4,
clk5,clk6,clk7,clk8,reset1,reset2,reset3,q);
input data, clk1, clk2,clk3,clk4,clk5,clk6,
clk7,clk8,reset1,reset2,reset3;
output q;
wire gateClk1,
r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,
r14,r15,r16,r17,r18,gateClk2,gateClk3,
gateClk4,gateClk5;
assign r8=r1 || r2 || r3;
        assign r9=r4 || r5 || r6 || r7;
        assign r10=r8 & r9;
assign gateClk1=r12 || r13 || r14;
assign gateClk2=r15 & r16 & r17;
assign gateClk3=gateClk1 & gateClk2;
        dff U1(data,clk1,reset1,r1);
        dff U2(data,clk1,reset2,r2);
        dff U3(data,clk3,reset1,r3);
        dff U4(data,clk1,reset1,r4);
        dff U5(data,clk2,reset2,r5);
        dff U6(data,clk2,reset1,r6);
        dff U7(data,clk4,reset1,r7);
        dff U8(r10,clk3,reset3,r11);
        dff U9(r11,clk4,reset2,r12);
        dff U10(r11,clk4,reset2,r13);
        dff U11(r11,clk4,reset2,r14);
        dff U12(r11,clk5,reset2,r15);
        dff U13(r11,clk4,reset2,r16);
        dff U14(r11,clk7,reset2,r17);
       dff U15(gateClk3,clk5,reset1,r18);
       dff U16(r18,clk6,reset2,q);
       dff U17(r18,clk7,reset2,q);
      dff U18(r18,clk6,reset1,q);
endmodule
```

//test.sgdc

```
current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d2
clock -name clk3 -domain d3
clock -name clk4 -domain d4
clock -name clk5 -domain d5
clock -name clk6 -domain d6
clock -name clk7 -domain d7
reset -name reset1 -sync
```

//attributes.sgdc

```
current_design top
cdc_matrix_attributes
-src_clock_limit 2
-gen_clock_limit 2
-sync_reset_limit 1
-async_reset_limit 1
-domain_limit 1
-crossing_limit 8
-src_per_dest_limit 0
-crossing_per_clock_pair_limit 0
```

For the above example, the *Setup_req01* rule generates the following *cdc_matrix* report:

```
Section A : CDC matrix summary that does not follow the prescribed limit
*********************************************************************
---------------------------------------------------------------------
  S. No.    Type                Number     Prescribed     Exceeded
                                           Limit          Limit
---------------------------------------------------------------------
    1.      Primary Clocks       7            2           YES
    2.      Sync Resets          1            1           NO
    3.      Async Resets         0            1           NO
    4.      Generated Domains    7            1           YES
    5.      Crossings            11           8           YES
---------------------------------------------------------------------

Section B : Top 5 crossings that have source count exceeding the limit of 0
************************************************
------------------------------------------------
 S. No.    Destination-          Number Of Sources
           Instance/Port
------------------------------------------------
  1.       top.U8.q_reg           6
  2.       top.U15.q_reg          5
  3.       top.U13.q_reg          1
  4.       top.U11.q_reg          1
  5.       top.U10.q_reg          1
------------------------------------------------

Section C : Top 5 crossings between clocks exceeding the limit of 0
*************************************************************************************
-------------------------------------------------------------------------------------
 S. No. Source Clock(s) Dest Clock(s) Source Domain  Dest Domain  Number Of
                                      Id             Id           Crossings
-------------------------------------------------------------------------------------
  1.    top.clk3        top.clk4       2             3            4
  2.    top.clk4        top.clk5       3             4            4
  3.    top.clk1        top.clk3       0             2            3
  4.    top.clk2        top.clk3       1             2            2
  5.    top.clk5        top.clk6       4             5            2
-------------------------------------------------------------------------------------
```

FIGURE 15. The CDC Matrix Report

For information on the sections in the above report, see *The CDC Matrix Report*.

Default Severity Label

Error | Info

Rule Group

SETUP

Reports and Related Files

The CDC Matrix Report

Ac_topology01

Generates a module topology report for the blocks instantiated at the top level

When To Use

Use this rule to during the top-down constraint generation for blocks and generation of abstract views of blocks.

Description

The *Ac_topology01* rule generates *The Module Topology Report* that shows the dependency among the modules that are instantiated at the top level.

Checking this dependency helps you to identify the modules for which the top-down constraint migration should be done first followed by generating their abstract view. For details, see *Example Code and/or Schematic*.

The Need to Know Blocks Dependency

Top-down constraint generation works at the SoC level, and therefore, requires loading the entire design. This may cause capacity issues for large SoC designs.

This arises the need to perform top-down constraint generation in steps so that all the blocks are not considered for top-down constraint generation in one go.

So the need is to generate constraints for blocks one-by-one using the top-down constraint migration. However, you cannot pick blocks in a random order as there can be dependency among the blocks. To know this dependency, the *Ac_topology01* rule is used.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

This rule reports the following message:

```
[INFO] Module topology for design '<design-name>' generated.
Refer to report module_topology.rpt for details
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Example 1

Consider the following file specified for SpyGlass analysis:

```
module top(in1, in2 , in3, out1, out2);
  input in1, in2 , in3;
  output out1, out2;
  wire b1_out1,b1_out2;
  wire b2_out1,b2_out2;
  block b1(.in1(in1), .in2(in2), .out1(b1_out1), .out2(b1_out2));
  block b2(.in1(b1_out1), .in2(b1_out2), .out1(b2_out1), .out2(b2_out2));
  block b3(.in1(b2_out1), .in2(b2_out2), .out1(out1), .out2(out2));
endmodule
module block(in1, in2 ,out1, out2);
  input in1, in2;
  output out1, out2;
  assign out1 = in1 & in2;
  assign out2 = in1 & in2;
endmodule
```

For the above example, the *Ac_topology01* rule generates the following *module_topology* report:

S. No.	Instance Name	Module Name	Topological Order	Verification Order
1.	b1	block	1	1
2.	b2	block	2	2
3.	b3	block	3	3

FIGURE 16. The Module Topology Report

In the above report, the *Topological order of instances* indicates the following dependency between the `b1`, `b2`, and `b3` instances:

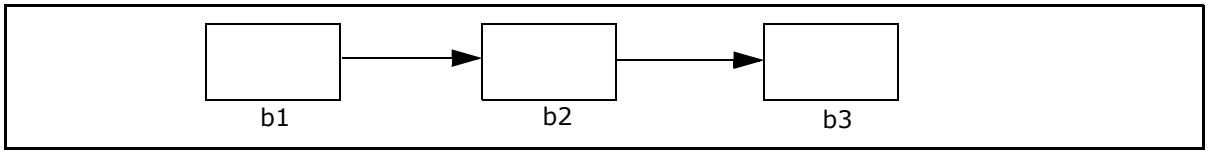


FIGURE 17. Example of module dependency

Top-Down Constraint Migration Based on module_topology Report

Based on the dependency shown in *Figure 17*, generate constraint for instances in the sequence described in the following steps:

1. Black box the `b2` and `b3` modules and generate constraints for `b1` by using the top-down constraint migration.

   Once the constraints for A are generated, generate the abstract view for `b1`.

2. Black box the `b1` and `b3` modules and generate constraints for `b2` by using the top-down constraint migration and the abstract view of `b1`.

   Once the constraints for `b2` are generated, generate the abstract view for `b2`.

3. Black box the `b1` and `b2` modules and generate constraints for `b3` by using the top-down constraint migration and the abstract views of `b1` and `b2`.

   Once the constraints for `b3` are generated, generate the abstract view for `b3`.

Example 2

Consider the following figure showing dependency between instances:

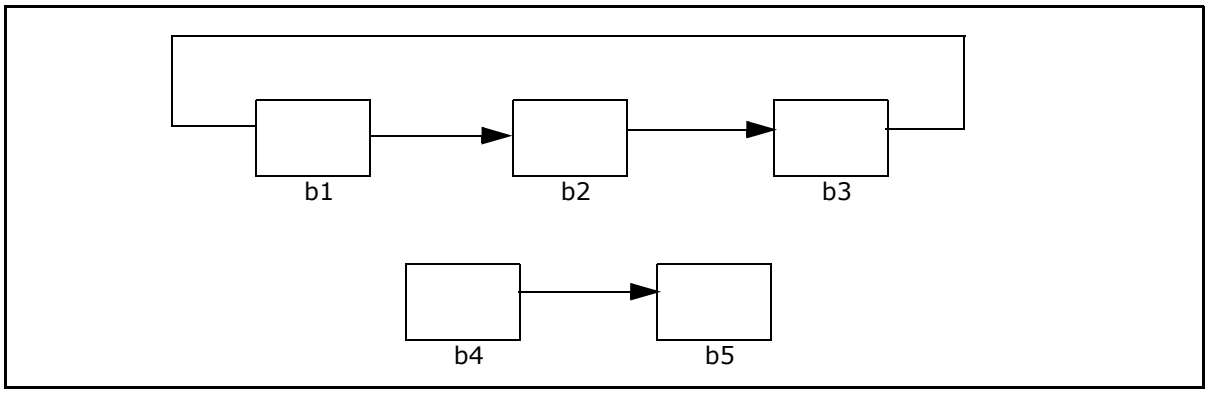


FIGURE 18. Modules Dependency

In the above example, the `b1`, `b2`, and `b3` instances have circular dependency. Therefore, they are assigned the same topological order as 1 in *The Module Topology Report*.

However, topological ordering is possible between `b4` and `b5` as `b5` is dependent on `b4`. So `b4` and `b5` are assigned the ordering as 4 and 5, respectively.

The following figure shows *The Module Topology Report* for this scenario:

S. No.	Instance Name	Module Name	Topological Order	Verification Order
1.	b1	block	1	1
2.	b2	block	1	2
3.	b3	block	1	3
4.	b4	block	2	4
5.	b5	block	3	5

FIGURE 19. Modules Dependency

Based on the *Verification order of instances* in the above report, perform top-down constraint migration similar to the steps described in *Top-Down Constraint Migration Based on module_topology Report*.

Default Severity Label

Info

Rule Group

SETUP

Reports and Related Files

The Module Topology Report

Ac_svasetup01

Setup issues in SVA constraints

When to Use

Use this rule to parse SVA constraints and report issues related with these constraints.

Prerequisites

Specify the following project-file command:

```
set_option enableSVA yes
```

Description

The *Ac_svasetup01* rule parses SVA constraints and reports issues related with these constraints.

For details, refer to the *Using SystemVerilog Assertions* application note.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

This rule reports different messages based on the issues in SVA constraints. All these messages are described in the *Using SystemVerilog Assertions* application note.

Potential Issues

Refer to the *Using SystemVerilog Assertions* application note.

Consequences of Not Fixing

Refer to the *Using SystemVerilog Assertions* application note.

How to Debug and Fix

Refer to the *Using SystemVerilog Assertions* application note.

Example Code and/or Schematic

Refer to the *Using SystemVerilog Assertions* application note.

Default Severity Label

Warning

Rule Group

SETUP

Reports and Related Files

No report or related file

Formal Setup Rules

The SpyGlass Formal Setup rules are as follows:

Rule	Reports
Ac_clockperiod01	Any of the missing -period or -edge arguments of the *clock* constraint.
Ac_clockperiod02	Clocks whose periods are optimized by SpyGlass for lower design cycle. This helps in faster functional analysis.
Ac_resetvalue01	Missing `-value` field of the *reset* constraint defined in an SGDC file
Ac_sanity01	Issues with the user-specified property files (using the fa_propfile parameter)
Ac_sanity02	Non-tristate nets that have multiple drivers
Ac_sanity03	Loops in the design
Ac_sanity04	Over-constraining in a design
Ac_sanity06	Issues in distributed computing flow

The Formal Setup Rules are used for the setup of the *CDC Verification Rules*. These rules are run through SpyGlass CDC solution Setup Manager and are also a part of the `cdc_verify` goal. Please refer to *SpyGlass CDC solution Methodology Guide* for details.

Ac_clockperiod01

Reports a violation if any of the -period or -edge argument is not specified in the clock constraint

When to Use

Use this rule to detect clocks for which any of the period or edge value is not defined.

Prerequisites

Enable this rule by specifying the following command in a project file:

```
set_goal_option addrules Ac_clockperiod01
```

By default, this rule is switched off.

Description

The *Ac_clockperiod01* rule reports a violation if any of the `-period` or `-edge` argument is missing in the *clock* constraint.

You must specify both the arguments to the *clock* constraint.

By default, SpyGlass considers the value 10 ns for a period and a half duty cycle for an edge.

Parameter(s)

check_edge: The default value is yes. Set this parameter to `no` to report a violation if none of the `-period` or `-edge` argument is specified.

Constraint(s)

clock (Mandatory): Use this constraint to specify clock signals.

Messages and Suggested Fix

The following message appears if any of the *-period* or *-edge* argument is missing in the *clock* constraint:

```
[AcCP1_1] [WARNING] For top design unit '<du-name>', <period | edge> is not defined for <num> (<percentage>%) clocks. Refer file: '<file-name>' for details
```

The arguments of the above message are explained below:

Argument	Description
<du-name>	Name of the design unit
<num>	Number of clocks for which the period/edge is not defined
<percentage>	Percentage of clocks for which period/edge has not been defined
<file-name>	Name of the spreadsheet file

Potential Issues

This violation appears if your design contains clocks for which any of the period or edge value is not defined.

Consequences of Not Fixing

Clocks defined without any of the period or edge value may result in inaccurate functional analysis.

Secondly, for the same period clock crossings, the *Ac_cdc01* rules perform data hold checks. However, if it is the case of automatically-inferred periods, the *Ac_cdc01* rule checks are not performed on such crossings. This may result in wrong analysis of a design as cases of potential data loss may get missed in such cases.

How to Debug and Fix

To debug this violation, perform the following steps:

1. Open the *Spreadsheet Viewer* of this rule.
2. In the spreadsheet, check the clocks for which the *Period* and/or *Edge list* columns are blank.
3. Add appropriate values in the *-period* and *-edge* arguments for such clocks in their *clock* constraint specification in the SGDC file.

Example Code and/or Schematic

Consider the following specification of the *clock* constraint in an SGDC file:

```
clock -name clk1 -value rtz
```

In the above case, the *Ac_clockperiod01* reports a violation as no period and edge is defined for the `clk1` clock.

Default Severity Label

Warning

Rule Group

`ADV_CLOCKS`

Report and Related File

- *The CDC Report*
- Ac_clockperiod01.csv: This file contains details of clock names, their periods, and edge list.

Ac_clockperiod02

Reports clocks for which period optimization has been done

When to Use

Use this rule to detect clocks whose period values are optimized.

Prerequisites

- Enable this rule by using the following command in a project file:

  ```
  set_goal_option addrules Ac_clockperiod02
  ```

- Run any of the *CDC Verification Rules* or set the *formal_setup_rules_check* parameter to `yes`.

Description

The *Ac_clockperiod02* rule reports clocks whose periods are optimized by SpyGlass for lower design cycle. This helps in faster functional analysis.

Parameter(s)

None

Constraint(s)

clock (Mandatory): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears if period values of clocks have been optimized:

[AcCP2_1] [INFO] For top design unit '<du-name>', Period has been rounded for <num> (<percentage>) clocks. Refer file: '<file-name>' for details

The arguments of the above message are explained below:

Argument	Description
<du-name>	Name of the design unit
<num>	Number of clocks whose period values are optimized
<percentage>	Percentage of clocks whose period values are optimized
<file-name>	Name of the spreadsheet file

Potential Issues

This violation appears if your design contains clocks for which period values have been optimized.

Consequences of Not Fixing

Optimization of period values may result in a slight mismatch in expected results.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the *Spreadsheet Viewer* for this rule.
2. In the spreadsheet, check the clocks whose periods have been optimized by SpyGlass for lower design cycle.
3. Review and modify the period/edge list in the *clock* constraint specification in the SGDC file if the periods are to be adjusted correctly to maintain the initial clock period ratio.

Example Code and/or Schematic

Consider an example in which the clock periods of two clocks belonging to the same domain are `3.33` and `6.61`. In this case, SpyGlass may optimize the clock periods to `3.33` and `6.66`, respectively, to optimize the design period and design cycle. This rule reports such clocks in a design.

Default Severity Label

Info

Rule Group

`ADV_CLOCKS`

Report and Related File

- *The CDC Report*
- Ac_clockperiod02.csv: This file contains details of clocks, their actual period values, and their optimized values.

Ac_resetvalue01

Reports a violation if the -value argument is not specified for the reset constraint.

When to Use

Use this rule to detect resets for which no active value is specified.

Prerequisites

Enable this rule by using the following command in a project file:

```
set_goal_option addrules Ac_resetvalue01
```

By default, this rule is switched off.

Description

The *Ac_resetvalue01* rule reports a violation if the *-value* argument of the *reset* constraint is not specified.

By default, SpyGlass assumes the value of such resets to be active high.

Parameter(s)

None

Constraint(s)

reset (Mandatory): Use this constraint to specify reset signals.

Messages and Suggested Fix

The following message appears to report design units that have resets without active values:

```
[AcRv1_1] [WARNING] For top design unit '<du-name>', active value is not defined for <num> (<percentage>%) resets. Refer file: '<file-name>' for details
```

The arguments of the above message are explained below:

Argument	Description
<du-name>	Name of the design unit
<num>	Number of resets for which the active value is not specified
<percentage>	Percentage of resets for which the active value is not specified
<file-name>	Name of the spreadsheet file

Potential Issues

This violation appears if your design contains resets for which no active value is defined.

Consequences of Not Fixing

Resets without a proper active value may result in inaccurate functional analysis.

During functional analysis, resets are turned off and put into an inactive state. Therefore, if there is a reset for which the active value is incorrectly provided, the entire functional analysis can go wrong.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the *Spreadsheet Viewer* window of this rule.
2. In the spreadsheet, check the resets for which the *value* column is blank.
3. Specify the correct active value in the *-value* field for such resets in their *reset* constraint specification in the SGDC file.

Example Code and/or Schematic

Consider the following reset constraint specification:

```
reset -name r1
```

For the above example, the *Ac_resetvalue01* rule reports a violation as the `-value` argument is not specified for the reset constraint.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Report and Related File

- *The CDC Report*
- Ac_resetvalue01.csv: This spreadsheet file contains details of resets and their type (synchronous/asynchronous).

Ac_sanity03

Reports loops in a design

When to Use

Use this rule during functional analysis to detect loops in a design.

Prerequisites

Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_sanity03* rule reports the following loops in a design:

- All combinational loops
- Loops involving clock to Q, preset to Q, or clear to Q paths of a flip-flop

If you want to check over constraining due to unstable combinational loops, run the *Ac_sanity04* rule.

NOTE: *By default, this rule is not run.*

Parameter(s)

None

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if a loop involving the net *<net-name>* is present in a design:

`[WARNING] Loops involving net '<net-name>' detected`

In the above message, name of the first-found user net, *<net-name>*, in an unstable loop is reported. In case of internally generated nets, *<synth_gen_net>* is displayed.

Potential Issues

This violation appears if your design contains loops.

Consequences of Not Fixing

Functional analysis cannot be performed in the presence of unstable combinational loops.

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following design containing a combinational loop:

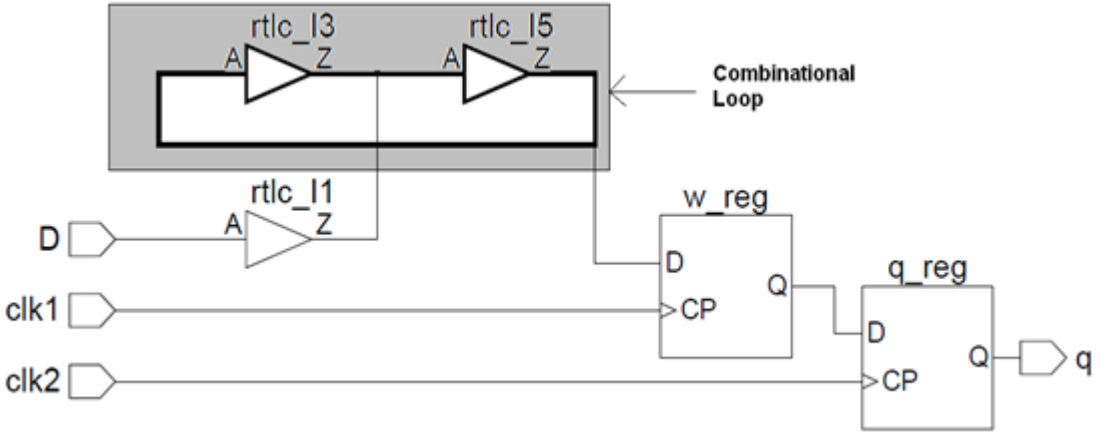


FIGURE 20. Loops in a Design

The *Ac_sanity03* rule reports a violation in the above case.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

No related files and reports

Ac_sanity04

Reports over-constraining in a design

When to Use

Use this rule in the pre-layout phase of a design to detect over-constraining in the design.

Prerequisites

Following are the prerequisites for running this rule:

- Run any of the *CDC Verification Rules* while running this rule.
- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_sanity04* rule reports over-constraining in a design that can arise in any of the following situations:

- If there are conflicting *set_case_analysis* and *reset* constraints on the same net

  Functional analysis is done with an inactive value of a reset. Therefore, if a *reset* active value and *set_case_analysis* are set to the same value for the same net, this rule reports a violation.
- If there are conflicting *set_case_analysis* and *clock* constraints on the same net
- If there are conflicting *reset* and *clock* constraints on the same net
- If there are unstable combinational loops
- If there are conflicting OVL constraints

NOTE: *By default, the Ac_sanity04 rule is switched off.*

Parameter(s)

None

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if a set of conflicting constraints are specified for a design:

```
[ERROR] There are conflicting constraints. Refer
file:'<file-name>' for details
```

Where *<file-name>* is the name of the file containing the conflicting constraints information.

Potential Issues

This violation appears if a set of conflicting constraints are specified for a design.

Consequences of Not Fixing

If you do not fix this violation, the rule does not perform any formal checks and may stop further processing.

How to Debug and Fix

To fix this violation, identify conflicting constraints in the *Overconstrain Info File* and resolve them.

Example Code and/or Schematic

Example 1

Consider the following constraints specified for a design:

```
set_case_analysis -name top.rst -value 0
reset -name top.rst -value 0
```

For the above example, the *Ac_sanity04* rule reports a violation because the active value specified by the *reset* and the *set_case_analysis* constraint is same for the same net.

Example 2

Consider the following constraints specified for a design:

```
clock -name clk1 -period 5
reset -name clk1 -value 0
```

For the above example, the *Ac_sanity04* rule reports a violation because of the conflicting *reset* and *clock* constraints on the same net.

Example 3

Consider the following figure:

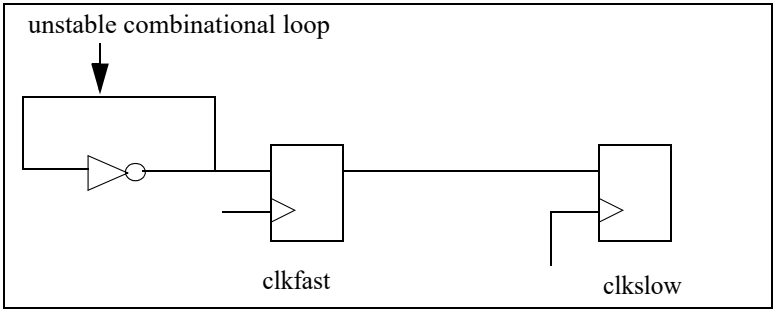


FIGURE 21. Example of an Unstable Combinational Loop

For the above example, the *Ac_sanity04* rule reports a violation because of the presence of an unstable combinational loop.

Example 4

Consider the following files specified during SpyGlass analysis:

```
//Verilog File                                                    //SGDC file

module top(input D,clk1,clk2,rst,output reg q);                   current_design top
reg w;
always @(posedge clk1 or negedge rst)                             clock -name clk1 -period 5
   if(!rst)                                                       clock -name clk2 -period 10
     w<=1'b0;                                                     reset -name rst -value 0
   else w<=D;
always @(posedge clk2 or negedge rst)
   if(!rst)
     w<=1 'b0;
   else q<=w;
assert_proposition #(0,1) constraint (1'b1,!rst); //Ties rst to 0
endmodule
```

For the above example, the *Ac_sanity04* rule reports a violation because of conflicting OVL constraints.

Default Severity Label

Error

Rule Group

ADV_CLOCKS

Reports and Related Files

Overconstrain Info File

Ac_sanity07

Reports synchronous clocks having the maximum cycle count greater than the specified limit

When to Use

Use this rule during functional analysis to detect the clocks having the *Maximum Cycle Count* greater than the specified limit.

Prerequisites

Specify clock signals by using the *clock* constraint.

Description

The *Ac_sanity07* rule reports the same domain synchronous clocks for which the *Maximum Cycle Count* is greater than the limit set by the *fa_c2c_max_cycles* parameter.

Cycle Count

A cycle count refers to the number of clock edges required to synchronize two clocks.

NOTE: *Edge calculation is done with respect to the lower bound of the design period.*

Maximum Cycle Count

The maximum cycle count is the total *Cycle Count* of the fastest clock and the slowest clock.

The fastest clock refers to the clock having the minimum period value and the slowest clock refers to the clock having the maximum period value.

Parameter(s)

fa_c2c_max_cycles: The default value is 100. Specify a positive integer value to set a limit for the *Maximum Cycle Count*.

Constraint(s)

clock (Mandatory): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the *Maximum Cycle Count* of the same domain synchronous clocks exceeds the limit specified by the *fa_c2c_max_cycles* parameter:

```
[WARNING] Clocks from domain '<domain-name>' have max cycle
'<max-cycle-count>', which is greater than the specified limit
of '<limit>'
```

Potential Issues

This violation appears if your design contains same domain synchronous clocks for which the *Maximum Cycle Count* is greater than the limit specified by the *fa_c2c_max_cycles* parameter.

Consequences of Not Fixing

If you do not fix this violation, the *Design Period* may be high resulting in a bad QoR of *CDC Verification Rules*.

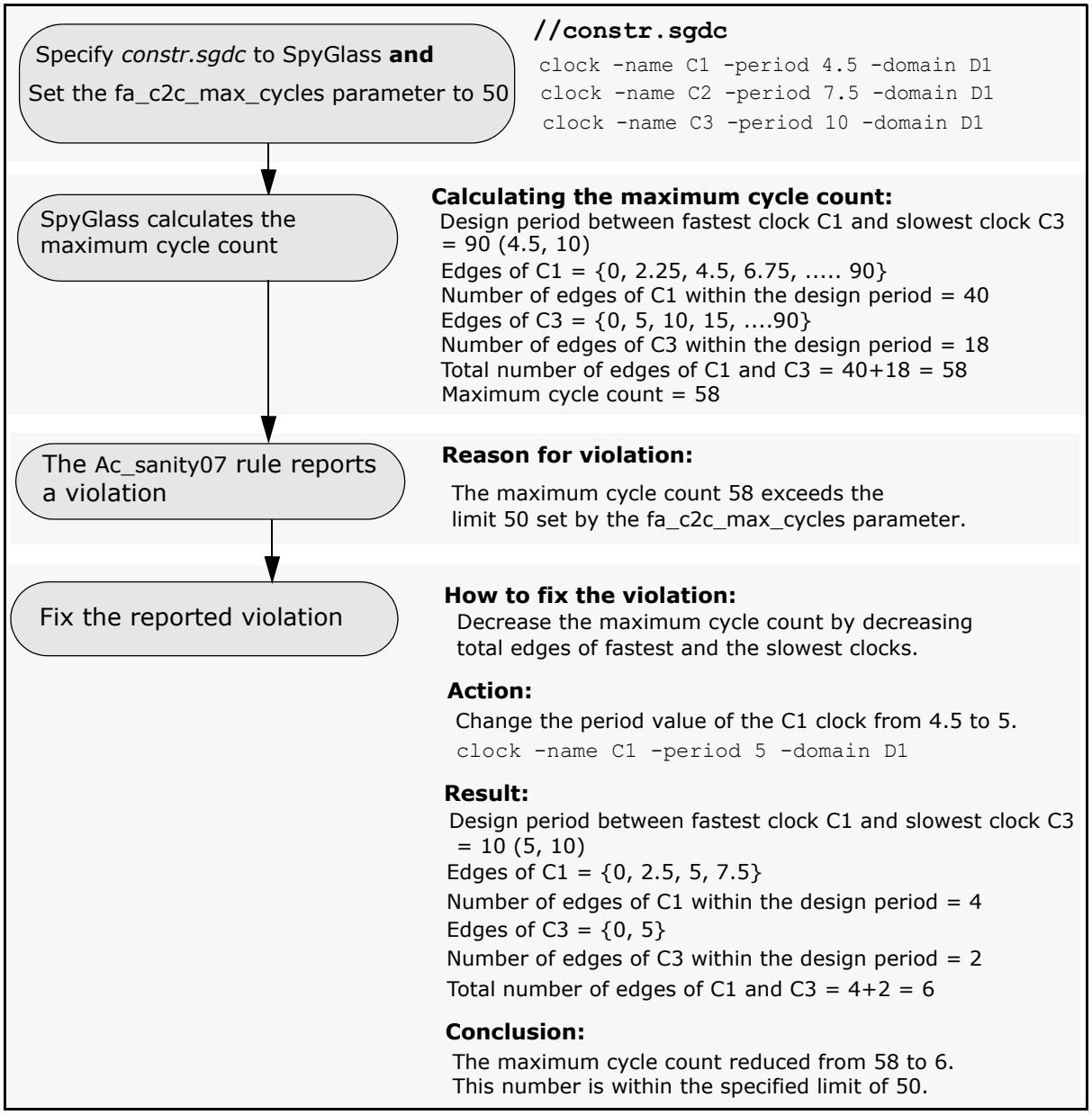
How to Debug and Fix

To fix this violation, modify the clock periods of the fastest and/or the slowest clocks such that the total number of edges (*Maximum Cycle Count*) of these clocks comes within the limit set by the *fa_c2c_max_cycles* parameter.

NOTE: *Modify the clock periods such that they are multiples of each other.*

See *Example Code and/or Schematic*.

Example Code and/or Schematic



Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

No report or related file

Clock Information Rules

The SpyGlass CDC solution has the following rules for generating clock information:

Rule	Reports
Clock_info01	Clock candidates in the design
Clock_info02	Prints the clock tree for specified clock signals
Clock_info03	Runs *Clock_info03a*, *Clock_info03b*, and *Clock_info03c* rules
Clock_info03a	Unconstrained clock nets
Clock_info03b	Flip-flops/latches where the data pins are tied to a constant value
Clock_info03c	Flip-flops or latches where the clock/enable pin is set to a constant
Clock_info05	MUX descriptions where two or more clock signals converge
Clock_info05a	Signals on which the set_case_analysis should be set to control MUXed clock selection
Clock_info05b	Combinational gates other than MUXes where two or more clock signals converge
Clock_info06	Clocks that are derived from specified clocks after frequency division
Clock_info07	Specified clocks that are derived from other clocks after frequency division
Clock_info14	Generates the data required to highlight specified clock domains in GUI windows using different colors
Clock_info15	Generates clock domain information for primary ports
Clock_info16	MUX descriptions without the Synopsys `infer_mux` pragma set where two or more clock signals converge
Clock_info17	Reports all synchronous clocks present inside a particular hierarchy
Clock_info18	Reports the number and percentage of input and output ports that are not constrained using input and output constraints

Clock_info01

Reports inferred signals that are likely to be clock signals

When to Use

Use this rule to:

- Find clocks in your design.
- View schematic back-annotation information, which is useful in distinguishing clocks from clock-gating logic.

Description

The *Clock_info01* rule reports different types of clocks in a design.

This rule traces back clock/enable signals of sequential elements till one of the following is reached:

Traced to	Clock Type
Primary inputs or undriven output pins of a library cell that have a defined clock attribute	Primary Clocks
Black box instances and instances of ASIC cells whose functional description is not available	Black box Clocks
Outputs of flip-flops and instances of ASIC cells	Derived Clocks
Hanging nets, clocks derived from disabled latches and tristates	Undriven Clocks
Outputs of latches or tristate or combinational gates if the fan-in of the gate does not have a Primary, black box, Derived, or Undriven clock	Gated Clock

Clock candidates reported by this rule can be:

- Used by other SpyGlass CDC solution rules.
- Used to verify that your design has all intended clock signals.

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a

MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *ignore_latches*: Default value is `yes`. Set this parameter to `no` to consider signals ending at latch enable terminals while deciding clock candidates
- *ignore_bus_clocks*: Default value is `1024`. Set this parameter to any positive integer value to ignore all vector signals of bus width greater or equal to that number. Other possible values are `yes` and `no`.
- *master_clock_limit*: The default value is 1000. Set this parameter to limit the number of master clocks for which the *generated_clock* constraint should be generated.
- *filter_named_clocks*: Default value is `rst reset, scan, set`. Set this parameter to a list of strings.
- *enable_generated_clocks*: Default value is `no`. Set this parameter to `yes` to dump *generated_clock* constraint for derived clocks.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *infer_constraint_from_abstract_blocks*: Default value is `no`. Set this parameter to a supported value to enable the supported rules infer clock, reset, set_case_analysis constraints from similar constraint defined in abstracted blocks.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location where a clock candidate *<clk-name>* of type *<clk-type>* is first used in a design:

[INFO] `Candidate clock: <clk-name> of type: <clk-type>`

Where *<clk-type>* can be `Primary Clock`, `Derived Clock`, `Gated Clock`, or `Undriven Clock`.

NOTE: *If the clock is a black box, Message 2 is reported.*

Potential Issues

This violation appears if your design contains clocks, such as primary clocks, derived clocks, gated clocks, or undriven clocks.

Consequences of Not Fixing

None

How to Debug and Fix

For information on debugging, click *How To Debug and Fix*.

Message 2

The following message appears at the location where a clock candidate *<clk-name>* of black box type is first used in a design:

`[INFO] Candidate clock: <clk-name> of type: Blackbox Clock (cell name: <bb-du-name>)`

Where *<bb-du-name>* is name of the black box cell name.

Potential Issues

This violation appears if your design contains clocks of a black box type.

Consequences of Not Fixing

None

How To Debug and Fix

To debug the violation reported by this rule, perform the following steps:

1. Open the spyglass_reports/clock-reset/autoclocks.sgdc file, and review the clocks reported as probable clocks.
2. View the *Incremental Schematic* of the violation message.
3. In the schematic, check the clock that displays path of the clock till one sequential element to which it is driving.
4. If this is not a real clock, remove this clock from the SGDC file.
5. Copy the final SGDC file to desired location.
6. You can also view case analysis settings along with the violation of this rule.

You can run this rule to find clocks in a design. In addition, the schematic back-annotation information generated by this rule is useful in distinguishing clocks from a clock-gating logic.

If you do not want to view messages by this rule, do not run this rule or waive this rule.

Example Code and/or Schematic

This rule detects all types of clock candidates in the following example:

```
module top (d, q, clk1, sr);
  input [3:0]d;
  input clk1, sr;
  output [3:0]q;

  reg [3:0]q;
  reg clk3;
  wire clk2, clk4, w1;

  BB I1(clk1, clk2, w1);

  always @(posedge clk1)
    q[0] = d[0];

  always @(negedge clk2)
    q[1] = d[1];

  always @(posedge clk3)
    q[2] = d[2];

  always @(posedge clk1)
    if (sr==1)
      clk3 = 0;
    else
      clk3 = ~clk3;

  always @(posedge clk4)
    q[3] = d[3];
endmodule
```

In the above example:

- The `clk1` clock signal of the `q[0]` flip-flop can be traced to the primary input, `clk1`. Therefore, this rule reports the following message:

```
Candidate clock: top.clk1 of type: Primary Clock
```

Following is the modular schematic for this message:

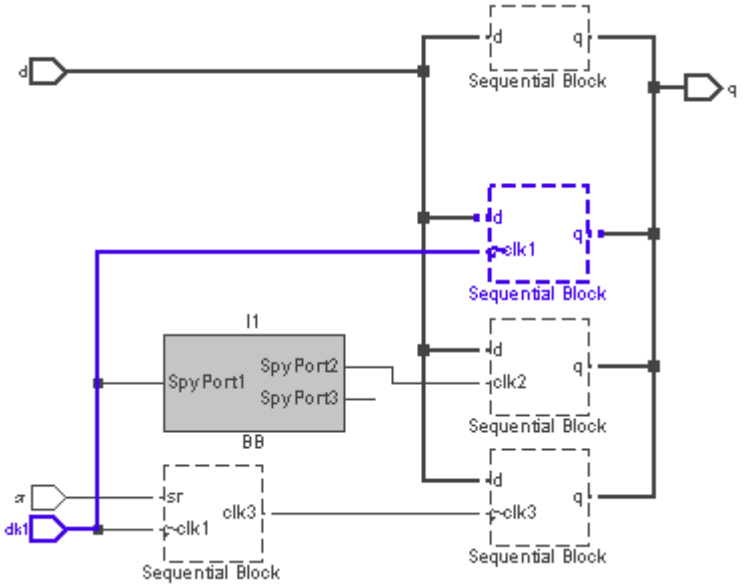


FIGURE 22. Example 1 - Modular Schematic for Clock_info01 Rule Violation

- The `clk2` clock signal of the `q[1]` flip-flop can be traced to the output of a black box instance, `I1`, of the `BB` black box. Therefore, this rule reports the following message:

  ```
  Candidate clock: top.clk2 of type: BlackBox Clock (cell
  name:BB)
  ```

  Following is the modular schematic for this message:

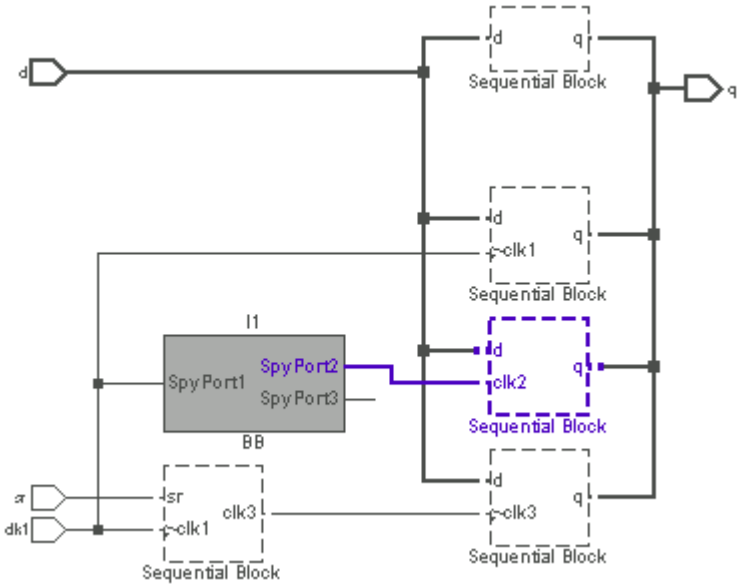


FIGURE 23. Example 2 - Modular Schematic for Clock_info01 Rule Violation

- The `clk3` clock signal of the `q[2]` flip-flop can be traced to the output of a flip-flop. Therefore, this rule reports the following message:

  ```
  Candidate clock: top.clk3 of type: Derived Clock
  ```

  Following is the modular schematic for this message:

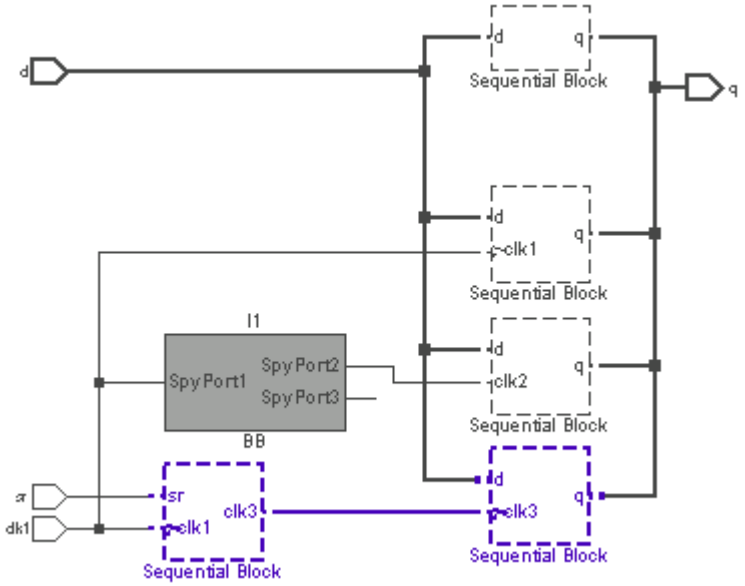


FIGURE 24. Example 3 - Modular Schematic for Clock_info01 Rule Violation

- The `clk4` clock signal of the `q[3]` flip-flop is a hanging net. Therefore, this rule reports the following message:

  ```
  Candidate clock: top.clk4 of type: Undriven Clock
  ```

  Following is the modular schematic for this message:

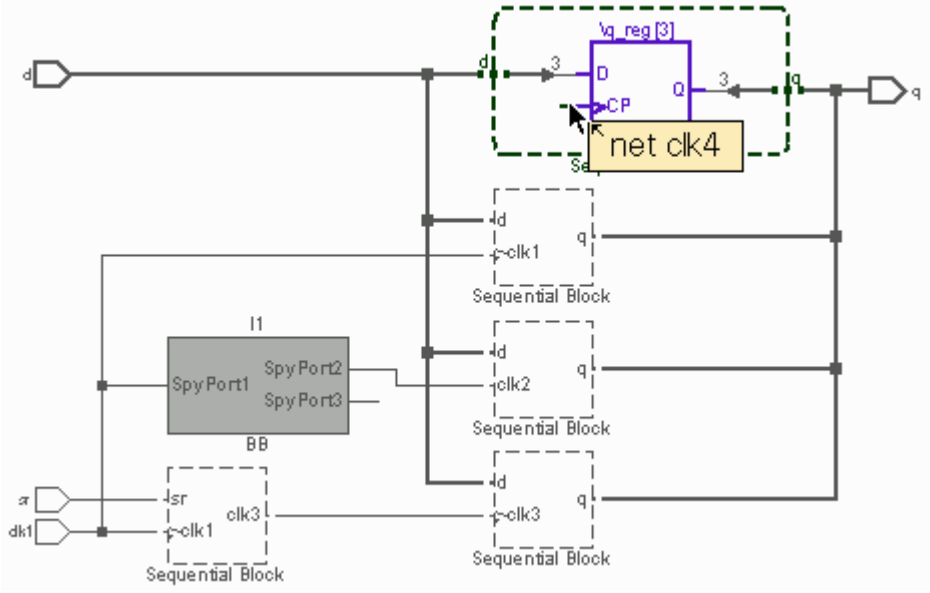


FIGURE 25. Example 4 - Modular Schematic for Clock_info01 Rule Violation

Schematic Details

The *Clock_info01* rule highlights the path from a clock source to a clock pin of a flip-flop.

Default Severity Label

Info

Rule Group

FIND

Reports and Related Files

- *The Clock-Reset-Summary Report*
- *autoclocks.sgdc*

  This file contains the following information:

  ❒ All primary and black box clocks with their respective domain names

- Gated clocks that are output of latches

  Definite and probable clocks are reported in separate sections in this file.

  A clock is reported as definite when it directly connects to a clock pin of a flip-flop or through a combinational logic where there is only one definite path existent (constrained by the *set_case_analysis* constraint). All other clocks are reported as probable.

  Undriven clocks are not reported in the autoclocks.sgdc file as clocks are not expected to be undriven.

- *generated_clocks.sgdc*

  This file is an SGDC format file that contains information about derived clocks. These clocks are not reported in the autoclocks.sgdc file.

  Such clocks can be controlled only through their respective source clocks (primary clocks or black box clocks).

  This file contains information about derived clocks in the form of the following constraints:

  - *clock*

    This constraint is generated if the *enable_generated_clocks* parameter is set to `no`.

  - *generated_clock*

    This constraint is generated if the *enable_generated_clocks* parameter is set to `yes`.

    If the source clock of the generated clock has multiple master clocks, the *generated_clock* constraint is generated for one master clock.

    However, if `all_master_clocks` is specified to the *clock_reduce_pessimism* parameter, the *generated_clock* constraints are generated with respect to all the master clocks.

    To set a limit of maximum number of master clocks for which the *generated_clock* constraints should be generated, use the *master_clock_limit* parameter.

Clock_info02

Prints a clock tree for the specified clock signals

When to Use

Use this rule to view a clock tree that hierarchically lists various clocks in your design.

Prerequisites

Specify clock signals by using the *clock* constraint and/or the *use_inferred_clocks* parameter.

Description

The *Clock_info02* rule prints a clock tree displaying clock signals in your design.

NOTE: *This is a runtime-intensive rule. Therefore, you are recommended not to run this rule in general. You should run this rule only when you view a clock tree.*

Rule Functioning

While traversing a design to generate clock trees, this rule does not traverse beyond sequential objects and black box instances found in the path. If you specify a library cell without a function that can be inferred as a sequential cell, this rule does not traverse beyond this cell instance. However, for combinational library cells, this rule traverses beyond the cell instance.

Depending upon the complexity of the library cell and its fan-out logic, the traversal through all outputs of the library cell instance may take too long and may result in a long runtime.

Therefore, it is recommended that you do not provide a library cell definition for such design units, such as memory gate cells for which the corresponding outputs are not expected to be driving clock nets in any case.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears to indicate that a clock tree has been generated:

`[INFO] Clock Tree generated`

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Click on the message in the GUI. The design clock tree appears, as shown in the following figure:

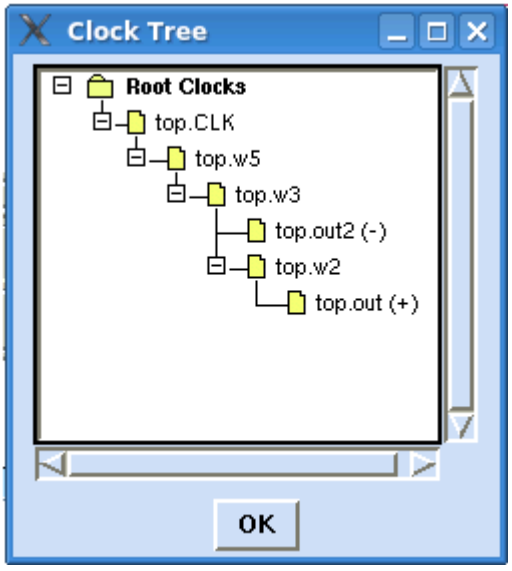


FIGURE 26. Clock Tree Generated by the Clock_info02 Rule

2. Explore the clock tree by expanding a clock of your interest.

   For leaf-level cells, such as flip-flops, latches, and black boxes, the triggering clock edge is indicated as + for positive edge and - for negative edge.
3. Open *The CKTree Report* to view data in a text format.
4. Open *The CKCondensedTree Report* to view a condensed view of the tree that shows only sequential instance count at each level.
5. Use the *Propagate_Clocks* rule if you are interested in viewing *Incremental Schematic* of a clock.

Example Code and/or Schematic

Consider the following design file and SGDC file provided as input for SpyGlass analysis:

// test.v

```
module test (d,clk1,clk2,q);
input d,
      clk1,
      clk2;
output q;
wire d,
     clk1,
     clk2;
reg q1, //internal reg
    q2, //internal reg
    q;
always @(posedge clk1)
begin
q1 <=d;
end
always @(negedge clk2)
begin
q2 <=q1;
end
always @(posedge clk2)
begin
q <= q2;
end
endmodule
```

// constr.sgdc

```
current_design test
clock -name clk1 -domain d1
clock -name clk2 -domain d2
-testclock
```

After running SpyGlass analysis, double-click on the violation of the *Clock_info02* rule. The following clock tree appears:

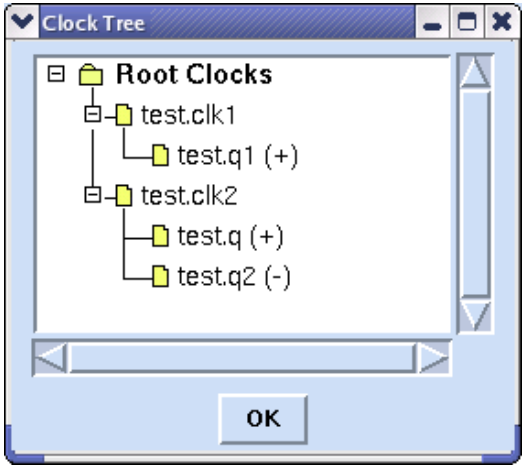


FIGURE 27. Example - Clock Tree Generated by the Clock_info02 Rule

Default Severity Label

Info

Rule Group

FIND

Report and Related File

- *The CKTree Report*
- *The CKCondensedTree Report*

Clock_info03

Cases not checked for clock domain crossings: Unconstrained clocks

Description

The Clock_info03 rule runs the *Clock_info03a*, *Clock_info03b*, and *Clock_info03c* rules.

Clock_info03a

Reports unconstrained clock nets

When to Use

Use this rule to identify cases that are not checked for clock domain crossings due to unconstrained clock nets.

Description

The *Clock_info03a* rule reports a violation in the following cases:

- If a clock net is driven by a combinational logic that blocks a clock defined in an SGDC file:

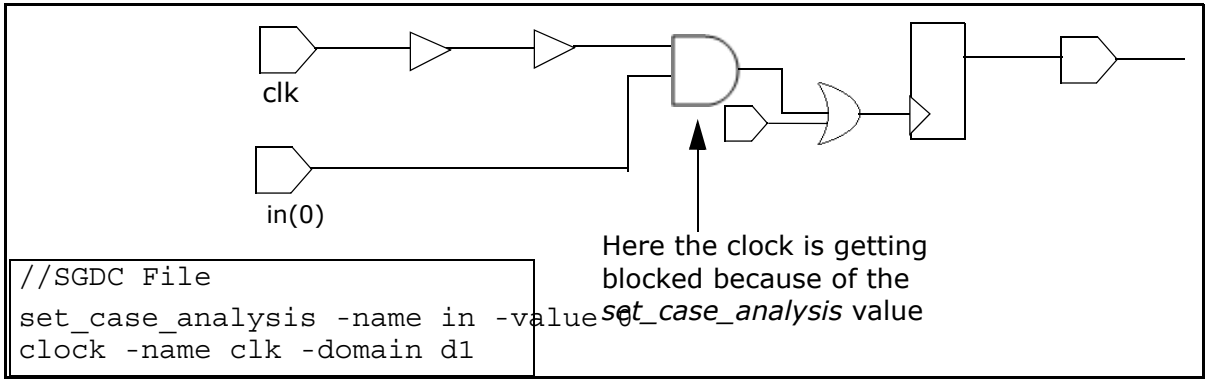


FIGURE 28. Clock_info03a Rule - Example 1

- If a clock net is driven by a signal that is a clock candidate but is not defined in an SGDC file, and the *use_inferred_clocks* parameter is set to `no`:

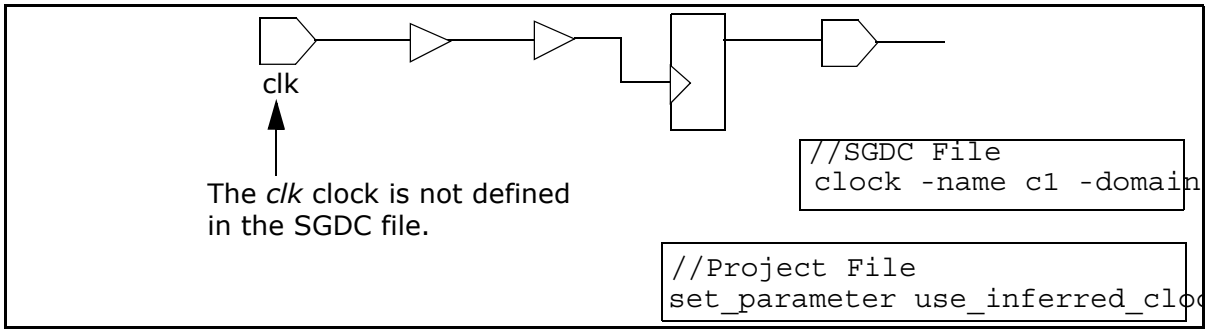


FIGURE 29. Clock_info03a Rule - Example 2

- If a clock net is driven by a black box and a clock defined in an SGDC file is reaching to this black box. Such clock is not propagated beyond the black box.

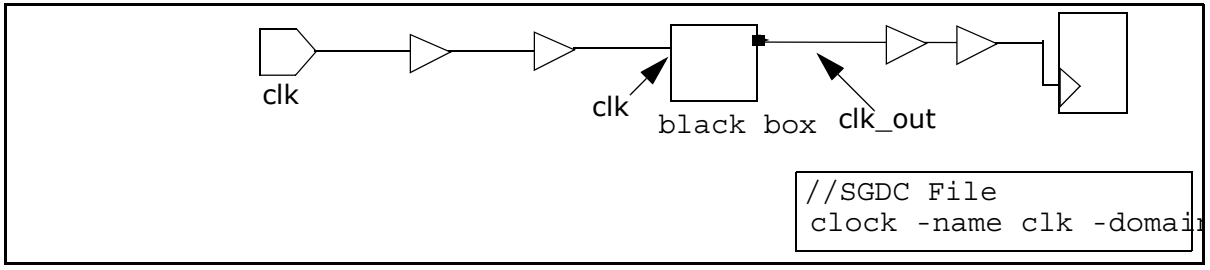


FIGURE 30. Clock_info03a Rule - Example 3

- If the design contains an unconnected clock net:

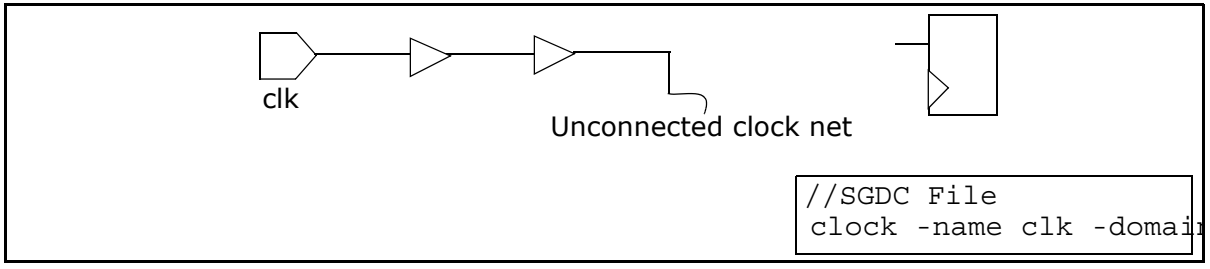


FIGURE 31. Clock_info03a Rule - Example 4

- If a clock net is driven by a latch output and the *clock_reduce_pessimism* parameter includes `latch_en`:

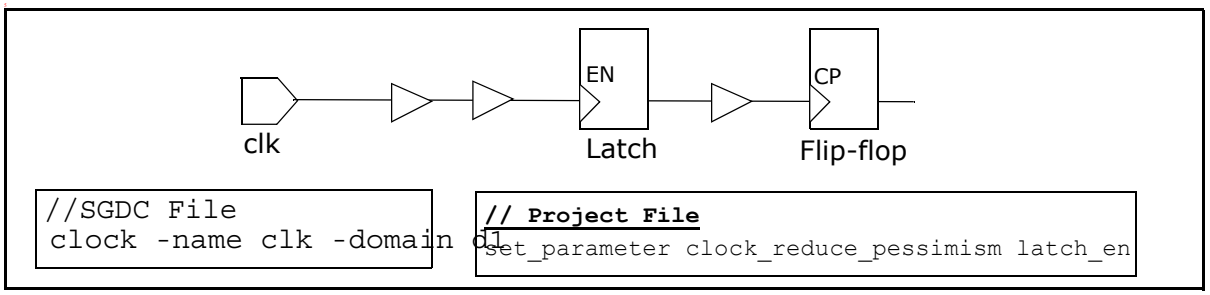


FIGURE 32. Clock_info03a Rule - Example 5

Rule Exceptions

This rule does not report a violation if you specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Parameter(s)

- *ignore_latches*: Default value is `yes`. Set this parameter to `no` to consider signals ending at latch enable terminals.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use the auto-generated clock information.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *cdc_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a

MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *report_common_clock*: Default value is `no`. Set this parameter to `yes` to report common clock source. If there is any buffer or inverter in the input side of the clock-pin of a flop, the rule ignores it and reports the next available net.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where an instance has unconstrained clock *<clk-name>* reaching to it:

[WARNING] Clock-Net '<clk-name>' is unconstrained

Potential Issues

This violation appears if your design contains unconstrained clock nets for flip-flops, latches, and sequential cells in a design.

Consequences of Not Fixing

If you do not fix this violation, clock domain crossing checks are not checked for unconstrained sequential elements.

In addition, if a clock is blocked and it does not reach to sequential elements, SpyGlass may generate incorrect functional results.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Back-trace the clock net that is reported as un-constrained.

Perform appropriate actions based on the following possibilities:

- The reported net is driven by a black box, and the clock defined in the SGDC file is reaching to the input of this black box. This typically happens with a PLL or similar clock generation cells.

  Action: Set the *assume_path* constraint or specify the *clock* constraint on the output of a black box. However, make sure that you set a domain for that clock to be the same domain as the clock driving the input of the black box.

  This can also happen in a gate-level netlist if you miss to supply a gate-library (defining that gate) to SpyGlass. Ensure that you compile the library by using the SpyGlass library compiler before using it.

- The reported net is unconnected.

  Action: Update the design.

- The reported net is driven by combinational logic that blocks the clock defined in the SGDC file.

  Action: Enable *Show Case Analysis* in the *Incremental Schematic* window, and check if constant propagation blocks the defined clock net.

- The reported net was driven by a signal that is a clock candidate, but is not defined in the SGDC file.

  Action: Refer to the messages of the *Propagate_Clocks* rule.

3. You can also view the list of unconstrained clock nets in *Section D* of *The Clock-Reset-Summary Report*.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

```
// test.v

module top2(d,d1,in,q,q1,clk1,clk2);
  input d,d1,clk1,clk2,in;
  output q,q1;
  wire clk = clk1 & in;
  reg q,q1;
  always @(posedge clk)
     q<=d;
  always @(posedge clk1)
     q1<=d1;
endmodule
```

```
// constraints.sgdc

current_design top1
clock -name "top1.clk"
set_case_analysis
   -name top1.en -value 0

current_design top
clock -name "top.clock"

current_design top2
clock -name "top2.clk2"
```

For the above example, the *Clock_info03a* rule reports a violation as the `top2.clk` clock-net is unconstrained.

The schematic of this violation is shown in the following figure:

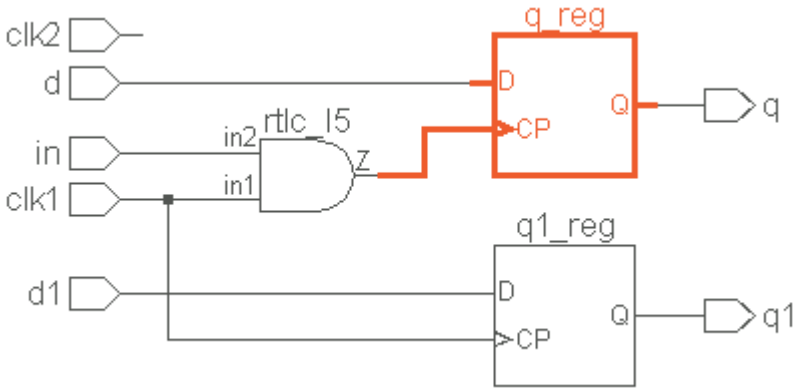


FIGURE 33. Schematic of Clock_info03a Rule Violation

To fix this violation, constraint the `clk1` input port.

Schematic Details

The *Clock_info03a* rule highlights the path from a clock net to a flip-flop clock pin or a latch enable pin.

Default Severity Label

Warning

Rule Group

INFORMATION

Reports and Related Files

The Clock-Reset-Summary Report

Clock_info03b

Reports sequential elements whose data pin is tied to a constant

When to Use

Use this rule to identify cases that are not checked for clock domain crossings as the data pin of a flip-flop/latch is tied to a constant.

Description

The *Clock_info03b* reports flip-flops and latches that cannot change value because of any of the following reasons, and therefore are not checked for clock domain crossings:

- Data is tied to a constant and no clear or reset.
- Data is tied to a 0 and only a clear pin.
- Data is tied to a 1 and only a set pin.

The following figure shows such flip-flops:

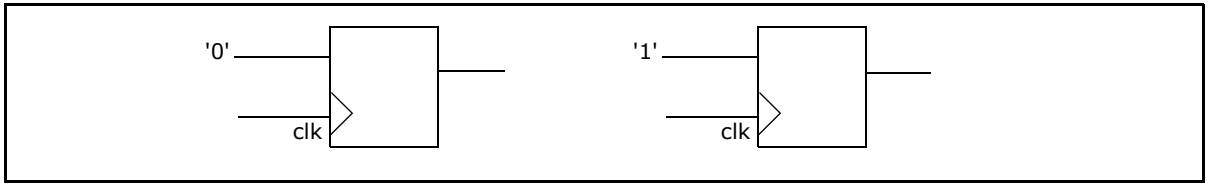


FIGURE 34. Data Pin of Flip-Flops Tied to a Constant Value

Such sequential elements are checked for clock domain crossings only if they contain asynchronous set/clear pins.

Rule Functioning in Case of Flip-Flops with a Reset Pin

For flip-flops with a reset pin, this rule functions in the following manner:

- Reports flip-flops with a clear pin if the data pin is tied to 0, as shown in the following figure:

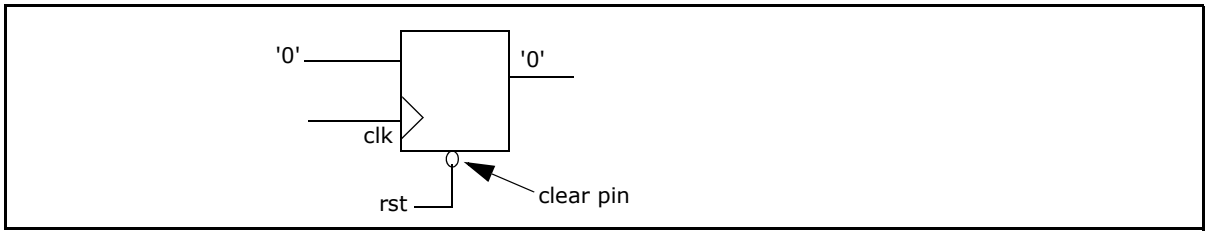


FIGURE 35. Flip-Flop with a Clear Pin and the Data Pin Tied to 0

A violation appears in the above scenario because the output always remains 0 irrespective of the value at clock and clear pins.

- Reports flip-flops with a preset pin if the data pin is tied to 1, as shown in the following figure:

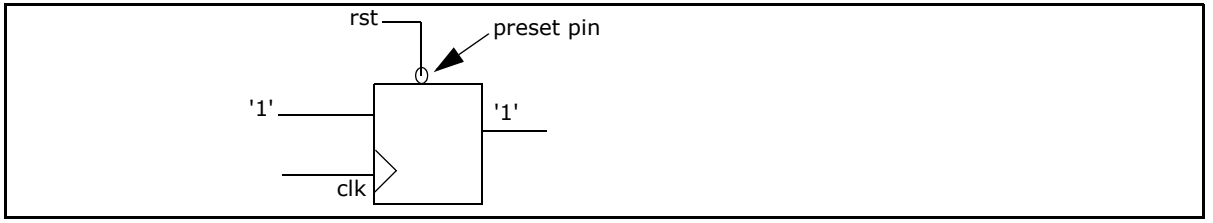


FIGURE 36. Flip-Flop with a Preset Pin and the Data Pin Tied to 1

A violation appears in the above scenario because the output always remains 1 irrespective of the value at clock and preset pins.

- Does not report flip-flops with a clear pin if the data pin is tied to 1, as shown in the following figure:

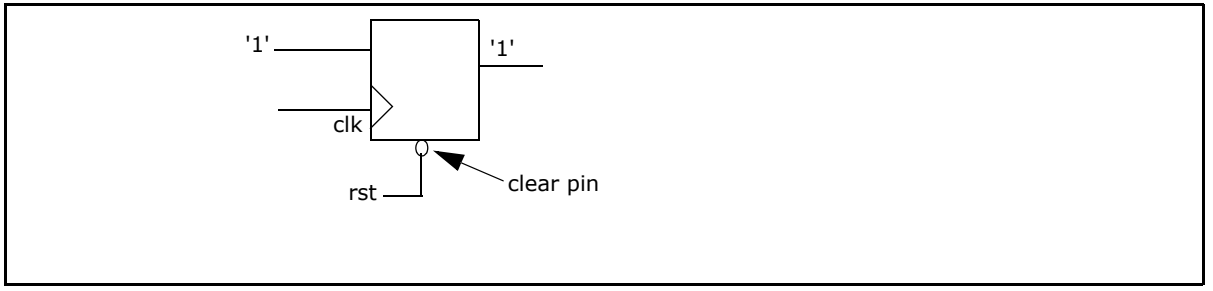


FIGURE 37. Flip-Flop with a Clear Pin and the Data Pin Tied to 1

No violation appears in the above scenario because when the clear pin changes from 1 to 0, the output will change to 1 at the next clock positive/negative edge.

- Does not report flip-flops with a preset pin if the data pin is tied to 0, as shown in the following figure:

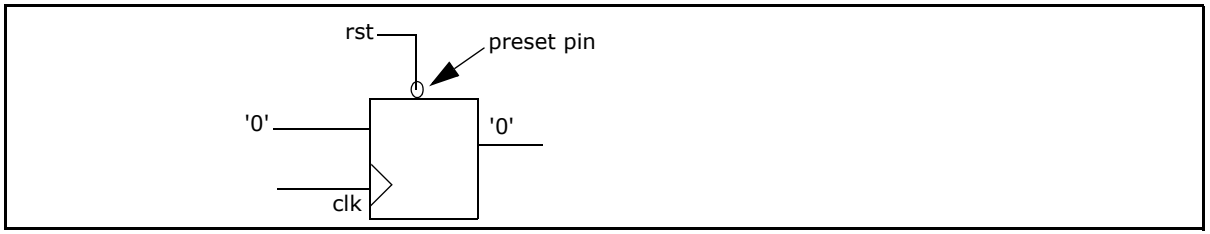


FIGURE 38. Flip-Flop with a Preset Pin and the Data Pin Tied to 0

No violation appears in the above scenario because when the preset pin changes from 1 to 0, the output will change to 0 at the next clock positive/negative edge.

Parameter(s)

ignore_latches: Default value is `yes`. Set this parameter to `no` to consider signals ending on latch enable terminals.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where the output of a sequential element *<name>* (type *<type>*) triggered by a clock *<clk-name>* is set when its data pin is tied to a constant value *<value>*:

[INFO] Data pin of <type> <name>, clocked by <clk-name>, is tied to constant value <value>

Where *<type>* can be `flop`, `latch`, or `library-cell`.

NOTE: *For RTL designs,* `<name>` *is the name of the output net of the flip-flop/latch/*

clock-gating cell. For netlist designs, if the report_inst_for_netlist parameter is set to yes, `<name>` *is* `<inst-name>.<data-pin-name>`*. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains flip-flops and latches that cannot change values.

Consequences of Not Fixing

If you do not fix this violation, clock domain crossing violations are not reported for flip-flops and latches that cannot change value.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Enable *Show Case Analysis* to see constant propagation in the schematic that is causing the data input to be constant.

   Check the case analysis settings carefully.

If the constant value of data pin is intentional, disable/waive this rule. Else, change the logic.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

// test.v

```
module top(clk,ck,clock,q,q1);
input clk,ck,clock;
output q,q1;
reg q;
reg q1;
wire CLK;

reg temp;

assign CLK = clk & ck;
wire d,d1;
assign d = 1'b0;
assign d1 = 1'b1;

always @(posedge CLK)
   {q,temp}<={temp,d};

always @(posedge CLK)
   q1<=d1;

endmodule
```

// constraints.sgdc

```
current_design top1
clock -name "top1.clk"
set_case_analysis
   -name top1.en -value 0

current_design top
clock -name "top.clock"

current_design top2
clock -name "top2.clk2"
```

For the above example, the *Clock_info03b* rule reports a violation as the data pin of the `top.temp` flip-flop is tied to a constant value `0`.

The schematic of this violation is shown below:

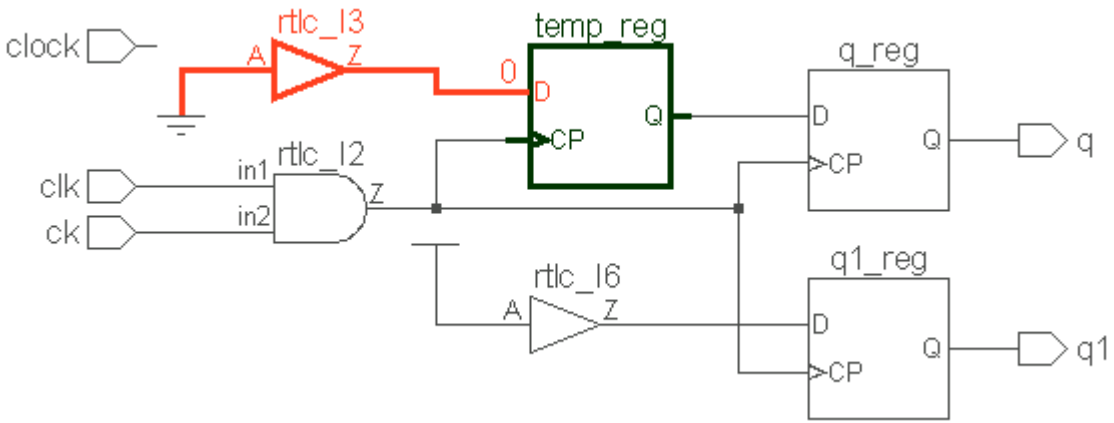


FIGURE 39. Schematic of Clock_info03b Rule Violation

Schematic Highlight

The *Clock_info03b* rule highlights the data path of a flip-flop, latch, or clock-gating cell whose data pin is tied to a constant value.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

- *The Clock-Reset-Summary Report*
- *The Clock-Reset-Detail Report*

Clock_info03c

Reports sequential elements whose clock pin is tied to a constant

When to Use

Use this rule to check correctness of SpyGlass CDC solution setup.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Clock_info03c* rule reports sequential elements for which a clock pin is set to a constant. See *Figure 40*.

In this case, the clock pin of reported flip-flops, latches, or sequential library cells is:

- Constrained by case analysis settings (specified using the *set_case_analysis* constraint).
- Connected to supply nets.
- Connected to tied-off/tied-on cells.

Such sequential elements are not checked for clock domain crossings.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use the auto-generated clock information.
- *ignore_latches*: Default value is `yes`. Set this parameter to `no` to consider signals ending at latch enable terminals while deciding clock candidates.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *check_reset_for_constclock*: Specifies if the `Clock_info03c` rule checks for set and preset pins on flop whose clock pin is tied to constant value.

- *report_quasi_static_on_clock*: Specifies the rule that should report a violation when a quasi_static signal is reaching to the clock pin of a flop.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location where a flip-flop, latch, or a sequential cell *<name>*, triggered by a clock pin of a clock net, *<clk-net>*, is set when that clock pin is tied to a constant and the clock net is other than the supply net:

```
[INFO] Clock pin <clk-net>, of <flip-flop | latch | library
cell> <name>, is <tied to constant value> | <driven by
quasi_static signal>
```

Here, *<clk-net>* can be a hierarchal name of a clock pin instead of a clock net if the clock net is a supply net.

Potential Issues

This violation appears if your design contains sequential elements for which the clock pin is tied to a constant.

Consequences of Not Fixing

If you do not fix this violation, clock domain crossing is not checked for such sequential elements.

In addition, such constant clocks may be unintentional constant clocks whose presence in a design may result in functional issues.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Enable *Show Case Analysis* and see the constant propagation in the schematic that is causing the clock to be constant.

   Check the case analysis settings carefully.

Message 2

If the check_reset_for_constclock parameter is set to `yes`, the rule reports the following messages:

- If a flop has only reset pins:

```
Clock pin <clk-net>, of <flip-flop | latch | library cell>
<name>, <tied to constant value 0|1> | <driven by quasi_static
signal> and it has asynchronous reset(s) (''<pin1-
name>":<status>, "<pin2-name>":<status>, …)
```

- If a flop has only preset pins:

```
Clock pin <clk-net>, of <flip-flop | latch | library cell>
<name>, <tied to constant value 0|1> | <driven by quasi_static
signal> and it has asynchronous set(s) ("<pin1-name>":<status>,
…)
```

- If a flop has both reset and preset pins:

```
Clock pin <clk-net>, of <flip-flop | latch | library cell>
<name>, <tied to constant value 0|1> | <driven by quasi_static
signal> and it has asynchronous reset(s) (''<pin1-
name>":<status>, "<pin2-name>":<status>, …) and asynchronous
set(s) ("<pin1-name>":<status>, …)
```

Where, status can be any of the following:

- UNCONNECTED
- CONNECTED
- CONSTANT ACTIVE
- CONSTANT INACTIVE

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v
module top1(d,en,q,clk1);
  input d,clk1,en;
  output q;
  wire clk = clk1 & en;
  reg q;
  always @(posedge clk)
     q<=d;
endmodule
```

```
// constraints.sgdc

current_design top1
clock -name "top1.clk"
set_case_analysis
   -name top1.en -value 0

current_design top
clock -name "top.clock"

current_design top2
clock -name "top2.clk2"
```

In the above example, the `top1.clk` clock pin of the `top1.q` flip-flop is tied to constant value. This is shown in the following schematic:

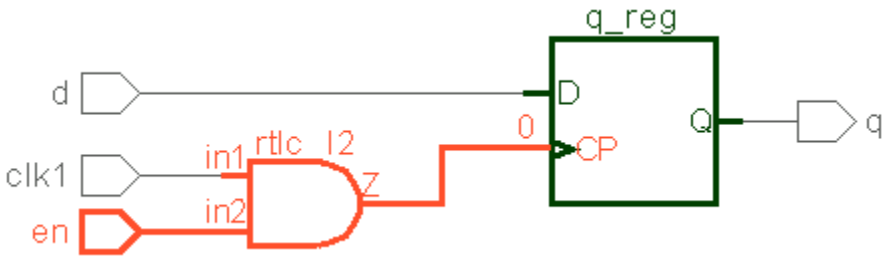


FIGURE 40. Schematic of Clock_info03c Rule Violation

To fix this violation, apply appropriate constant value on the `en` signal to enable the clock signal.

Schematic Details

The *Clock_info03c* rule highlights the path from a clock pin to a net where the constant value is implied.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

The Clock-Reset-Summary Report

Clock_info05

Reports clock signals converging on a MUX

When to Use

Use this rule to detect muxes on which multiple clock signals converge.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Clock_info05* rule reports MUX descriptions where two or more clock signals converge. See *Figure 42*.

By default, this rule reports a violation if the output of the MUX (where clocks converge) is captured by the clock pin of a sequential element. If you set the *cdc_reduce_pessimism* parameter to `clock_on_ports`, this rule also reports a violation if the MUX output is captured by a port.

Difference Between Reporting of the Clock_info05 and Clock_info05b Violations

The *Clock_info05* rule reports a violation when the output of a MUX reaches the clock pin of any flip-flop, whereas the *Clock_info05b* rule reports combinational gates other than MUX gates where two or more clocks converge.

Suppressing Clock_info05 Violations for Same Domain Clock Signals

To suppress *Clock_info05* violations for the same domain clock signals that converge on a MUX when no *set_case_analysis* constraint is applied on the MUX select pin, specify the following *clock_reduce_pessimism* parameter setting:

```
set_parameter clock_reduce_pessimism +ignore_same_domain
```

Rule Exceptions

This rule does not report a violation if the output of such muxes does not drive a clock pin of a sequential element, as shown in the following figure:

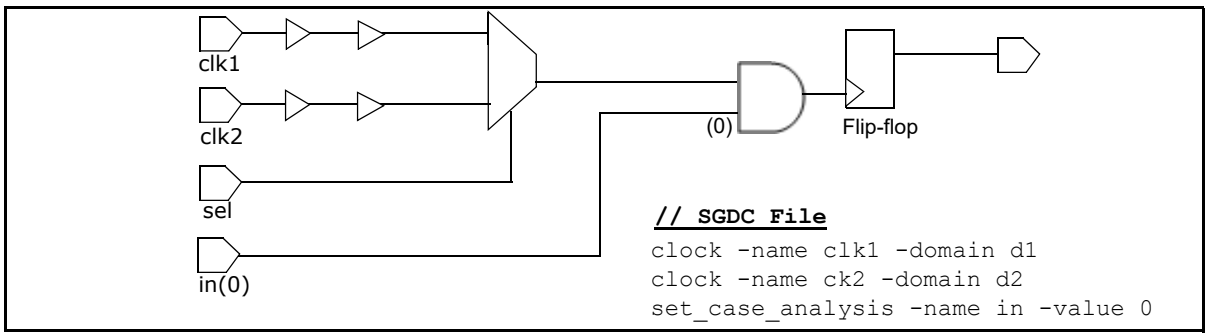


FIGURE 41. Clock_info05 Rule Exception

Parameter(s)

- *clocks_pair*: Default value is `NULL`. Specify a list of clock signal names in pairs to this parameter.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set the value of this parameter to `clock_on_ports` to report muxes where clocks converge and the mux output is captured by a clock pin or a port. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.

- *enable_glitchfreecell_detection*: Default value is `no`. Set this parameter to `yes` to report glitch-free multiplexers in a design.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *all_converging_clocks*: Default value is `no`. Set this parameter to `yes` to report all the clocks converging on a mux.
- *filter_clock_converge_on_cdc*: Filters clock signals that converge on a mux that is reaching the source or destination of a CDC crossing.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location of a MUX description where clocks (*<clk1-name>* and *<clk2-name>*) converge:

```
[INFO] Clock signals <clk1-name> and <clk2-name> converge at
mux <obj-type> <inst-name>. Missing set_case_analysis
constraint on control signal <control-net-name>
```

Details of the arguments of the above violation message are described in the following table:

Argument	Description
<control-net-name>	The hierarchical name of the source net of the MUX select pin.
<obj-type>	`output` in case of RTL designs. `instance` in case of netlist designs if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `output`.
<inst-name>	*<hier-out-net-name>* in case of RTL designs. This is the hierarchical name of the net connected to the MUX output. *<hier-inst-name>* in case of netlist designs if the *report_inst_for_netlist* parameter is set to yes. Otherwise, it is same as in case of RTL designs.

Potential Issues

This violation appears if multiple clocks reach to a MUX that does not have the *set_case_analysis* constraint specified on its select pin.

Consequences of Not Fixing

If an appropriate mode is not specified, a design is not configured in the desired functional or test mode. This may result in inaccurate clock domain crossings analysis.

How to Debug and Fix

To debug the violation reported by this rule, perform the following steps:

1. Open the *Spreadsheet Viewer* window to view the clock-pairs affected by the same MUX select signal.
2. View the *Incremental Schematic* of the violation message.
3. In the schematic, check the MUX where clock signals converge with the un-constrained select signal.

To find the signals that need to be constrained, perform the following steps:

1. Turn on the *Setup_clock01* rule and look for the auto_case_analysis.sgdc file in the current working directory.
2. Review each constraint in the generated SGDC file.
3. Copy the relevant constraints in current SGDC file.

Alternatively, you can perform the following steps to find the signals that need to be constrained:

1. Back-trace the select-signal in the *Incremental Schematic* window of the *Clock_info05* violation message till it hits the input ports, black box outputs, or flip-flops.
2. Apply the *set_case_analysis* constraint on appropriate signals in the SGDC file.

If signals or connected nets are already constrained, perform the following steps:

1. Enable *Show Case Analysis* in *Incremental Schematic* window.

   This schematic should show where constant propagation is blocked or takes 'x' value.
2. Apply correct constant value to signals by using *set_case_analysis* constraint.

Message 2

The following message is reported when the `all_converging_clocks` parameter is set to `yes` and if at least two clocks converge at two different input pins.

[Info] <num_converging_clocks> Clock signals converge at mux output <mux_output>. Missing set_case_analysis constraint on the control signal <select_net>

Details of the arguments of the above violation message are described in the following table:

Argument	Description
<num_converging_clocks>	The number of clocks converging on a mux
<mux_output>	Name of the mux where clocks are converging
<select_net>	Name of the signal on which the set_case_analysis constraint is missing

Potential Issues

This violation appears if multiple clocks reach to a MUX that does not have the *set_case_analysis* constraint specified on its select pin.

Consequences of Not Fixing

If the *set_case_analysis* constraint is not specified on the select pin, it may result in inaccurate clock domain crossings analysis.

How to Debug and Fix

To debug the violation reported by this rule, specify the *set_case_analysis* constraint on the select pin.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

// test.v

```
module test(clk1,clk2,sel,d,in,q);
input clk1;
input clk2;
input d,in;
input sel;

output q;
reg q;
reg temp_clk;

always@(posedge clk2)
     temp_clk <= in;

assign clk = (sel)?clk1:temp_clk;

always@(posedge clk)
    q <= d;
endmodule
```

// constraints.sgdc

```
current_design test
clock -name clk1 -domain d1
clock -name clk2 -domain d2
```

For the above example, the *Clock_info05* rule reports a violation as the `clk1` and `clk2` clock signals converge on a MUX in the clock path of the `test.q` flip-flop. This is shown in the following schematic:

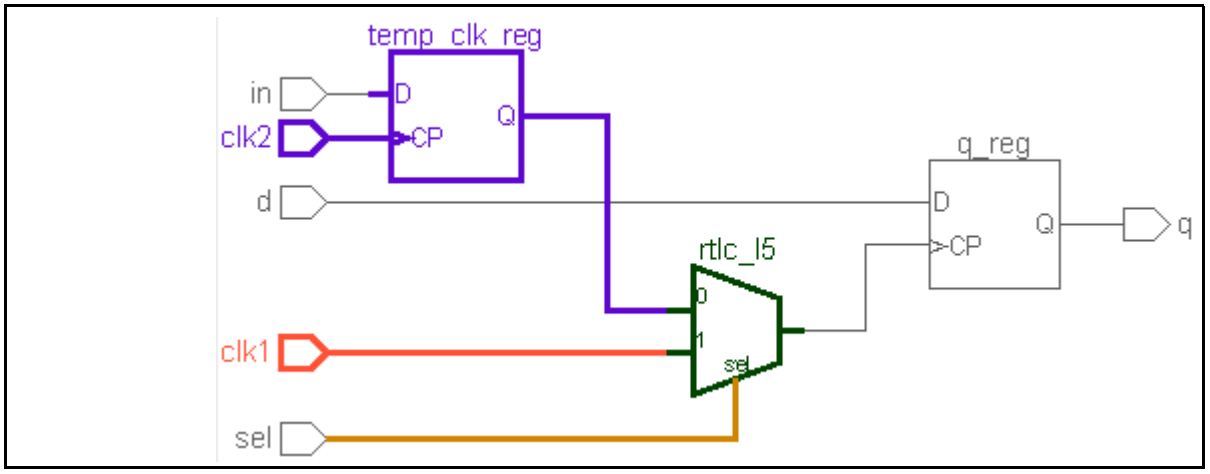


FIGURE 42. Schematic of Clock_info05 Rule Violation

To fix this violation, constraint the `sel` select signal to enable propagation of an appropriate clock signal.

Schematic Highlight

The *Clock_info05* rule highlights paths from different clock sources to MUX input terminals in different colors.

Default Severity Label

Info

Rule Group

`INFORMATION`

Report and Related File

- clock_info05.csv: This is a rule-based spreadsheet that contains details of violations of this rule. The last column in the report lists the name of the corresponding message-based spreadsheet.
- clock_info05_<unique-number>.csv: This is a message-based spreadsheet that contains details of a particular violation of this rule.

Clock_info05a

Reports signals which should be constrained for muxed clock selection

When to Use

Use this rule to detect signals on which the *set_case_analysis* constraint should be applied to control clock selection at muxes.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Clock_info05a* rule reports signals on which the *set_case_analysis* constraint should be set to control muxed clock selection.

During execution, this rule traverses back from a clock pin of sequential elements till a MUX output pin is found. Then it reports the net connected to a select pin of the MUX (ignoring buffers and inverters) if constant value is not seen on MUX select path.

Rule Exceptions

This rule does not report a violation if all data inputs of a MUX are connected to a constant and the clock signal reaches the select pin of the MUX, as shown in the following figure:

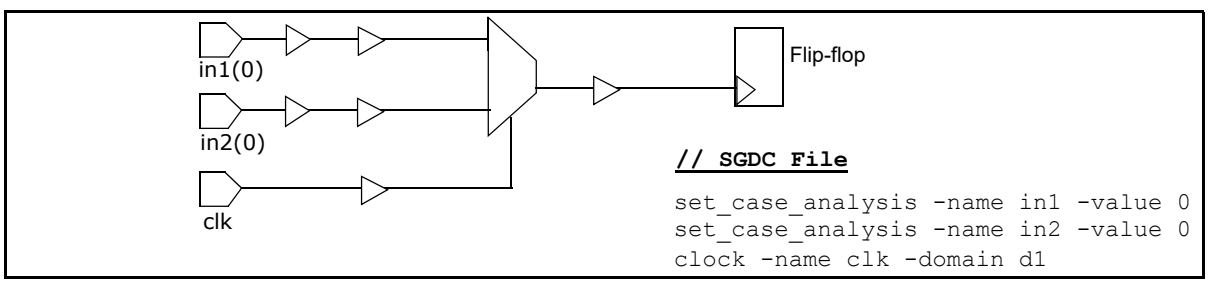


FIGURE 43. Clock_info05a Rule Exception

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *report_clock_tag_names*: Default value is `no`. Set this parameter to `yes` to enable the rule report the logical names of the clocks in the violation message.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for a signal *<sig-name>* on which the `set_case_analysis` constraint should be set to control MUXed clock selection:

```
[INFO] Inferred signal on which to set a set_case_analysis for muxed clock selection: <sig-name>
```

NOTE: *If <sig-name> is an internally generated net name, the following message is displayed instead of the net name:*

```
(complex expression) - User needs to identify from schematic' is displayed.
```

Potential Issues

The violation appears if a clock hits a MUX that has no *set_case_analysis* defined on its select pin.

Consequences of Not Fixing

A design is not configured in the desired functional or test mode if an appropriate mode is not specified. This may result in an inaccurate clock domain crossings analysis.

How to Debug and Fix

To debug the violation reported by this rule, open the *Spreadsheet Viewer* window to view a list of signals that can be constrained with the *set_case_analysis* constraint.

If internal net names are reported, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Back-trace the select signal path till it hits the input ports, black box outputs, or flip-flops.
3. Apply *set_case_analysis* constraint on appropriate signals in the SGDC file.

If the signals or connected nets are already constrained, perform the following steps:

1. Run the *Info_Case_Analysis* rule.
2. View the schematic along with the violation rule.

   The schematic should show where constant propagation is blocked or takes 'x' value.
3. Apply correct constant value to signals by using the *set_case_analysis* constraint.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

// test.v

```
module test(clk1,clk2,sel1,sel2,in1,out);

input clk1,clk2;
input sel1,sel2;
input in1;
output reg out;
wire mclk;
assign cntrl = sel1 & sel2;
assign mclk = (cntrl)? clk1 : clk2;
always@(posedge mclk)
    out <= in1;

endmodule
```

// constraints.sgdc

```
current_design test
clock -name clk1
clock -name clk2
```

In the above example, the *Clock_info05a* rule reports a violation as no *set_case_analysis* constraint is set on the `test.cntrl` signal. This is shown in the following schematic:

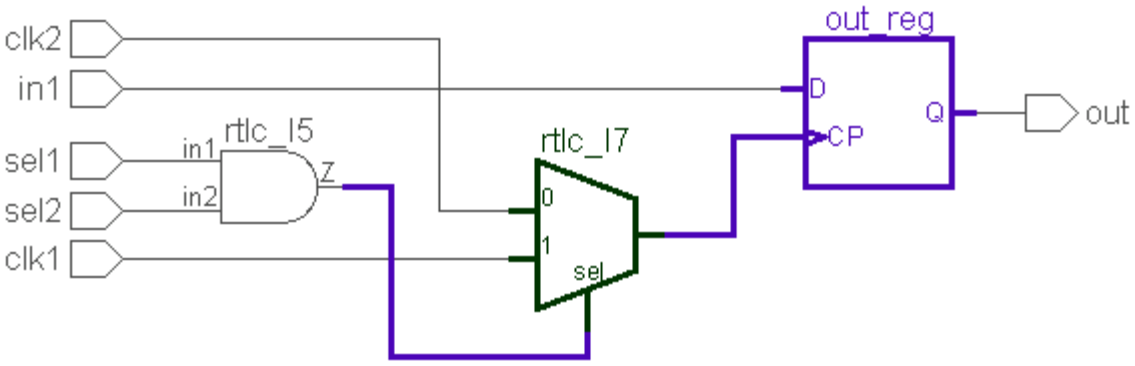


FIGURE 44. Schematic of Clock_info05a Rule Violation

To fix this violation, apply the *set_case_analysis* constraint on one or both the `sel1` and `sel2` select pins so that appropriate clocks are propagated through the MUX.

Schematic Details

The *Clock_info05a* rule highlights the path from a MUX output to a clock pin of a flip-flop.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

Clock_info05a.csv: This file is generated in the <*wdir*>/spyglass_reports/clock-reset/ directory.

Clock_info05b

Reports clock signals converging at a combinational gate other than a MUX

When to Use

Use this rule to detect clock signals converging on a combinational gate other than a MUX.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Clock_info05b* rule reports clock signals that converge on a combinational gate other than a MUX.

NOTE: *The Clock_info05 rule reports MUX gates where two or more clocks converge.*

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *report_clock_tag_names*: Default value is `no`. Set this parameter to `yes` to enable the rule report the logical names of the clocks in the violation message.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Specify this constraint on a clock signal to suppress the *Clock_info05b* rule violation for the convergence using that clock signal.

Messages and Suggested Fix

The following message appears at the location of a combinational gate other than a MUX gate where clocks *<clk-name-list>* converge:

[INFO] Clock Signals '<clk-name-list>' converge at combinational gate

Potential Issues

This violation appears if you design contains multiple clocks that reach to a combinational gate other than a MUX.

Consequences of Not Fixing

Your design may not be configured in the desired functional or test mode if an appropriate mode is not specified. This may result in inaccurate clock domain crossings analysis.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, view the combinational gate and connected clock signals.
3. Review the convergence gate, and make sure that it is intentional. Otherwise, change the logic to eliminate it.
4. You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

// test.v

```
module test(clk1,d,q,clk2,en);
input clk1;
input clk2;
input d;
input en;
output q;
reg q;
reg clock;
wire fclk;
wire tclk;
assign clockby2 = !clock;
always@(posedge clk1)
    clock <= clockby2;
assign tclk = clock&en;
assign fclk = tclk & clk2;
always@(posedge fclk)
    q <= d;
endmodule
```

// constraints.sgdc

```
current_design test
clock -name clk1 -domain d1
clock -name clk2 -domain d2
```

In the above example, two clock-signals converge on the `AND` gate in the clock path of the `q_reg` flip-flop. This is shown in the following schematic:

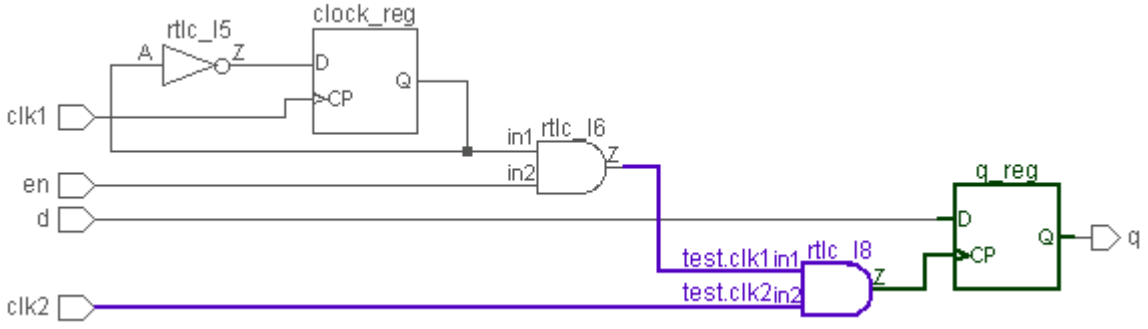


FIGURE 45. Schematic of Clock_info05b Rule Violation

To fix this violation, apply the *set_case_analysis* constraint on the `en` signal to block further propagation of the `clk1` clock.

Schematic Details

The *Clock_info05b* rule highlights path from clock signals through a combinational gate to a clock pin of a sequential element (one of the sequential elements in case of paths existing to clock pins of multiple sequential elements).

Default Severity Label

Info

Rule Group

INFORMATION

Report and Related File

No related reports or files

Clock_info05c

Reports unconstrained MUXes which do not receive clocks in all its data inputs

When to Use

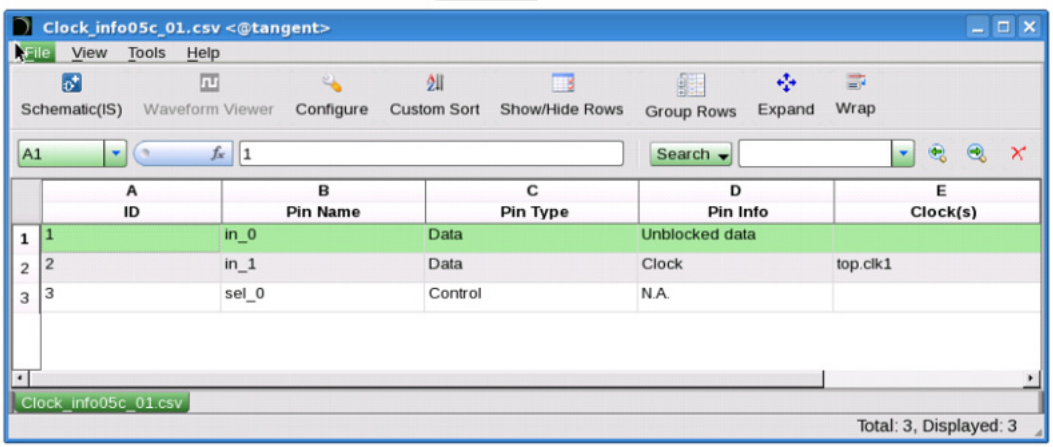
Use this rule to detect unconstrained MUXes which do not receive clocks in all its data inputs.

Description

It reports a violation when not all the pins of a MUX receive clocks from top-level when the MUX output reaches to a clock pin of sequential elements. It is mandatory to have following conditions satisfied:

- One clock reaching to at least one data pin of the MUX
- At least one pin which is unblocked (non-constant and non-quasi) without any clock reaching to it

The rule reports one violation per mux and it shows the message spreadsheet as below.



Where,

- **<Pin Name>** shows the name of input pin of a MUX
- **<Pin Type>** shows the data or control

- **<Pin Info>** shows "Clock", "Blocked", "Constant", "Quasi-Static", "Unblocked data", "Hanging" or "N.A.". For some cases, it can be all except "Unblocked data".
- **<Clock(s)>** shows top-level clocks reaching to the data pin in case of clock.

The rule spreadsheet shows information in the following format:

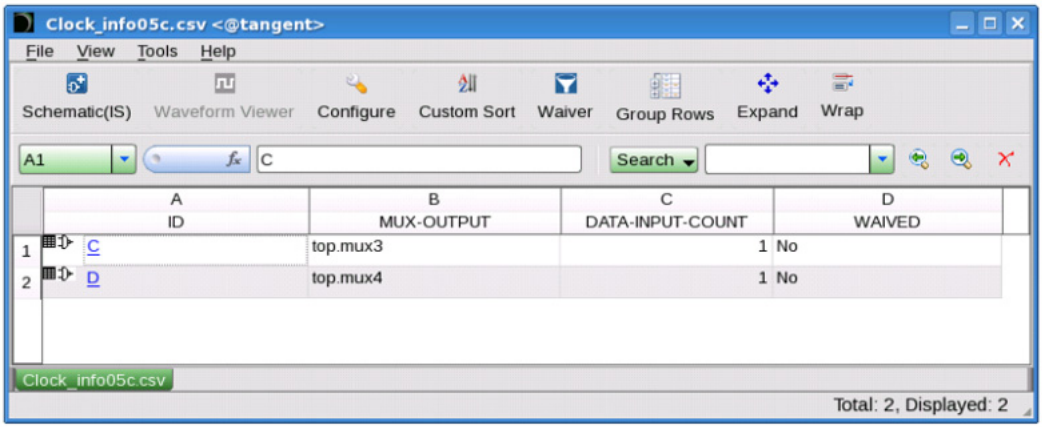


FIGURE 46.

Where,

- **<DATA-INPUT-COUNT>** shows the count of unblocked data inputs which needs action from user.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *report_clock_tag_names*: Default value is `no`. Set this parameter to `yes` to enable the rule report the logical names of the clocks in the violation message.

- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Specify this constraint on a clock signal to suppress the *Clock_info05b* rule violation for the convergence using that clock signal.

Messages and Suggested Fix

The following message appears at the location of an unconstrained MUX which does not receive clocks in all its data inputs:

```
[WARNING] No clock is reaching to <count>  data <input|inputs>
of unconstrained MUX <output|instance> <output name>
```

Potential Issues

This violation appears if you design contains unconstrained MUXes which do not receive clocks in all its data inputs.

Consequences of Not Fixing

This may result in inaccurate clock domain crossings analysis.

How to Debug and Fix

To debug the violation of this rule, either connect all data pins to clock sources or tie constant to MUX select lines.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

```
// test.v
module top (input in, inp, clk1, en3,
output d3);
reg mux3;

always begin
case(en3)
0: mux3 = clk1;
1: mux3 = inp;
endcase
end

ff f3 (in, mux3, d3);

endmodule

module ff (input in, clk, output reg out);
always @(posedge clk )
out <= in;
endmodule
```

```
// constraints.sgdc

current_design top
clock -name clk1
```

In the above example, the Clock_info05c rule reports a violation as only one clock signal is received on one of the MUX inputs. This is shown in the following schematic:

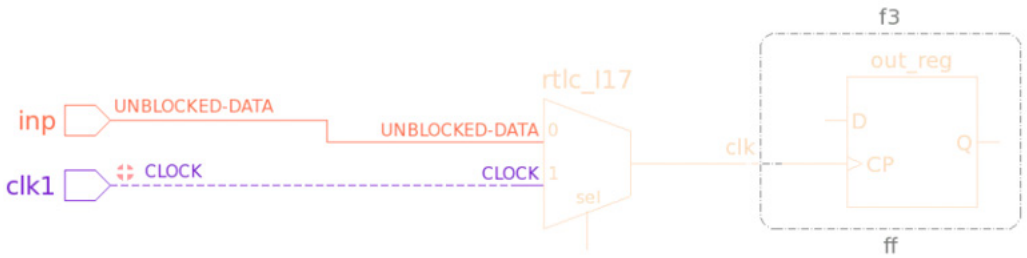


FIGURE 47. Schematic of Clock_info05b rule violation

To fix the violation, either connect all data pins to clock sources or tie constant to MUX select lines.

Schematic Details

The *Clock_info05c* rule highlights the data pins in different colors based on following categories:

CLOCK, UNBLOCKED-DATA, BLOCKED-DATA, CONSTANT, QUASI-STATIC

Default Severity Label

Warning

Rule Group

INFORMATION

Report and Related File

No related reports or files

Clock_info06

Reports nets derived from user-specified clocks

When to Use

Use this rule to detect nets derived from user-specified clock signals.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_info06* rule reports nets that are derived from the user specified clocks after frequency division.

Performing Frequency Division Checks

Frequency division checks are limited to a factor of 2. The *Clock_info06* rule reports only those clock dividers (frequency division by 2) in a design that match with any of the following topologies:

- Divide By 2 - Topology 1

  This topology is described in the following example and the equivalent circuit that follows:

```
module divBy2_T1(clock, clockby2, out, in);
  input clock, in;
  output clockby2, out;

  reg clk, out;

  assign clockby2 = !clk;
  always @(posedge clock)
    clk = clockby2;

  always @(posedge clockby2)
```

```
    out <= in;
endmodule
```

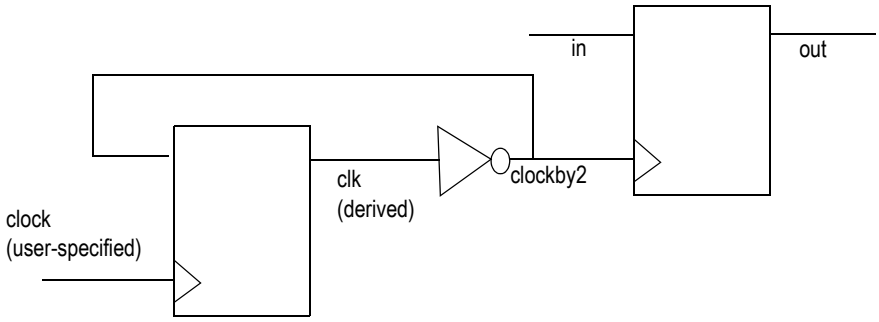


FIGURE 48. Divide By 2 - Topology 1

- Divide By 2 - Topology 2

  This topology is described in the following example and the equivalent circuit that follows:

```
module divBy2_T2(clock, clockby2, data, out);
  input clock, data;
  output clockby2, out;
  reg clockby2, out;

  always @(posedge clock)
    clockby2 = !clockby2;
  always @(posedge clockby2)
    out <= data;
endmodule
```

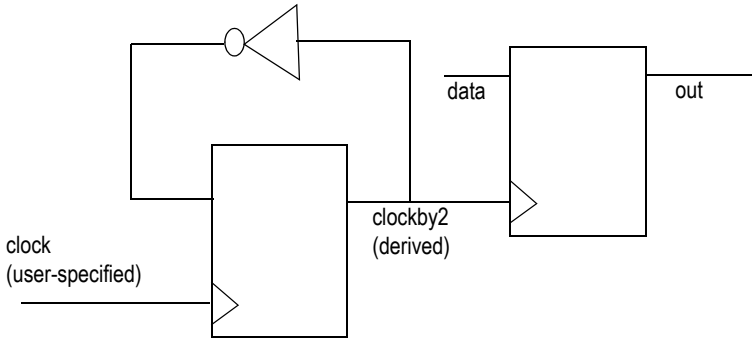


FIGURE 49. Divide By 2 - Topology 2

NOTE: *For both the topologies, buffers or buffer equivalents in term of inverters are allowed in any of the paths.*

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where the signal *<net-name>*, which is derived from the user-specified clock signal *<clk-name>*, is first set:

```
[INFO] Clock '<net-name>' is derived from user-specified clock
'<clk-name>'
```

Potential Issues

This violation appears when the user-specified clocks propagate in the design after frequency division.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following schematic:

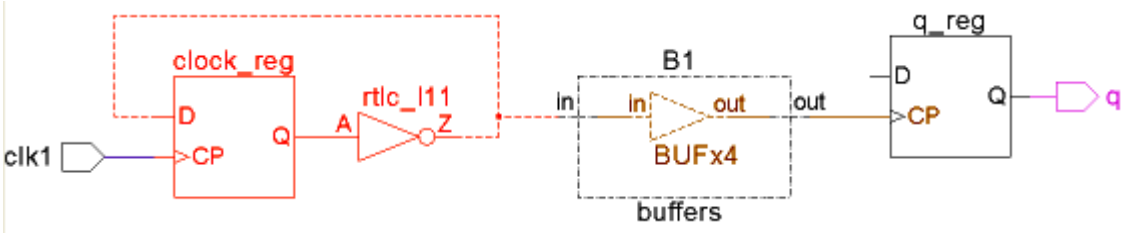


FIGURE 50. Schematic of Clock_info06 Rule Violation

In the above schematic, `clock` is derived from the user-specified clock `clk1`. Therefore, the *Clock_info06* rule reports a violation.

Schematic Details

The *Clock_info06* highlights the following paths in different colors:

- The path from the source clock to the dividing clock pin of a flip-flop
- The path from the dividing output of a flip-flop back to its input pin

- The path from the divided clock to any one sequential element

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

No report or related file

Clock_info07

Reports user-specified clocks that are derived from other clocks

When to Use

Use this rule to identify correct usage of clocks defined in a design.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use automatically generated clock signals
- By using a combination of both the above methods

Description

The *Clock_info07* rule reports user-specified clocks that are derived from other clocks after frequency division.

Frequency division checks are limited to a factor of 2. The *Clock_info07* rule reports only those clock dividers (frequency division by 2) in a design that match any of the specified topologies (as described in the *Clock_info06* rule).

NOTE: *Please note the following points:*

- *The Clock_info06 rule reports clocks derived from user-specified clocks after frequency division whereas the Clock_info07 rule reports user-specified clocks that are derived from other clocks after frequency division.*
- *Primary clocks are automatically propagated through frequency dividers. Therefore, derived clocks have the same domain as the primary clocks. Specifying derived clocks in an SGDC file can result in a conflict.*

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where the user-specified clock signal *<derived-net-name>* is derived from another clock *<clock-net-name>*:

[INFO] `User-specified clock '<derived-net-name>' is derived from clock '<clock-net-name>'`

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Example 1

Consider the following example:

```
module divBy2_T1(clock, clockby2, out, in);
  input clock, in;
  output clockby2, out;
  reg clk, out;
  assign clockby2 = !clk;
```

```
  always @(posedge clock)
    clk = clockby2;
  always @(posedge clockby2)
    out <= in;
endmodule
```

The following figure shows the circuit of this example:

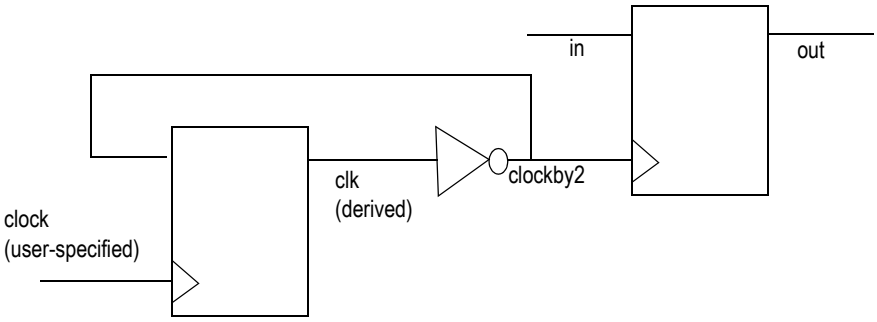


FIGURE 51. Scenario 1 of Clock_info07 Rule Violation

For the above example, the *Clock_info07* rule reports the following violation when the `clk` signal specified by the *clock* constraint is analyzed:

```
User-specified clock 'divBy2_TI.clk' is derived from clock
'divBy2_TI.clock'
```

Example 2

Consider the following example:

```
module divBy2_T2(clock, clockby2, data, out);
  input clock, data;
  output clockby2, out;
  reg clockby2, out;
  always @(posedge clock)
    clockby2 = !clockby2;
  always @(posedge clockby2)
    out <= data;
endmodule
```

The following figure shows the circuit of this example:

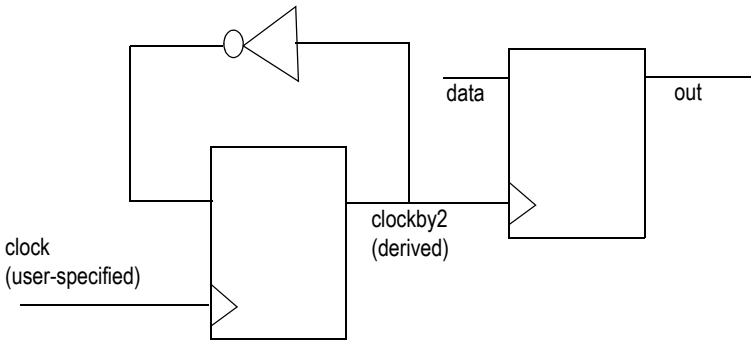


FIGURE 52. Scenario 2 of Clock_info07 Rule Violation

For the above example, the *Clock_info07* rule reports the following violation when the `clockby2` signal specified by the *clock* constraint is analyzed:

```
User-specified clock 'divBy2_T2.clockby2' is derived from clock
'divBy2_T2.clock'
```

Schematic Details

The *Clock_info07* rule highlights the following information in different colors in the schematic:

- The path from the source clock to the dividing clock pin of a flip-flop
- The path from the dividing output of a flip-flop back to its input pin

Default Severity Label

Info

Rule Group

`INFORMATION`

Reports and Related Files

No report or related file

Clock_info14

Highlights signals of different domains in different colors

When to Use

Use this rule to analyze clock distribution in a design.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By using auto-generated clock signals by setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_info14* rule highlights clock domains in different colors in RTL and schematic.

A different color is used to highlight clock domains to distinguish them from each other. Signals where two or more clock domains merge are shown in a color different from the color of the merging clock domains. You can set the color scheme as per your requirements.

This rule is also used to view clock crossings in a design. If signals on two sides of an assignment statement are shown in different colors in the RTL, it indicates a clock crossing.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when clock domains in the design unit *<du-name>* are identified and are highlighted in different colors:

[INFO] Clock domain highlight information populated for <du-name>

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

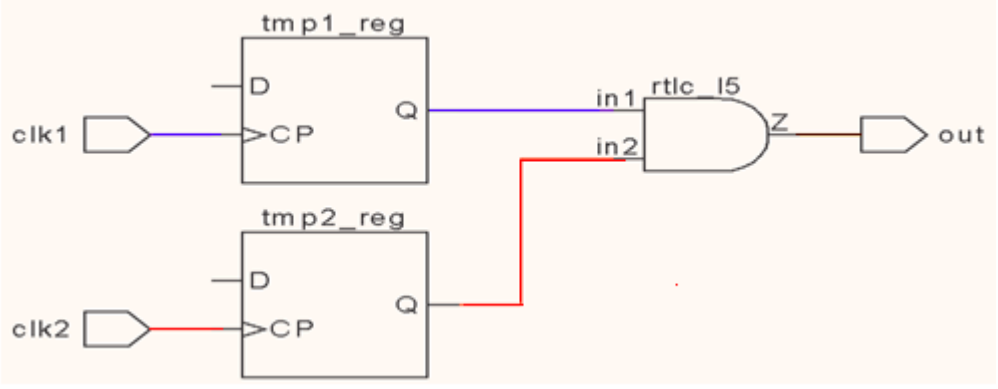


FIGURE 53. Schematic of Clock_info14 Rule Violation

In the above schematic, the paths of `clk1` and `clk2` are highlighted in different colors. In addition, the net connected to the `out` port is

highlighted in a different color because the paths of `clk1` and `clk2` merge on this net.

Default Severity Label

Info

Rule Group

`INFORMATION`

Reports and Related Files

No report or related file

Clock_info15

Generates the PortClockMatrix report and abstracted model for input ports

When to Use

Use this rule to view clock domain information for primary ports.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By using auto-generated clock signals by setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_info15* rule generates the following:

- Clock domain information for primary ports in *The PortClockMatrix Report*
- Constraints on input ports in the *Input Port Constraints File*

  However, constraints are not generated on the input ports that are already specified by any user-defined constraint, *such as clock, quasi_static, or set_case_analysis*.

  Use the generated constraints in the SoC flow for block-level SpyGlass CDC solution verification and abstraction.

Parameter(s)

- *report_indirect_port_clock*: Default value is `no`. Set this parameter to `yes` to generate an enhanced PortClockMatrix report, which also shows clocks that are indirectly connected to input/output ports.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. Other possible

values are `bbox`, `output_not_used`, `hanging_net`, `skip_unused_paths`, and `all`.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears to indicate that port clock information has been generated:

[ClkI15_1] [INFO] `Port-clock information generated for PortClockMatrix report`

Potential Issues

None

Consequences of Not Fixing

Information in the SGDC file and port-clock matrix should be consistent. If it is not consistent, ensure that such mismatch is intentional. Otherwise, it may result in incorrect SpyGlass CDC solution analysis.

How to Debug and Fix

To debug the violation of this rule, open *The PortClockMatrix Report* and check the following:

- Ports for which direct/indirect clocks are mentioned

  Search the *input* constraint for such ports in the SGDC file. If it is not defined, add it with correct clock name. If it is defined with a different clock, review and modify it appropriately.

- Ports for which direct/indirect clocks are not mentioned

  Such ports are connected to pure combinational paths, and may need to

be reviewed.

Review the ports defined as blocked/unconnected, and make sure it is as expected.

Message 2

The following message appears to indicate that port clock information has been generated:

```
[ClkI15_2] [INFO] Abstracted sgdc file for input ports of block
'<block-name>' is generated
```

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

None

Message 3

The following message appears to indicate that port clock information for the block *<block-name>* is not generated:

```
[ClkI15_3] [INFO] Abstracted sgdc file for input ports of block
'<block-name>' is not generated
```

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

None

Example Code and/or Schematic

Example 1 - The PortClockMatrix Report Generation

Consider the following example:

```
module flop(D,C,Q);
```

```
  input D,C;
  output Q;
  reg Q;
  always @ (posedge C)
    Q<=D;
endmodule


module top(I1, C1, C2, O1);
  input I1,C1,C2;
  output O1;
  reg O1;
  wire w = C2 & C1;
  flop F1(I1, w, O1);
endmodule
```

For the above example, the *Clock_info15* rule generates the following report containing clock domain information for primary ports:

```
*****************************************************
               Input Port - Clock Matrix
*****************************************************
-----------------------------------------------------
S. No.       Input      Attribute       Direct Clocks
-----------------------------------------------------
1.           C1         clock
2.           C2         clock
3.           I1         top.C1, top.C2
-----------------------------------------------------

*****************************************************
              Output Port - Clock Matrix
*****************************************************
-----------------------------------------------------
S. No.     Output     Attribute     Direct Clocks
-----------------------------------------------------
1.         O1                       top.C1, top.C2
-----------------------------------------------------
```

Default Severity Label

Info

Rule Group

INFORMATION

Related Reports

- *The PortClockMatrix Report*
- *Input Port Constraints File*
- *The SynchInfo Report*

Clock_info16

Reports clocks converging on a MUX that does not have the Synopsys infer_mux pragma set on it

When to Use

Use this rule to detect MUXes on which multiple clocks converge, but no Synopsys `infer_mux` pragma is defined for the MUX in the RTL.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By using the automatically generated clocks after setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_info16* rule reports a violation when two or more clocks converge on a MUX for which no Synopsys `infer_mux` pragma is defined in the RTL.

By default, this rule reports a violation if the output of the MUX (where clocks converge) is captured by the clock pin of a sequential element. If you set the *cdc_reduce_pessimism* parameter to `clock_on_ports`, this rule also reports a violation if the MUX output is captured by a port.

Rule Exceptions

This rule does not report a violation if the output of MUXes does not drive a clock pin of a sequential element.

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set the value of this parameter to `clock_on_ports` to report muxes where clocks converge and the mux output is captured by a clock pin or a port. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location of a MUX description where the clocks *<clk1-name>* and *<clk2-name>* converge, but no Synopsys `infer_mux` pragma is set for the MUX:

[INFO] Clock signals <clk1-name> and <clk2-name> converge at mux <obj-type> <inst-name> (control signal <control-net-name>), which does not have a synopsys infer_mux pragma set on it

The arguments of the above message are explained below:

Argument	Description
<control-net-name>	The hierarchical name of the source net of the MUX select pin
<obj-type>	`output` in case of RTL designs. `instance` in case of netlist designs if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `output`.
<inst-name>	*<hier-out-net-name>* in case of RTL designs. This is the hierarchical name of the net connected to the MUX output *<hier-inst-name>* in case of netlist designs if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs.

Potential Issues

This violation appears if your design contains multiple clock signals that converge on a MUX, and the Synopsys `infer_mux` pragma is not set on the MUX in the RTL.

Consequences of Not Fixing

If you do not fix this violation, some synthesis tools may optimize the MUX logic. As a result, analysis may be different between RTL and synthesized netlist.

How to Debug and Fix

To fix this violation, check the RTL block reported by this rule and apply the Synopsys `infer_mux` pragma on the MUX.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v

module top(sys_clk, test_clk, in, mode, out);
input sys_clk, test_clk, in, mode;
output out;
reg w_clk, out;
always@(sys_clk, test_clk)
begin
    case(mode) // no pragma defined
        2'b0 : w_clk = sys_clk;
        2'b1 : w_clk = test_clk;
    endcase
end
always @(posedge w_clk)
    out <= in;
endmodule
```

```
// constraints.sgdc

current_design top
clock -name sys_clk -domain d1
clock -name test_clk -domain d2
```

For the above example, the *Clock_info16* rule reports a violation for the line highlighted in red.

The following figure shows the schematic for this violation:

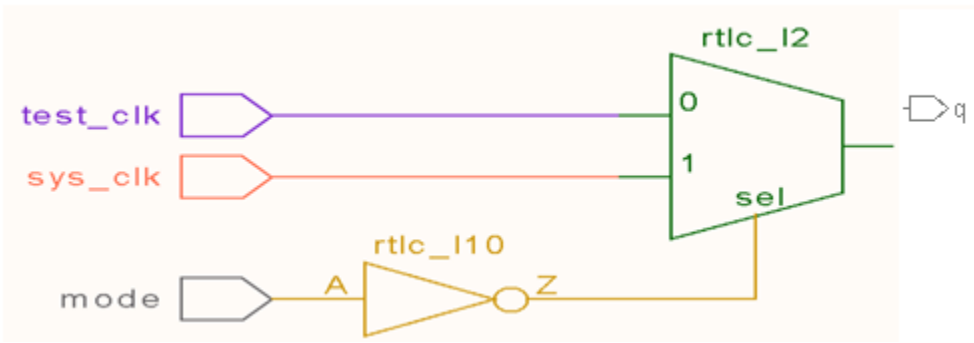


FIGURE 54. Schematic of Clock_info16 Rule Violation

To fix this violation, specify the `infer_mux` pragma on the `case` statement in the RTL, as shown below:

```
case (mode) // synopsys infer_mux
```

This will ensure that the MUX is not optimized during synthesis.

Schematic Details

This rule highlights the paths from different clock sources to the MUX input terminals.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

No report or related file

Clock_info17

Reports all the synchronous clocks present in a hierarchy

When to Use

Use this rule to check multiple *Synchronous Clocks* in a hierarchy.

Prerequisites

Specify a list of hierarchies for which *Synchronous Clocks* should be reported by using the *report_sync_clk_for_hier* parameter.

Description

The *Clock_info17* rule reports all *Synchronous Clocks* present in a hierarchy.

You must run this rule for the following reasons:

- This rule is important for physical design and is relevant for physical sub-chips only.
- Timing closure of sub-chips that has multiple synchronous input clocks is difficult because the closure is dependent on how well clocks are balanced.

Rule Exceptions

The *Clock_info17* rule does not report a violation if a clock and its corresponding derived clock is used in a particular hierarchy even though they are synchronous to each other.

Parameter(s)

report_sync_clk_for_hier: Default value is `NULL`. Specify a comma-separated list of hierarchies for which top-level synchronous clock signals should be reported.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when multiple *Synchronous Clocks* *<clk-net-list>* are used in the hierarchy *<hier-name>*:

```
[INFO] Hierarchy '<hier-name>' receives synchronous clocks
'<clk-net-list>'
```

Potential Issues

This message appears if a design hierarchy receives multiple *Synchronous Clocks*.

Consequences of Not Fixing

Not applicable

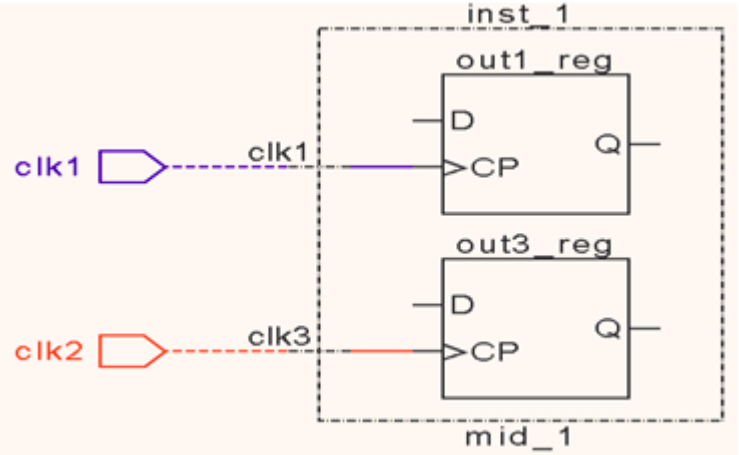
How to Debug and Fix

This is an informational rule.

Open the schematic of this rule to see multiple clocks from the same domain reaching to a particular hierarchy.

Example Code and/or Schematic

Consider the following schematic of this rule:




```
// constr.sgdc

current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d1
```

FIGURE 55. Schematic of Clock_info17 Rule Violation

For the above example, the *Clock_info17* rule reports a violation because the `clk1` and `clk2` clocks of the same domain `d1` are reaching to the instance `inst_1`.

Schematic Details

This rule highlights the path of primary clocks till the clock pin of a sequential element inside the hierarchy.

Default Severity Label

Info

Rule Group

`INFORMATION`

Reports and Related Files

No report or related file

Clock_info18

Reports unconstrained ports

When to Use

Use this rule to detect ports for which no SGDC constraint is specified.

Description

The *Clock_info18* rule reports a violation if input, output, or black box ports are not specified through any SGDC constraint.

This rule reports the following information:

- The number and percentage of top-level input and output ports that are not constrained by any of the *input*, *output*, *clock*, *reset*, *set_case_analysis*, or *abstract_port* constraints.
- The number and percentage of ports of all black boxes in a design that are neither:
  - Constrained by using any of the *clock*, *reset*, *abstract_port*, *assume_path,* or *signal_in_domain* constraints, nor
  - Hanging, nor
  - Net connected to the output port of that black box is blocked.

For a bus port, all bits of the bus are counted separately.

Parameter(s)

check_input_coverage: Default value is `no`. Set this parameter to `yes` to report violations only for the unconstrained input ports in the top-level design.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears if input and output ports are not constrained:

```
[ClkI18_1] [WARNING] For top design unit '<design-name>',
'<num-input-ports>' ('<per-input-ports>') input port(s) and
'<num-output-ports>' ('<per-output-ports>') output ports are
unconstrained. Refer report CKSGDCInfo for details
```

The arguments of the above message are explained below:

Argument	Description
<design-name>	Name of the top design unit
<num-input-ports>	Number of input ports that are not constrained with *input* constraint
<per-input-ports>	Percentage of input ports that are not constrained with *input* constraint
<num-output-ports>	Number of output ports that are not constrained with *output* constraint
<per-output-ports>	Percentage of output ports that are not constrained with *output* constraint

Potential Issues

This violation appears if your design contains input and output ports that are not constrained by using any of the *input*, *output*, *clock*, *reset*, *set_case_analysis* or *abstract_port* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass CDC verification ignores top-level input and output ports that are not constrained by using any of the specified constraint.

How to Debug and Fix

To fix this violation, specify input and output ports of a design by using any of the *input*, *output*, *clock*, *reset*, *set_case_analysis* or *abstract_port* constraint.

Message 2

The following message appears if all the input and output ports are

constrained:

```
[ClkI18_2] [INFO] All ports in the top design unit '<du-name>'
are constrained using either 'input', 'output', 'clock',
'reset', 'set_case_analysis' or 'abstract_port' constraints.
Refer report CKSGDCInfo for details
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 3

The following warning message appears if black box ports are not constrained:

```
[ClkI18_3] [WARNING] For black box '<black-box-name>', '<num-
ports>' ('<percentage-ports>') port(s) are unconstrained. Refer
report CKSGDCInfo for details
```

The arguments of the above message are explained below:

Argument	Description
<black-box-name>	Name of a black box
<num-ports>	Total number of unconstrained black box ports
<percentage-ports>	Percentage of ports not constrained

Potential Issues

This violation appears if black box ports of a design are not specified through any of the *clock*, *reset*, *abstract_port*, or *assume_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the black box input and output ports that are not specified through any of the specified constraint.

How to Debug and Fix

To fix this violation, specify black box ports by using any of the *clock*, *reset*, *abstract_port*, or *assume_path* constraint.

Message 4

The following message appears if all ports of the black box *<black-box-name>* are constrained:

[ClkI18_4] [INFO] All the ports of the black-box '<black-box-name>' have been constrained. Refer report CKSGDCInfo for details

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 5

The following message appears if input and output ports are not constrained:

[ClkI18_5] [WARNING] For top design unit '<design-name>', '<num-input-ports>' ('<per-input-ports>') input port(s) are unconstrained. Refer report CKSGDCInfo for details

The arguments of the above message are explained below:

Argument	Description
<design-name>	Name of the top design unit
<num-input-ports>	Number of input ports that are not constrained with *input* constraint
<per-input-ports>	Percentage of input ports that are not constrained with *input* constraint

Potential Issues

This violation appears if your design contains input ports that are not constrained by using any of the *input*, *clock*, *reset*, *set_case_analysis* or *abstract_port* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass CDC verification ignores top-level input ports that are not constrained by using any of the specified constraints.

How to Debug and Fix

To fix this violation, specify input ports of a design by using any of the *input*, *output*, *clock*, *reset*, *set_case_analysis* or *abstract_port* constraints.

Message 6

The following message appears if all the input ports are constrained:

```
[ClkI18_6] [INFO] All input ports in the top design unit '<du-
name>' are constrained using either 'input', 'clock', 'reset',
'set_case_analysis' or 'abstract_port' constraints. Refer
report CKSGDCInfo for details
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v

module top(ck1, ck2, in, mode, out);
  input ck1, ck2, in, mode;
  output out;
   …
   …
 endmodule
```

```
// constraints.sgdc

current_design top
clock -name ck1 -domain d1
clock -name ck2 -domain d2
abstract_port -module top -ports in
              -clock ck1
```

In the above example, the input port `mode` and the output port `out` are not specified through any constraint. Therefore, the *Clock_info18* rule reports a warning.

Default Severity Label

Info | Warning

Rule Group

`INFORMATION`

Reports and Related Files

The CKSGDCInfo Report

Clockmatrix01

Shows clock relationship matrix

When to Use

Use this rule to see the clock relationships in a design.

Description

The `Clockmatrix01` rule generates a spreadsheet report listing the clock relationships defined in the SGDC file for all clocks in the design. If the `sg_clock_group` constraint is used, the clock domain is inferred based on the `sg_clock_group` constraint.

The spreadsheet is saved in the `SpyGlass_reports` directory.

Parameter(s)

`sta_based_clock_relationship`: Default value is no. Set this parameter to `only_scg` or `scg_functional` to enable the `Clockmatrix01` rule to generate the clock relationship spreadsheet based on the `sg_clock_group` constraint.

Constraint(s)

- *clock*: Use this constraint to specify clock signals.
- *generated_clock*: (Optional): Use this constraint to specify generated/ derived clocks.
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

Message 1

The following message appears when the spreadsheet is generated:

`[INFO] Clock Relation Matrix for design <design_name>, based on the sg_clock_group constraints defined in the SGDC`

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following SGDC file:

```
current_design top
clock     -name clk1     -domain d1     -tag T1
clock     -name clk2     -domain d2     -tag T2
clock     -name clk3     -domain d3     -tag T2
sg_clock_group     -group1 { T1 }     -group2 { T2 }
```

FIGURE 56. Example SGDC file

In the above example, the spreadsheet generated by the `Clockmatrix01` rule shows that the `clk1` and `clk2` as asynchronous and all other clocks are synchronous as shown in the figure below.

	A	B	C	D	E
	Clock Name	Filename:Line	T1	T2	T2
1	T1	test.sgdc:2	NA	A (SCG)	A (SCG)
2	T2	test.sgdc:3	A (SCG)	NA	S
3	T2	test.sgdc:4	A (SCG)	S	NA

FIGURE 57. Spreadsheet Generated by the Clockmatrix01 rule

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

This rule generates the clock_<design_name>_clock_relationship_matrix.csv file in the spyglass_reports/clock-reset directory.

Reset Information Rules

The SpyGlass CDC solution has the following rules for generating reset information:

Rule	Reports
Ar_syncrst_setupcheck01	Reports constant value on functional flip-flops in synchronous reset deassert mode
Reset_info01	Asynchronous and synchronous preset and clear candidates in the design
Ar_glitch01	Detects glitch-prone circuits
Reset_info02	Prints the preset/clear tree for specified preset/ clear signals
Reset_info09a	Unconstrained asynchronous reset nets
Reset_info09b	Asynchronous reset nets that are tied to constant value

Ar_syncrst_setupcheck01

Reports constant values on functional flip-flops in the synchronous reset deassert mode

When to Use

Use this rule to verify polarity of user-defined synchronous resets.

Prerequisites

Specify the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ar_syncrst_setupcheck01* rule reports a violation if a synchronous reset leads to a constant value on a *Functional Flip-Flop* in the deassert mode.

The `enable_const_prop_thru_seq` command is always on, and therefore, the reset values are propagated beyond the sequential elements.

NOTE: *The Ar_syncrst_setupcheck01 rule check is performed only for flops that have synchronous resets and not for flops with asynchronous resets.*

Rule Exceptions

This rule does not report a violation in the following cases:

- When a *Functional Flip-Flop* is tied to a constant value in any of the following ways:
  - Constrained by the *set_case_analysis* constraint
  - Connected to supply nets
  - Connected to tied-off or tied-on cells
- When the constant 1 reaches to the enable pin during the deassert mode, no violation is reported on the enable pin.
- When the enable is 0 and a constant reaches on the data pin during the deassert mode, no violation is reported on the data pin.

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears if the *<type>* pin of the *<element-name>* element is held constant at the deassertion of the *<reset-name>* synchronous reset:

```
[SRST01] [ERROR] At deassertion of synchronous reset '<reset-name>','<type>' pin of '<element-name>' is held constant at '<value>'
```

The arguments of the above message are explained below:

Argument	Description
<reset-name>	User-specified or automatically-inferred reset signal

Argument	Description
<type>	The possible values are `data` and `enable`
<element-name>	Is the output net name of the instance that is held constant during the deassertion of the synchronous reset. For RTL designs, this argument is the name of the output net of the corresponding flip-flop or latch. For library cells which do not have functional attributes, such as memory cells, this argument is the name of the input pin. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if a synchronous reset in the design leads to constant value on a *Functional Flip-Flop* in the deassert mode. See *Case 1*.

Consequences of Not Fixing

If you do not fix this violation, a constant value reaches the *Functional Flip-Flop* during deassertion of the synchronous reset signal.

How to Debug and Fix

To debug this violation, open the incremental schematic. Check the resets that are causing the flip-flops to be in deassert mode at inactive values.

To fix this violation, correct the values of the resets in the SGDC file.

Message 2

The following message appears if you do not specify an active value for the *<reset-name>* synchronous reset in the SGDC file:

[SRST03] [ERROR] `Active value for synchronous reset '<reset-name>' is not specified in SGDC`

Potential Issues

This violation appears if you do not specify an active value for a synchronous reset in the design.

Consequences of Not Fixing

If you do not fix this violation then incorrect value of synchronous reset may cause some of the functional flip-flops to be held constant during the deassertion.

How to Debug and Fix

To fix this violation, specify an active value of the reset in the SGDC file.

Message 3

The following message appears if you specify the *<reset-val>* active value that is not supported for the *<reset-name>* synchronous reset:

```
[SRST04] [ERROR] Unsupported active value '<reset-val>'
specified for synchronous reset '<reset-name>' is ignored
```

The arguments of the above message are explained below:

Argument	Description
<reset-name>	User-specified or automatically-inferred reset signal
<reset-val>	Active value of the reset specified in the SGDC file

Potential Issues

This violation appears if you specify an active value that is not supported for a synchronous reset in the design. For example, the value `X` is not supported.

Consequences of Not Fixing

If you do not fix this violation then incorrect value of synchronous reset

may cause some of the functional flip-flops to be held constant during the deassertion.

How to Debug and Fix

To fix this violation, specify a valid active value of the reset in the SGDC file.

Message 4

The following message appears if during deassertion of the *<reset-name>* synchronous reset, the data pin of the *<element-name>* element is held constant because of an unknown value reaching the enable pin of a *Functional Flip-Flop*:

```
[SRST02] [WARNING] Data pin of '<element-name>' may be held
constant at '<constant-value>' either due to deassertion of
synchronous reset '<reset-name>' or data tied to constant
```

Potential Issues

This violation appears if the enable reaching the enable pin of a *Functional Flip-Flop* is unknown, and the data pin of the flip-flop becomes constant when a synchronous reset deasserts. For details, see *Case 3*.

Consequences of Not Fixing

If you do not fix this violation, a constant value reaches the *Functional Flip-Flop*.

How to Debug and Fix

To debug this violation, open the incremental schematic. Check the resets that are causing the flip-flops at inactive values.

To fix this violation, correct the values of the resets in the SGDC file.

Example Code and/or Schematic

Consider the scenario shown in the following figure:

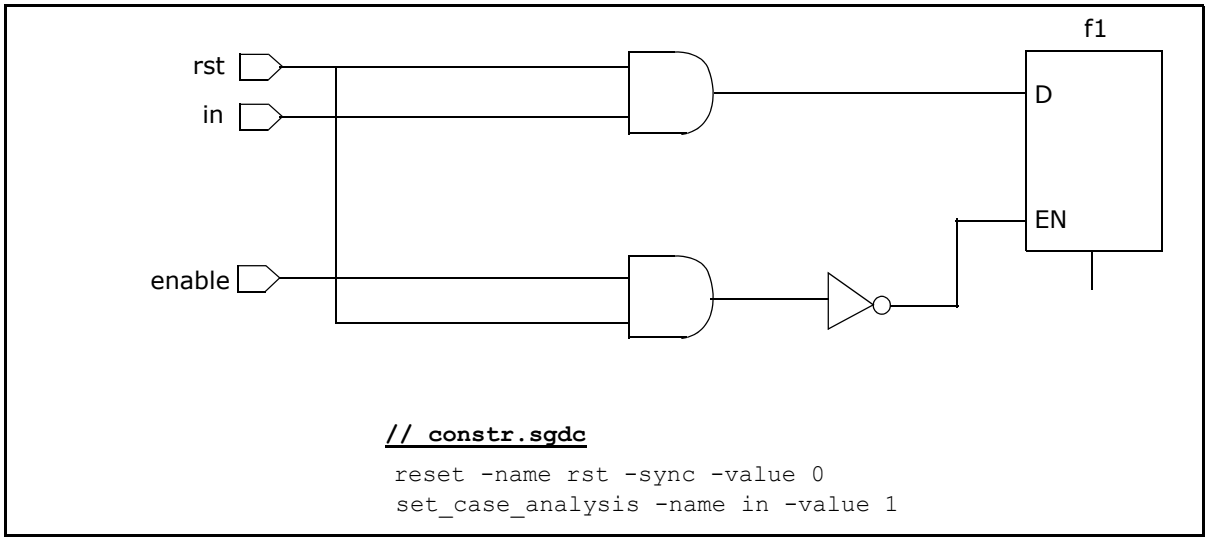


FIGURE 58.

In the above scenario, the *Ar_syncrst_setupcheck01* rule violation (error, warning, or no violation) depends on three cases: *Case 1*, *Case 2*, and *Case 3*.

Case 1

For the scenario shown in *Figure 58*, consider that you set the value of enable to 1, as shown below:

```
set_case_analysis -name enable -value 1
```

In this case, during deassertion of the synchronous reset `rst`, the value 1 reaches to the `D` pin of the `f1` flip-flop and `EN` becomes 0. Since the constant on `D` is due to synchronous reset deassertion only, the *Ar_syncrst_setupcheck01* rule reports an error (*Message 1*).

Case 2

For the scenario shown in *Figure 58*, consider that you set the value of enable to 0, as shown below:

```
set_case_analysis -name enable -value 0
```

In this case, the *Ar_syncrst_setupcheck01* rule does not report any violation because the constant reaching at the enable path is generated by the *set_case_analysis* constraint instead of reset deassertion.

Case 3

For the scenario shown in *Figure 58*, consider that you do not specify any value for `enable`.

Since the value of `enable` is unknown, the *Ar_syncrst_setupcheck01* rule reports a warning (*Message 4*) only if the `rst` reset is present in the input cone of `enable`. Since this holds true in *Figure 58*, a warning appears.

However, if `rst` was not present in the input cone of `enable`, this rule would not have reported a violation.

Default Severity Label

Error

Rule Group

`SETUP`

Reports and Related Files

A spreadsheet file containing all violations of this rule

Ar_syncrstTree

Prints the synchronous reset tree

When to Use

Use this rule to detect all the flip-flops with synchronous resets (*reset* `-sync`).

Prerequisites

Specify resets by using the *reset* constraint with the `-sync` argument.

Description

The *Ar_syncrstTree* rule generates a synchronous reset tree for the synchronous resets (*reset* `-sync`) specified in an SGDC file.

Rule Exceptions

While traversing the design to generate a reset tree, this rule does not traverse beyond the following objects:

- Black box objects that are specified without *assume_path*
- Sequential elements that are not flip-flops

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use the auto-generated clock information.

Constraint(s)

reset (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.

Messages and Suggested Fix

The following message appears to indicate that a synchronous reset tree is generated:

```
[INFO] Synchronous Reset Tree generated.
```

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Double-click on the message in GUI.

   This displays the design *Sync Reset Tree* window, as shown in the following figure:

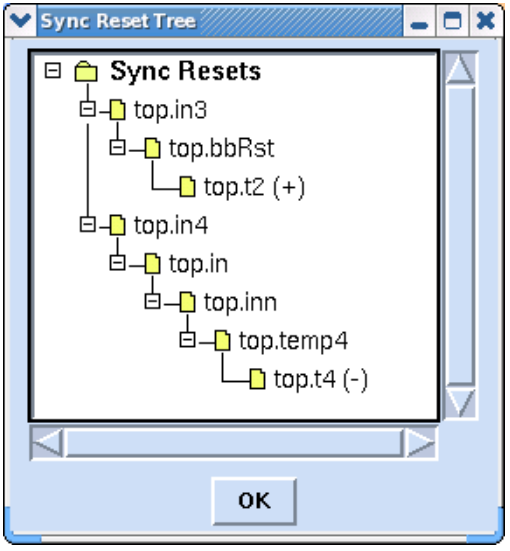


FIGURE 59. The Design Reset Tree Window

2. Review the reset tree and the sequential cells/black boxes driven by it.

   The reset tree shows a reset polarity for each leaf-level cell, such as a flip-flop, a latch, or a black box. In the reset tree:

   - The (+) suffix indicates a positive reset polarity.

- The `(-)` suffix indicates a negative reset polarity.
- The `(x)` suffix indicates that reset polarity could not be uniquely determined.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

//test.v

```
module top(clk,d, in1, in2, in3,in4);
  input clk,d, in1, in2, in3, in4;
  reg t1, t2, t3, Q, t4;
  wire temp1, temp2, temp3,temp4;
  always @ (clk)
   begin
          if (clk == 1'b1)
              Q <= d;
   end  // End Latch
 always@(posedge clk)
    if(Q)
       t1 = 1'b0;
    else
       t1 = d;
BB C1(in1,in2,temp2,temp3);
wire bbRst= temp2 & in3;
always@(posedge clk)
   if(bbRst)
     t2 = 1'b0;
   else
     t2 = d;
always@(posedge clk)
   if(temp1)
      t3 = 1'b0;
   else
      t3 = d;
wire in = !in4;
wire inn = in;
FD1 A(.D(inn),.CP(clk),.Q(temp4));
always@(posedge clk)
 if(temp4)
  t4 = 1'b0;
   else
  t4 = d;
endmodule

module BB(in1,in2,q1,q2);
 input in1,in2;
 output q1,q2;
endmodule
```

// constr.sgdc

```
current_design top
reset -sync -name in3
reset -sync -name in4
```

For the above example, the *Ar_syncrstTree* rule generates the following reset tree:

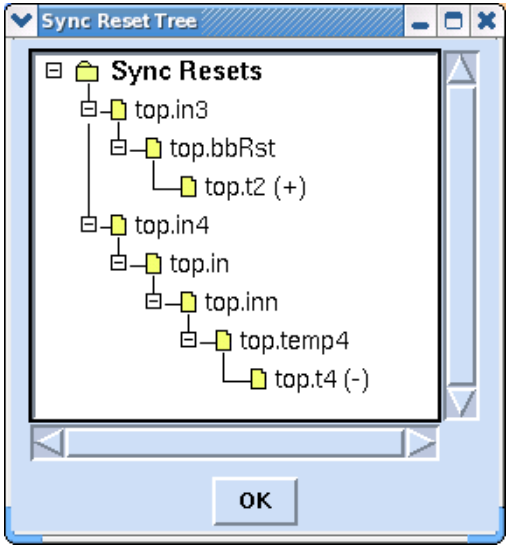


FIGURE 60. Synchronous reset tree generated by the Ar_syncrstTree rule

Default Severity Label

Info

Rule Group

FIND

Reports and Related Files

The SyncRstTree Report

Ar_glitch01

Glitch in reset paths

When To Use

Use this rule to detect glitch-prone *reset* circuits.

Description

Some glitches on the reset pin of a flip-flop might result in a metastable state. For example, consider an active low reset pin. A glitch of the form 010 on the logic driving the reset pin does not result in a metastable state. However, a glitch of the form 101 on the reset pin can cause metastable behavior.

Intuitively, a 010 glitch cannot cause a metastable state because after the transient transition to 1, the flip-flop would return to the reset state of 0 although it might produce a temporary output of 1 on the Q pin of the flop as shown in the following figure:

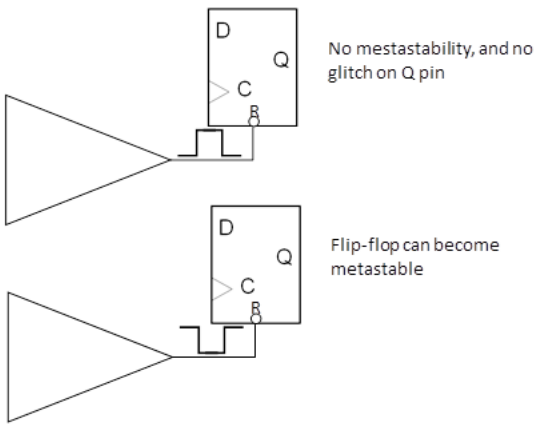


FIGURE 61.

However, as shown in the above figure, a 101 glitch can cause metastability because the flip-flop can be in two possible states after the glitch (either 0 if the glitch was long enough to cause the flip-flop to reset

its value or 1 if the glitch wasn't long enough for the flip-flop to reset its value). Note that this analysis assumes that data and clock pins of the flop are not changing at the same time as reset pin.

NOTE: *Spyglass CDC provides the Reset_check07 rule that reports presence of combinational logic on reset paths. Similarly, when a reset diverges and converges back, it is reported by the Ar_converge01 rule. These two rules detect specific structures in reset path that are prone to glitch. The Ar_glitch01 rule detects all the possible cases where multiple input changes at the input of the reset pin and can result in glitch.*

The *Ar_glitch01* rule reports glitch-prone *reset* logic.

Consider the following logic driving the active-low reset pin of a flip-flop:

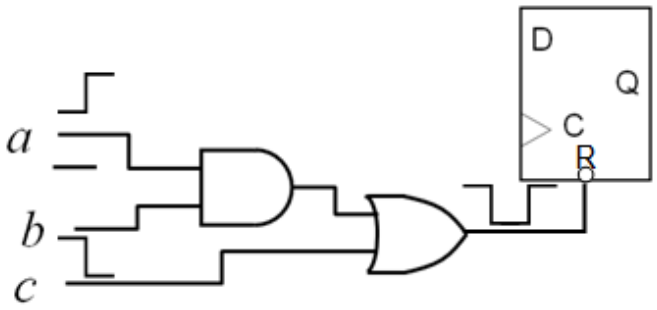


FIGURE 62.

In the above logic, multiple input changes at the input of the reset pin can result in a 101 glitch when:

- The steady value of a function before and after the change is equal to 1, and
- There is an intermediate transition (say `t`) that the function can pass through while changing values so that the value of the function at `t` is 0.

For example, the transition in *Figure 62* happens from `a'bc (011)` to `abc' (110)`. During this transition, if `c` changes to 0 before `a` changes to 1, the intermediate transition `a'bc'(010)` occurs, which generates an intermediate value of 0. As this transition sequence results in the 101 transition on the active-low reset pin, the flip-flop can become metastable.

Resets Considered by the Ar_Glitch01 Rule

The *Ar_glitch01* rule considers all the inferred reset signals in the reset paths for glitch checking. For example, this rule infers the following resets:

- Primary resets specified by the *reset* constraint. This scenario is shown in the following schematic:

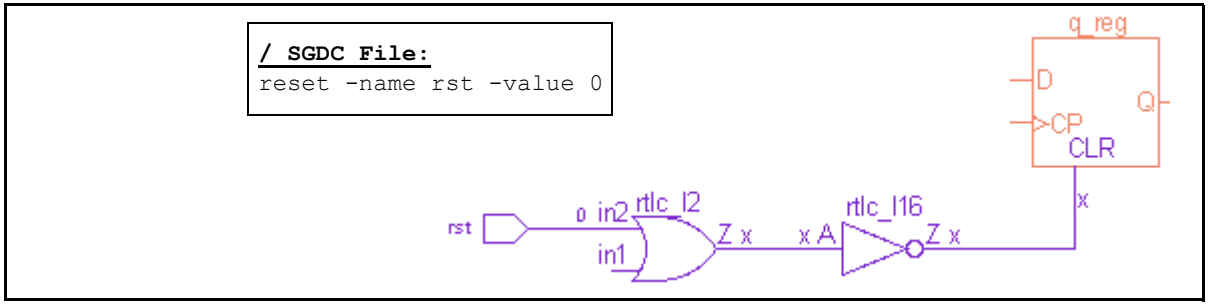


FIGURE 63.

- *Derived Resets*
- Resets from the output of the flip-flops *without* any preset/clear reset pin, as explained in the following figure:

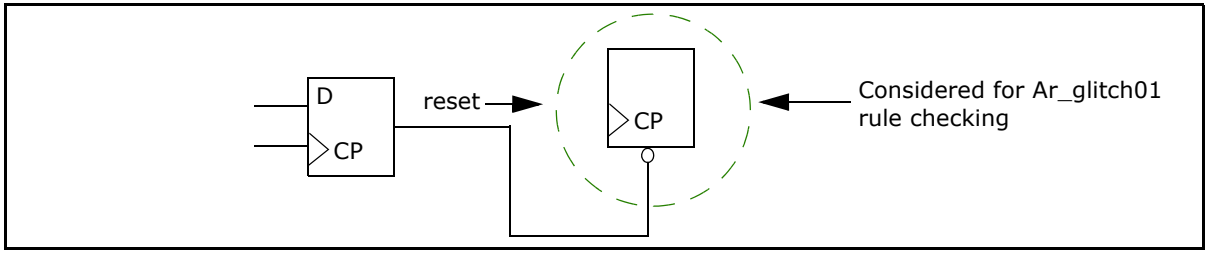


FIGURE 64.

- Resets from the output of a black box. This scenario is shown in the following figure:

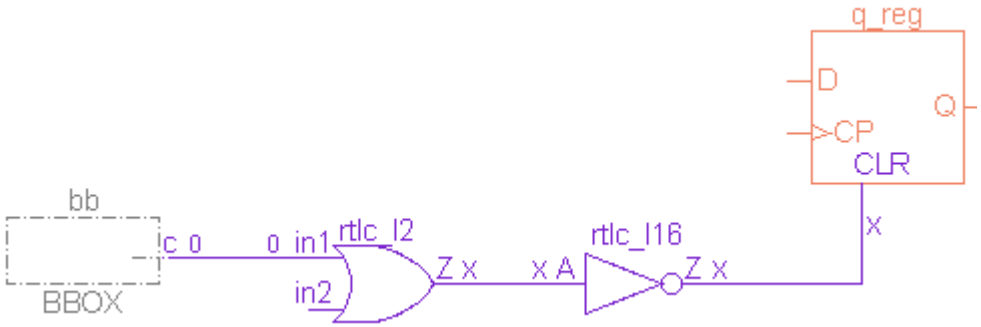


FIGURE 65.

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *ignore_set_case*: Default value is `none`. Set this parameter to `Ar_glitch01` to ignore simulation of block path traversal performed by set_case_analysis constraint.
- *ignore_bus_resets*: Default value is `yes`. Set this parameter to `no` to generate reset vector nets, which are not struct nets, in the *autoresets.sgdc* and the `generated_resets.sgdc` file.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *reset* (Optional): Use this constraint to specify reset signals in your design.

Messages and Suggested Fix

The following message appears if the design contains a glitch-prone logic:

[WARNING] Signal <signal-name> driving <pin-name> pin of element <element-name> has glitch

The arguments of the above message are explained below:

Argument	Description
<signal-name>	Name of the net driving the reset/set pin of a flip-flop
<pin-name>	Set/reset pin of a flip-flop or latch (active low or active high)
<element-name>	Name of the flip-flop or latch

Potential Issues

This violation appears if your design contains glitch-prone *reset* logic.

Consequences of Not Fixing

If you do not fix this violation, your design will contain metastability issues.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(d[1:0],clk1,clk2,sel,rst1,
           rst2,rst3,q[1:0]);
  input clk1,clk2,rst1,rst2,rst3,sel;
  input [1:0] d;
  output reg [1:0] q;

  wire ff3,q5;
  reg q3,q4,q2_2,q3_2,q_2,w1;

  assign ff3 = rst1 & rst2;
  assign q5 = ff3 || rst3;

  always @(posedge clk1 or posedge q5)
  if(q5 == 1'b1)
  q <= 2'b00;
  else
  q <= d;
endmodule
```

// constr.sgdc

```
current_design top

clock -name clk1
clock -name clk2

reset -name rst1 -value 0
reset -name rst2 -value 0
reset -name rst3 -value 0
```

For the above example, the *Ar_glitch01* rule reports the following three

violations:

```
Violation 1: Signal top.rst1 driving 'clear' pin of element
'top.q[1:0]' has glitch
Violation 2: Signal top.rst2 driving 'clear' pin of element
'top.q[1:0]' has glitch
Violation 3: Signal top.rst3 driving 'clear' pin of element
'top.q[1:0]' has glitch
```

For violation 1, the **Incremental Schematic** is as follows:

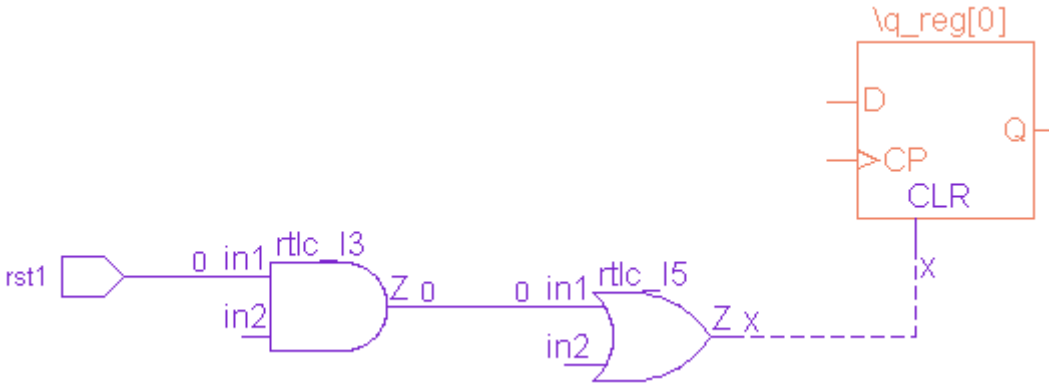


FIGURE 66. Schematic of the Ar_glitch01 rule violation

Schematic Highlight

Path from the specified reset to the set/reset pin of the flip-flop/latch/ sequential cell.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

No report or related file

Reset_info01

Reports signals that are likely to be asynchronous and synchronous preset and clear signals

When to Use

Use this rule to find asynchronous and synchronous preset and clear signals in a design.

Description

The *Reset_info01* rule reports asynchronous and synchronous preset and clear signals in a design.

This rule identifies the following types of asynchronous/synchronous preset or clear signals:

Traced to	Reset Type
Primary inputs	Primary Presets/Clears
Black box instances and instances of ASIC cells whose functional description is not available	Black box Presets/Clears
Outputs of flip-flops	Derived Presets/Clears
Hanging nets	Undriven Presets/Clears
Outputs of latches or tristate gates	Gated Preset/Clear

The following figure shows the scenario in which this rule reports a violation:

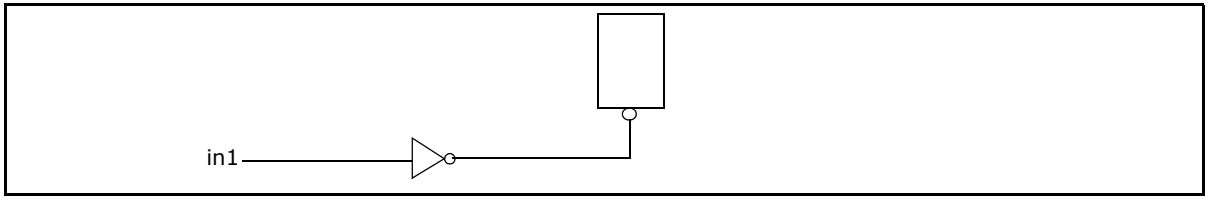


FIGURE 67. The Reset_info01 Rule Violation

In the above scenario, `in1` is used asynchronously as it reaches the asynchronous reset pin of the flip-flop. Therefore, the *Reset_info01* rule reports a violation in this case.

Rule Exceptions

This rule does not report presets or clears tied to a constant value in a design or specified as a constant signal by using the *set_case_analysis* constraint.

Parameter(s)

- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *ignore_bus_resets*: Default value is `yes`. Set this parameter to `no` to generate reset vector nets, which are not struct nets, in the *autoresets.sgdc* and the `generated_resets.sgdc` file.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *infer_constraint_from_abstract_blocks*: Default value is `no`. Set this parameter to a supported value to enable the supported rules infer clock, reset, set_case_analysis constraints from similar constraint defined in abstracted blocks.

Constraint(s)

- *sync_reset_style* (Optional): When a primary, black box, undriven, or gated type reset does not follow synchronous reset style specified by

this constraint, the *Reset_info01* rule does not report such source resets. In addition, the autoresets.sgdc file also contains these kind of resets in a commented form.

- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location where an asynchronous preset candidate *<rst-name>* of type *<rst-type>* is first used in a design:

[RstI1_1] [INFO] Candidate <Set | Clear>: <rst-name> of type <rst-type>

Where *<rst-type>* can be:

- `Primary Set`, `BlackBox Set`, `Derived Set`, or `Undriven Set` for `set` type of signals.
- `Primary Clear`, `BlackBox Clear`, `Derived Clear`, or `Undriven Clear` for `Clear` type of signals.

Potential Issues

This violation appears if your design contains an asynchronous preset signals or asynchronous clear signal.

Consequences of Not Fixing

None

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

The following message appears at the location where a synchronous preset candidate *<rst-name>* of type *<rst-type>* is first used in the design:

[RstI1_2] [INFO] Candidate synchronous <Set | Clear>: <rst-name> of type <rst-type>

Where *<rst-type>* can be:

- `Primary Synchronous Set`, `BlackBox Synchronous Set`, `Derived Synchronous Set`, or `Undriven Synchronous Set` for Set type of signals.
- `Primary Synchronous Clear`, `BlackBox Synchronous Clear`, `Derived Synchronous Clear`, or `Undriven Synchronous Clear` for `clear` type of signals.

Potential Issues

This violation appears if your design contains a synchronous preset or clear signal.

Consequences of Not Fixing

None

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the spyglass_reports/clock-reset/autoresets.sgdc file to review the resets reported as probable resets.
2. View the *Incremental Schematic* for each reset reported by this rule.
3. If this is not a real reset, remove it from the SGDC file and copy the final SGDC file to the desired location.
4. You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

Consider the following design file given as an input for SpyGlass analysis:

```
module top(data, out, clock, reset, enable);
  input data, clock, reset, enable;
  output out;
  reg out;
```

```
  wire rst;
  wire w1, w2;
  assign w1 = reset;
  assign w2 = enable;
  assign rst = w1 & w2;
  bbox I1(rst, data);
  always @(posedge clock or posedge rst)
    if(rst)
      out <= 0;
    else
      out <= data;
endmodule
```

For the above example, the *Reset_info01* rule reports the `top.enable` and `top.reset` signals.

The following schematic shows the path from the reset source to the reset pin of the flip-flop for the `top.enable` signal:

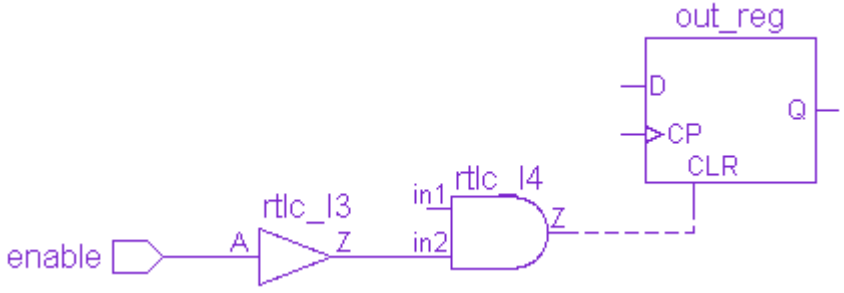


FIGURE 68. Schematic of the Reset_info01 Rule Violation

Schematic Details

The *Reset_info01* rule highlights the path from a reset source to a reset pin of a flip-flop in the schematic.

Rule Group

`FIND`

Default Severity Label

Info

Reports and Related Files

- *The Clock-Reset-Summary Report*
- autoresets.sgdc: This file contains all primary resets and black box presets/clears specified in an SGDC format.

NOTE: *Undriven presets/clears are not reported in the* autoresets.sgdc *file as presets/clears are not expected to be undriven.*

- generated_resets.sgdc: This file contains all derived presets/clears.

NOTE: *Resets that are categorized as definite and probable are reported in autoresets.sgdc and generated_resets.sgdc files only.*

- A spreadsheet file that contains the following two tabs:
  - *Asynchronous Reset*: This tab contains primary and black box asynchronous reset and its active value (similar to autoresets.sgdc)
  - *Synchronous Reset*: This tab contains primary and black box synchronous reset and its active value (similar to generated_resets.sgdc)

  If the *gen_sync_reset_style_info* parameter is set to `yes`, this spreadsheet also shows the following information:
  - Load
  - Presence of combinational logic in reset path
  - Polarity
  - Presence of the `sync_set_reset` pragma
  - Reset usage in first if of sequential block

Reset_info02

Prints an asynchronous preset and clear tree

When to Use

Use this rule to generate a reset tree.

Prerequisites

Specify reset signals by using the *reset* constraint or the *use_inferred_resets* parameter.

Description

The *Reset_info02* rule prints a preset/clear tree for the specified preset/clear signals.

Rule Exceptions

While traversing a design to generate reset trees, this rule does not traverse beyond following objects:

- Black box pins without *assume_path*
- Sequential library cells without a functional arc
- Pins other than the resets of sequential elements

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

Constraint(s)

- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears to indicate that a reset tree is generated:

`[INFO] Reset Tree generated`

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Click on the message in GUI.

   This displays the design *Reset Tree* window, as shown in the following figure:

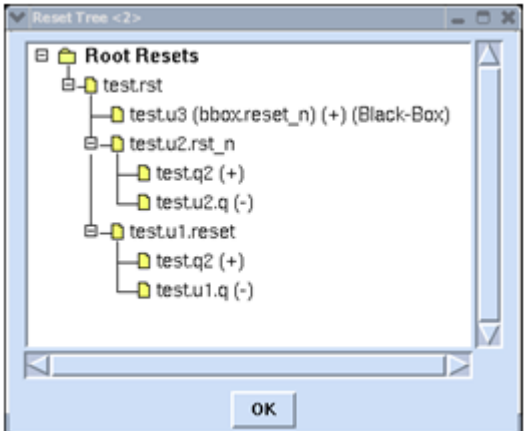


FIGURE 69. The Design Reset Tree Window

2. Review the reset tree and the sequential cells/black boxes driven by it.

The reset tree shows a reset polarity for each leaf-level cell, such as a flip-flop, a latch, or a black box. In the reset tree:

- The `(+)` suffix indicates a positive reset polarity.
- The `(-)` suffix indicates a negative reset polarity.
- The `(x)` suffix indicates that reset polarity could not be uniquely determined.

The reset tree report is generated in a text format under ./spyglass_reports.

Example Code and/or Schematic

Consider the following design file given as an input for SpyGlass analysis (the *use_inferred_resets* parameter is set to `yes` in this case):

```
module top(data, out, clock, reset, enable);
  input data, clock, reset, enable;
  output out;
  reg out;
  wire rst;
  wire w1, w2;
  assign w1 = reset;
  assign w2 = enable;
  assign rst = w1 & w2;
  bbox I1(rst, data);
  always @(posedge clock or posedge rst)
    if(rst)
      out <= 0;
    else
      out <= data;
endmodule
```

After SpyGlass analysis, double-click on the message of the *Reset_info02* rule. The following reset tree is generated:

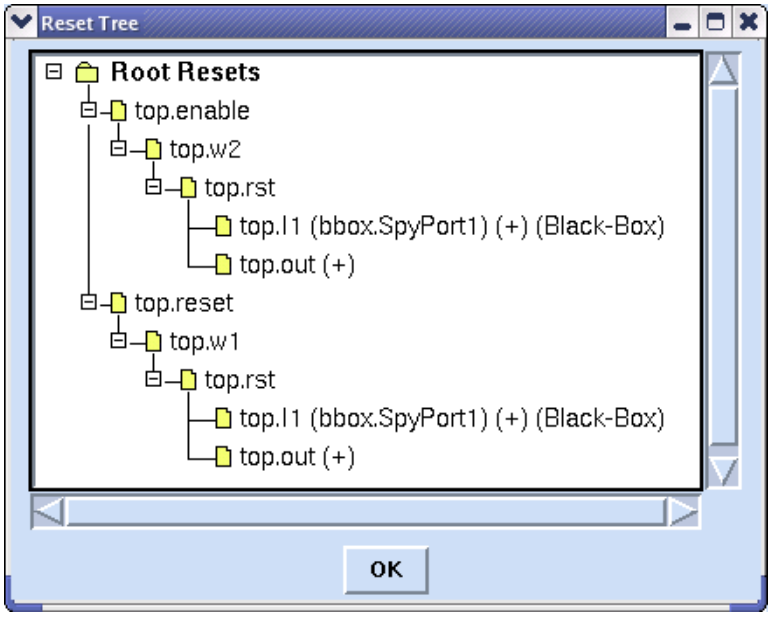


FIGURE 70. The Design Reset Tree Window

Default Severity Label

Info

Rule Group

FIND

Reports and Related Files

The RSTree Report

Reset_info09

Reports unconstrained asynchronous reset nets and reset nets tied to a constant value

Description

The Reset_info09 rule runs *Reset_info09a* and *Reset_info09b* rules.

Reset_info09a

Reports unconstrained asynchronous reset nets

When to Use

Use this rule to check correctness of setup for reset checks by detecting unconstrained asynchronous nets in a design.

Description

The *Reset_info09a* rule reports unconstrained asynchronous reset nets.

The following nets are considered as unconstrained:

- Reset nets that are not specified in the constraints file.
- Reset nets that are specified in the constraints file, but are not propagated due to the presence of some unspecified cells.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *Reset_info09a_filter_on_constant_clock*: Default value is `no`. Set this parameter to `yes` to report violation messages of the `Reset_info09a` rule for flops whose clock pin is receiving a constant value.
- *report_common_reset*: Default value is `no`. Set this parameter to `yes` to enable the Reset_info09a rule to find the common reset source by skipping buffers/inverters and MUX/combo gates acting as buffer.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location of a flip-flop instance where a net *<net-name>* connected to a reset pin is unconstrained:

`[INFO] Reset net <net-name> is unconstrained`

Potential Issues

This violation appears if your design contains either of the following:

- Reset nets that are not specified in an SGDC file
- Reset nets that are specified in an SGDC file but are not propagated because of blocked paths

Consequences of Not Fixing

If all sequential elements in a design are not controlled by external resets, you may get incorrect functional results.

It might also have impact on other reset-related rule checking if the issues reported by this rule are not fixed.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Back-trace the reset net that is reported as un-constrained.

   Perform appropriate actions based on the following possibilities:

   - The reported net is driven by a black box, and the reset defined in the SGDC file is reaching to this black box.

     Action: Set the *assume_path* constraint.

   - The reported net is unconnected.

     Action: Update the design.

   - The reported net is driven by a combinational logic that blocks the reset defined in the SGDC file.

     Action: Enable *Show Case Analysis* in the *Incremental Schematic* window, and view if constant propagation blocks the defined reset net.

   - The reported net was driven by the signal that is a reset candidate, but it not defined in the SGDC file.

     Action: Refer to the messages of the *Propagate_Resets* rule.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

// test.v

```
module test(clk1,rst,in,out);
input clk1,rst,in;
output reg out;
BUF B1(.A(rst),.Z(nrst));
always@(posedge clk1 or negedge nrst)
    if(!nrst)
     out <= 1 'b0;
    else
     out <= in;
endmodule
```

// constraints.sgdc

```
current_design test
reset -name rst
```

For the above example, the *Reset_info09a* rule reports a violation as the `test.nrst` reset net is unconstrained. This is shown in the following schematic:

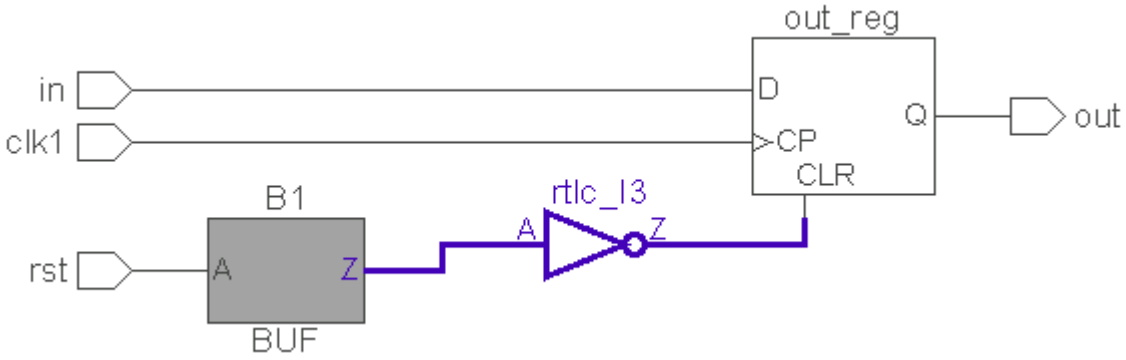


FIGURE 71. Schematic of the Reset_info09a Rule Violation

To fix this violation, perform the following actions:

- Provide a functional definition of the `BUF` library cell for propagation of the `rst` signal.
- Apply the following *reset* constraint on a net connected to the output pin of `BUF`:

  ```
  reset -name nrst -value 0
  ```

Schematic Details

The *Reset_info09a* rule highlights the path from an unconstrained reset net to the flip-flop reset pin in schematic.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

No report and related file

Reset_info09b

Reports asynchronous reset nets that are tied to a constant value

When to Use

Use this rule to detect asynchronous reset nets in a design that are tied to constant value.

Description

The *Reset_info09b* rule reports asynchronous resets (specified in an SGDC file or automatically-inferred) that are tied to an active or inactive constant value.

Asynchronous reset nets are tied to a constant value in any of the following ways:

- Constrained by *set_case_analysis* constraint
- Connected to supply nets
- Connected to tied-off/on cells

Consider the scenario shown in the following figure:

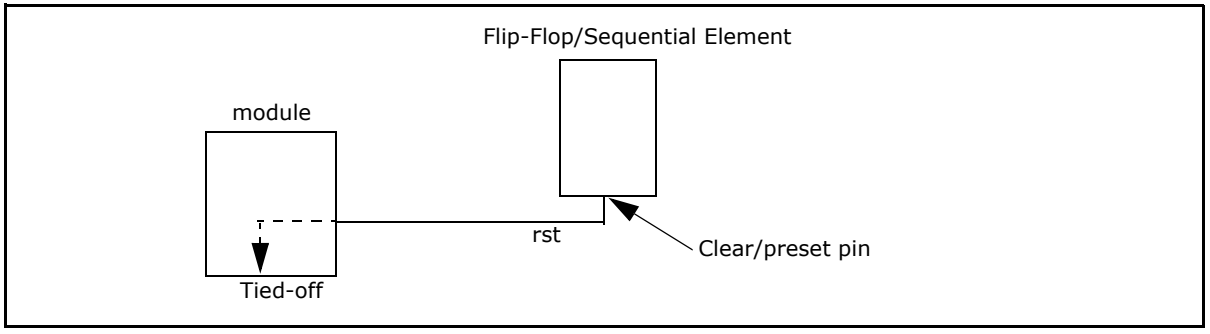


FIGURE 72. The rst reset connected to a tied-off cell

In the above scenario, the *Reset_info09b* rule reports a violation for the `rst` reset as it is connected to a tied-off cell.

Parameter(s)

use_inferred_resets: Default value is `no`. Set this parameter to `yes` to use

auto-generated reset information.

Constraint(s)

- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *sgdc*: (Optional): Use this constraint to specify a block-level SGDC file to be imported or specify blocks for which block-level SGDC file is to be generated.

Messages and Suggested Fix

The following message appears at the location of a flip-flop instance where the reset net *<net-name>* is tied to its inactive/active value with a constant value *<value>*:

[INFO] Reset net <net-name> is tied to its inactive value with a constant value <0|1>

[ERROR] Reset net <net-name> is tied to its active value with a constant value <0|1>

Potential Issues

This violation appears if your design contains sequential elements whose reset or set pin is tied to constant value.

Consequences of Not Fixing

Such constant resets may be unintentional constant resets in your design and may result in functional issues.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Enable *Show Case Analysis* to check constant propagation in the schematic that is causing the reset to be constant.
3. Check the case analysis settings.

Example

Consider the following files specified in SpyGlass analysis:

// test.v

```
module test(clk1,in,out,rst,preset);

input clk1;
input in,rst,preset;
output out;
reg t1;
wire reset,en;
assign en = 1'b1;     // tied at inactive value
assign reset = rst | en;
always@(posedge clk1 or negedge reset) // RTL flip-flop
    if(!reset)
     t1 <= 1'b0;
    else
     t1 <= in;
 assign out = t1;

endmodule
```

// constraints.sgdc

```
current_design test
clock -name clk1
reset -name rst
```

In the above example, the *Reset_info09b* rule reports a violation as the `test.reset` reset net is tied to constant value 1. This is shown in the following schematic:

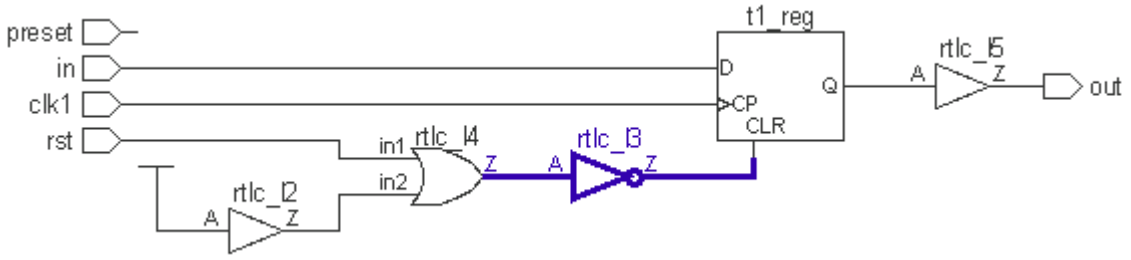


FIGURE 73. Schematic of the Reset_info09b Rule Violation

To fix this violation, review and change the supply net connected to the `OR` gate.

Schematic Details

The *Reset_info09b* rule highlights the path from a reset pin to a net where

the constant value is implied.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

No report or related file

Clock and Reset Information Rules

The SpyGlass CDC solution has the following rule for generating clock and reset information:

Rule	Reports
Clock_Reset_info01	SpyGlass CDC solution usage matrix

Clock_Reset_info01

Generates the Clock-Reset Matrix

When to Use

Use this rule to find the clock domains of a design where an asynchronous reset is applied and vice-versa.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Specify reset signals in any of the following ways:

- By using the `reset` constraint
- By setting the *use_inferred_resets* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Clock_Reset_info01* rule generates a clock-reset usage matrix for the specified clock and preset/clear signals.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use the auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use the auto-generated reset information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.

- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Message Details

The following message appears when this rule is run:

`[INFO] Clock-Reset Matrix information generated for Clock-Reset-Summary report`

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open *The Clock-Reset-Summary Report*, and refer to *Section F: Clock-Reset Matrix Section*.
2. Look along a column length-wise to know the clock names and domains where a particular reset is used.
3. Look along row-wise to know the asynchronous resets used in a particular domain.

The *Clock_Reset_info01* rule is an informational rule. Since, this rule generates information for *The Clock-Reset-Summary Report*, not running it would mean that the corresponding report section is not generated.

Example Code and/or Schematic

Consider a module, `myDU`, with the following clock-reset usage:

- `clk1` and `preset1` are used together.

- clk1 & clk2 are used with preset12.
- clk2 and rst2 are used together.
- clk1 or clk2 is not used with rst3.
- clk3 is not used with any of preset1, preset12, or rst2.
- clk3 and rst3 are used together.

The clk1 and clk2 clocks are in the same domain, and the clk3 clock is in a different domain.

Now consider that you specify the following design constraints specifications while analyzing your design:

```
current_design myDU
  clock -name myDU.clk1 -domain A
  clock -name myDU.clk2 -domain A
  clock -name myDU.clk3 -domain B
  reset -name myDU.preset1 -value 1
  reset -name myDU.preset12 -value 1
  reset -name myDU.rst2 -value 1
  reset -name myDU.rst3 -value 1
```

In the above scenario, this rule generates the following matrix for each design unit:

```
Resets     myDU.preset1  myDU.preset12        myDU.rst2   myDU.rst3
=================================================================
Domains
 A         myDU.clk1     myDU.clk1,myDU.clk2  myDU.clk2     -
 B             -              -                  -        myDU.clk3
```

The dash character (-) in a matrix cell indicates that there is no interaction between clock-domain and reset signal.

Default Severity Label

Info

Rule Group

INFORMATION

Reports and Related Files

Section F: Clock-Reset Matrix Section of *The Clock-Reset-Summary Report*

Reset Synchronization Rules

The SpyGlass CDC solution has the following rules for checking reset synchronization status:

Rule	Reports
Ar_resetcross01	Reset domain crossings between sequential elements of the same clock domain
Ar_asyncdeassert01	Reports if reset signal is asynchronously de-asserted
Ar_syncdeassert01	Reports if reset signal is synchronously de-asserted or not de-asserted at all
Ar_sync01	Reports synchronized reset signals in the design
Ar_unsync01	Reports unsynchronized reset signals in the design
Reset_sync01	Asynchronous reset signals that are not de-asserted synchronously with the corresponding clock signal
Reset_sync02	Asynchronous resets used in a clock domain and generated in one of its asynchronous clock domains
Reset_sync03	Multi-flop reset synchronizers in the design
Reset_sync04	Asynchronous resets synchronized more than once in the same clock domain
SGDC_cdc_define_transition01	Checks for compatible values in cdc_define_transition

Using the Reset Domain Crossing (RDC) Flow

NOTE: *The RDC flow works only with the Advanced_RDC license feature.*

The *Ar_resetcross01* rule reports a warning for the reset-domain crossings between the sequential elements of the same clock domain. Such crossings may cause metastability issues.

However, if you know the crossings that do not cause metastability issues, you can filter such crossings to reduce noise from the *Ar_resetcross01* rule.

To filter RDCs, specify an enable condition for these crossings by using the `qualifier` constraint so that these crossings are qualified as synchronized. Such crossings are then reported as informational messages instead of warnings.

For details, see *Enhancements to the qualifier Constraint* and *Example of Specifying an Enable Condition*.

The *Setup_rdc01* identifies reset domain crossings that have the same asynchronous resets at both the source and destination but with side inputs in between the resets.

The *Ac_resetcross01* rule reports invalid reset ordering between sequential elements of the same clock domain.

The *RFPSetup* rule reports a violation if the reset_filter_path constraint is not used to filter reset domain crossings. The *SGDC_qualifier23* rule perform sanity checks related to the qualifier constraint.

The following table lists the parameters that support the RDC flow.

enable_sim_check_rdc	*enable_sync_check_rdc*
ignore_qualifier_mismatch_rdc	*rdc_reduce_pessimism*
show_unsync_qualifier_rdc	*enable_multiflop_sync*
enable_diff_clkdom_rdc	*report_sync_rdc*
rdc_allow_sync_reset	

The *The SynchInfo Report* report shows the synchronized RDCs.

Enhancements to the qualifier Constraint

To filter a reset-domain crossing reported by the *Ar_resetcross01* rule,

specify an enable condition by using one of the *Syntax of the qualifier Constraint in the RDC Flow*.

In this syntax, `-name` (existing argument) and `-rdc` (new argument) of the `qualifier` constraint are the mandatory arguments. All the new arguments are described in the topic, *New Arguments of the qualifier Constraint*.

The `-name` argument accepts the name of an enable signal to be used to filter reset-domain crossings. See *Example of Specifying an Enable Condition*.

While specifying an enable signal, ensure the following:

- The enable signal should be either in the same clock domain as the destination element (that is, driven in the same clock domain) or driven from a port or black box for which no clock domain is specified.
- The enable signal should be either in the same destination reset domain (that is, driven in the same reset domain as that of the destination element) or driven from a port/black box for which no reset domain is specified.

New Arguments of the qualifier Constraint

The `qualifier` constraint is enhanced to support the following new arguments:

- *-rdc* (Mandatory)

  Use this argument to enable the RDC flow in which you can specify an enable signal to filter reset-domain crossings.

- *-from_rst <src-rst>* (Optional)

  Use this argument to specify the source reset in a reset-domain crossing.

- *-to_rst <dest-rst>* (Optional)

  Use this argument to specify the destination reset in a reset-domain crossing.

See also:

- *Arguments Applicable in the RDC Flow*
- *Arguments Invalid in the RDC Flow*

Syntax of the qualifier Constraint in the RDC Flow

The following usages of the qualifier constraint are applicable in the RDC flow:

Usage 1:

```
qualifier
  -rdc
  -name
  -from_rst
  -to_rst
```

Usage 2:

```
qualifier
  -rdc
  -name
  from_rst
  -to_rst
  -to_clk
```

Usage 3:

```
qualifier
  -rdc
  -name
  -from_rst
  -to_rst
  -to_domain
```

Usage 4:

```
qualifier
  -rdc
  -name
  -from_obj
  -to_obj
```

Arguments Applicable in the RDC Flow

The following existing arguments of the `qualifier` constraint **are applicable** in this flow:

-rdc	-from_rst	-to_rst	-to_clk	-to_domain
-from_obj	-to_obj	-name		

Arguments Invalid in the RDC Flow

The following existing arguments of the `qualifier` constraint are **not applicable** in this flow:

-from_clk	-from_domain	-thru_obj	-crossing	-type
-enable	-ignore	-strict		

Example of Specifying an Enable Condition

Consider the example shown in the following figure:

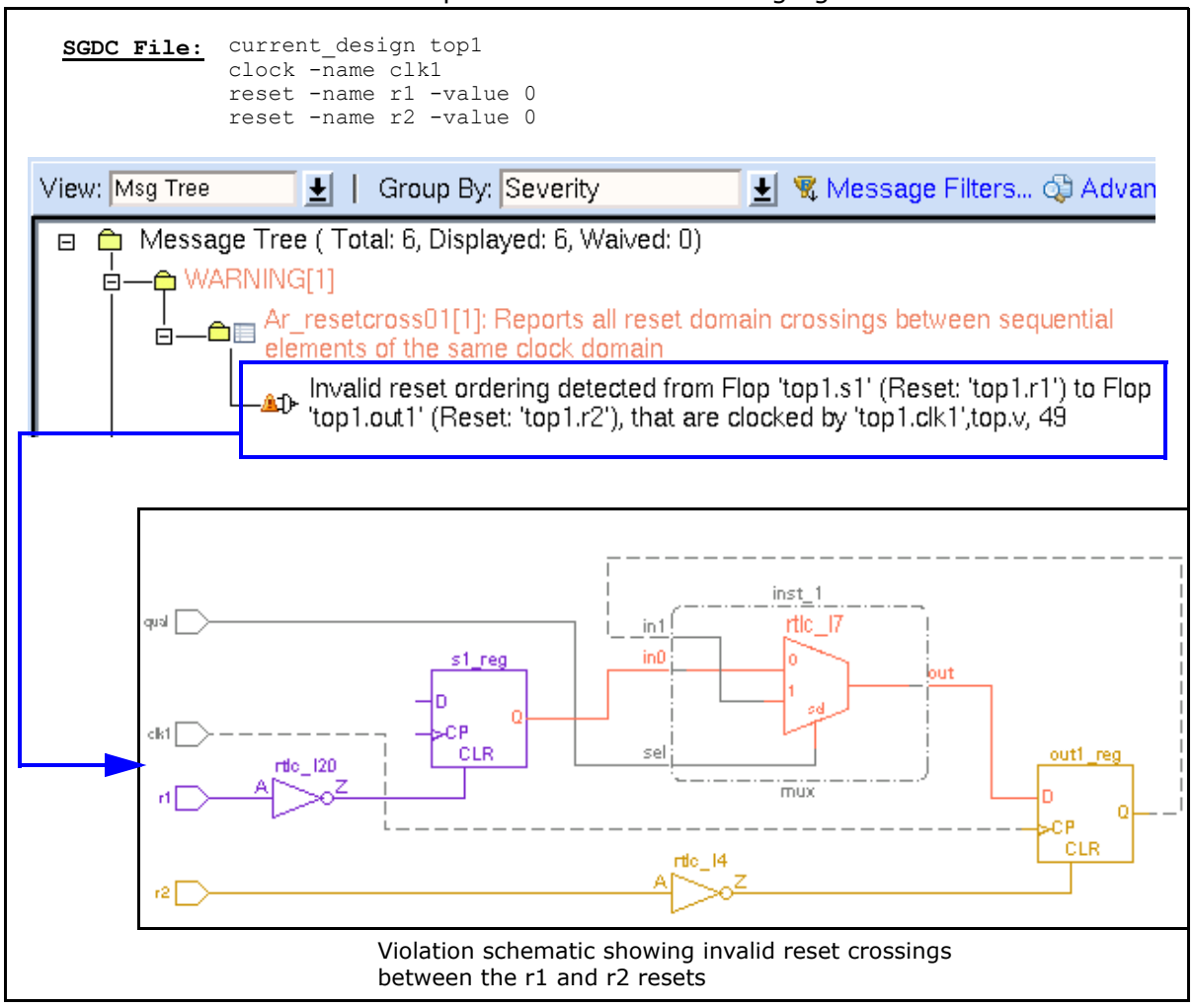


FIGURE 74. Schematic showing invalid reset crossings

To filter the above-reported crossing so that it is not reported as invalid, specify an enable signal for the crossing by using the following `qualifier` constraint:

```
qualifier -from_rst r1 -to_rst r2 -name qual -rdc
```

The above constraint qualifies the crossing between the r1 and r2 resets as synchronized. Therefore, these crossings are reported as synchronized by the *Ar_resetcross01* rule.

The following figure shows the reporting of these crossings as synchronized after you specify an enable signal using the `qualifier` constraint:

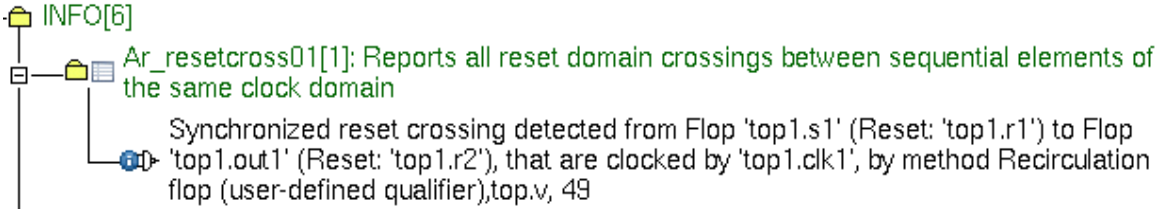


FIGURE 75. Crossings reported as synchronized crossings

Ac_resetcross01

Reports invalid reset ordering between sequential elements of the same clock domain

When to Use

Use this rule when different asynchronous resets are used in flip-flops, which are connected in the data path.

Prerequisites

Specify the following information before running this rule:

- Specify reset signals by using the `reset` constraint.
- Enable this rule by specifying the `set_goal_option addrules {Ac_resetcross01}` command in a project file.

Description

The details of the *Ac_resetcross01* rule are covered under the following topics:

- *Reason for the Ac_resetcross01 Rule Violation*
- *Features of the Ac_resetcross01 Rule*
- *Rule Exceptions*

NOTE: *The Ac_resetcross01 rule will be deprecated in a future release. The functionality of the Ac_resetcross01 rule is covered by the Ar_resetcross01 rule.*

Reason for the *Ac_resetcross01* Rule Violation

The *Ac_resetcross01* rule reports a violation if there are paths between two sequential elements that are clocked by the same clock domain but have different asynchronous resets/sets. Such scenario is shown in the following figure:

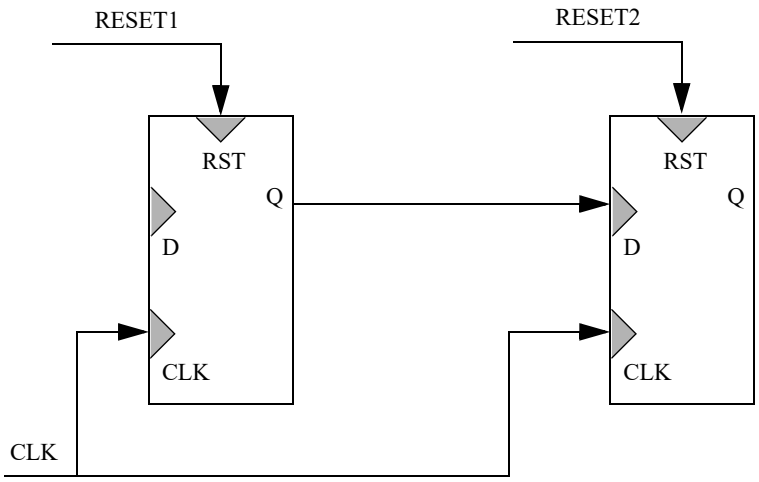


FIGURE 76. Scenario for the Ac_resetcross01 Rule Violation

In the above figure, there is always a possibility of set/clear on the first flip-flop while the second flip-flop is active.

Features of the Ac_resetcross01 Rule

The *Ac_resetcross01* rule has the following features:

- This rule checks only if both the connected sequential elements are clocked by the same clock domain. If the clock domains are different, the `Ac_unsync01/Ac_unsync02` and `Ac_sync01/Ac_sync02` rules report a violation.
- This rule ignores combinational logic between sequential elements.
- This rule stops traversal on black boxes, sequential elements, blocked paths, and primary ports.
- This rule checks for flip-flops, latches, or other sequential library cells that have asynchronous set or reset pins.
- This rule checks if there is a path between sequential elements through data or enable pins. For clock paths, this rule continues propagation beyond clock gate enables.

- Reset propagation on the sequential elements is used by the `Propagate_Resets` rule.
- This rule checks for both preset and clear pins of sequential element.
- If multiple resets reach to sequential elements and a reset is different for both source and destination, this rule reports violation with respect to the resets that are different.

Rule Exceptions

This rule does not report a violation for the cases in which:

- A reset is always inactive because of the `set_case_analysis` constraint or power/ground connections.
- A destination instance does not have a preset or clear pin.

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- `report_all_reset_cross`: Default value is `no`. Set this parameter to `yes` to report violations on reset crossings in which the destination receives no clear/set signal.
- `report_for_single_busbit`: Default value is `no`. Set this parameter to `yes` to report violations for single bit of a bus.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see `Possible Values to the reset_reduce_pessimism Parameter`.

- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Constraint(s)

- *reset* (Mandatory): Use this constraint to specify reset signals in a design.
- *define_reset_order* (Optional): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location where an invalid reset ordering is found between the source output net and the destination output net:

```
[WARNING] Invalid reset ordering detected from <cell-type>
'<src-net>' (Reset: '<source-reset>') to <cell-type> '<dest-
net>' (Reset: <dest-reset>) that are clocked by '<clock-name>'
```

NOTE: *If the* `report_inst_for_netlist` *parameter is set to yes, Message 2 is reported.*

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-net>	Source output net
<source-reset>	Source reset
<dest-net>	Destination output net
<dest-reset>	Destination reset
<clock-name>	Comma-separated list of clock names reaching the destination flip-flop

Potential Issues

This violation appears if your design contains flip-flops that are connected in a data path, but are triggered by different asynchronous resets.

Consequences of Not Fixing

If you do not fix this violation, it may result in setup/hold violations at the destination flip-flop because of an asynchronous reset trigger on the source flip-flop.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

- Open the incremental schematic of the violation.
- Analyze the `Reset Cone` of the source and destination flip-flops in the schematic.
- If the order of resets is such that the destination flip-flop is reset first, define it as a valid reset order by using the `define_reset_order` constraint.

  For example, the following schematic shows a valid reset order:

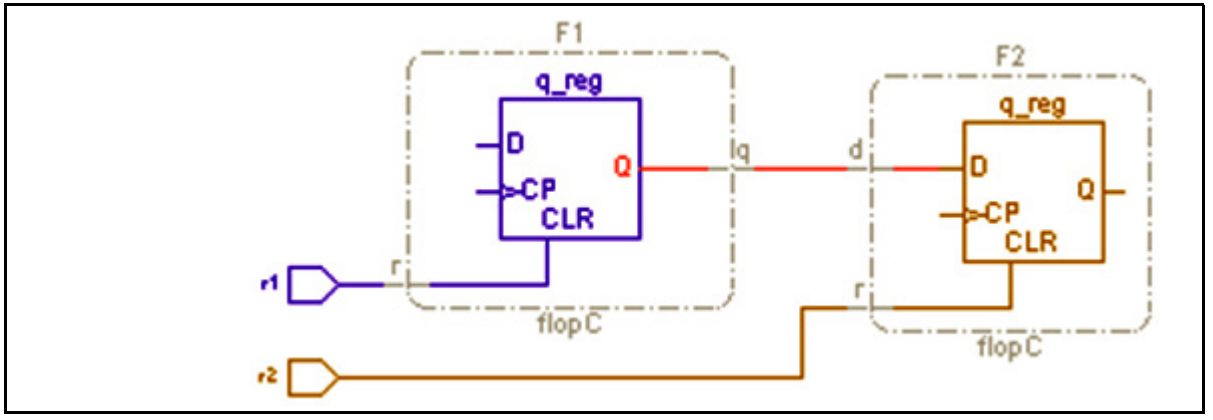


FIGURE 77. Example of a valid reset order

In the above example, specify the following constraint to define a valid reset order:

```
define_reset_order -from r1 -to r2
```

- However, if the order of resets is not known or if the source flip-flop is reset first, make necessary corrections in the design so that no setup/hold violations are reported.

Message 2

The following message appears when the `report_inst_for_netlist` parameter is set to `yes`, and an invalid reset ordering is found from the source instance pin to the destination instance pin:

```
[WARNING] Invalid reset ordering detected from <cell-type>
instance pin '<src-pin>' (Reset: '<src-reset>') to <cell-type>
instance pin '<dest-pin>' (Reset: <dest-reset>) that are
clocked by '<clock-name>'
```

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-pin>	Source instance pin
<src-reset>	Reset reaching source
<dest-pin>	Destination instance pin
<dest-reset>	Reset reaching destination
<clock-name>	Comma-separated list of clock names reaching the destination flip-flop

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v
module test(clk1,clk2,clk3,rst1,rst2,in1,in2,q1,q2);
input clk1,clk2,clk3;
input rst1,rst2;
input in1,in2;
output q2,q1;
reg t1,t2,t3;
always@(posedge clk1 or negedge rst1)
    if(!rst1)
     t1 <= 1'b0;
     else
      t1 <= in1;
always@(posedge clk2 or posedge rst2)
    if(rst2)
     t2 <= 1'b0;
    else
     t2 <= t1;
always@(posedge clk3 or posedge rst2)
    if(rst2)
     t3 <= 1'b0;
    else
     t3 <= in2;
assign q1 = t2;
assign q2 = t3;
endmodule
```

```
// constr.sgdc
current_design test
clock -name clk1 -domain d1
clock -name clk2 -domain d1
clock -name clk3 -domain d1
reset -name rst1 -async
reset -name rst2 -async
```

For the above example, the *Ac_resetcross01* rule reports a violation because of an invalid reset ordering present between the `t1` flip-flop (reset `rst1`) and the `t2` flip-flop (reset `rst2`).

The following figure shows the spreadsheet generated in this case:

A	B	C	D	E	F	G
ID	SOURCE	SOURCE RESET	DEST.	DEST. RESET	CLOCKS	WAIVED
5	test.t1	test.rst1	test.t2	test.rst2	test.clk2	No

FIGURE 78. Spreadsheet generated by the Ac_resetcross01 rule

To view the schematic of the violation shown in the above spreadsheet, click the link *5* in the *ID* column of the above spreadsheet and then click  .

The following figure shows the schematic of this violation:

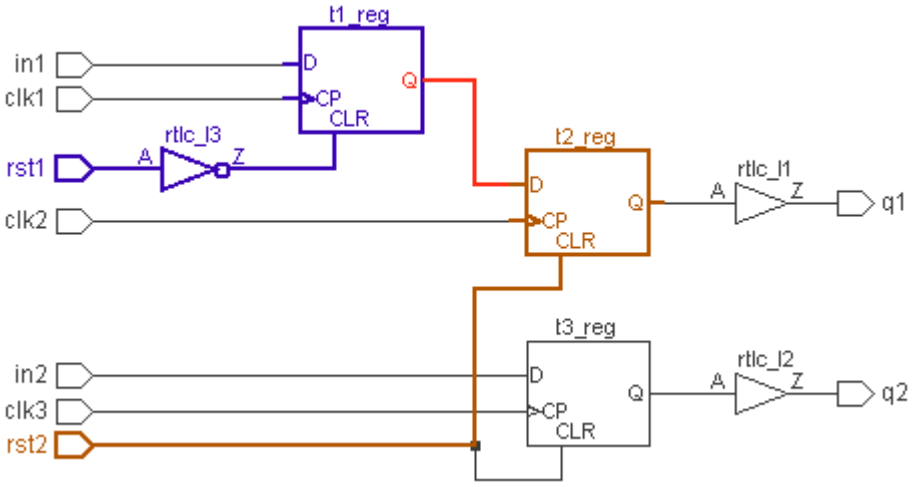


FIGURE 79. Schematic of the Ac_resetcross01 Rule Violation

Schematic Details

The *Ac_resetcross01* rule highlights the following information in the schematic:

- Source instance and its reset path
- Destination instance and its reset path
- Path between source and destination instance

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

- The CKSGDCInfo Report
- Ac_resetcross01.csv. See *Figure 78*.

Ar_resetcross01

Reports all reset domain crossings between sequential elements

When to Use

Use this rule when different asynchronous resets are used in flip-flops, which are connected in the data path.

Prerequisites

Specify the following information before running this rule:

- Specify reset signals by using the `reset` constraint.
- Enable this rule by specifying the `set_goal_option addrules {Ar_resetcross01}` command in a project file.
- Use the `Advanced_RDC` license.

Description

The details of the *Ar_resetcross01* rule are covered under the following topics:

Reason for the *Ar_resetcross01* Rule Violation

The *Ar_resetcross01* rule reports a violation if there are paths between two sequential elements that have different asynchronous resets/sets. Such scenario is shown in the following figure:

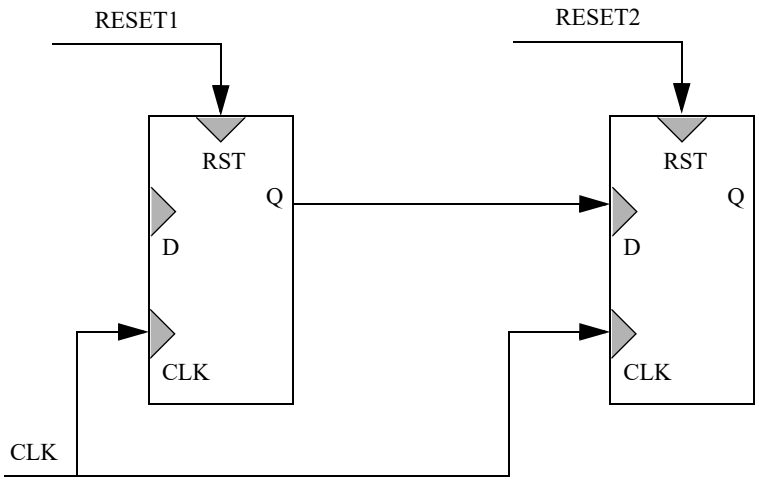


FIGURE 80. Scenario for the Ar_resetcross01 Rule Violation

In the above figure, there is always a possibility of set/clear on the first flip-flop while the second flip-flop is active.

Reasons for Unsynchronized Crossings Reported by Ar_resetcross01

The Ar_resetcross01 rule reports unsynchronized crossings because of the following reasons (starting from the highest priority):

Reason - Gating logic not accepted: qualifier merges with reconverging source

Reason - Gating logic not accepted: qualifier convergence is invalid

Reason - Gating logic not accepted: qualifier invalid

Reason - Gating logic not accepted: gate-type invalid

Reason - Qualifier not propagated: clock reaching at qualifier does not match destination

Reason - Qualifier not propagated: reset reaching at qualifier does not match destination

Reason - Qualifier not found

NOTE: *These reasons are reported only if the show_unsync_qualifier_rdc parameter is set to yes.*

- Reason - Gating logic not accepted: qualifier merges with reconverging source

  This reason is reported when a source diverges and the diverged source path merges with a valid qualifier before re-converging. For example, SpyGlass reports this reason in the scenario shown in Figure 79 where s1_reg is diverging and one of the diverged path merges with the qualifier before re-converging on the AND gate.

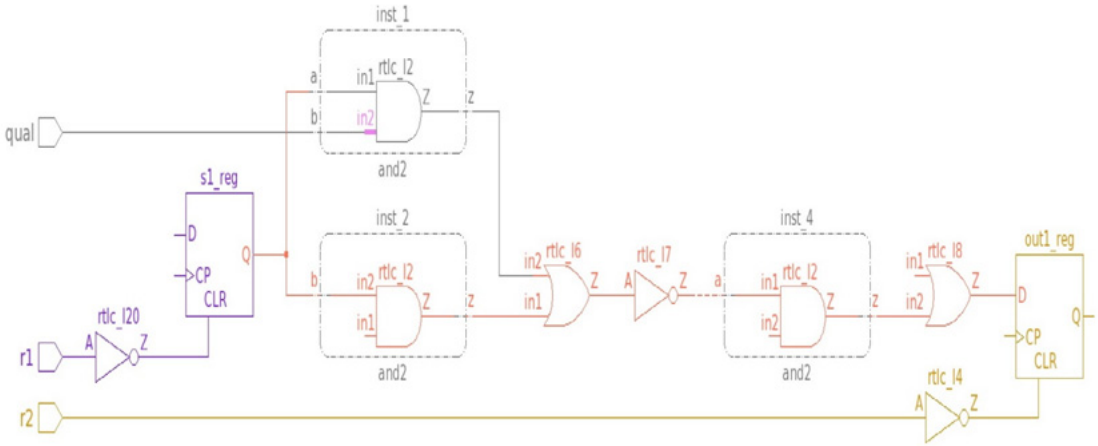


FIGURE 81. Qualifier merges with reconverging source

- Reason - Gating logic not accepted: qualifier convergence is invalid

  This reason is reported when the source and qualifier converges at any gate but in an invalid manner. For example, as shown in the following figure, in a Mux-Select based Synchronization scheme, even if a Mux is a valid gate for source and qualifier convergence but this reason is reported if:

  - ❒ Any source drives the MUX select input
  - ❒ None of the Mux data pin is without source
  - ❒ No qualifier drives the MUX select input

  If the *show_unsync_qualifier_rdc* parameter is set to `backward`, only the above reason is reported. If the parameter is set to `yes`, the invalid qualifier convergence reason is replaced with one of the following specific reasons:

"Gating logic not accepted: source drives MUX select input";

"Gating logic not accepted: source and qualifier drive MUX data inputs";

"Gating logic not accepted: only sources drive MUX data inputs; atleast one destination domain signal should drive a MUX data input";

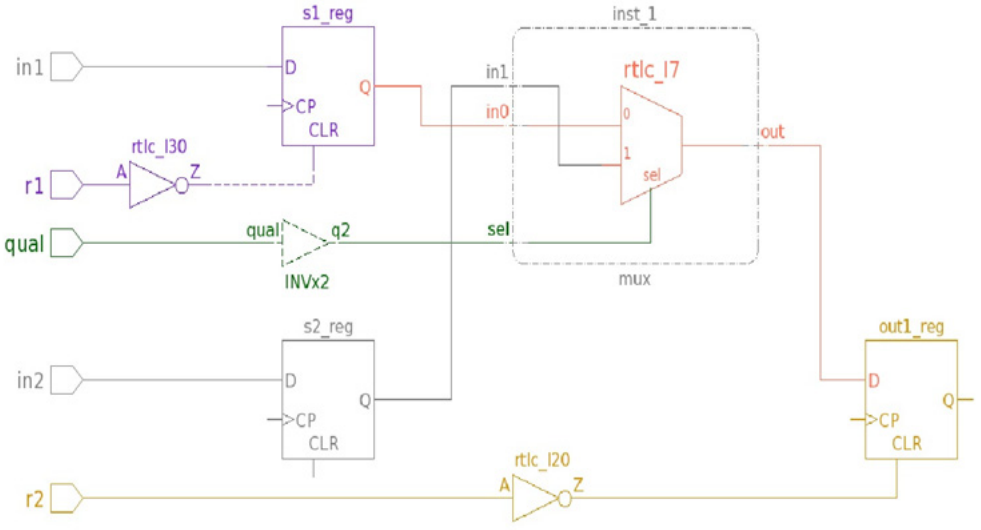


FIGURE 82. Invalid qualifier convergence

- Reason - Gating logic not accepted: qualifier invalid

  This reason is reported when attributes, such as source or destination clock/reset/object, specified with the qualifier constraint does not match with the source/destination data for a RDC crossing.

  If the *show_unsync_qualifier_rdc* parameter is set to `backward`, only the above reason is reported. If the parameter is set to `yes`, the invalid qualifier reason is replaced with one of the following specific reasons:

  `Gating logic not accepted: destination object does not`

```
match qualifier constraint";

"Gating logic not accepted: source object does not match
qualifier constraint";

"Gating logic not accepted: destination clock does not
match qualifier constraint";

"Gating logic not accepted: source clock does not match
qualifier constraint";

"Gating logic not accepted: destination reset does not
match qualifier constraint";

"Gating logic not accepted: source reset does not match
qualifier constraint";
```

- Reason - Gating logic not accepted: gate-type invalid

  This reason is reported when a source converges with a valid qualifier signal at an invalid combinational gate.

  For example, SpyGlass reports this reason in the following scenario as the qualifier merges with the XOR gate, which is considered invalid:

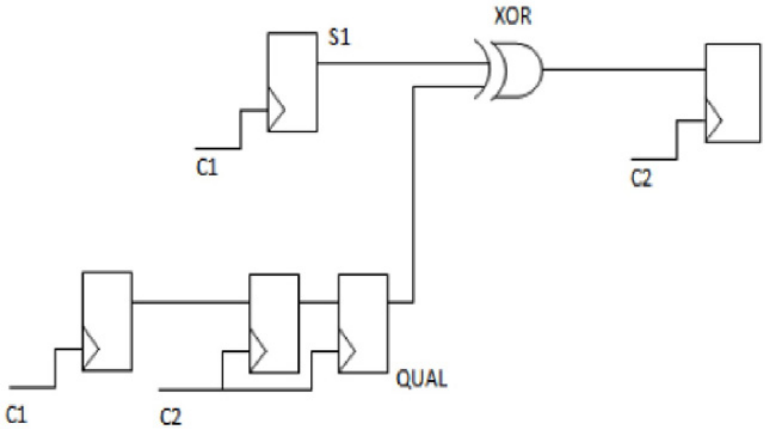


FIGURE 83. Invalid gate type

- Reason - Qualifier not propagated: clock reaching at qualifier does not match destination

  This reason is reported if clock(s) reaching at the qualifier does not match with the clock(s) of the destination object that is involved in reset domain crossing.

- Reason - Qualifier not propagated: reset reaching at qualifier does not match destination

  This reason is reported if reset(s) reaching at the qualifier does not match with the reset(s) of the destination object that is involved in reset domain crossing.

- Reason - Qualifier not found

  This reason is reported when SpyGlass cannot find any synchronizing structure at the destination of a crossing. This is reported in case of a missing qualifier or if SpyGlass cannot find the qualifier.

Features of the Ar_resetcross01 Rule

The *Ar_resetcross01* rule has the following features:

- This rule ignores combinational logic between sequential elements.
- This rule stops traversal on black boxes, sequential elements, blocked paths, and primary ports.
- This rule checks for sources flip-flops, latches, black boxes, ports, or other sequential library cells that have asynchronous set or reset pins.
- This rule checks for destinations having flip-flops or latches.
- This rule checks if there is a path between sequential elements through data or enable pins. For clock paths, this rule continues propagation beyond clock gate enables.
- Reset propagation on the sequential elements is used by the `Propagate_Resets` rule.
- This rule checks for both preset and clear pins of sequential element.
- If multiple resets reach to sequential elements and a reset is different for both source and destination, this rule reports violation with respect to the resets that are different.
- This rule reports an informational message if a crossing is synchronized by any of the following methods:

- *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)*

  To disable this rule to consider synchronization from this scheme, set the `expected_ckcells_file` parameter to `no`.

- *For Control path crossings: Synchronizing Cell Synchronization Scheme (for RDC)*

  To disable this rule to consider synchronization from this scheme, set the `expected_ckcells_file` parameter to `no`.

- *For Data path crossings: Synchronized Enable Synchronization Scheme (for RDC)*

  To disable this rule to consider synchronization from this scheme, set the `enable_sync` parameter to `no`.

- *For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)*

  To disable this rule to consider synchronization from this scheme, set the `enable_mux_sync` parameter, `enable_or_sync` parameter, or the `enable_and_sync` parameter, as applicable, to `no`.

- *For Data path crossings: Clock-Gating Cell Synchronization Scheme (for RDC)*

  To disable this rule to consider synchronization from this scheme, set the `enable_clock_gate_sync` parameter to `no`.

- This rule does not report any violation message if a crossing is synchronized by the following method:
  - Dynamically switching off the destination clock

    SpyGlass performs simulation by applying active values on source resets and checks if it is blocking the destination clock. If the destination clock is found switched off, metastability due to assertion of source reset will not be captured by destination.

    To disable this rule to perform the above simulation, set the *enable_sim_check_rdc* parameter to `no`.
- This rule checks only if the destination has either preset or clear pin.

Reducing Noise Due to the Ar_resetcross01 Violations

SpyGlass CDC provides the following approaches to reduce noise in the

Ar_resetcross01 rule. These approaches do not reduce the number of violations but reports unsynchronized crossings as synchronized. Use the following approaches to reduce noise pertaining to control path crossings and data path crossings:

- *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)*
- *For Control path crossings: Synchronizing Cell Synchronization Scheme (for RDC)*
- *For Data path crossings: Synchronized Enable Synchronization Scheme (for RDC)*
- *For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)*
- *For Data path crossings: Clock-Gating Cell Synchronization Scheme (for RDC)*
- *Specifying the Paths of Reset Crossings as False Paths* (No violations are reported when this approach is used)
- Setting the *report_sync_rdc* parameter to `none`. (No violations are reported when this approach is used)

For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)

This scheme marks those reset domain crossings as synchronized where flip-flops are in a synchronization flip-flops arrangement. You can set the number of flip-flops in the synchronization chain by using the `num_flops` constraint.

The following figure shows an example of this scheme:

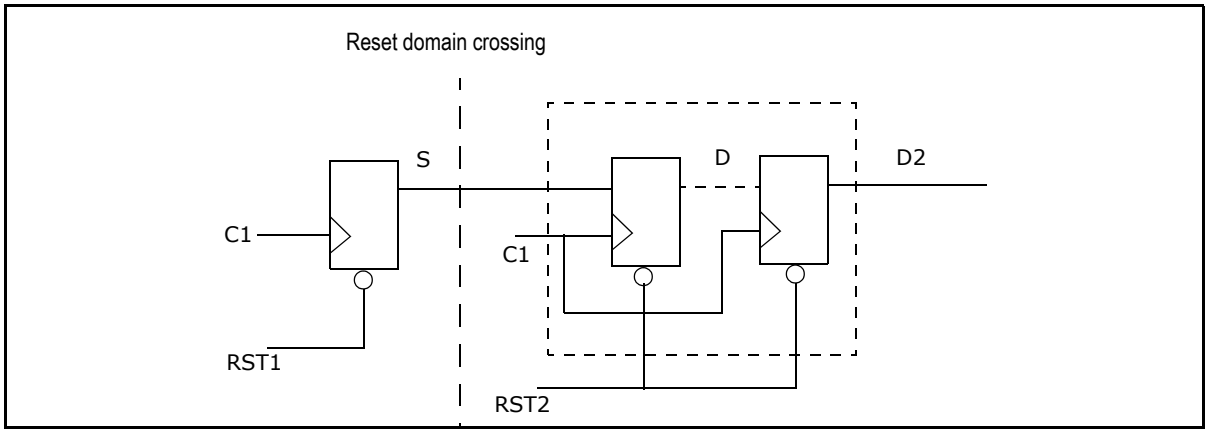


FIGURE 84. Adding multi-flop synchronization to reduce Ar_resetcross01 violations

In the above figure, the `D2` flip-flop is acting as the synchronizer for the reset crossing.

To disable this scheme, set the *enable_multiflop_sync* parameter to `no`.

By default, two flip-flops are considered as a valid synchronizer structure. To set a greater number of synchronizer flip-flops, use the `num_flops` constraint. Ensure to specify the `-rdc` argument with the `num_flops` constraint in this case.

For Control path crossings: Synchronizing Cell Synchronization Scheme (for RDC)

This scheme marks those reset domain crossings as synchronized in which the destination object is an instance of a synchronizing cell specified the `sync_cell` constraint.

The following figure shows the example of a crossing synchronized by this scheme:

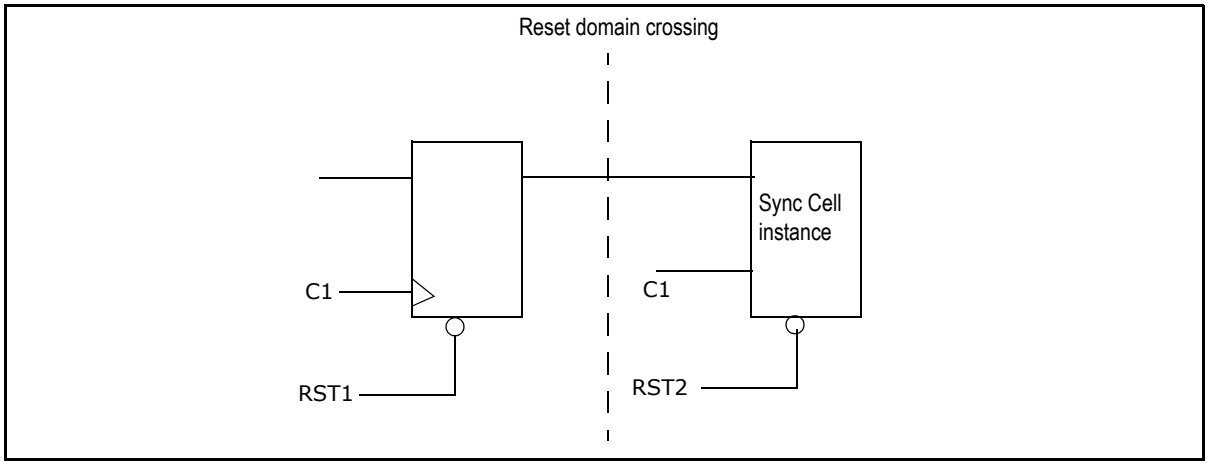


FIGURE 85. Synchronizing Cell Synchronization Scheme

To consider a reset crossing as synchronized, specify the destination instance as a synchronizing cell by using the `sync_cell` constraint with the `-rdc` argument.

In this scheme:

- The destination object should be any of the following:
  - A design unit instance (soft instance)

    That is, a design object can be a cell instance that contains a functional description.
  - A black box instance (hard instance)

    That is, a design object can be a cell instance that contains no functional description.

  This distinction is significant because unlike hard instances, soft instances specified by using the `sync_cell` constraint must be determined to describe a flip-flop for each destination bit. Otherwise, the control crossing is not considered as synchronized by this scheme.

For Data path crossings: Synchronized Enable Synchronization Scheme (for RDC)

This scheme marks those reset domain crossings as synchronized where

the first flip-flop in the destination reset domain is enabled by a signal synchronized to the destination reset and the reset crossing is in the data path.

The reset domain crossing is marked as synchronized if the following conditions are met:

- Enable pin is driven by a signal synchronized to the destination.
- Qualifier signal is merging at this enable pin.
- Qualifier signal should be either in the same destination reset domain or driven from a port/bbox that does not have a specified domain.
- Qualifier signal should either be in the same destination clock domain (i.e. driven in same clock domain as that of the destination element) or driven from a port/bbox that does not have a specified domain.
- The `enable_sync` parameter is set to yes.

By default, this scheme is always run. Set the `enable_sync` parameter to `no` to disable this scheme.

The following figure shows the example of a crossing synchronized by this scheme:

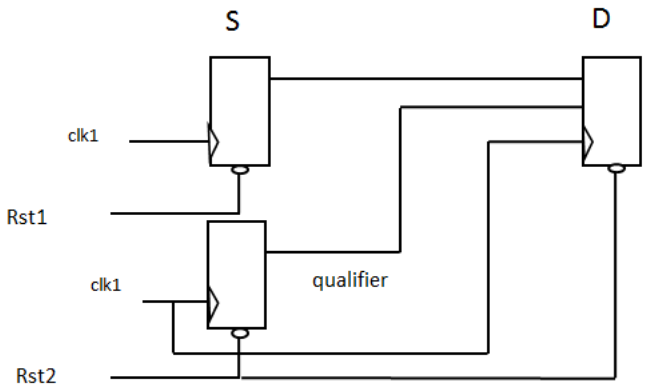


FIGURE 86. Synchronized Enable Synchronization Scheme

For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)

This includes following types of synchronization schemes:

- Recirculation-Mux based Synchronization scheme
- Mux-Select based Synchronization scheme
- Glitch Protect Cell based Synchronization scheme
- AND based Synchronization scheme
- OR based Synchronization scheme

The Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell) marks those reset domain crossings as synchronized where the destination flip flop in a RDC is driven by a Macro. A Macro can be any of glitch-protect-cell, and gate, or gate, mux gate or recirculation mux gate.

The reset domain crossing is marked as synchronized if the following conditions are met:

- Source to destination path must be blocked at Macro pin (which acts as its enable).
- This enable pin of the macro should be driven by a signal (qualifier) synchronized to the destination.
- Qualifier signal should be either in the same destination reset domain or driven from port/BBox which have no domain specified.
- Qualifier signal should be either in the same destination clock domain (i.e. driven in same clock domain as that of destination element), or driven from port/bbox which have no domain specified.

The Glitch Protect Cell based sync scheme is supported when macro/gating cell is provided by using the `glitch_protect_cell` parameter.

The Mux-based sync scheme is supported when the `enable_mux_sync` parameter is set to support mux and recirculation mux based synchronization (depending upon the value of this parameter).

The AND based sync scheme is supported when the `enable_and_sync` parameter is set to `yes`.

The OR based sync scheme is supported when the `enable_or_sync` parameter is set to yes.

The following figure shows the example of a crossing synchronized by this scheme:

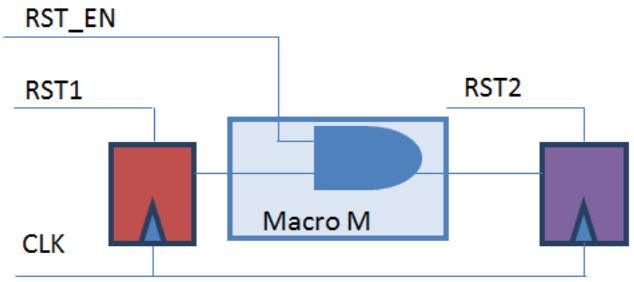


FIGURE 87. Reset Enable Logic based Synchronization Scheme for RDC

For Data path crossings: Clock-Gating Cell Synchronization Scheme (for RDC)

The Clock-Gating Cell Synchronization Scheme marks those reset domain crossings as synchronized where:

- The clock path of the destination flip-flop has a clock-gating cell.
- The enable pin of clock-gating-cell is driven by a signal synchronized to the destination.
- Qualifier signal should be merging at this enable pin.
- Qualifier signal should either be in the same destination reset domain or driven from a port/bbox that does not have a specified domain.
- Qualifier signal should be either in same destination clock domain (i.e. driven in same clock domain as that of destination element) or driven from a port/bbox that does not have a specified domain.
- This scheme is supported when the `enable_clock_gate_sync` parameter is set to `yes`.

The following figure shows the example of a crossing synchronized by this scheme:

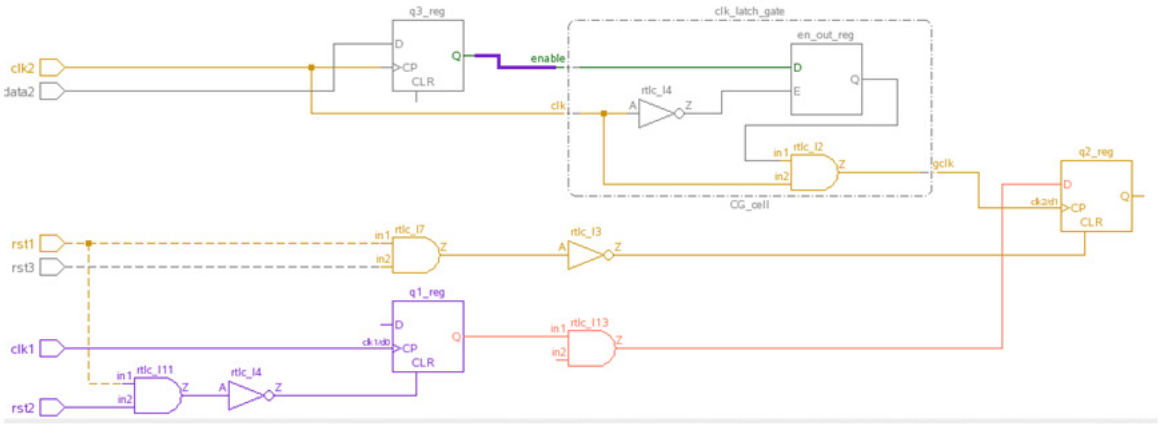


FIGURE 88. Clock-Gating Cell Synchronization Scheme

Specifying the Paths of Reset Crossings as False Paths

To suppress the *Ar_resetcross01* rule checking on certain paths, specify those paths by using the `reset_filter_path` constraint.

Rule Exceptions

This rule does not report a violation for the cases in which:

- A reset is always inactive because of the `set_case_analysis` constraint or power/ground connections.
- A destination instance does not have a preset or clear pin.
- A crossing is found synchronized by simulation checks.

Parameter(s)

- *dump_sync_info*: Default value is `no`. Set this parameter to `yes` to generate `The SynchInfo Report` and `The CrossingInfo Report` reports in the default format. Other possible values are `no`, `detailed`, and `detailed_mod`.
- *dump_inst_type*: Default value is `all`. Set this parameter to `flop` to generate destinations and synchronizers that are flip-flops in `The SynchInfo Report` and `The CrossingInfo Report`.

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- `report_all_reset_cross`: Default value is `no`. Set this parameter to `yes` to report violations on reset crossings in which the destination receives no clear/set signal.
- `report_for_single_busbit`: Default value is `no`. Set this parameter to `yes` to report violations for single bit of a bus.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see `Possible Values to the reset_reduce_pessimism Parameter`.
- *expected_ckcells_file*: Default value is `yes`. Set this parameter to `no` to ignore the *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)* and *For Control path crossings: Synchronizing Cell Synchronization Scheme (for RDC)* for synchronizing the destination.
- *enable_sim_check_rdc*: Default value is `yes`. Set this parameter to `no` to disable simulation to check if the destination clock is switched off (while the source reset is active) so that no metastability is captured at the destination.
- *report_sync_rdc*: Default value is `all`. Set this parameter to `one` to report only one source of the synchronized destination. Other possible value is `none`.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see `Allowed Values of the cdc_reduce_pessimism Parameter`.

- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *ignore_num_rtl_buf_invs*: Default value is `many`. Set this parameter to `one` to allow one buffer and inverter. Other possible values are `two` and `none`.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *For Control path crossings: Conventional Multi-Flop Synchronization Scheme (for RDC)*.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *enable_diff_clkdom_rdc*: Default value is `no`. Set this parameter to `yes` to report reset domain crossings having different clock domains.
- *glitch_protect_cell*: Default value is `NULL`. Specify a comma or space-separated list of glitch protection cell names for the *For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)*.
- *enable_and_sync*: Default value is `no`. Set this parameter to `yes` to enable the *For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)*.
- *enable_or_sync*: Default value is `no`. Set this parameter to `yes` to enable the *For Data path crossings: Reset Enable Logic based Synchronization Scheme for RDC (Mux, AND, OR, Glitch Protect Cell)*.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *For Data path crossings: Synchronized Enable Synchronization Scheme (for RDC)*.

- *enable_clock_gate_sync*: Default value is `yes`. Set this parameter to `no` to disable the *For Data path crossings: Clock-Gating Cell Synchronization Scheme (for RDC).*
- *show_unsync_qualifier_rdc*: Default value is `no`. Set this parameter to `yes` to report enable and potential qualifiers for unsynchronized RDC crossings.
- *rdc_reduce_pessimism*: Default value is `none`. Set this parameter to `reset_filter` so that the Ar_resetcross01 rule ignores reset domain crossings in some specific scenarios.
- *ignore_qualifier_mismatch_rdc*: Default value is `no`. Set this parameter to `yes` to report clock and reset mismatches between the qualifier and the destination object of a reset-domain crossing.
- *rdc_report_all_resets*: Default value is `no`. Set this parameter to `yes` to enable the Ar_resetcross01 rule to report all resets in the reset domain crossing in rule-based spreadsheet.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *rdc_allow_sync_reset*: Default value is `none`. Set the value of the parameter to `both` specify synchronous resets, in addition to asynchronous resets, in the `-to_rst` argument of the `reset_filter_path` constraint.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.
- *msg_inst_mod_report*: Default value is `auto`. Set the value of the parameter to one of the *supported values for the Ar_resetcross01 rule* to specify the source or the destination parent module to report in the csv.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *reset* (Mandatory): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in your design.

- *define_reset_order* (Optional): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *num_flops* (Optional): Use this constraint to specify the minimum number of flip-flops required in a synchronizer chain.
- *sync_cell* (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *reset_filter_path* (Optional): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.
- *abstract_port* (Optional): Use the `-reset` argument of this constraint to specify the reset name assigned to the port of an abstract view.
- *qualifier* (Optional): Use this constraint with the -rdc argument to specify a qualifier for synchronizing a reset domain crossing.

Messages and Suggested Fix

Message 1

The following message appears to report an invalid reset crossing:

[WARNING] Unsynchronized reset crossing detected from <cell-type> '<src-net>' (Reset: '<source-reset>') to <cell-type> '<dest-net>' (Reset: <dest-reset>) that are clocked by '<clock-name>'

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-pin>	Source instance pin
<src-reset>	Reset reaching source
<dest-pin>	Destination instance pin
<dest-reset>	Reset reaching destination
<clock-name>	Comma-separated list of clock names reaching the destination flip-flop

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 2

The following message appears to report an invalid reset crossing:

```
[WARNING] Unsynchronized reset crossing detected from <cell-type> '<src-net>' (Reset: '<source-reset>') to <cell-type> '<dest-net>' (Reset: <dest-reset>) that are clocked by '<clock-name>'. Reason: <unsync-reason>
```

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-pin>	Source instance pin
<src-reset>	Reset reaching source
<dest-pin>	Destination instance pin
<dest-reset>	Reset reaching destination
<clock-name>	Comma-separated list of clock names reaching the destination flip-flop
<unsync-reason>	Reason for unsynchronized crossing

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 3

The following informational message appears to indicate a synchronized reset crossing:

`[INFO] Synchronized reset crossing detected from <cell-type> '<src-net>' (Reset: '<source-reset>') to <cell-type> '<dest-net>'(Reset: <dest-reset>), that are clocked by '<clock-name>', by method <synchronization-method>`

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-net>	Source instance net
<source-reset>	Reset reaching source
<dest-net>	Destination instance net
<dest-reset>	Reset reaching destination
<clock-name>	Comma-separated list of clock names reaching the destination flip-flop
<synchronization-method>	Reason for synchronized crossing

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 4

The following message appears to report an invalid reset crossing:

```
[WARNING] Unsynchronized reset crossing detected from <cell-type> '<src-net>' (Reset: '<source-reset>', Clock: '<source-clock>') to <cell-type> '<dest-net>' (Reset: <dest-reset>, Clock: '<dest-clock>')
```

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-pin>	Source instance pin
<src-reset>	Reset reaching source
<dest-pin>	Destination instance pin
<dest-reset>	Reset reaching destination
<source-clock>	Comma-separated list of source clock names reaching the destination flip-flop
<dest-clock>	Comma-separated list of destination clock names reaching the destination flip-flop

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 5

The following message appears to report an invalid reset crossing:

```
[WARNING] Unsynchronized reset crossing detected from <cell-type> '<src-net>' (Reset: '<source-reset>', Clock: '<source-clock>') to <cell-type> '<dest-net>' (Reset: <dest-reset>,
```

Clock: '<dest-clock>'). Reason: <unsync-reason>

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-pin>	Source instance pin
<src-reset>	Reset reaching source
<dest-pin>	Destination instance pin
<dest-reset>	Reset reaching destination
<source-clock>	Comma-separated list of source clock names reaching the destination flip-flop
<dest-clock>	Comma-separated list of destination clock names reaching the destination flip-flop
<unsync-reason>	Reason for unsynchronized crossing

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 6

The following informational message appears to indicate a synchronized reset crossing:

[INFO] Synchronized reset crossing detected from <cell-type> '<src-net>' (Reset: '<source-reset>', Clock: '<source-clock>') to <cell-type> '<dest-net>'(Reset: <dest-reset>, Clock: '<dest-clock>'), by method <synchronization-method>

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-net>	Source instance net
<source-reset>	Reset reaching source
<dest-net>	Destination instance net
<dest-reset>	Reset reaching destination
<source-clock>	Comma-separated list of source clock names reaching the destination flip-flop
<dest-clock>	Comma-separated list of destination clock names reaching the destination flip-flop
<synchronization-method>	Reason for synchronized crossing

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

The following schematic shows the violation of the *Ar_resetcross01* rule:

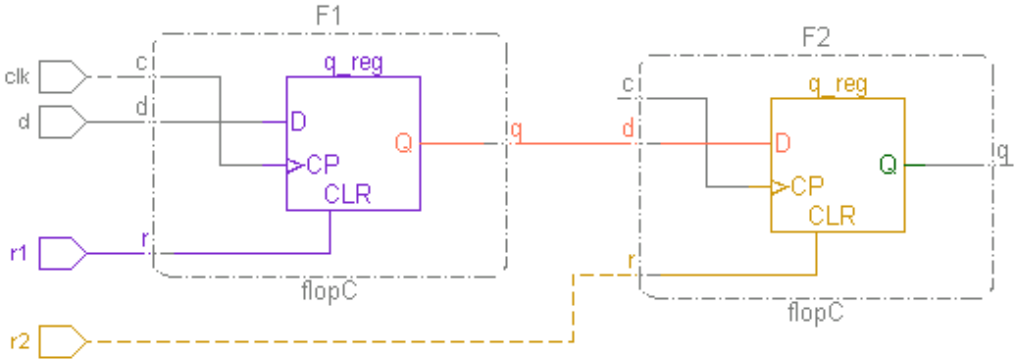


FIGURE 89.

In the above example, the *Ar_resetcross01* violation appears because of the invalid reset ordering between the `F1.q` flip-flop (reset `r1`) and the `F2.q` flip-flop (reset `r3`).

Default Severity Label

Warning

Rule Group

`ADV_CLOCKS`

Reports and Related Files

- `The CKSGDCInfo Report`
- Ar_resetcross01.csv

Ar_resetcross_matrix01

You can generate a spreadsheet for Reset Domain Crossing Matrix View by running the `Ar_resetcross_matrix01` rule.

When to Use

Use this rule to generate a spreadsheet containing the summary of synchronized and unsynchronized crossings between each pair of resets.

Prerequisites

Before running this rule:

- Use the Advanced_RDC license feature
- Run Ar_resetcross01 rule

Description

The *Ar_resetcross_matrix01* rule reports all the reset domain crossings, which are extracted by Ar_resetcross01 rule, in a design. For information on this spreadsheet, see *Reports and Related Files*.

View this spreadsheet to identify any unexpected reset-domain crossings (RDC), overall RDC risk in a design, reset domain crossings congestion area, setup issues causing unusual crossing distribution, and understand the architecture of these crossings.

Parameter(s)

- `report_for_single_busbit`: Default value is `no`. Set this parameter to `yes` to report violations for single bit of a bus.
- `report_all_reset_cross`: Default value is `no`. Set this parameter to `yes` to report violations on reset domain crossings in which the destination receives no clear/set signal.
- *enable_sync_check_rdc*: Default value is `yes`. Set this parameter to `no` to ignore the Conventional Multi-Flop Synchronization Scheme (for RDC) and Synchronizing Cell Synchronization Scheme (for RDC) for synchronizing the destination.
- *enable_sim_check_rdc*: Default value is `yes`. Set this parameter to `no` to disable simulation to check if the destination clock is switched off (while the source reset is active) so that no metastability is captured at the destination.

- *report_sync_rdc*: Default value is `all`. Set this parameter to `one` to report only one source of the synchronized destination. Other possible value is `none`.
- `handle_combo_arc`: Default value is `no`. Set this parameter to `yes` to enable the clock/reset to propagate from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- `sync_reset`: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- `ignore_num_rtl_buf_invs`: Default value is `many`. Set this parameter to `one` to allow one buffer and inverter. Other possible values are `two` and `none`.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the Conventional Multi-Flop Synchronization Scheme.
- `use_inferred_clocks`: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- `use_inferred_resets`: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *rdc_allow_sync_reset*: Default value is `none`. Set the value of the parameter to `both` specify synchronous resets, in addition to asynchronous resets, in the `-to_rst` argument of the `reset_filter_path` constraint.

Constraint(s)

- `reset_filter_path` (Optional): Use this constraint to specify false paths so that reset domain crossings along these paths are ignored from rule checking.
- `define_reset_order` (Optional): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- `num_flops` (Optional): Use this constraint to specify the minimum number of flip-flops required in a synchronizer chain.

- `sync_cell` (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- `reset` (Optional): Use this constraint to specify reset signals in a design.
- `clock` (Optional): Use this constraint to specify clock signals in a design.
- `abstract_port` (Optional): Use the `-reset` argument of this constraint to specify the reset name assigned to the port of an abstract view.

Messages and Suggested Fix

This rule reports the following informational message:

[INFO] '<num>' Reset Domain Crossings identified for '<reset-num>' resets for design '<top-name>'

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

To open the spreadsheet, double-click the message.

Analyze the reset domain crossings shown in the spreadsheet to:

- Check any unexpected reset-domain crossing.
- Check the overall RDC risk in a design.
- Understand the architecture of these crossings.

Example Code and/or Schematic

See *Reports and Related Files*.

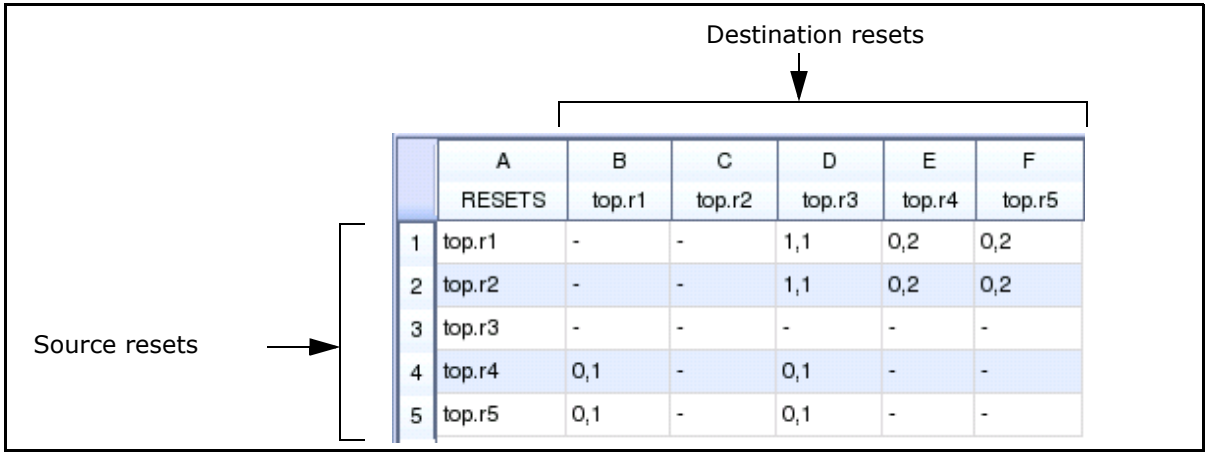
Default Severity Label

Info

Reports and Related Files

The *Ar_resetcross_matrix01* rule generates the *ResetCrossingMatrix.csv* spreadsheet. This spreadsheet is a matrix-like structure in which each column and row header lists all the resets.

The following figure shows a sample spreadsheet generated by this rule:



	A RESETS	B top.r1	C top.r2	D top.r3	E top.r4	F top.r5
1	top.r1	-	-	1,1	0,2	0,2
2	top.r2	-	-	1,1	0,2	0,2
3	top.r3	-	-	-	-	-
4	top.r4	0,1	-	0,1	-	-
5	top.r5	0,1	-	0,1	-	-

FIGURE 90.

In the above spreadsheet:

- A hyphen (`-`) indicates no reset-domain crossing.
- The value pair (<*unsync*>,<*sync*> format) indicates a reset domain crossing.

  For example, this spreadsheet indicates zero unsynchronized crossing and two synchronized crossings between the `r2` and `r4` resets. Similarly, there is one unsynchronized and one synchronized crossing between the `r2` and `r3` resets.

Setup_rdc01

Reports all the potential reset domain crossings between sequential elements having same reset domains

When to Use

Use this rule to identify reset domain crossings that have the same asynchronous resets at both the source and destination but with side inputs in between the resets.

Prerequisites

Specify the following information before running this rule:

- Specify reset signals by using the `reset` constraint.
- Enable this rule by specifying the `set_goal_option addrules {Ar_resetcross01, Setup_rdc01}` command in a project file.
- Use the `Advanced_RDC` license.

Description

Unlike the Ar_resetcross01 rule, the Setup_rdc01 rule reports cases where the resets are the same at both the source and destination reset domain crossings and side inputs exist between the resets.

The Setup_rdc01 rule allows you to define the reset_filter_path constraint to filter out desired reset crossings by using the source reset, destination reset, source object, destination object, or the clock.

The details of the Setup_rdc01 rule are covered under the following topics:

- *Reason for the Setup_rdc01 Rule Violation*
- *Features of the Setup_rdc01 Rule*
- *Reducing Noise Due to the Setup_rdc01 Violations*
- *Rule Exceptions*

Reason for the Setup_rdc01 Rule Violation

The Setup_rdc01 rule reports a violation if there are paths between two sequential elements which are clocked by the same clock domain and have the same asynchronous resets/sets but side inputs in between the resets exist as shown in the following figure:

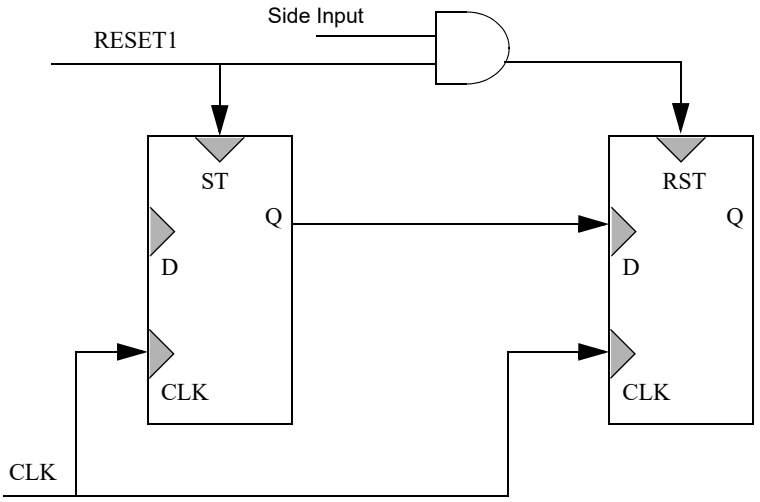


FIGURE 91. Scenario for the Setup_rdc01 Rule Violation

Features of the Setup_rdc01 Rule

The Setup_rdc01 rule has the following features:

- This rule ignores combinational logic between sequential elements.
- This rule stops traversal on black boxes, sequential elements, blocked paths, and primary ports.
- This rule checks for sources flip-flops, latches, black boxes, ports, or other sequential library cells that have asynchronous set or reset pins.
- This rule checks for destinations having flip-flops or latches.
- This rule checks if there is a path between sequential elements through data or enable pins. For clock paths, this rule continues propagation beyond clock gate enables.
- Reset propagation on the sequential elements is used by the `Propagate_Resets` rule.
- This rule checks for both preset and clear pins of sequential element.
- This rule checks only if the destination has either preset or clear pin.

Reducing Noise Due to the Setup_rdc01 Violations

Use the following approaches to reduce noise due to the Setup_rdc01 violations:

- *Specifying the Paths of Reset Crossings as False Paths*
- Setting the *report_sync_rdc* parameter to `none`.

Specifying the Paths of Reset Crossings as False Paths

To suppress the *Setup_rdc01* rule checking on certain paths, specify those paths by using the `reset_filter_path` constraint.

Rule Exceptions

This rule does not report a violation for the cases in which:

- A reset is always inactive because of the `set_case_analysis` constraint or power/ground connections.
- A destination instance does not have a preset or clear pin.
- A crossing is found synchronized by simulation checks.

Parameter(s)

- `dump_sync_info`: Default value is `no`. Set this parameter to `yes` to generate `The SynchInfo Report` and `The CrossingInfo Report` reports in the default format. Other possible values are `no`, `detailed`, and `detailed_mod`.
- `dump_inst_type`: Default value is `all`. Set this parameter to `flop` to generate destinations and synchronizers that are flip-flops in `The SynchInfo Report` and `The CrossingInfo Report`.
- `report_inst_for_netlist`: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- `filter_named_resets`: Default value is `clk`, `clock`, `scan`. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- `filter_named_clocks`: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.

- report_all_reset_cross: Default value is no. Set this parameter to yes to report violations on reset crossings in which the destination receives no clear/set signal.
- report_for_single_busbit: Default value is no. Set this parameter to yes to report violations for single bit of a bus.
- reset_reduce_pessimism: Default value is filter_unused_synchronizer, same_data_reset_flop. Set the value of this parameter to all_potential_resets to detect all the potential resets in a design. For information on the other possible values, see Possible Values to the reset_reduce_pessimism Parameter.
- *enable_sim_check_rdc*: Default value is yes. Set this parameter to no to disable simulation to check if the destination clock is switched off (while the source reset is active) so that no metastability is captured at the destination.
- *report_sync_rdc*: Default value is all. Set this parameter to one to report only one source of the synchronized destination. Other possible value is none.
- cdc_reduce_pessimism: Default value is mbit_macro, no_convergence_at_syncreset, no_convergence_at_enable. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see Allowed Values of the cdc_reduce_pessimism Parameter.
- handle_combo_arc: Default value is no. Set this parameter to yes so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- sync_reset: Default value is no. Set this parameter to yes to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- ignore_num_rtl_buf_invs: Default value is many. Set this parameter to one to allow one buffer and inverter. Other possible values are two and none.

- `use_inferred_clocks`: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- `use_inferred_resets`: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *enable_diff_clkdom_rdc*: Default value is `no`. Set this parameter to `yes` to report reset domain crossings having different clock domains.

Constraint(s)

- `reset` (Mandatory): Use this constraint to specify reset signals in a design.
- `clock` (Optional): Use this constraint to specify clock signals in your design.
- `define_reset_order` (Optional): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- `set_case_analysis` (Optional): Use this constraint to specify case analysis conditions.
- `reset_filter_path` (Optional): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

Message 1

The following message appears to report an invalid reset crossing:

```
[Warning] Potential invalid reset ordering detected from <cell-type> '<src-net>' (<source-reset>) to <cell-type> '<destnet>'
(Reset: <dest-reset>) having same reset '<reset>', that are clocked by '<clock-name>'
```

The arguments of the above message are explained below:

Argument	Description
<cell-type>	It can be flop, latch, or sequential cell.
<src-net>	Source instance net
<source-reset>	Reset reaching source

Argument	Description
<destnet>	Destination instance net
<dest-reset>	Reset reaching destination
<clock-name>	Comma-separated list of clock names reaching the destination flip-flop

Potential Issues

This violation appears if your design contains flip-flops that are connected in a data path, but are triggered by the same asynchronous resets.

Consequences of Not Fixing

If you do not fix this violation, it may result in setup/hold violations at the destination flip-flop because of the presence of side inputs between the source and the destination resets.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

- Open the incremental schematic of the violation.
- Analyze the `Reset Cone` of the source and destination flip-flops in the schematic.
- If the source and destination are coming at the same time and the side inputs are not causing issues in the reset assertions, filter the violation by using the `define_reset_order` or the `reset_filter_path` constraint.

Example Code and/or Schematic

The following schematic shows the violation of the *Setup_rdc01* rule:

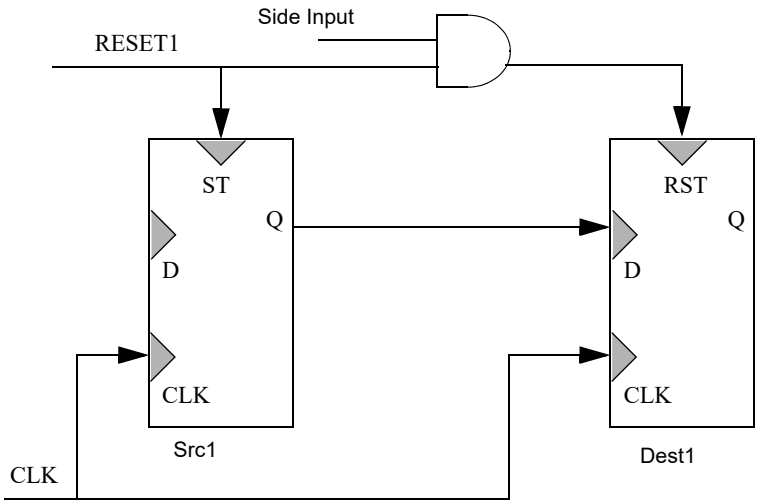


FIGURE 92.

In the above example, the *Setup_rdc01* violation appears because of the side input between the resets of the `Src1` source and the `Dest1` destination.

Default Severity Label

Warning

Rule Group

`ADV_CLOCKS`

Reports and Related Files

- `The CKSGDCInfo Report`

RFPSetup

Reports a violation if the reset_filter_path constraint is not used to filter reset domain crossings

When to Use

Use this rule for reset_filter_path setup check

Prerequisites

Specify the `reset_filter_path` constraint.

Description

The *RFPSetup* rule reports a violation if the `reset_filter_path` constraint is not used to filter any reset domain crossing in the current design.

Parameter(s)

None

Constraint(s)

`reset_filter_path` (Mandatory): Use this constraint to specify reset paths so that the reset domain crossings across these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

The following message appears if the reset_filter_path constraint is not used to filter any reset domain crossing.

```
[WARNING] 'reset_filter_path' constraint is not used to filter any crossing in the current design <current_design>
```

Potential Issues

This violation appears if a `reset_filter_path` constraint is mentioned in the .sgdc file but is not used to filter any reset domain crossing in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the specific constraint.

How to Debug and Fix

To fix this violation, specify a correct `reset_filter_path` constraint for existing reset domain crossings.

Example Code and/or Schematic

Consider the following constraints specification:

```
current_design "top"
clock -name "top.clk"
reset -name r1
reset -name r2
reset -name r3
reset_filter_path -from_rst w1 -to_rst r3
```

For the above example, the *RFPSetup* rule reports a violation because the `from_rst` reset is incorrect.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier23

qualifier's clocks or resets does not matches with the clocks and resets of the destination object

When to Use

Use this rule to perform sanity checks related to the `qualifier` constraint.

Prerequisites

Specify the `qualifier` constraint.

Description

The *SGDC_qualifier23* rule reports a violation if the reset/clock driving the enable signal (qualifier -rdc) is different from the reset/clock of the sequential element which it encounters during forward propagation.

The *SGDC_qualifier23* rule does not report a violation if an enable signal reaches at least one sequential element whose reset/clock matches with that of the enable signal.

However, the rule reports a violation if multiple resets drive the enable signal.

By default, qualifier is not propagated further if any mismatch is found between enable signal and the sequential element.

You can use the *ignore_qualifier_mismatch_rdc* parameter so that the SGDC_qualifier23 rule does not stop the qualifier propagation and reports a violation with Warning severity instead of an Error severity in this case. For example, consider the following schematic:

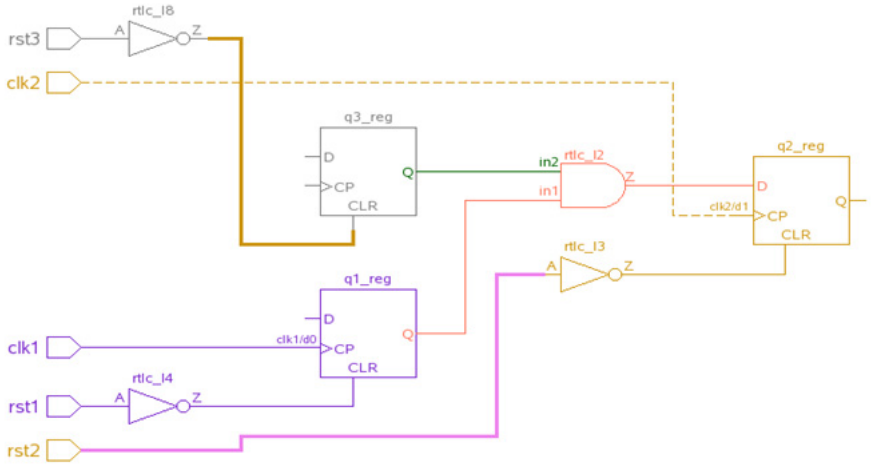


FIGURE 93.

Also consider the following constraints:

```
qualifier -name q3 -rdc //(qualifies path from q1_reg to
q2_reg)
reset_filter_path -from_rst rst3 -to_rst rst2 //(filters out
RDC from q3_reg to q2_reg)
```

In the above schematic, qualifier flop and destination resets are different. In this case, if *ignore_qualifier_mismatch_rdc* set to yes and reset_filter_path is used to filter non issues, which removes RDC between them, the qualifier is considered as valid. Therefore in this case, the reset domain crossing between q1_reg and q2_reg is reported as Synchronized by User defined And gate based qualifier if the `enable_and_sync` parameter is set to `yes`.

Parameter(s)

ignore_qualifier_mismatch_rdc: Default value is `no`. Set the value to `yes` to not stop qualifier propagation and report a Warning message instead of an Error message.

Constraint(s)

`qualifier` (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

Message 1

The following message appears when the clock reaching the qualifier signal does not match with the destination clock:

[WARNING] Constraint 'qualifier': Clock <clk_name> reaching at qualifier <qual_net_name> does not match with the clock <dest_clk_name> of destination object <dest_obj>, Qualifier is not propagated

Potential Issues

This violation appears if you have specified the `qualifier` constraint on the wrong net/terminal/port.

Consequences of Not Fixing

If you do not fix this violation, the qualifier might be considered as invalid. This might not be the intended behavior.

How to Debug and Fix

To fix this violation, ensure that the `qualifier` constraint is defined on the correct net/terminal/port.

Message 2

The following message appears when the reset reaching the qualifier signal does not match with the destination reset:

[WARNING] Constraint 'qualifier': Reset <reset_name> reaching at qualifier <qual_net_name> does not match with the reset <dest_rst_name> of destination object <dest_obj>

Potential Issues

This violation appears if you have specified the `qualifier` constraint on the wrong net/terminal/port.

Consequences of Not Fixing

If you do not fix this violation, the qualifier might be considered as invalid. This might not be the intended behavior.

How to Debug and Fix

To fix this violation, ensure that the `qualifier` constraint is defined on the correct net/terminal/port.

Message 3

The following message appears when the clock reaching the qualifier signal does not match with the destination clock:

[ERROR] Constraint 'qualifier': Clock <clk_name> reaching at qualifier <qual_net_name> does not match with the clock <dest_clk_name> of destination object <dest_obj>, Qualifier is not propagated

Potential Issues

This violation appears if you have specified the `qualifier` constraint on the wrong net/terminal/port.

Consequences of Not Fixing

If you do not fix this violation, the qualifier is considered as invalid. This might not be the intended behavior.

How to Debug and Fix

To fix this violation, ensure that the `qualifier` constraint is defined on the correct net/terminal/port.

Message 4

The following message appears when the reset reaching the qualifier signal does not match with the destination reset:

[ERROR] Constraint 'qualifier': Reset <reset_name> reaching at qualifier <qual_net_name> does not match with the reset <dest_rst_name> of destination object <dest_obj>; Qualifier is not propagated

Potential Issues

This violation appears if you have specified the `qualifier` constraint on the wrong net/terminal/port.

Consequences of Not Fixing

If you do not fix this violation, the qualifier is considered as invalid. This might not be the intended behavior.

How to Debug and Fix

To fix this violation, ensure that the `qualifier` constraint is defined on the correct net/terminal/port.

Message 5

The following message appears when multiple resets are found in the fanin cone of the qualifier signal:

[ERROR] Constraint 'qualifier : <reset_count> Resets (<reset_name1>, <reset_name2>, ...) in the fanin cone of the qualifier <qual_net_name>, Qualifier propagation is stopped

Potential Issues

This violation appears when multiple resets are found in the fanin cone of the qualifier signal.

Consequences of Not Fixing

If you do not fix this violation, the qualifier is considered as invalid. This might not be the intended behavior.

How to Debug and Fix

To fix this violation, ensure that the `qualifier` constraint is defined on the correct net/terminal/port.

Message 6

The following message appears when when multiple resets are found in the fanin cone of the qualifier signal:

[WARNING] Constraint 'qualifier : <reset_count> Resets (<reset_name1>, <reset_name2>, ...) in the fanin cone of the qualifier <qual_net_name>, Qualifier is still propagated

Potential Issues

This violation appears when multiple resets are found in the fanin cone of the qualifier signal.

Consequences of Not Fixing

If you do not fix this violation, the qualifier might be considered as invalid. This might not be the intended behavior.

How to Debug and Fix

To fix this violation, ensure that the `qualifier` constraint is defined on the correct net/terminal/port.

Example Code and/or Schematic

Consider the example shown in the following figure:

The following schematic shows the violation of the *SGDC_qualifier23* rule:

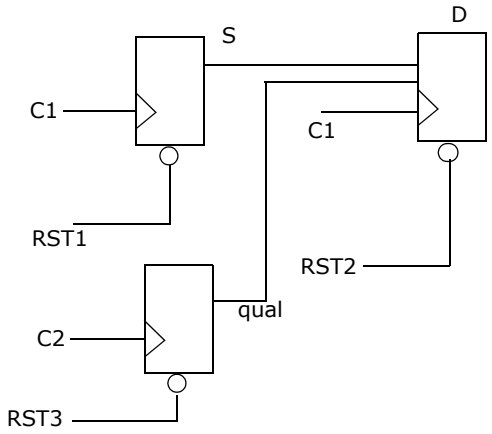


FIGURE 94.

In the above example, the *SGDC_qualifier23* violation appears because of the mismatch of the `C2` clock reaching the `qual` qualifier with that of the destination clock `C1`.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_define_transition01

Checks for compatible values in cdc_define_transition

When to Use

Use this rule to check for incompatible values in the *cdc_define_transition* constraint.

Description

This rule is run by default when the *cdc_define_transition* constraint is specified.

The SGDC_cdc_define_transition01 rule checks for incompatible values in the *cdc_define_transition* constraint. It reports the cases where both 00 and 11 are provided in the `-type` argument of the *cdc_define_transition* constraint.

Parameter(s)

None

Constraint(s)

- *cdc_define_transition* (Mandatory): Used to specify the permitted transitions of any net.

Messages and Suggested Fix

The following message appears:

`[ERROR] Constraint "cdc_define_transition" cannot accept both 00 and 11 for same net`

Potential Issues

This violation is reported because you have specified both 00 and 11 glitch for the same net in the `-type` argument of the *cdc_define_transition* constraint.

Consequences of Not Fixing

The net on which the constraint has been applied will not be checked for glitch.

How to Debug and Fix

Review the0 *cdc_define_transition* constraint specification.

Example Code and/or Schematic

The *SGDC_cdc_define_transition01* rule reports a violation for the following constraint specification because both 00 and 11 have been defined for the same net:

```
cdc_define_transition -name top.A -type {00, 11}
```

Default Severity Label

Error

Rule Group

None

Reports and Related Files

No report or related file

Ar_cross_analysis01

Reports clock domain crossings on the reset path in a design

When to Use

Use this rule to identify all asynchronous clock domain crossings on reset paths in a design.

Prerequisites

The prerequisites of using this rule are as follows:

- Enable this rule by specifying the `set_goal_option addrules {Ar_cross_analysis01}` command in a project file.
- Use the `Advanced_CDC` license.

Description

If a source from an asynchronous clock domain reaches the asynchronous preset or clear pin of a sequential element, it may cause metastability issues. The `Ar_cross_analysis01` rule performs crossing detection and synchronization checks and reports all the clock domain crossings on reset paths in a design.

This rule reports all the reset path crossings in the form of a spreadsheet in *The Ar_cross_analysis01 Spreadsheet*.

Parameter(s)

- All the parameters of *The Ac_sync_group Rules*

Constraint(s)

- All the constraints of *The Ac_sync_group Rules*

Messages and Suggested Fix

Message 1

The following message appears to report unsynchronized and synchronized crossings:

[ERROR] `Design <design-name> contains unsynchronized and synchronized crossings`

Potential Issues

This violation appears if there is a clock domain crossing in the reset path and the destination is not properly synchronized leading to metastability.

Consequences of Not Fixing

Same as the *Consequences of Not Fixing* of the *Ar_unsync01* rule.

How to Debug and Fix

To fix this violation, add a proper synchronization circuit for the signal in the reset path.

Message 2

The following message appears to report unsynchronized crossings:

`[ERROR] Design <design-name> contains unsynchronized crossing`

Potential Issues

This violation appears if there is a clock domain crossing in the reset path and the destination is not properly synchronized leading to metastability.

Consequences of Not Fixing

Same as the *Consequences of Not Fixing* of the *Ar_unsync01* rule.

How to Debug and Fix

To fix this violation, add a proper synchronization circuit for the signal in the reset path.

Message 3

The following info message appears to report synchronized crossings:

`[ERROR] Design <design-name> contains synchronized crossing`

Potential Issues

This is an informational message that reports asynchronous clock domain crossings in the reset path for which all the sources are synchronized.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the following code in the SGDC file and the schematic:

```
current_design top

clock -name clk1
clock -name clk2
clock -name clk3

input -name rst -clock clk1
```

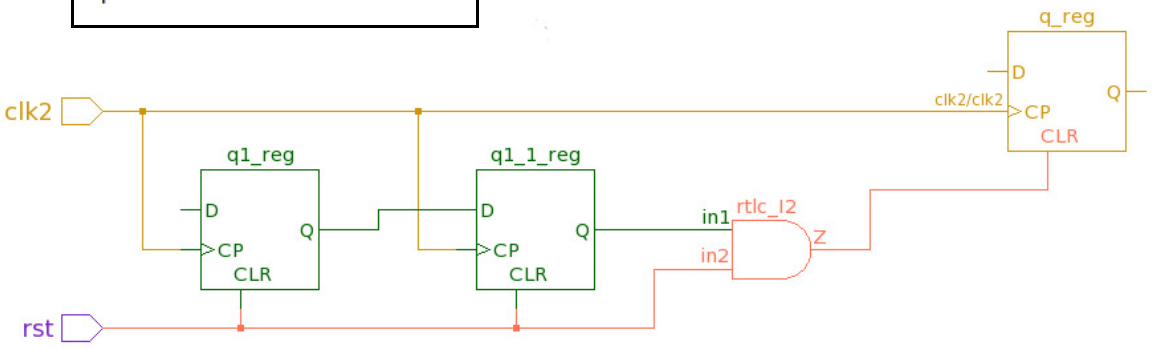


FIGURE 95. SGDC file and Schematic

For the above example, the Ar_cross_analysis01 rule reports synchronized crossings.

To view the details of these crossings, double-click on the violation of this rule to open *The Ar_cross_analysis01 Spreadsheet*.

Default Severity Label

Info

Rule Group

ADV_CLOCKS

Reports and Related Files

- *The Ar_cross_analysis01 Spreadsheet*
- *The SynchInfo Report*

Ar_asyncdeassert01

Reports if reset signal is asynchronously de-asserted

When To Use

Use this rule to perform metastability checks at a *Functional Flip-Flop*.

Prerequisites

The *Ar_asyncdeassert01* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The *Ar_asyncdeassert01* rule reports reset/preset pins of sequential elements that are asynchronously de-asserted because of the following reasons:

- Improper De-assertion

  The *Functional Flip-Flop* is not synchronously de-asserted for the inactive value of the reset signal, as shown in the following example:

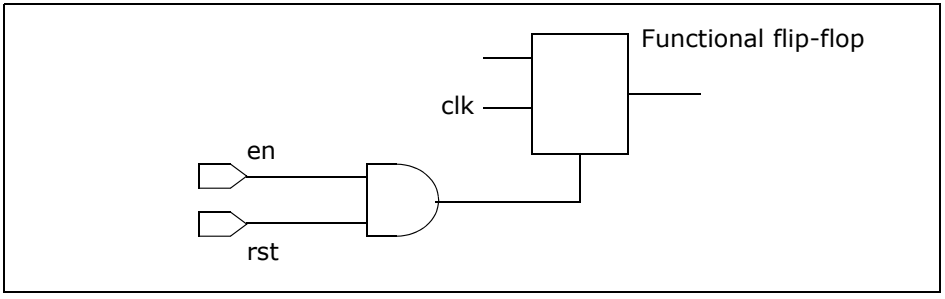


FIGURE 96. Improper De-assertion

- Domain-mismatch

  Domain mismatch occurs in any of the following cases:

  - The clock domain of a *Reset Flip-Flop* does not match the clock domain of a *Functional Flip-Flop*.

- If a reset is constrained by using the *input* constraint the clock specified by the `-clock` argument is not in the domain of the clock of the *Functional Flip-Flop*.

The following figure shows an example of domain mismatch:

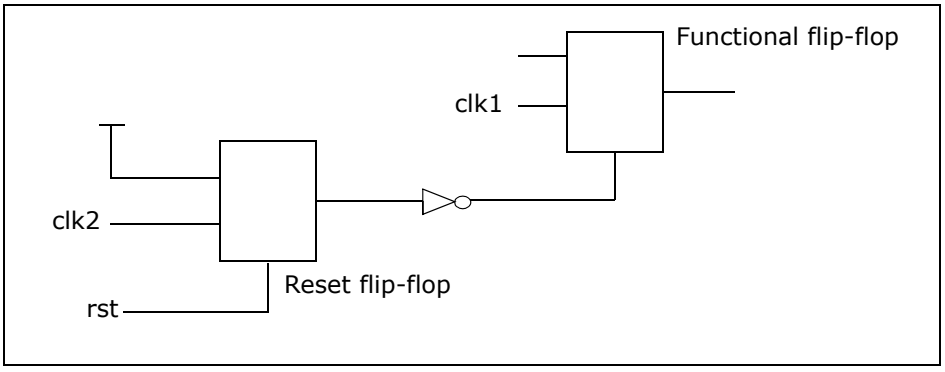


FIGURE 97. Domain Mismatch

- Unconstrained source clock

  The clock of a *Reset Flip-Flop* is unconstrained. The following figure shows an example of an unconstrained clock:

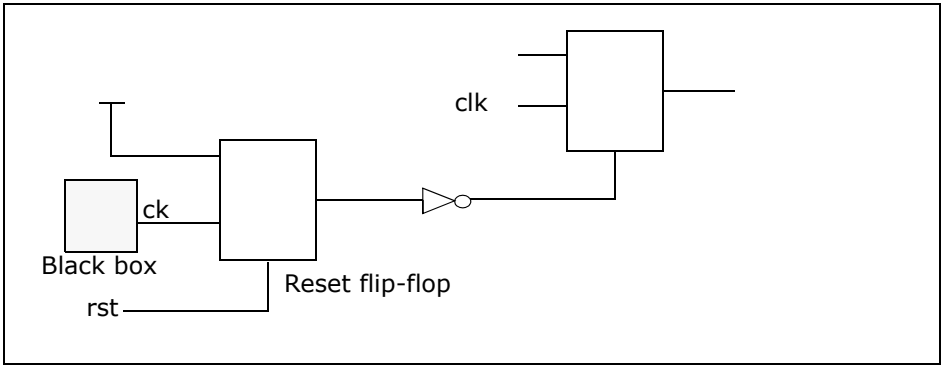


FIGURE 98. Unconstrained Clock

In the above example, the ck pin is not constrained by using the *clock*

constraint.

This rule reports one flip-flop per *Reset Cone*. You can expand the reset cone in the schematic to view other flip-flops or set the *enable_reset_cone_spreadsheet* parameter to `yes` to list all the flip-flops for the violation in a spreadsheet.

Parameter(s)

- *deassert_mode*: Default value is `none`. Set this parameter to a comma-separated list of *Possible Values of the deassert_mode Parameter* to filter violations of this rule.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.
- *reset_sync_depth*: Default value is 8. Set this parameter to a positive integer value to specify the number of flip-flops that are a part of the longest reset synchronizer chain in a design.
- *reset_synchronize_cells*: Default value is NULL. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the

potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *enable_reset_cone_spreadsheet*: Default value is `no`. Set this parameter to `yes` to enable SpyGlass CDC to generate a spreadsheet for each violation message reported by the *Ar_unsync01*, *Ar_asyncdeassert01*, and *Reset_sync02* rules. The generated spreadsheet includes all similar flops that are candidates for the reported violation.

Constraint(s)

- *assume_path*: (Optional) Use this constraint to specify paths that exist between the input pins and the output pins of black boxes.
- *set_case_analysis*: (Optional) Use this constraint to specify case analysis settings in a design.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock* (Optional): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset_filter_path*: (Optional) Use this constraint to specify reset paths so that the reset crossings across these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

Message 1

This rule reports the following message:

```
[RstAD1_1] [ERROR] Reset signal '<sig-name>' for '<pin-name>'
pin of <element-type> '<element-name>', is asynchronously de-
asserted relative to clock signal '<clock-signal>'. Reason:
```

`<reason>.`

The arguments of the above message are explained below:

Argument	Description
<sig-name>	User-specified or automatically-inferred reset signal
<pin-name>	Can be clear or set
<element-type>	Can be flop or latch
<element-name>	For netlist designs, if the *report_inst_for_netlist* parameter is set to yes, <element-name> is the output pin name of a functional flip-flop. Else, it is output net of *Functional Flip-Flop*.
<clock-signal >	Clock driving the *Functional Flip-Flop*
<reason>	Can be one of the following: • Improper de-assertion • Unconstrained clock source

Potential Issues

This violation appears if your design contains asynchronous reset signals that are asserted asynchronously but are not de-asserted synchronously with the corresponding clock signal.

Consequences of Not Fixing

Same as the *Consequences of Not Fixing* of the *Reset_sync01*.

How to Debug and Fix

To fix this violation, perform any of the following actions based on your requirement:

- Use the *reset_synchronize_cells* parameter to specify a black box, library cell, or a module that has a custom reset synchronizer.
- Use the `input` constraint to specify input constraint on reset source if it is synchronized outside the block.
- Use the *reset_synchronizer* constraint to specify a net/port <check from definition>

Message 2

This rule reports the following message:

```
[RstAD1_2] [ERROR] Reset signal '<sig-name>' for '<pin-name>' pin of <element-type> '<element-name>', is asynchronously de-asserted relative to clock signal '<clock-signal>'. Reason: Domain mismatch. Source clock: <source-clock>
```

The arguments of the above message are explained below:

Argument	Description
<sig-name>	User-specified or automatically-inferred reset signal
<pin-name>	Can be clear or set
<element-type>	Can be flop or latch
<element-name>	For netlist designs, if the *report_inst_for_netlist* parameter is set to yes, <element-name> is the output pin name of a functional flip-flop. Else, it is output net of *Functional Flip-Flop*.
<clock-signal >	Clock driving the *Functional Flip-Flop*
<source-clock>	Source clock name

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

Same as the *Consequences of Not Fixing* of the *Reset_sync01*.

How to Debug and Fix

See *How to Debug and Fix*.

Example Code and/or Schematic

Consider the following schematic:

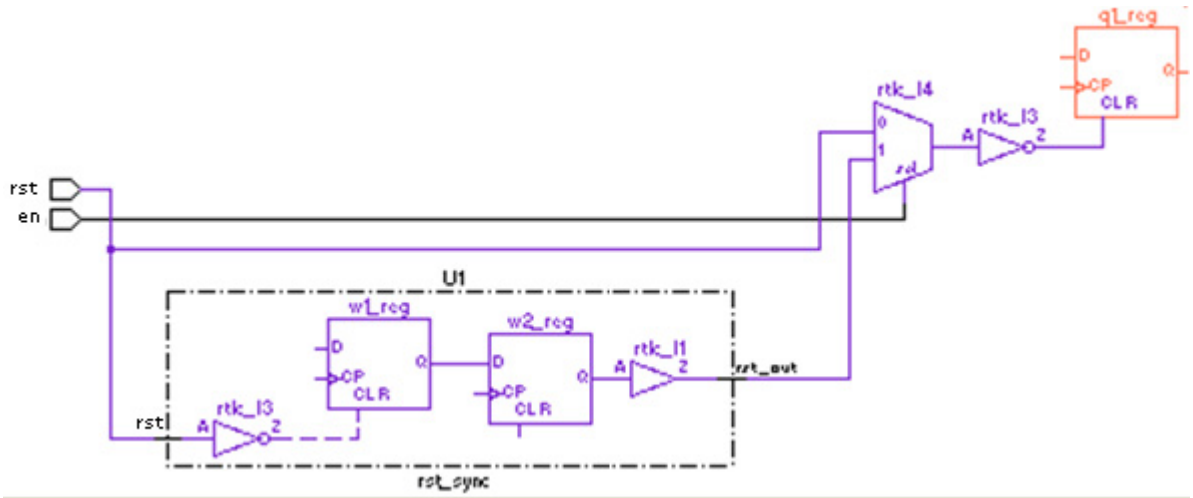


FIGURE 99. Schematic of the Ar_asyncdeassert01 Rule Violation

In this case, the reset may or may not be de-asserted synchronously depending upon the sequencing of the `en` enable and the `rst` reset.

De-assertion in this case is asynchronous if en is 0.

Default Severity Label

Error

Rule Group

Ar_sync_group

Reports and Related Files

The SynchInfo Report

Ar_syncdeassert01

Reports if reset signal is synchronously de-asserted or not de-asserted at all

When To Use

Use this rule to perform metastability checks at a *Functional Flip-Flop*.

Prerequisites

The *Ar_syncdeassert01* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The *Ar_syncdeassert01* rule reports reset/preset pins of sequential elements when they satisfy any of following conditions:

- If a reset is specified by using the *input* constraint and it is used by a flip-flop whose domain matches with the domain of a clock specified by the `-clock` argument of the `input` constraint. An example is shown in the following figure:

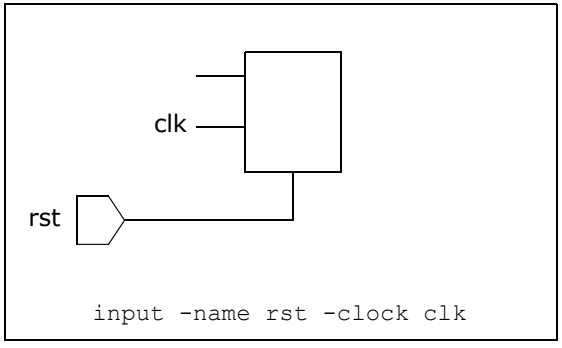


FIGURE 100. Scenario 1 - Ar_syncdeassert01 Rule Violation

- If at least one *Reset Flip-Flop* exists in the path of a reset and a *Functional Flip-Flop*. The functional flip-flop is de-asserted appropriately for the inactive value of the specified reset, as shown in the following figure:

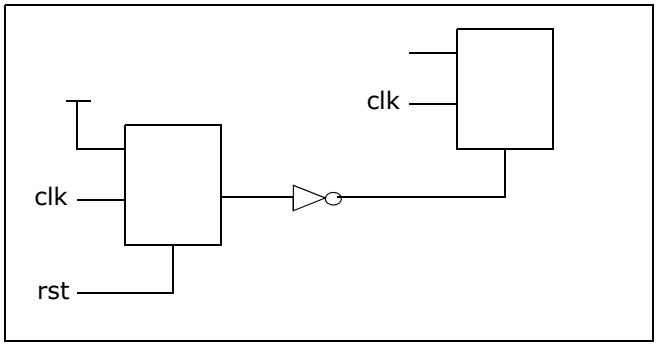


FIGURE 101. Scenario 2 - Ar_syncdeassert01 Rule Violation

- If its path is blocked such that the *Functional Flip-Flop* is never deasserted.

This rule reports one flip-flop per *Reset Cone*. You can expand the reset cone in the schematic to view other flip-flops.

Parameter(s)

- *deassert_mode*: Default value is `none`. Set this parameter to a comma-separated list of *Possible Values of the deassert_mode Parameter* to filter violations of this rule.
- *reset_num_flops*: Default value is 2. Specify a positive integer value, greater than one, to specify different number of flip-flops.
- *reset_synchronize_cells*: Default value is NULL. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.

- *reset_sync_depth*: Default value is 8. Set this parameter to a positive integer value to specify the number of flip-flops that are a part of the longest reset synchronizer chain in a design.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Constraint(s)

- *assume_path*: (Optional) Use this constraint to specify paths that exist between the input pins and the output pins of black boxes.
- *set_case_analysis*: (Optional) Use this constraint to specify case analysis settings in a design.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock* (Optional): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.
- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.

- *reset_filter_path*: (Optional) Use this constraint to specify reset paths so that the reset crossings across these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

This rule reports the following message:

```
[INFO] Reset signal <sig-name>' for '<pin-name>' pin of
<element-type> '<element-name>', synchronously de-asserts
relative to clock signal '<clock-signal>'
```

The arguments of the above message are explained below:

Argument	Description
<sig-name>	User-specified or automatically-inferred asynchronous reset signal
<pin-name>	Can be clear or set
<element-type>	Can be flop or latch
<element-name>	For netlist design, if the *report_inst_for_netlist* parameter is specified, it is the output pin name of functional flop. Otherwise, it is output net name of the functional flop.
<clock-signal >	Clock signal driving the *Reset Cone*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule. Disable or waive this rule if not required.

Example Code and/or Schematic

Consider the following schematic:

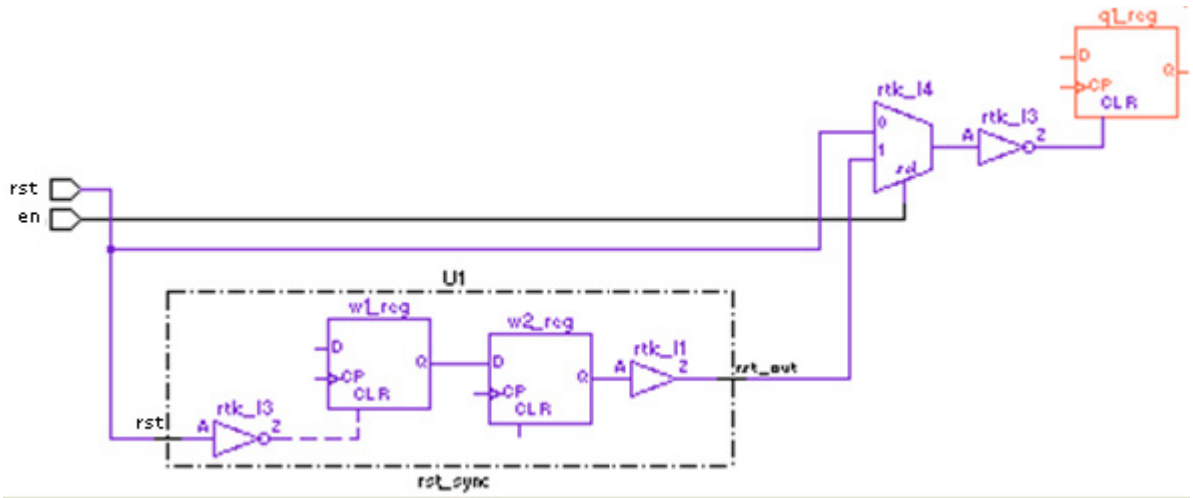


FIGURE 102. Schematic of the Ar_syncdeassert01 Rule Violation

In this case, the reset may or may not be de-asserted synchronously depending upon the sequencing of the `en` enable and the `rst` reset.

De-assertion in this case is synchronous if en is 1.

Default Severity Label

Info

Rule Group

Ar_sync_group

Reports and Related Files

The SynchInfo Report

Ar_sync01

Reports synchronized reset signals in the design

When To Use

Use this rule to check reset synchronization issues in your design.

Prerequisites

The *Ar_sync01* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The *Ar_sync01* rule reports reset pins of flip-flops that have at least one reset synchronizer in any of the paths between an asynchronous reset source and flip-flops.

This rule reports one flip-flop per *Reset Cone*. You can expand the reset cone in the schematic to view other flip-flops.

Synchronization Methods Reported by this Rule

This rule reports the following synchronization methods in its violation message:

- `Multi-flop reset synchronizer`

  This synchronization method is reported if a multi-flop synchronizer chain is found between a reset source and a flip-flop, as shown in the following figure:

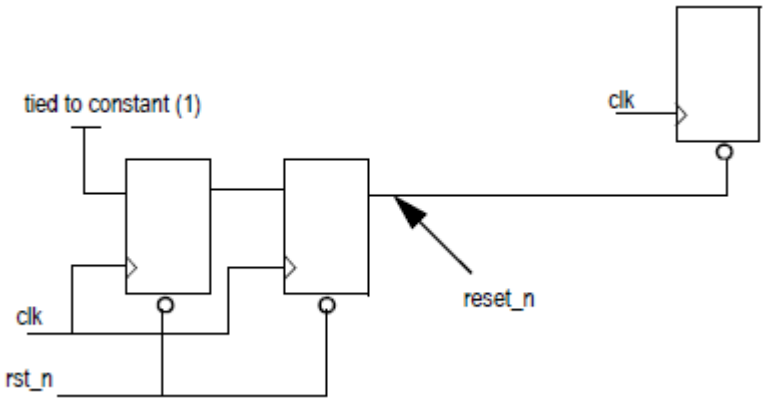


FIGURE 103. Multi-Flop Synchronizer Chain between Reset Source and Flip-Flop

- `Reset constrained by using the input constraint`

  This synchronization method is reported if an input constraint has been specified on a reset source. In this case, the clock domain specified by the input constraint matches with the clock domain of the flip-flop where it is used as a reset.

- `User-defined reset synchronizer`

  This synchronization method is reported if a reset synchronizer specified by the *reset_synchronize_cells* parameter or by the *reset_synchronizer* constraint is found between a reset source and a flip-flop.

Reporting of Hanging Nets

Consider the scenario shown in the following figure:

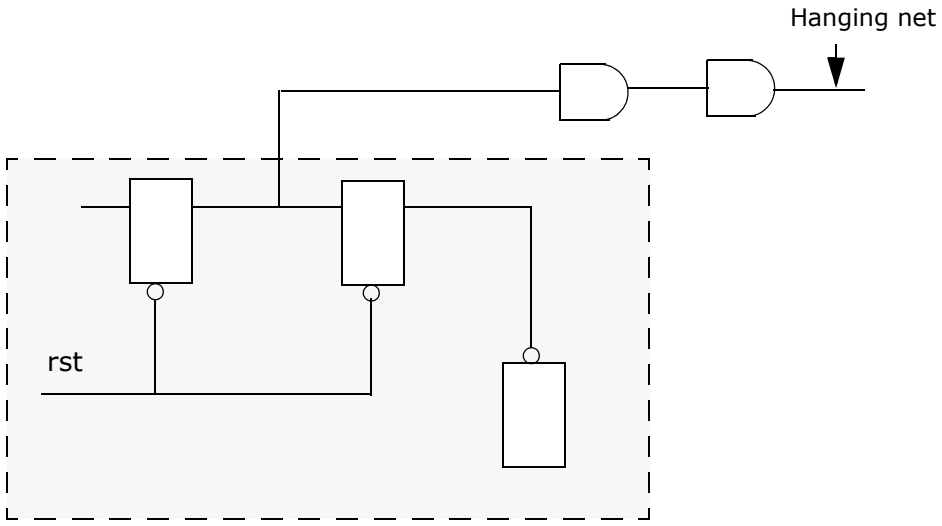


FIGURE 104. Hanging Net

In the above scenario, by default, the *Ar_sync01* rule does not ignore the hanging net coming from a combinational logic and does not report the grey area as valid synchronization.

When you set the *cdc_reduce_pessimism* parameter to `skip_unused_paths`, the *Ar_sync01* rule ignores the hanging net and considered the grey area as valid synchronization.

NOTE: *By default, this rule is switched off.*

Parameter(s)

- *reset_synchronize_cells*: Default value is NULL. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *reset_num_flops*: Default value is 2. Specify a positive integer value, greater than one, to specify different number of flip-flops.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.
- *enable_glitchfreecell_detection*: Default value is `no`. Set this parameter to `yes` to report glitch-free multiplexers in a design.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. Other possible values are `bbox`, `output_not_used`, `hanging_net`, `skip_unused_paths`, and all.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock* (Optional): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.

- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *num_flops -reset* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *reset_filter_path*: (Optional) Use this constraint to specify reset paths so that the reset crossings across these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

This rule reports the following message for one flip-flop on a per net basis:

```
[INFO] Reset signal '<sig-name>' for '<pin-name>' pin of
<object-type> '<object-name>', clocked by '<clock-name>', is
synchronized by method: <method>
```

The arguments of the above message are explained below:

Argument	Description
<sig-name>	Reset signal name
<pin-name>	Can be clear or set
<object-type>	Object type, such as flop, latch, or library-cell
<object-name>	For a netlist design, it refers to the output pin name of reporting flip- flop if the *report_inst_for_netlist* is specified. Else, it refers to the output net name.
<clock-name>	Clock signal of the reporting flip-flop
<method>	Any of the following synchronization method: • Multi-flop reset synchronize • User-defined reset synchronizer • Reset constrained through input constraint • Reset constrained through abstract_port constraint

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule. Disable or waive this rule if you do not want to view its messages.

Example Code and/or Schematic

Consider the following schematic of this rule:

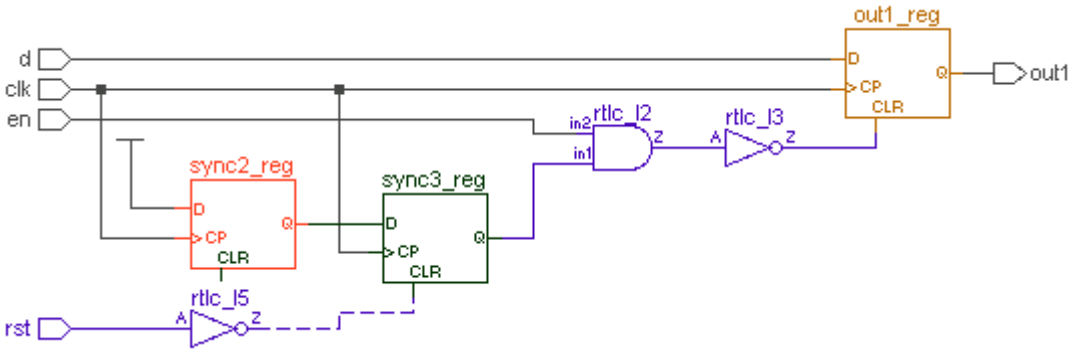


FIGURE 105. Schematic of the Ar_sync01 Rule Violation

In the above example, a multi-flop reset synchronizer is present between the reset source and flip-flop.

Schematic Highlight

Following details are highlighted in different colors:

- Reset path
- Destination flop
- Reset synchronizer used for synchronizing a flop

Default Severity Label

Info

Rule Group

Ar_sync_group

Reports and Related Files

The SynchInfo Report

Ar_unsync01

Reports unsynchronized reset signals in the design

When To Use

Use this rule to check reset synchronization issues in your design.

Prerequisites

The *Ar_unsync01* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The *Ar_unsync01* rule reports reset pins of flip-flops that do not have a reset synchronizer in any of the paths between an asynchronous reset source and flip-flops.

This rule reports following reasons in the violation message:

- Missing synchronizer

  This reason is reported if a reset synchronizer is not present in any of the paths.

- Different domain synchronizer

  This reason is reported if a reset synchronizer of different domain is present, as shown in the following figure:

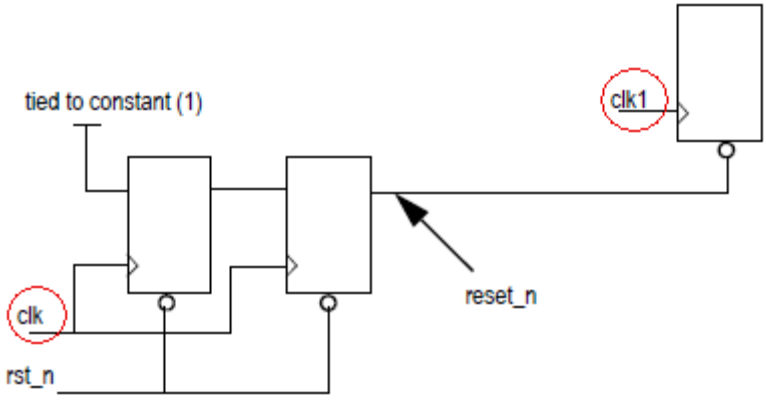


FIGURE 106. Different Domain Synchronizer

- Invalid reset synchronizer

  This reason is reported if an invalid reset synchronizer is present.

  A reset synchronizer is considered invalid in the following cases:

  - When multiple fan-out of flip-flops in a synchronizer chain are present, as shown in the following figure:

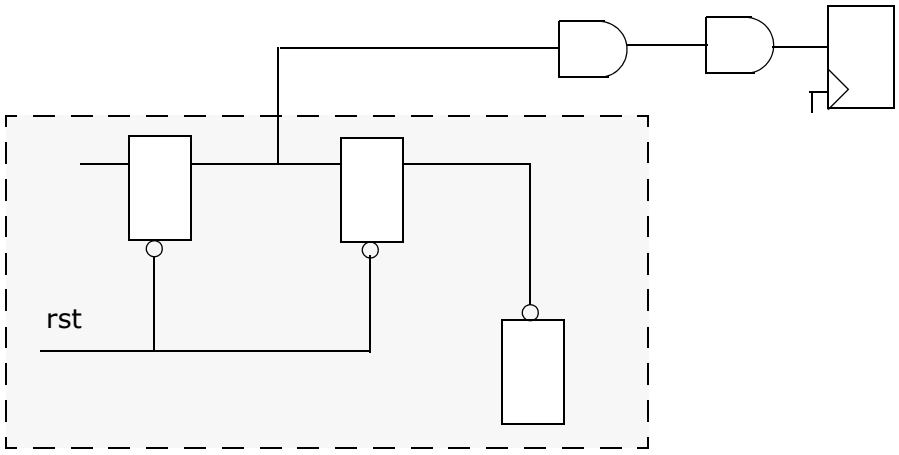


FIGURE 107. Multiple fan-outs of flip-flops in a synchronizer chain

  - Number of flip-flops in a chain are less than the value specified by the *reset_num_flops* parameter.

    For example, this rule reports a violation when the *reset_num_flops* parameter is set to 3 in the following scenario:

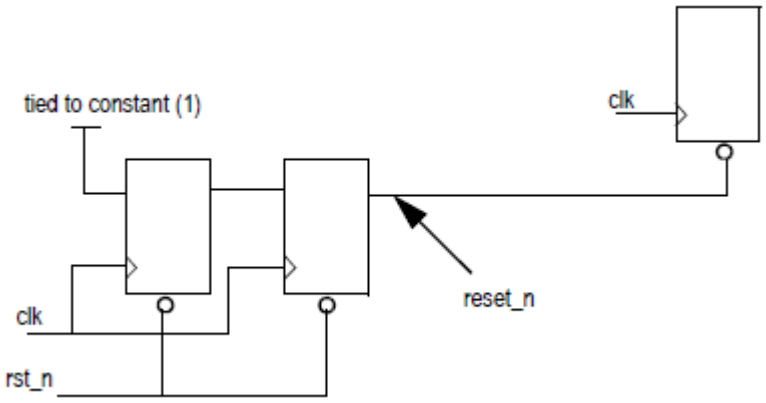


FIGURE 108. Number of Flip-Flops in a chain Less than the Value Specified

This rule reports one flip-flop per *Reset Cone*. You can expand the reset cone in the schematic to view other flip-flops or set the *enable_reset_cone_spreadsheet* parameter to `yes` to list all the flip-flops for the violation in a spreadsheet.

NOTE: *By default, this rule is switched off.*

Parameter(s)

- *reset_synchronize_cells*: Default value is NULL. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *reset_num_flops*: Default value is 2. Specify a positive integer value, greater than one, to specify different number of flip-flops.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.

- *enable_glitchfreecell_detection*: Default value is `no`. Set this parameter to `yes` to report glitch-free multiplexers in a design.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *enable_reset_cone_spreadsheet*: Default value is `no`. Set this parameter to `yes` to enable SpyGlass CDC to generate a spreadsheet for each violation message reported by the *Ar_unsync01*, *Ar_asyncdeassert01*, and *Reset_sync02* rules. The generated spreadsheet includes all similar flops that are candidates for the reported violation.

Constraint(s)

- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock* (Optional): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.
- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *num_flops -reset* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

- *set_case_analysis*: (Optional) Use this constraint to specify case analysis settings in a design.
- *reset_filter_path*: (Optional) Use this constraint to specify reset paths so that the reset crossings across these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

This rule reports the following message for one flip-flop on a per net basis:

```
[ERROR] Reset signal '<sig-name>' for '<pin-name>' pin of
<object-type> '<object-name>', clocked by '<clock-name>', is
unsynchronized by reason: <reason>
```

The arguments of the above message are explained below:

Argument	Description
<sig-name>	Reset signal name
<pin-name>	Pin name
<object-type>	Object type, such as flop, latch, or library-cell
<object-name>	Name of a flip-flop, latch, or a library cell (This rule reports a violation for one flip-flop on a per net basis.)
<clock-name>	Clock name
<reason>	Any of the following synchronization reason: • Missing synchronizer • Invalid reset synchronizer • Different domain synchronizer

Potential Issues

This violation appears if a reset synchronizer is missing in your design or the design contains a reset synchronizer that is not well built.

Consequences of Not Fixing

See the *Consequences of Not Fixing* of the *Reset_sync01*.

How to Debug and Fix

To fix this violation, add a proper synchronization circuit for the reset signal.

Example Code and/or Schematic

Consider the following schematic of a violation that is reported because of an invalid reset synchronizer:

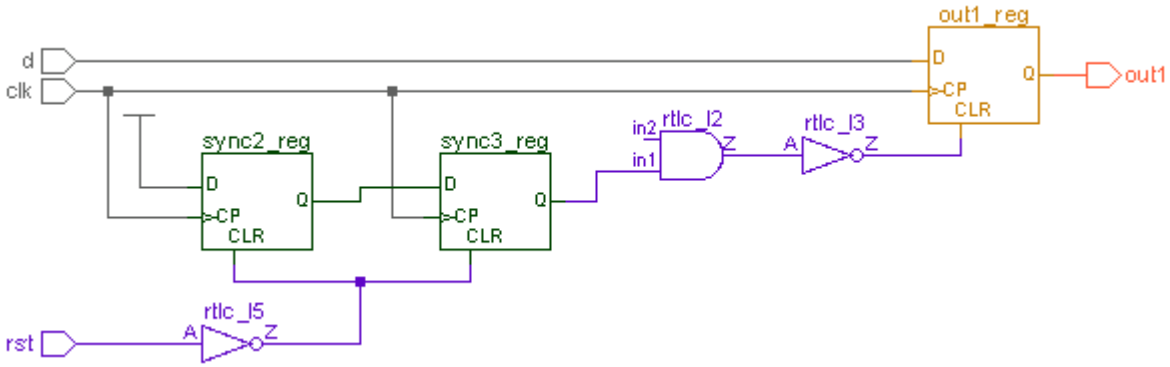


FIGURE 109. Schematic of the Ar_unsync01 Rule Violation

The above violation appears when the *reset_num_flops* parameter is set to `3`.

To fix this violation, increase the synchronizer flip-flop chain to 3 flip-flops.

Schematic Highlight

Following details are highlighted in the schematic:

- Reset path
- Destination flop
- Reset synchronizer used for synchronizing a flop

Default Severity Label

Error

Rule Group

Ar_sync_group

Reports and Related Files

- *The SynchInfo Report*
- Ar_unsync01.csv: This is the rule-based spreadsheet that contains details of all violations of this rule.

  You can double-click a row in the spreadsheet to see a list of all the flops that are in the same reset cone as the flop listed in the row. Note that this list is available only if the *enable_reset_cone_spreadsheet* parameter set to `yes`.

Reset_sync01

Reports asynchronous reset signals that are not de-asserted synchronously

NOTE: *The Reset_sync01 rule will be deprecated in a future release. The rule is not included in CDC GuideWare goals now and do not perform checks until specifically included in the user-defined goal options. In this case, the rule performs the checks and SpyGlass includes a deprecation message in both the spyglass.out and spyglass.log files.*

When to Use

Use this rule to detect asynchronous reset signals that are not de-asserted synchronously.

NOTE: *It is recommended to use the Ar_unsync01 and Ar_asyncdeassert01 rules instead of this rule.*

Prerequisites

Specify reset signals in any of the following ways:

- By using the *reset* constraint
- By setting the *use_inferred_resets* parameter to `yes` to use auto-generated reset signals
- By using a combination of both the above methods

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Reset_sync01* rule reports asynchronous reset signals that are asserted asynchronously but are not de-asserted synchronously with the corresponding clock signal.

This rule considers an asynchronous reset as synchronized if:

- It is specified by using the *input* constraint and

- It is used by a flip-flop where the domain of the flip-flop matches with the domain of a clock specified in the `-clock` argument of the `input` constraint.

This rule reports one flip-flop per *Reset Cone*. You can expand the reset cone in the schematic to view other flip-flops.

Rule Behavior While Traversing from an Asynchronous Reset Source

While traversing forward from an asynchronous reset source to find synchronizer modules:

- If all paths from an asynchronous reset signal do not have synchronizers, the reset signal is not considered as synchronized and the *Reset_sync01* rule reports violations for such reset signal used in the design.
- This rule skips buffers, inverters, and sensitized paths before a synchronizer while traversing forward from a reset source.

Rule Exceptions

Following are the exceptions to the *Reset_sync01* rule:

- It does not check asynchronous reset signals that are not asserted asynchronously.
- When all flip-flops are asynchronously de-asserted by a reset, this rule reports all of them except the synchronizer chain flip-flops with respect to any one clock of a design.

  In such cases, use the *Ar_sync_group* rules instead of the *Reset_sync01* rule.

Checks Performed by the Reset_sync01 Rule

This rule performs the following checks:

- It checks for cases in which clock domain of flip-flops, where asynchronous reset is being used, is same as the clock domain of an input port of a reset (if specified), or clock domain of a flip-flop/latch/black box, which is generating the reset. If the domain is not same, rule-checking will be done by the *Reset_sync02* rule rather than the *Reset_sync01* rule.

- It checks whether an asynchronous reset signal is properly synchronized with respect to the same clock domain (as shown in the following figure) to ensure its synchronous de-assertion:

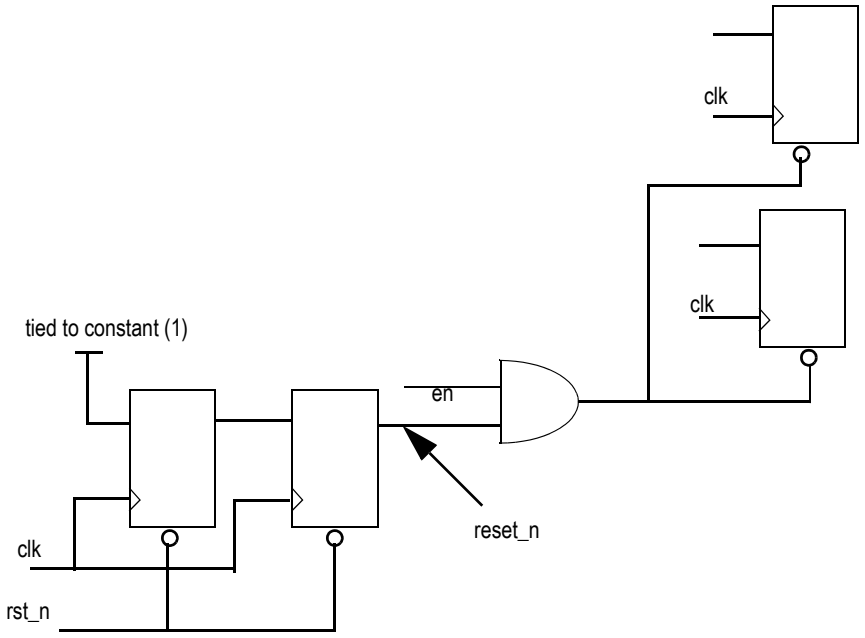


FIGURE 110. Check done by the Reset_sync01 Rule

In the above example, the `rst_n` reset signal is not used directly. It is synchronized with the `clk` clock to a synchronized reset signal `reset_n`, which is then used for all flip-flops triggered by the `clk` clock.

In the above example, if the *set_case_analysis* constraint is used to set the value of `en` to `1`, this rule does not report any violation. However, this rule reports a violation if the constraint is not specified for the enable net, `en`.

While checking for de-assertion, this rule uses inactive value of synchronous resets present in the cone of the asynchronous reset.

Parameter(s)

- *reset_synchronize_cells*: Default value is `NULL`. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *reset_sync_check*: Default value is `strict`. Set this parameter to `soft` to start de-assertion check on reset/clear pin of flip-flops instead of reset source so that you are not required to provide the *set_case_analysis* settings on other signals present in the cone of reset/clear pins of flip-flops.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis signals.

- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock* (Optional): Use this constraint to specify clock signals.
- *num_flops -reset* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears for reset signal *<rst-name>* of flip-flop *<flop-name>* that is not de-asserted synchronously with the corresponding clock signal *<clk-name>*:

```
[WARNING] Reset signal '<rst-name>' for flop '<flop-name>' is
not synchronously de-asserted relative to clock signal
'<clk-name>'
```

NOTE: *The above message is reported for one flip-flop per cone where the issue is present.*

The arguments of the above message are explained below:

Argument	Description
<rst-name>	Hierarchical name of the asynchronous reset signal
<flop-name>	For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<flop-name>* refers to the name of the flip-flop instance. Otherwise, *<flop-name>* is the name of the output net of the flip- flip-flop that is not de-asserted synchronously.
<clk-name>	Name of the destination clock

Potential Issues

This violation appears if your design contains asynchronous reset signals that are asserted asynchronously but are not de-asserted synchronously with the corresponding clock signal.

Consequences of Not Fixing

If an asynchronous reset signal is de-asserted asynchronously, the following problems may arise:

- Violation of reset recovery time

Reset recovery time refers to the minimum time required between a reset signal being de-asserted and a clock signal going high again. Missing a recovery time can cause signal integrity or metastability problems with the registered data outputs.

- Reset removal may occur in different clock cycles for different sequential elements

  Slight difference in propagation delays in either or both the reset signal and the clock signal can cause registers or flip-flops to exit the reset state before other sequential element.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message.

Perform the following steps to find the root cause of the problem:

1. Check if the combinational logic is present between the last synchronizer flip-flop and the flip-flop where it is used as a reset.

   This is not allowed if the combinational logic is not acting as a buffer/inverter and not allowing reset value to propagate.

   Enable *Show Case Analysis* in the schematic to check the constant value propagation. You may need to constraint the path by using *set_case_analysis* constraints.

2. Check if combinational logic is present between reset source and the synchronizer flip-flop.

   This is not allowed if the combinational logic is not acting as a buffer/inverter and not allowing reset value to propagate. Enable *Show Case Analysis* in the schematic to check the constant value propagation. You may need to constraint the path by using *set_case_analysis* constraints.

3. If you specify reset synchronizer cell, verify whether all the fan-out paths of reset net reaches reset synchronizer cell after ignoring inverters and buffers.

   Set the *reset_sync_check* parameter to `soft` for the rule to perform de-assertion check on reset/clear pin of flip-flops instead of the reset source.

   You may specify the *input* constraint on a reset port for the rule to assume these ports to be synchronized for the specified clock domain.

4. You can also view case analysis settings along with the violation of this rule.

There are several ways an asynchronous reset can be synchronized into the domain of a clock against which it will reset. A simple approach is to pass the reset through a metastability (for example, double flip-flop) structure synchronized to the clock. Another approach is to insert an additional flip-flop clocked on a negative edge of the clock, adding a half-cycle offset between the clock and the reset, further reducing chances of setup time problems between clock and reset.

If you want to assert reset asynchronously and de-assert synchronously, consider using a parallel OR and flip-flop structure, as shown in the following example:

```
...
assign final_reset = raw_reset || sync_reset;
always @(posedge clk)
  sync_reset <= raw_reset;
```

Example Code and/or Schematic

Consider the following schematic of a violation reported by the *Reset_sync01* rule:

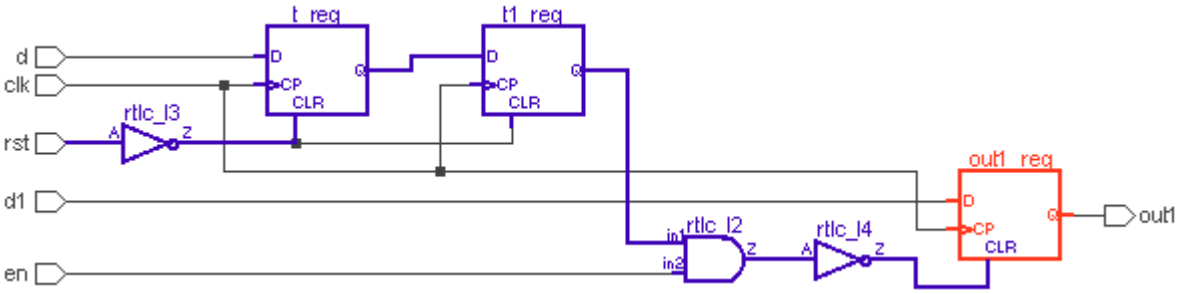


FIGURE 111. Schematic of the Reset_sync01 Rule Violation

In the above example, when the `rst` reset signal is de-asserted, the synchronized output does not de-assert the `out1` flip-flop due to unconstrained `en` input of the `rtlc_I2` AND gate.

To fix this violation, constraint the `en` input of the `rtlc_I2` AND gate to

value `1`. This will lead to propagation of synchronous output to the `out1` flip-flop.

Schematic Details

The *Reset_sync01* rule highlights the path from a reset signal to a reset pin of any one flip-flop.

Default Severity Label

Warning

Rule Group

`SYNCHRONIZATION`

Reports and Related Files

Reset_sync01.csv

Reset_sync02

Reports asynchronous reset signals that are generated in asynchronous clock domain or are generated from unconstrained source clocks

When to Use

Use this rule to detect asynchronous reset signals that are used in a particular clock domain but are generated in an asynchronous clock domain or are generated from an unconstrained source clock.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Specify reset signals in any of the following ways to identify valid reset synchronization structures in a design:

- By using the *reset* constraint
- By setting the *use_inferred_resets* parameter to `yes` to use auto-generated reset signals

In such cases, the *Reset_sync02* rule does not report a violation for reset pins of flip-flops that are a part of valid reset synchronizers.

Description

The *Reset_sync02* rule reports violations for:

- *Asynchronous Resets Generated in Asynchronous Clock Domains*.
- *Asynchronous Resets Generated from Unconstrained Source Clock*.

Asynchronous Resets Generated in Asynchronous Clock Domains

The *Reset_sync02* rule reports violations for asynchronous resets that are used in a particular clock domain but are generated in an asynchronous clock domain.

The following figure shows an example in which this rule reports a violation:

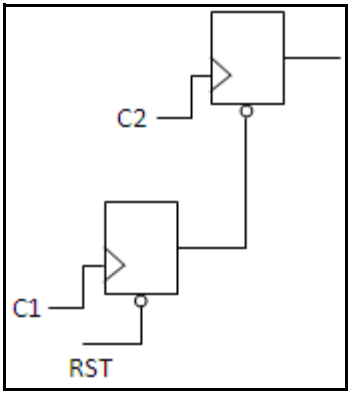


FIGURE 112. Scenario for the Reset_sync02 Rule Violation

This rule reports any one flip-flop per *Reset Cone* with respect to each clock domain. If a cone is driving multiple domains, this rule reports one violation per clock domain. You can expand the reset cone in the schematic to view other flip-flops or set the *enable_reset_cone_spreadsheet* parameter to `yes` to list all the flip-flops for the violation in a spreadsheet.

Asynchronous Resets Generated from Unconstrained Source Clock

The *Reset_sync02* rule reports violations for an unconstrained clock of a *Reset Flip-Flop*. The following figure shows the example of an unconstrained clock:

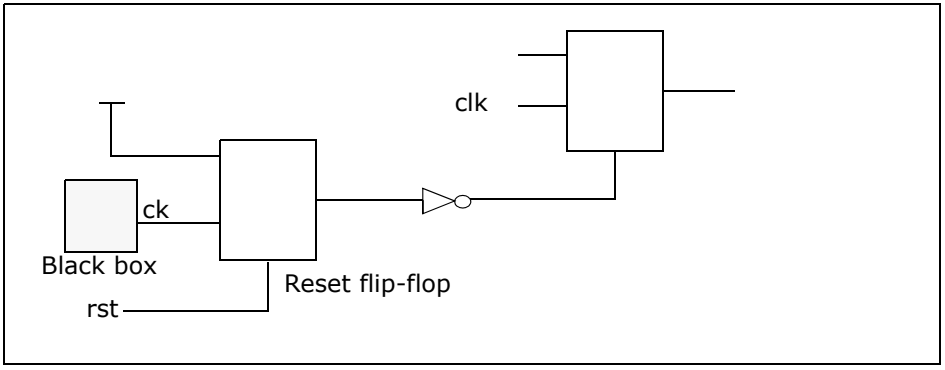


FIGURE 113. Unconstrained Clock

In the above example, the `ck` pin is not constrained by the *clock* constraint.

Rule Functioning

The *Reset_sync02* functions in the following manner:

1. It traverses from each asynchronous reset pin till an input port, a flip-flop output pin, a latch output pin, or a black box pin (ignoring combinational logic) is reached.
2. It reports a violation if the clock domain of a parent flip-flop of a reset signal is not same as the clock domain of the input port (if specified), flip-flop, latch, or black box (the clock domain of the first found clock signal connected to the black box is used).

Parameter(s)

- *report_all_flops*: Default value is `no`. Set this parameter to `yes` to report all flip-flops whose resets are generated in asynchronous domains.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to auto-detect clock signals.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.

- *reset_synchronize_cells*: Default value is `NULL`. Specify a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *reset_num_flops*: Default value is `2`. Specify a positive integer value, greater than one, to specify different number of flip-flops.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *enable_reset_cone_spreadsheet*: Default value is `no`. Set this parameter to `yes` to enable SpyGlass CDC to generate a spreadsheet for each violation message reported by the *Ar_unsync01*, *Ar_asyncdeassert01*, and *Reset_sync02* rules. The generated spreadsheet includes all similar flops that are candidates for the reported violation.
- *allow_unconstrained_reset_in_rfp*: Default value is `no`. Set this parameter to `yes` to enable the *reset_filter_path* constraint to accept unconstrained resets.
- *generate_rfp_suppressed_violations*: Default in `none`. Set this parameter to a supported rule to generate a report of the such suppressed rule violations.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to specify a clock domain on resets.
- *clock* (Optional): Use this constraint to specify clock signals.
- *input* (Optional): Use this constraint to specify clock domain at input ports.

- *reset* (Optional): Use this constraint to specify reset signals.
- *num_flops* with the `-reset` argument (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *reset_filter_path*: (Optional) Use this constraint to specify reset paths so that the reset crossings across these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

Message 1

The following message appears for reset signal *<rst-name>* of flip-flop of clock domain *<clkdomain1-name>* when the reset signal is being generated in an asynchronous clock domain *<clkdomain2-name>*:

[WARNING] Reset signal '<rst-name>' used to reset <obj-type> '<inst-name>' (domain '<clkdomain1-name>') is generated from domain '<clkdomain2-name>'

The arguments of the above message are explained below:

Argument	Description
<obj-type>	`signal` in case of RTL designs. `flop` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `signal`
<inst-name>	*<flip-flop/latch-output-net-name>* in case of RTL designs. *<flip-flop/latch-inst-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs

Potential Issues

This violation appears if your design contains asynchronous reset signals that are generated in an asynchronous clock-domain.

Consequences of Not Fixing

If you do not fix this violation, there can be in metastability issues in your design.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message.

Perform the following steps to find the root cause of the problem:

1. Verify whether the clock domain for both the reset generator flip-flop and the flip-flop where it is used as a reset is same.

   You can view the details of clocks reaching to the clock net (connected to the flip-flop clock pin) by performing the following steps:

   a. Right-click on the clock net connected to the flip-flop clock pin in the *Incremental Schematic* window.

   b. Select the *Show Debug Data->Clock-reset* option from the shortcut menu.

      This option is enabled if the *enable_debug_data* parameter is set to `yes`.

2. Enable *Show Case Analysis* in the schematic to check if *set_case_analysis* constraint is missing on any of the paths that should be blocked.
3. If the reset generator is expected to be a part of reset synchronizer, run and analyze the *Ar_sync01* and *Ar_unsync01* rules to verify whether it is detected as a reset synchronizer.

You may specify the *input* constraint on a reset port for the rule to assume these ports to be synchronized for the specified clock domain.

Message 2

The following message appears for reset signal *<rst-name>* of flip-flop of clock domain *<clkdomain1-name>* when the reset signal is generated from an unconstrained clock:

```
[WARNING] Reset signal '<rst-name>' used to reset <obj-type>
'<inst-name>' (domain '<clkdomain1-name>') is generated from
unconstrained clock
```

Potential Issues

This violation appears if your design contains asynchronous reset signals that are generated from an unconstrained clock.

Consequences of Not Fixing

If you do not fix this violation, there can be in metastability issues in your design.

How to Debug and Fix

See *How to Debug and Fix*.

Example Code and/or Schematic

Consider the following spreadsheet that shows the details of all the violations reported by the *Reset_sync02* rule:

A ID	B RESET	C RESET-CLOCK	D FLOP	E FLOP-CLOCK	F WAIVED
5	test.rst	vclk	test.q	test.clk	No
6	test.reset	test.clk2	test.q	test.clk	No
7	test.trst	test.clk	test.te...	test.clk2	No

FIGURE 114. Spreadsheet generated by the Reset_sync02 rule

In the above spreadsheet, each row represents one violation. You can double-click a row in the spreadsheet to see a list of all the flops that are in the same reset cone as the flop listed in the row. For example, double-clicking the row with ID 6 in the sample spreadsheet shown above, all flops that are in the same reset cone as the `test.q` flop are shown in a separate spreadsheet. Note that this list is available only if the *enable_reset_cone_spreadsheet* parameter set to `yes`.

To view the schematic of a violation (say ID 7), click the link 7 in the above spreadsheet and then click  . The following schematic appears:

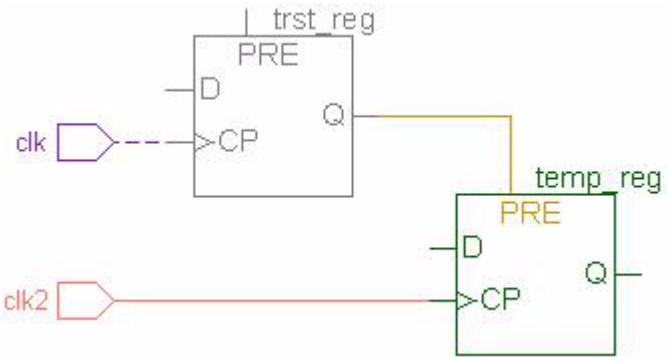


FIGURE 115. Schematic of the Reset_sync02 Rule Violation

In the above schematic, the `top.trst` asynchronous reset is generated in the asynchronous clock domain `clk` and is used in the `clk2` clock domain.

To fix this violation, add a reset synchronization logic of the `clk2` clock domain between the output of the `trst` flip-flop and the `pre` pin of the `temp` flip-flop to synchronously de-assert the `trst` flip-flop.

Schematic Details

The *Reset_sync02* rule highlights the following information in a different color:

- Clock path of a flip-flop generating an asynchronous reset
- Path from a flip-flop generating a reset signal to the flip-flop set/reset pin where it is being used
- Clock path of a flip-flop where a reset is being used
- Destination flip-flop where the reset is being used

Default Severity Label

Warning

Rule Group

`SYNCHRONIZATION`

Reports and Related Files

- Reset_sync02.csv. See *Figure 114*.
- *The SynchInfo Report*

Reset_sync03

Reports multi-flop reset synchronizers in a design

NOTE: *The Reset_sync03 rule will be deprecated in a future release. The rule is not included in CDC GuideWare goals now and do not perform checks until specifically included in the user-defined goal options. In this case, the rule performs the checks and SpyGlass includes a deprecation message in both the spyglass.out and spyglass.log files.*

When to Use

Use this rule to detect multi-flop reset synchronizers in a design.

NOTE: *It is recommended to use the Ar_sync01 and Ar_syncdeassert01 rules instead of this rule.*

Prerequisites

Specify reset signals in any of the following ways:

- By using the *reset* constraint
- By setting the *use_inferred_resets* parameter to `yes` to use auto-generated reset signals
- By using a combination of both the above methods

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use auto-generated clock signals
- By using a combination of both the above methods

Description

The *Reset_sync03* rule reports a violation if a multi-flop reset synchronizer is present in a design.

This rule reports one flip-flop per *Reset Cone*. You can expand the reset cone in the schematic to view other flip-flops.

Checks Performed

This rule performs the following checks if a multi-flop synchronizer output is reaching to an asynchronous set/reset pin of flip-flops:

- If a reset signal is de-asserted properly (as shown in the following figure), it is reported as an informational message.

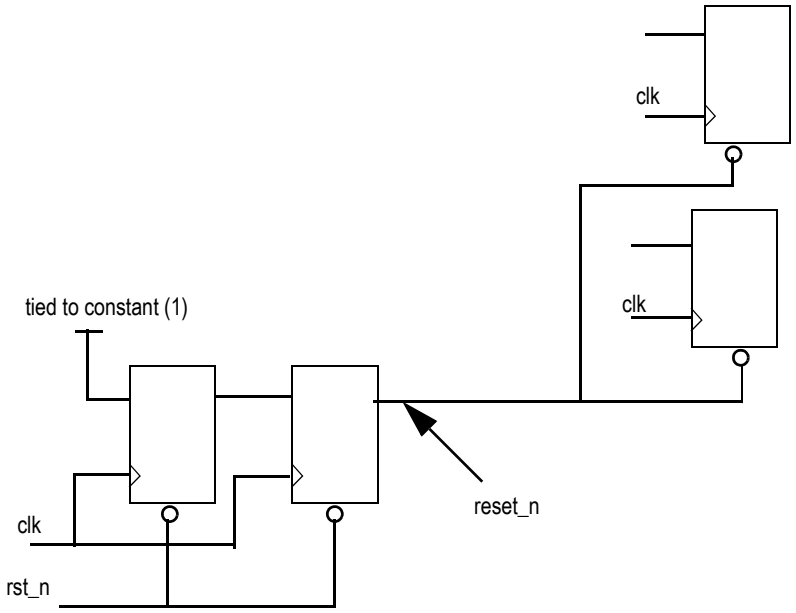


FIGURE 116. Reset Signal De-asserted Properly

- If de-assertion fails, it is reported as a warning. De-assertion may fail due to following reasons:
  - De-assertion circuit is improper even after having multi-flop synchronizer.
  - Multiple clocks are reaching to either synchronizer flip-flops or flip-flops where reset is being used.
  - *set_case_analysis* settings have not been provided, as shown in the following figure:

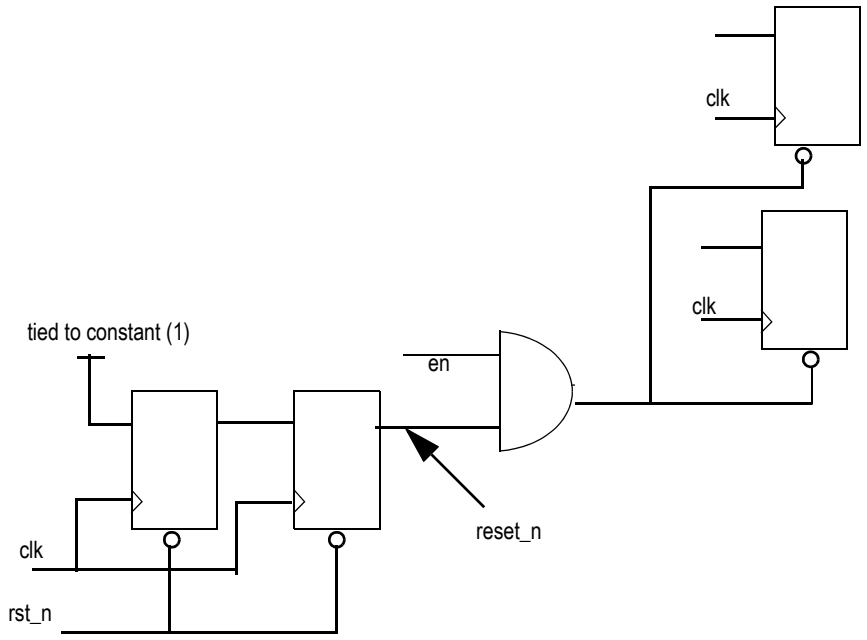


FIGURE 117. set_case_analysis Constraint not Specified for the Enable

In the above example, the *set_case_analysis* constraint has not been specified for the enable `en`.

While checking for de-assertion, the *Reset_sync03* rule uses inactive value of synchronous resets present in the cone of the asynchronous reset.

NOTE: *There may be some overlap with Reset_sync01 rule violations in case of a de-assertion failure.*

Parameter(s)

- *reset_sync_check*: Default value is `strict`. Set this parameter to `soft` to start de-assertion check on reset/clear pin of flip-flops instead of reset source so that you are not required to provide *set_case_analysis* settings on other signals present in cone of reset/clear pins of flip-flops.

- *reset_synchronize_cells*: Default value is `NULL`. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to specify a clock domain on resets.
- *clock* (Optional): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis conditions.
- *num_flops -reset* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

Message 1

The following informational message appears for reset signal *<rst-name>* of flip-flop *<flip-flop-name>* that is de-asserted properly:

[RSync3_1] [INFO] Reset signal '*<rst-name>*' for flop '*<flip-flop-name>*' is synchronously de-asserted relative to clock signal '*<clk-name>*' (reset synchronizer: '*<sync-flip-flop-name>*')

Potential Issues

This violation appears if your design contains a reset signal that is synchronously de-asserted by using multi-flop synchronizer relative to a clock of a reset flip-flop.

Consequences of Not Fixing

None

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

The following informational message appears for reset signal *<rst-name>* of flip-flop *<flip-flop-name>* if the reset signal is synchronized by the user-defined synchronizer cells specified using the *reset_synchronize_cells* parameter:

[RSync3_2] [INFO] Reset signal '*<rst-name>*' for flop '*<flip-flop-name>*' is synchronously de-asserted (User defined reset synchronizer)

Potential Issues

This violation appears if your design contains a reset signal that is synchronously de-asserted by using a user-defined reset synchronizer.

Consequences of Not Fixing

None

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 3

The following warning message appears for reset signal *<rst-name>* of flip-flop *<flip-flop-name>* if the reset signal is not de-asserted properly and a multi-flop synchronizer is detected:

[RSync3_3] [WARNING] Reset signal '*<rst-name>*' for flop '*<flip-flop-name>*' is synchronized by '*<sync-flip-flop-name>*' but is not synchronously de-asserted relative to clock signal '*<clk-name>*'

The arguments of the above message are explained below:

Argument	Description
<rst-name>	Name of the reset signal
<flip-flop-name>	For RTL designs, <flip-flop-name> is the name of the output net of the flip- flip-flop. For netlist designs, if the *report_inst_for_netlist* parameter is set to yes, <flip-flop-name> is the name of the flip-flop instance. Otherwise, the message details are same as for the RTL designs.
<sync-flip-flop-name>	Name of the reset synchronizer
<clk-name>	Name of the destination clock

Potential Issues

This violation is reported when a multi-flop synchronizer is present in the reset path but the reset signal is not de-asserted synchronously.

Consequences of Not Fixing

If you do not fix this violation, the following issues may appear in the design:

- Reset recovery time violation
- Reset removal may occur in different clock cycles for different sequential elements.

How to Debug and Fix

Debug the violation based on the type of violation reported, as described below:

- Reset signal is properly de-asserted synchronously either by multi-flop reset synchronizer in the design or by user-specified reset synchronizer

  This is an informational message and does not require debugging.

- The reset signal is not properly de-asserted

  To debug this violation message, view the *Incremental Schematic* of the violation message.

  The following figure illustrates the schematic of this rule:

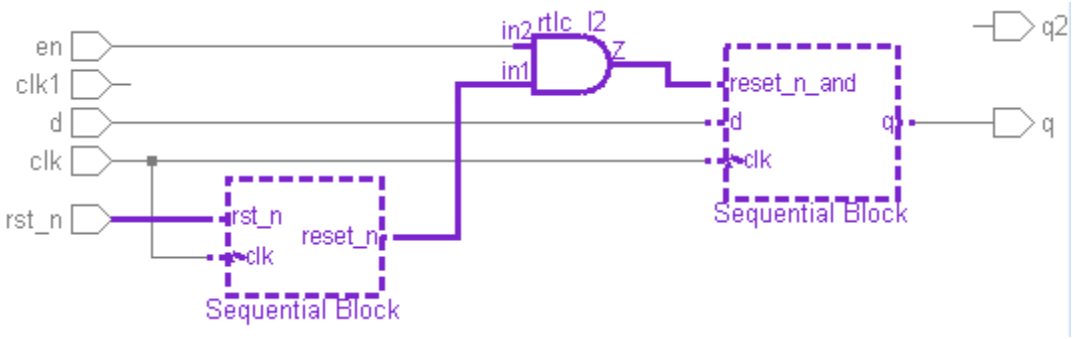


FIGURE 118. Schematic of the Reset_sync03 Rule Violation

In the schematic, you will notice that when the reset signal `rst_n` is de-asserted, the synchronized output does not de-assert the flip-flop `q_reg` due to unconstrained input `in2` of AND gate `rtlc_I2`.

Perform the following steps to find the root cause of the problem:

1. Check if the combinational logic is present between the last synchronizer flip-flop and the flip-flop where it is used as a reset.

   This is not allowed if the combinational logic is not acting as buffer/ inverter and not allowing reset value to propagate. Enable *Show Case Analysis* in the schematic to check the constant value propagation. You may need to constraint the path by using *set_case_analysis* constraint.

2. Check if combinational logic is present between reset source and the synchronizer flip-flop.

   This is not allowed if the combinational logic is not acting as buffer/ inverter and not allowing reset value to propagate. Enable *Show Case Analysis* in the schematic to check the constant value propagation. You

may need to constraint the path by using the *set_case_analysis* constraint.

You may set the *reset_sync_check* parameter to `soft` for the rule to perform de-assertion check on reset/clear pin of flip-flops instead of the reset source.

3. Multiple clocks are reaching to either synchronizer flip-flops or the flip-flops where reset is being used.

   You may view the details of clocks reaching to the clock net (connected to the flip-flop clock pin) by performing the following steps:

   a. Right-click on the clock net connected to the flip-flop clock pin in the *Incremental Schematic* window.

   b. Select the *Show Debug Data->Clock-reset* option from the shortcut menu.

      This option is enabled when the *enable_debug_data* parameter is set to `yes`.

   You may need to specify *set_case_analysis* constraints to allow propagation of only one of the clocks.

4. If you have specified reset synchronizer cell, verify whether all the fan-out paths of the reset net reaches reset synchronizer cell after ignoring inverters and buffers.

   You may view the details of clocks reaching to the clock net (connected to the flip-flop clock pin) by specifying the *input* constraint on a reset port for the rule to assume these ports to be synchronized for the specified clock domain.

5. You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

Consider the following schematic of a violation reported by the *Reset_sync03* rule:

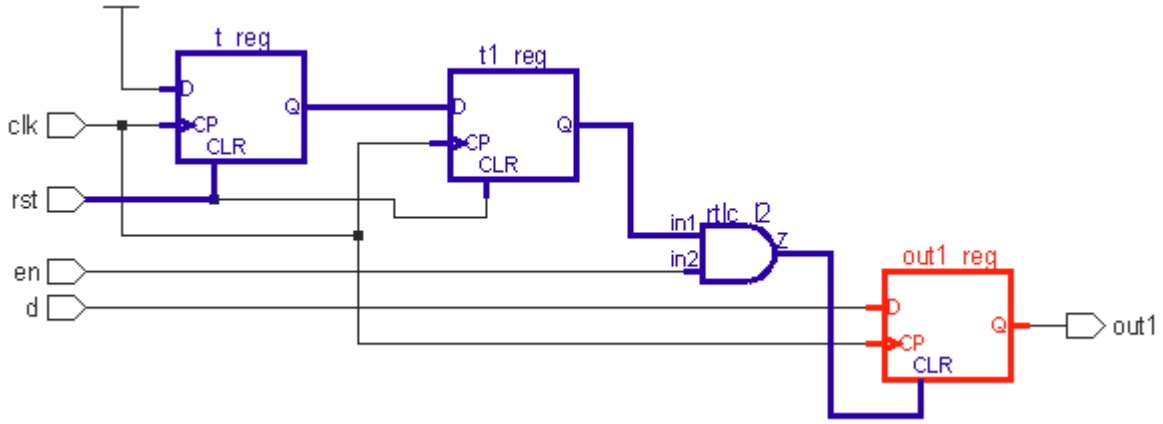


FIGURE 119. Schematic of the Reset_sync03 Rule Violation

In the above example, when the `rst` reset signal is de-asserted, the synchronized output does not de-assert the `out1` flip-flop due to unconstrained `en` input of the `rtlc_I2` AND gate.

To fix this violation, constraint the `en` input of the `rtlc_I2` AND gate to value `1`. This will lead to propagation of synchronous output to the `out1` flip-flop.

Schematic Details

The *Reset_sync03* rule highlights each of the following in a different color:

- Reset synchronizer and their reset path
- Path from output of set/reset of synchronizer flip-flop to the set/reset pin of the flip-flop where it is being used as reset

Rule Severity

Info/Warning

Rule Group

`SYNCHRONIZATION`

Reports and Related Files

Reset_sync04

Reports asynchronous resets that are synchronized more than once in the same clock domain

When to Use

Use this rule to detect the usage of distributed reset synchronization schemes, where reset synchronizers are placed at multiple hierarchies.

Prerequisites

Specify the following details before running this rule:

- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated reset signals when the *use_inferred_resets* parameter is set to `yes`
  - By using a combination of both the above methods
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clock signals when the *use_inferred_clocks* parameter is set to `yes`
  - By using a combination of both the above methods

Description

The *Reset_sync04* rule reports a violation for asynchronous resets that are synchronized multiple times in the same clock domain.

This rule reports any two destinations on which a source reset is synchronized in the same domain.

The following figure shows the scenario in which this rule reports a violation:

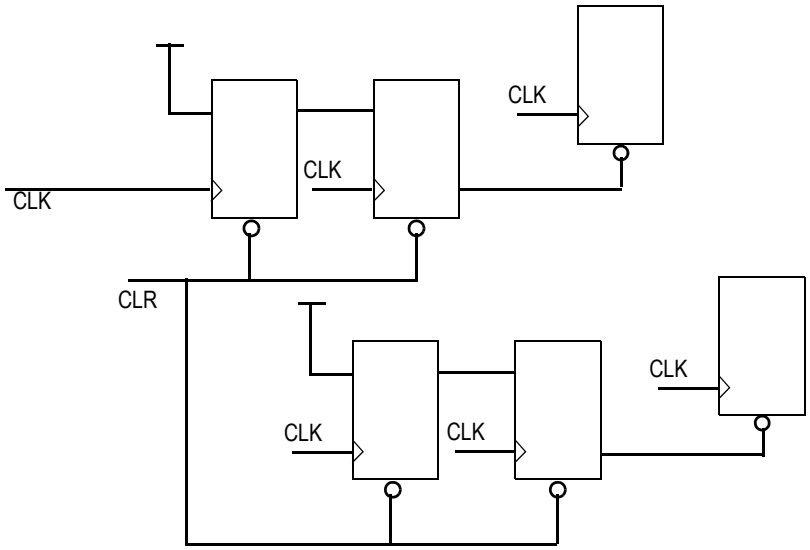


FIGURE 120. Scenario for the Reset_sync04 Rule Violation

This rule checks for the presence of multi-flop synchronizers irrespective of whether synchronous de-assertion is happening properly or not.

If more than two synchronizers exist in the same clock domain, any two synchronizers are reported by the *Reset_sync04* rule.

Rule Exceptions

The *Reset_sync04* rule does not check for resets that are synchronized by the synchronize cells specified by the *reset_synchronize_cells* parameter.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *num_flops -reset* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use automatically-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use automatically-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *dump_detailed_info*: Default value is `none`. Set this parameter to a supported value to enable the rule to include detailed information in the generated rule/message-based spreadsheet.

Messages and Suggested Fix

The following message appears for the reset signal *<rst-name>* if it is synchronized at multiple synchronizers in the same clock domain *<clk-name>*:

[WARNING] Asynchronous reset signal '*<rst-name>*' is synchronized at least twice (at '*<sync-flop1-name>*' and

'*<sync-flop2-name>*') relative to the clock signal '*<clk-name>*'

Where, *<sync-flop1-name>* and *<sync-flop2-name>* are two of the many synchronizers at which the reset signal is synchronized.

NOTE: *For RTL designs, <sync-flop1-name> and <sync-flop2-name> are the names of the output net of the flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to* yes*, these are the names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains a reset signal that is synchronized at multiple synchronizers in the same clock domain.

Consequences of Not Fixing

If you do not fix this violation, reset removal may happen in different clock cycles for different synchronizers.

Slight differences in propagation delays in the reset signal or the clock signal can result in different values at the output of different synchronizers. This may result in data coherency issues.

How to Debug and Fix

To fix this violation, synchronize the reset signal once and then use it.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:

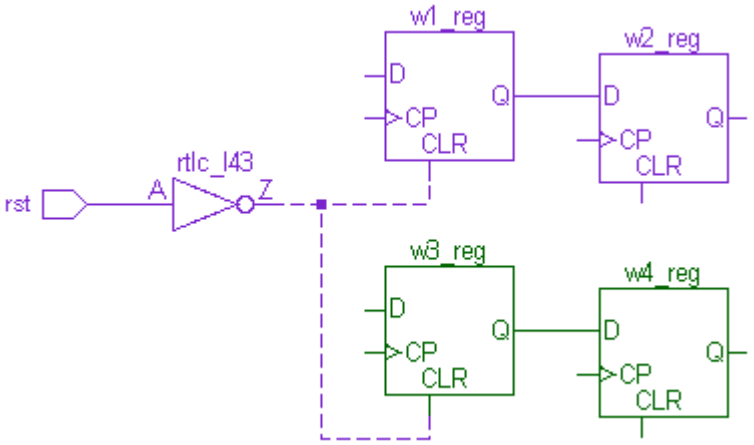


FIGURE 121. Schematic for the Reset_sync04 Rule Violation

In the above example, the *Reset_sync04* rule reports a violation because the reset signal `rst` is being synchronized by the following reset synchronizer chains:

- The chain consisting of the instance `w1_reg` and `w2_reg`
- The chain consisting of the instance `w3_reg` and `w4_reg`

To fix this violation, remove any one synchronizer chain so that the modified design contains only one reset synchronizer.

Schematic Details

The *Reset_sync04* rule highlights each of the following information in a different color:

- Instance chain of the first synchronizer
- Instance chain of the second synchronizer

Default Severity Label

Warning

Rule Group

SYNCHRONIZATION

Reports and Related Files

The SynchInfo Report

CDC Verification Rules

The following table lists the Clock-Domain Crossings (CDC) verification rules:

Rule	Reports
Ac_unsync01	Asynchronous clock domain crossings for scalar signals that have at least one unsynchronized source
Ac_unsync02	Asynchronous clock domain crossings for vector signals having at least one unsynchronized source
Ac_sync01	Asynchronous clock domain crossings for scalar signals that have all the sources synchronized.
Ac_sync02	Asynchronous clock domain crossings for vector signals that have all sources synchronized
Ac_coherency06	Reports signals that are synchronized more than once in the same clock domain
Clock_sync05	Multi-sample inputs
Ac_crossing01	Generates spreadsheet for Crossing Matrix view
Clock_sync03	Runs the *Clock_sync03a* and *Clock_sync03b* rules
Clock_sync03b	Reports convergence of signals from different domains
Clock_sync06	Multi-transition outputs
Clock_sync08a	Bus-bits that are not synchronized using the recommended techniques (specific case)
Clock_sync09	Signals at clock domain crossings that are synchronized at more than one place
Ac_conv01	Same-domain signals that are synchronized with multi-flop and sync-cell synchronizers to another domain and that are converging after any number of sequential elements
Ac_conv03	Different-domain signals that are synchronized with multi-flop and sync-cell synchronizers to another domain and that are converging before encountering a sequential element
Advanced Clock Functional Verification Rules	
Ac_cdc01	Runs the *Ac_cdc01a*, *Ac_cdc01b*, and *Ac_cdc01c* rules

Rule	Reports
Ac_cdc01a	Fast-to-slow clock crossings where the data generated by the source register (fast clock) is not stable long enough for the destination register (slow clock) to properly capture the data when the clock crossing is found to be synchronized by the multi-flop synchronization scheme
Ac_cdc01b	Fast-to-slow clock crossings where the data generated by the source register (fast clock) is not stable long enough for the destination register (slow clock) to properly capture the data when the clock crossing is found to be synchronized by a scheme other than the multi-flop synchronization scheme
Ac_cdc01c	Fast-to-slow clock crossings where the data generated by the source register (fast clock) is not stable long enough for the destination register (slow clock) to properly capture the data when the clock crossing is found to be un-synchronized
Ac_cdc08	Synchronized multi-bit control buses where the bus is not Gray encoded
Ac_clockperiod03	Clocks with design cycles greater than the threshold value
Ac_conv02	Same-domain signals that are synchronized with multi-flop and sync-cell synchronizers to another domain and that are converging before encountering a sequential element
Ac_conv04	All the control bus clock domain crossings that do not follow gray encoding
Ac_conv05	The status of gray encoding check performed on the signals specified by the *gray_signals* constraint
Ac_datahold01a	Reports synchronized data clock domain crossings where data can be unstable
Clock_sync03a	Reports convergence of signals from same source domain separately synchronized in a single destination domain

To learn how to use Advanced Functional Verification Rules, see *Performing Functional Analysis in SpyGlass CDC*.

Note that while running these rules, formal modeling of a port is done with respect to the first clock in following cases:

- If multiple clocks are specified for the same port through the *abstract_port* constraints
- If multiple *abstract_port* constraints are applied on the same port with different clocks

Ac_unsync01

Reports unsynchronized clock domain crossing for scalar signals

When to Use

Use this rule to find unsynchronized clock domain crossings for scalar signals in a design.

Prerequisites

Specify the following details before running this rule:

- Specify the `Advanced_CDC` and `adv_checker` license features.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clock information after setting the *use_inferred_clocks* parameter to `yes`

Description

The *Ac_unsync01* rule reports asynchronous clock domain crossings for the scalar signals having at least one unsynchronized source.

This rule also reports the reason for unsynchronized crossings. For details, see *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules* and *Reasons for Synchronized Crossings Reported by Ac_sync_group Rules*.

This rule belongs to *The Ac_sync_group Rules*.

NOTE: *The Ac_unsync01 rule is switched off by default.*

Parameter(s)

See *Parameters of the Ac_sync_group Rules*.

Constraint(s)

See *Constraints of the Ac_sync_group Rules*.

Messages and Suggested Fix

The following message appears when an unsynchronized crossing is detected between a source and destination:

```
[AcUSync1_1] [ERROR] Unsynchronized Crossing: destination
<type1> <name1>, clocked by <clock-name1>, source <type2>
<name2>, clocked by <clock-name2>. Reason: <reason> [Total
Sources: <count1> (Number of source domains: <count2>)]
```

The arguments of the above message are explained below:

Argument	Description
<type1>	Can be flip-flop, latch, black box, or primary output
<name1>	By default, destination net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, destination instance name is reported for netlist designs. To report the hierarchical pin name of the destination instance, set the *report_instance_pin* parameter to yes.
<clock-name1>	One of the clock that is reaching the destination
<type2>	flip-flop, latch, black box, or primary output
<name2>	By default, source net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, source instance name is reported for netlist designs. To report the hierarchical pin name of the source instance, set the *report_instance_pin* parameter to yes.
<clock-name2>	One of the clock that is reaching the source
<reason>	Failure reason. For details, see *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules*.
<count1>	Total number of sources involved in a crossing. Each bus group is counted as a single source.
<count2>	Total number of source domains involved in the crossing

Potential Issues

This violation appears if an unsynchronized data transfer occurs between different domain scalar signals.

Consequences of Not Fixing

If you do not fix this violation, unsynchronized crossings may cause metastability issues in a design. This may cause functional issues resulting

in chip failure.

How to Debug and Fix

If the number of violations is huge (say thousands of violations), the setup is possibly incorrect and incomplete. For such cases, see *Debugging Large Number of Violations*.

To perform the root cause analysis of the problem, perform the following steps:

1. If both source and destination instances reside in an IP module, specify the *ip_block* constraint for the module.
2. If the crossing involves a black box, ensure that the black box is configured properly with either of the following settings:
   - ❒ A domain is assigned to the black box pin (use the *abstract_port* constraint
   - ❒ A pin is defined as a feed through for the black box (use the *assume_path* constraint)

   If you do not want to view black box crossings, set the *cdc_reduce_pessimism* parameter to `bbox`.
3. Check if the crossing output is blocked, unused, or hanging.

   Use the *cdc_reduce_pessimism* parameter if you do not want to view such crossings.
4. If the source flip-flop is a static signal, no synchronization may be required. Declare such signal by using the *quasi_static* or *cdc_false_path* constraint.
5. Analyze the violation from the failure reason reported in the violation message.

   For details on the reason, see *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules* topic.
6. View the *Incremental Schematic* of the violation message to analyze the cause of failure.
7. Based on the source and destination flip-flops, perform appropriate actions as described below:
   - ❒ If the source flip-flop is a control signal and there is no valid synchronizer, determine the cause by exploring the destination flip-flop and fix it by inserting synchronizer flip-flops in the crossing.

- ❒ If this is a data crossing, perform appropriate actions as described below:
  - If the crossing is from the memory of an asynchronous FIFO (for example, SpyGlass CDC solution did not automatically recognized the structures), specify the *fifo* constraint to let SpyGlass CDC solution know about this asynchronous FIFO.
  - If the crossing involves a handshake (for example, SpyGlass CDC solution did not automatically identified the handshake), specify the *qualifier* constraint for handshake control line (for example, the synchronized request).
  - For any other ad-hoc data synchronization, identify the signal in charge of data crossing synchronization. For details, see *Identifying the Signal in Charge of Data-Crossing Synchronization*.

8. If there are crossings that you do not want to review, waive them by using the *cdc_false_path* constraint.
9. You can also view case analysis settings along with the violation of this rule.

Identifying the Signal in Charge of Data-Crossing Synchronization

Perform an appropriate action based on the following situations:

- If the control signal is not synchronized

  Action: Insert a synchronizer.

- If the synchronized control signal is controlling the crossing through an AND gate

  Action: Set the *enable_and_sync* parameter to `yes` to allow this synchronization scheme.

- If the synchronized control signal is controlling the crossing through gates other than the following:

  - ❒ Enable of a destination flip-flop
  - ❒ Selection of a recirculation MUX around the destination flip-flop
  - ❒ Clock-gating cell at the destination flip-flop

  Action: Follow the standard data synchronization techniques to avoid glitches on the crossing.

Debugging Large Number of Violations

Open the spreadsheet of this rule. See *Spreadsheet Support in Ac_sync_group Rules*.

To debug violations through the spreadsheet, perform the following actions:

- Cross-probe to RTL code and view schematic of the violation.
- Use filtering and sorting to debug the related messages.

  You can filter messages based on source and destination clock pairs, failure reason, and crossing module for easy debug.
- Waive or apply *cdc_false_path* constraint directly from the spreadsheet.

Example Code and/or Schematic

Consider the following rule-based spreadsheet generated by the *Ac_unsync01* rule:

A	B	C	D	E	F	G	
ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	REASON	TOTAL SOURCES	TOTAL
14	top.s1	top.clk1	top.out1	top.clk2	Gating logic not accepted: source drives MUX select input	3	1
11	top.s2	top.clk1	top.out2	top.clk2	Qualifier not found	1	1

FIGURE 122. Rule-Based Spreadsheet of the Ac_unsync01 Rule

The above spreadsheet shows the details of all violations (in separate rows) reported by the *Ac_unsync01* rule.

To view details of each violation, click the link in the *ID* column corresponding to a particular violation in the above spreadsheet.

When you click this link, the message-based spreadsheet appears. For example, when you click *14* in the *ID* column of the above spreadsheet, the following spreadsheet appears:

A Schematic	B Type	C Signal Name	D Failure Reason	E Synchronization Scheme	F Clock Names
15	Destination flop	top.out1	unsynchronized destination	N.A.	top.clk2
15	Source flop	top.s1	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1
16	Source flop	top.s4	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1
17	Source flop	top.s6	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1
17	Qualifier (detected)	top.d5	N.A.	Conventional multi-flop for metastability technique	top.clk2
18	Potential Qualifier	top.d3	N.A.	Conventional multi-flop for metastability technique	top.clk2

FIGURE 123. Message-Based Spreadsheet of the Ac_unsync01 Rule

The above spreadsheet provides details of the violation selected in the rule-based spreadsheet. These details include the following:

- All sources of the destination
- Source clock names
- Synchronization or unsynchronized status for each source and clock domain tag
- All valid qualifiers synchronizing the source signals

- Potential qualifier signals, which when synchronized make a source as synchronized

To view the schematic of the violation reported in the above spreadsheet, click the link in the *Schematic* column of a row in the spreadsheet and then click  in the toolbar of the spreadsheet window.

For example, the following figure shows the schematic of the unsynchronized crossing containing the `top.s6` source flip-flop:

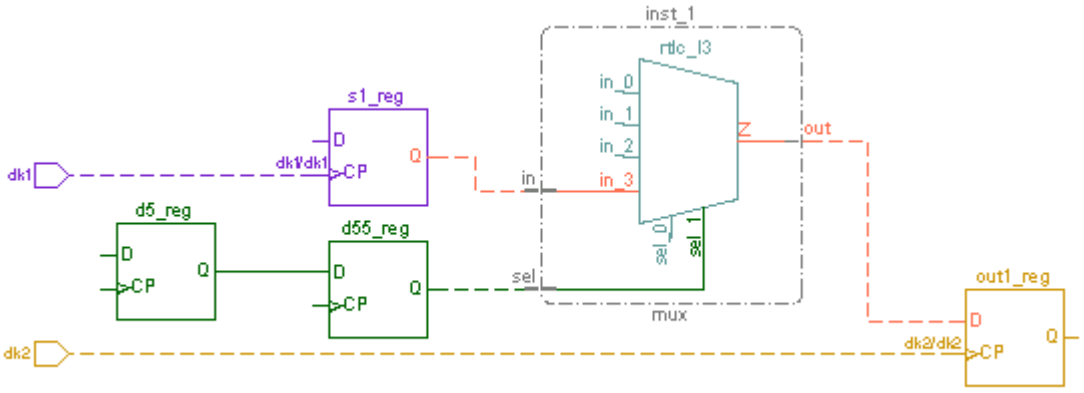


FIGURE 124. Schematic of the Ac_unsync01 Rule Violation

Schematic Details

The *Ac_unsync01* rule highlights the following information in different colors in the schematic:

- Source clock and the source instance or port
- Destination clock and the destination instance or port
- Crossing path including combination logic
- Qualifier signal, multi-flop synchronizer, or any other valid synchronizer
- Potential qualifier signal converging with the source

Default Severity Label

Error

Rule Group

```
Ac_sync_group
```

Reports and Related Files

- Ac_unsync01.csv: This is a *Rule-Based Spreadsheet* that contains details of all violations of this rule.
- ac_unsync_<unique-number>.csv: This is a *Message-Based Spreadsheet* that contains details of a particular violation of this rule.
- *The Ac_sync_group_detail Report*
- *The Ac_sync_qualifier Report*
- *The CrossingInfo Report*
- *The SynchInfo Report*
- *The Clock-Reset-Summary Report*
- *The Clock-Reset-Detail Report*

Ac_unsync02

Reports for unsynchronized clock domain crossings for vector signals

When to Use

Use this rule to find unsynchronized clock domain crossings for vector signals in a design.

Prerequisites

Specify the following details before running this rule:

- Specify the `Advanced_CDC` and `adv_checker` license features.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clock information after setting the *use_inferred_clocks* parameter to `yes`

Description

The *Ac_unsync02* rule reports asynchronous clock domain crossings for vector signals that have at least one unsynchronized source.

This rule also reports the reason for unsynchronized crossings. For information on the reasons, see *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules* and *Reasons for Synchronized Crossings Reported by Ac_sync_group Rules*.

This rule belongs to *The Ac_sync_group Rules*.

NOTE: *The Ac_unsync02 rule is switched off by default.*

Parameter(s)

See *Parameters of the Ac_sync_group Rules*.

Constraint(s)

See *Constraints of the Ac_sync_group Rules*.

Message Details

The following message appears when an unsynchronized crossing is detected between a source and destination:

```
[AcUSync2_1] [ERROR] Unsynchronized Crossing: destination
<type1> <name1>, clocked by <clock-name1>, source <type2>
<name2>, clocked by <clock-name2>. Reason: <reason> [Total
Sources: <count1> (Number of source domains: <count2>)]
```

The arguments of the above message are explained below:

Argument	Description
<type1>	Can be flip-flop, latch, black box, or primary output
<name1>	By default, destination net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, destination instance name is reported for netlist designs. To report the hierarchical pin name of the destination instance, set the *report_instance_pin* parameter to yes.
<clock-name1>	One of the clock that is reaching the destination
<type2>	Can be flip-flop, latch, black box, or primary output
<name2>	By default, source net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, source instance name is reported for netlist designs. To report the hierarchical pin name of the source instance, set the *report_instance_pin* parameter to yes.
<clock-name2>	One of the clock that is reaching the source
<reason>	Failure reason. For details on failure reasons, see *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules*.
<count1>	Total number of sources involved in a crossing. Each bus group is counted as a single source.
<count2>	Total number of source domains involved in the crossing

Potential Issues

This violation appears if an unsynchronized data transfer occurs between different domain vector signals.

Consequences of Not Fixing

If you do not fix this violation, unsynchronized crossings may cause metastability issues in a design. This may cause functional issues resulting in chip failure.

How to Debug and Fix

If the number of violations is huge (say thousands of violations), the setup is possibly incorrect and incomplete. For such cases, refer to the *Debugging Large Number of Violations* topic.

To debug and fix this violation, perform the following steps:

1. If both source and destination instances reside in an IP module, specify the *ip_block* constraint for the module.
2. If the crossing involves a black box, ensure that the black box is configured properly with either of the following settings:
   - ❒ A domain is assigned to the black box pin (use the *abstract_port* constraint
   - ❒ A pin is defined as a feed through for the black box (use the *assume_path* constraint)

   If you do not want to view black box crossings, use *cdc_reduce_pessimism* parameter.
3. Check if the crossing output is blocked, unused, or hanging.

   Use the *cdc_reduce_pessimism* parameter if you do not want to view such crossings.
4. If the source flip-flop is a static signal, no synchronization may be required. Declare such signal by using the *quasi_static* or *cdc_false_path* constraint.
5. Analyze the violation from the failure reason reported in the violation message.

   For details on each failure reason, see *Reasons for Unsynchronized Crossings Reported by Ac_sync_group Rules* topic.
6. View the *Incremental Schematic* of the violation message to analyze the cause of failure.

7. Based on source and destination flip-flops, perform appropriate actions as described below:
   - ❒ If the source flip-flop is a control signal and there is no valid synchronizer, determine the cause by exploring the destination flip-flop and fix it by inserting synchronizer flip-flops in the crossing.
   - ❒ If this is a data crossing, perform appropriate actions as described below:
     - If the crossing is from the memory of an asynchronous FIFO (for example, SpyGlass CDC solution did not automatically recognized the structures), specify the *fifo* constraint to let SpyGlass CDC solution know about this asynchronous FIFO.
     - If the crossing involves a handshake (for example, SpyGlass CDC solution did not automatically identified the handshake), specify the *qualifier* constraint for handshake control line (for example, the synchronized request).
     - For any other ad-hoc data synchronization, identify the signal in charge of data crossing synchronization. For details, see *Identifying the Signal in Charge of Data-Crossing Synchronization*.
8. If there are crossings that you do not want to review, waive them by using the *cdc_false_path* constraint.
9. You can also view case analysis settings along with the violation of this rule.

Identifying the Signal in Charge of Data-Crossing Synchronization

Perform the following actions based on the following cases:

- If the control signal is not synchronized

  Action: Insert a synchronizer.

- If the synchronized control signal is controlling the crossing through an AND gate

  Action: Set the *enable_and_sync* parameter to allow this synchronization scheme.

- If the synchronized control signal is controlling the crossing through gates other than the following:
  - ❒ Enable of a destination flip-flop

- ❒ Selection of a recirculation MUX around the destination flip-flop
- ❒ Clock-gating cell at the destination flip-flop

Action: Follow standard data synchronization techniques to avoid glitches on the crossing.

Example Code and/or Schematic

Consider the following rule-based spreadsheet generated by the *Ac_unsync02* rule:

A	B	C	D	E	F	G	H
ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	REASON	TOTAL SOURCES	TOTAL SOURCE DOMAINS
B	top.s2	top.clk1	top.out_bus[7:4]	top.clk2	Qualifier not found	5	1

FIGURE 125. Rule-Based Spreadsheet of the Ac_unsync02 Rule

In the above rule-based spreadsheet, click *B* in the *ID* column to display the message-based spreadsheet, as shown in the following figure:

A Schematic	B Type	C Signal Name	D Failure Reason	E Synchronization Scheme	F Clock Names
4	Destination flop	top.out_bus[7:4]	unsynchronized destination	N.A.	top.clk2
4	Source flop	top.s2	Qualifier not found	N.A.	top.clk1
5	Source flop	top.s3	Qualifier not found	N.A.	top.clk1
6	Source flop	top.s1	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1
7	Source flop	top.s4	Gating logic not accepted: source drives MUX select input	N.A.	top.clk1
8	Source flop	top.s6	Gating logic not accepted: source drives MUX selec...	N.A.	top.clk1
8	Qualifier (detected)	top.d5	N.A.	Conventional multi-flop for metastability technique	top.clk2
9	Potential Qualifier	top.d3	N.A.	Conventional multi-flop for metastability technique	top.clk2

FIGURE 126. Message-Based Spreadsheet of the Ac_unsync02 Rule

The above spreadsheet shows the following details:

- All sources of the destination
- Source clock names
- Synchronization or unsynchronized status for each source and clock domain tag
- All valid qualifiers synchronizing the source signals
- Potential qualifier signals, which when synchronized make a source as synchronized

To view the schematic of the violation reported in the above spreadsheet, click the link in the *Schematic* column of a row in the spreadsheet and then click  in the toolbar of the spreadsheet window.

For example, the following figure shows the schematic of the unsynchronized crossing containing the `top.s4` source flip-flop:

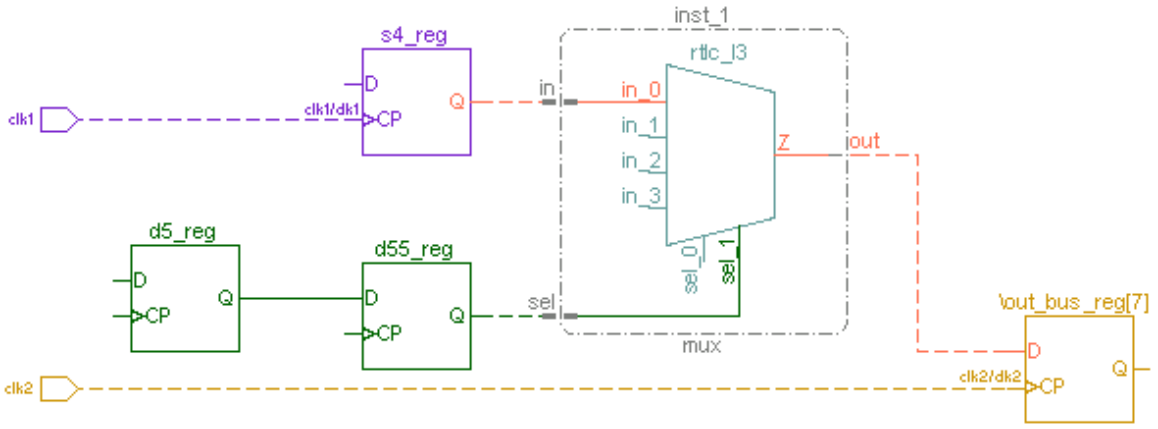


FIGURE 127. Schematic of the Ac_unsync02 Rule Violation

Schematic Details

The *Ac_unsync02* rule highlights the following information in different colors in the schematic:

- Source clock and the source instance or port
- Destination clock and the destination instance or port
- Crossing path including combination logic
- Qualifier signal, multi-flop synchronizer, or any other valid synchronizer
- Potential qualifier signal converging with the source

Default Severity Label

Error

Rule Group

`Ac_sync_group`

Reports and Related Files

- Ac_unsync02.csv: This is a *Rule-Based Spreadsheet* that contains details of all violations of this rule.
- ac_unsync_<unique-number>.csv: This is a *Message-Based Spreadsheet* that contains details of a particular violation of this rule.
- *The Ac_sync_group_detail Report*
- *The Ac_sync_qualifier Report*
- *The CrossingInfo Report*
- *The SynchInfo Report*
- *The Clock-Reset-Summary Report*
- *The Clock-Reset-Detail Report*

Ac_sync01

Reports synchronized clock domain crossings for scalar signals

When to Use

Use this rule to find synchronized clock domain crossings for scalar signals in a design.

Prerequisites

Specify the following details before running this rule:

- Specify the `Advanced_CDC` and `adv_checker` license features.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clock information after setting the *use_inferred_clocks* parameter to `yes`

Description

The *Ac_sync01* rule reports asynchronous clock domain crossings for scalar signals that have all sources synchronized.

To know about the cases in which a source is considered as synchronized, see *Reasons for Synchronized Crossings Reported by Ac_sync_group Rules*.

NOTE: *Please note the following points:*

- *The Ac_sync01 rule belongs to The Ac_sync_group Rules.*
- *The Ac_sync01 rule is switched off by default.*

Parameter(s)

See *Parameters of the Ac_sync_group Rules*.

Constraint(s)

See *Constraints of the Ac_sync_group Rules*.

Messages and Suggested Fix

Message 1

The following message appears at the line where the destination of type *<type1>* object of clock domain *<clock-name1>* receives a signal from source of type *<type2>* object of clock domain *<clock-name2>*:

```
[AcSync1_1] [INFO] Synchronized Crossing: destination <type1>
<name1>, clocked by <clock-name1>, source <type2> <name2>,
clocked by <clock-name2>, by method: <method> [Total Sources:
<count1> (Number of source domains: <count2>)]
```

The arguments of the above message are explained below:

Argument	Description
<type1>	flip-flop, latch, black box, or primary output
<name1>	By default, destination net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, destination instance name is reported for netlist designs. To report the hierarchical pin name of the destination instance, set the *report_instance_pin* parameter to yes.
<clock-name1>	One of the clock that is reaching the destination
<type2>	flip-flop, latch, black box, or primary output
<name2>	By default, source net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, source instance name is reported for netlist designs. To report the hierarchical pin name of the source instance, set the *report_instance_pin* parameter to yes.
<clock-name2>	One of the clock that is reaching the source
<method>	Synchronization method
<count1>	Total number of sources involved in a crossing. Each bus group is counted as a single source.
<count2>	Total number of source domains involved in the crossing

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule that reports asynchronous clock domain crossings for scalar signals for which all sources are synchronized.

To view details of such crossings to check if everything is as intended, see the following:

- *Rule-Based Spreadsheet*
- *Message-Based Spreadsheet*
- Schematic. See *Viewing Schematic Through the Spreadsheet*

If you do not want to view the message of this rule, waive or disable this rule.

Example Code and/or Schematic

Consider the following rule-based spreadsheet generated for the *Ac_sync01* rule:

A ID	B SOURCE	C SOURCE CLOCK	D DEST.	E DEST. CLOCK	F METHOD	G TOTAL SOURCES	H OTAL SOURCE DOMAI
4	top.s5	top.clk1	top.d5	top.clk2	Conventional multi-flop for metastability technique	1	1
5	top.s3	top.clk1	top.d3	top.clk2	Conventional multi-flop for metastability technique	1	1
6	top.s2	top.clk1	top.d2	top.clk2	Conventional multi-flop for metastability technique	1	1

FIGURE 128. Rule-Based Spreadsheet of the Ac_sync01 Rule

The above spreadsheet shows the details of all violations (in separate rows) of the *Ac_sync01* rule.

To view further details of each violation, click the link in the *ID* column corresponding to a particular violation in the above spreadsheet.

When you click this link, another spreadsheet appears. For example, when you click 5 in the *ID* column of the above spreadsheet, the following spreadsheet appears:

A	B	C	D	E	F
Schematic	Type	Signal Name	Synchronization Scheme	Clock Names	Internal Clock Domain Tag
2	Destination flop	top.d3	Conventional multi-flop for metastability technique	top.clk2	1
2	Source flop	top.s3	Conventional multi-flop for metastability technique	top.clk1	0

FIGURE 129. Message-Based Spreadsheet of the Ac_sync01 Rule

To view the schematic of the violation reported in the above spreadsheet, click the link in the *Schematic* column and then click in the tool bar of the spreadsheet window.

For example, the following figure shows the schematic of the synchronized crossing containing the `top.s3` source flip-flop:

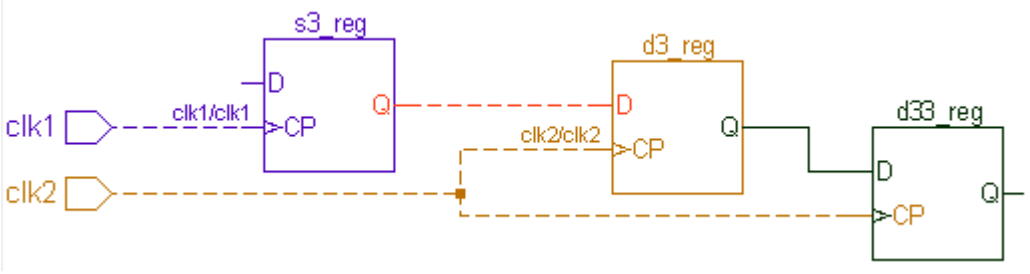


FIGURE 130. Schematic of the Ac_sync01 Rule Violation

Schematic Details

The *Ac_sync01* rule highlights the following information in different colors in the schematic:

- Source clock and the source instance or port
- Destination clock and the destination instance or port
- Crossing path including combination logic
- Qualifier signal, multi-flop synchronizer, or any other valid synchronizer

Default Severity Label

Info

Reports and Related Files

- Ac_sync01.csv: This is a *Rule-Based Spreadsheet* that contains details of all violations of this rule.
- ac_sync_<unique-number>.csv: This is a *Message-Based Spreadsheet* that contains details of a particular violation of this rule.
- *The Ac_sync_group_detail Report*
- *The Ac_sync_qualifier Report*
- *The CrossingInfo Report*
- *The SynchInfo Report*
- *The Clock-Reset-Summary Report*
- *The Clock-Reset-Detail Report*

Ac_sync02

Reports synchronized clock domain crossings for vector signals

When to Use

Use this rule to find synchronized clock domain crossings for vector signals in a design.

Prerequisites

Specify the following details before running this rule:

- Specify the `Advanced_CDC` and `adv_checker` license features.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clock information after setting the *use_inferred_clocks* parameter to `yes`

Description

The *Ac_sync02* rule reports asynchronous clock domain crossings for vector signals that have all sources synchronized.

To know the cases in which a source is considered as synchronized, see *Reasons for Synchronized Crossings Reported by Ac_sync_group Rules*.

NOTE: *Please note the following points:*

- *The Ac_sync02 rule belongs to The Ac_sync_group Rules.*
- *The Ac_sync02 rule is switched off by default.*

Parameter(s)

See *Parameters of the Ac_sync_group Rules*.

Constraint(s)

See *Constraints of the Ac_sync_group Rules*.

Messages and Suggested Fix

The following message appears at the line where the destination of type *<type1>* object of the clock domain *<clock-name1>* receives a signal from source of type *<type2>* object of the clock domain *<clock-*

name2>:

```
[AcSync2_1] [INFO] Synchronized Crossing: destination <type1>
<name1>, clocked by <clock-name1>, source <type2> <name2>,
clocked by <clock-name2>, by method: <method> [Total Sources:
<count1> (Number of source domains: <count2>)]
```

The arguments of the above message are explained below:

Argument	Description
<type1>	flip-flop, latch, black box, or primary output
<name1>	By default, destination net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, destination instance name is reported for netlist designs. To report the hierarchical pin name of the destination instance, set the *report_instance_pin* parameter to yes.
<clock-name1>	One of the clock that is reaching the destination
<type2>	flip-flop, latch, black box, or primary output
<name2>	By default, source net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, source instance name is reported for netlist designs. To report the hierarchical pin name of the source instance, set the *report_instance_pin* parameter to yes.
<clock-name2>	One of the clock that is reaching the source
<method>	Synchronization method
<count1>	Total number of sources involved in a crossing. Each bus group is counted as a single source.
<count2>	Total number of source domains involved in the crossing

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule that reports asynchronous clock domain crossings for vector signals for which all sources are synchronized.

To view details of such crossings to check if everything is as intended, see the following:

- *Rule-Based Spreadsheet*
- *Message-Based Spreadsheet*
- Schematic. See *Viewing Schematic Through the Spreadsheet*

If you do not want to view the message of this rule, waive or disable this rule.

Example Code and/or Schematic

Consider the following rule-based spreadsheet generated for the *Ac_sync02* rule:

A	B	C	D	E	F	G	H
ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	METHOD	TOTAL SOURCES	TOTAL SOURCE D
10	top.s1	top.clk1	top.out_bus[3:0]	top.clk2	Recirculation flop	4	1

FIGURE 131. Rule-Based Spreadsheet of the *Ac_sync02* Rule

In the above spreadsheet, total number of sources in the crossing is displayed as 4. However, the *SOURCES* column of this spreadsheet displays only one source.

To view details of all sources in the crossing reported in the above spreadsheet, click the link in the *ID* column of the above spreadsheet. When you click this link, the message-based spreadsheet appears showing details of all sources and the qualifier synchronizing these sources. The following figure shows this spreadsheet:

A	B	C	D	E	F
Schematic	Type	Signal Name	Synchronization Scheme	Clock Names	al Clock Doma
10	Destination flop	top.out_bus[3:0]	Recirculation flop	top.clk2	1
10	Source flop	top.s1	Recirculation flop	top.clk1	0
11	Source flop	top.s2	Recirculation flop	top.clk1	0
12	Source flop	top.s3	Recirculation flop	top.clk1	0
13	Source flop	top.s4	Recirculation flop	top.clk1	0
13	Qualifier (detected)	top.d5	Conventional multi-flop for metastability technique	top.clk2	1

FIGURE 132. Message-Based Spreadsheet of the Ac_sync02 Rule

To view the schematic of a crossing containing a particular source, click in the *Schematic* column of a row corresponding to that source, and then click in the toolbar of the spreadsheet window.

For example, the following figure shows the schematic of the synchronized crossing containing the `top.s2` source:

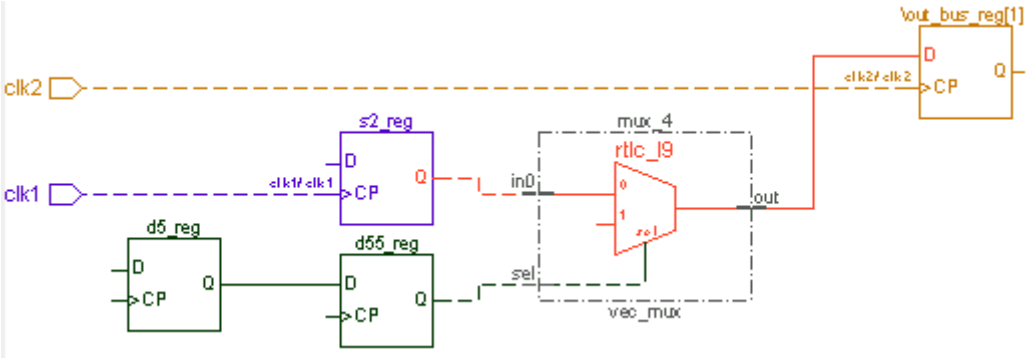


FIGURE 133. Schematic of the Ac_sync02 Rule Violation

Schematic Details

The *Ac_sync02* rule highlights the following information in different colors

in the schematic:

- Source clock and the source instance or port
- Destination clock and the destination instance or port
- Crossing path including combination logic
- Qualifier signal, multi-flop synchronizer, or any other valid synchronizer

Default Severity Label

Info

Reports and Related Files

- Ac_sync02.csv: This is a *Rule-Based Spreadsheet* that contains details of all violations of this rule.
- ac_sync_<unique-number>.csv: This is a *Message-Based Spreadsheet* that contains details of a particular violation of this rule.
- *The Ac_sync_group_detail Report*
- *The Ac_sync_qualifier Report*
- *The CrossingInfo Report*
- *The SynchInfo Report*
- *The Clock-Reset-Summary Report*
- *The Clock-Reset-Detail Report*

Ac_coherency06

Reports signals that are synchronized multiple times in the same clock domain

When to Use

Use this rule to detect signals that are synchronized multiple times in the same clock domain.

Description

The *Ac_coherency06* rule reports signals that are synchronized multiple times in the same clock domain by using any of the following synchronization schemes:

- *Conventional Multi-Flop Synchronization Scheme*
- *Synchronizing Cell Synchronization Scheme*
- *Qualifier Synchronization Scheme Using qualifier -crossing*

The following figure shows the rule-violating scenario:

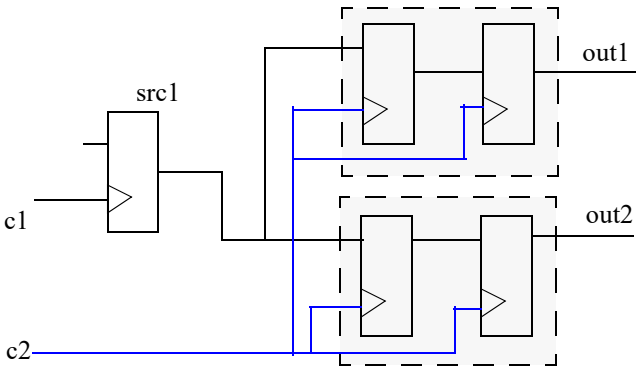


FIGURE 134. Scenario for the Ac_coherency06 Rule Violation

In the above figure, `src1` is synchronized multiple times in the same domain by using the *Conventional Multi-Flop Synchronization Scheme*.

Rule Features

Following are the features of the *Ac_coherency06* rule:

- It reports one violation per source per destination domain.
- It uses synchronization information from the *Ac_sync01* and *Ac_sync02* rules.

Rule Exceptions

The *Ac_coherency06* rule does not report a violation in the following cases:

- If a source is quasi-static.
- If a source is synchronized by using enable based synchronization schemes (that is, schemes other than the schemes mentioned in *Description*).
- If at one place, a source is synchronized by using control synchronization schemes (the schemes mentioned in *Description*) and at another place, it is synchronized by using an enable based synchronization scheme (that is, a scheme other than the schemes mentioned in *Description*).

  Such scenario is shown in the following figure:

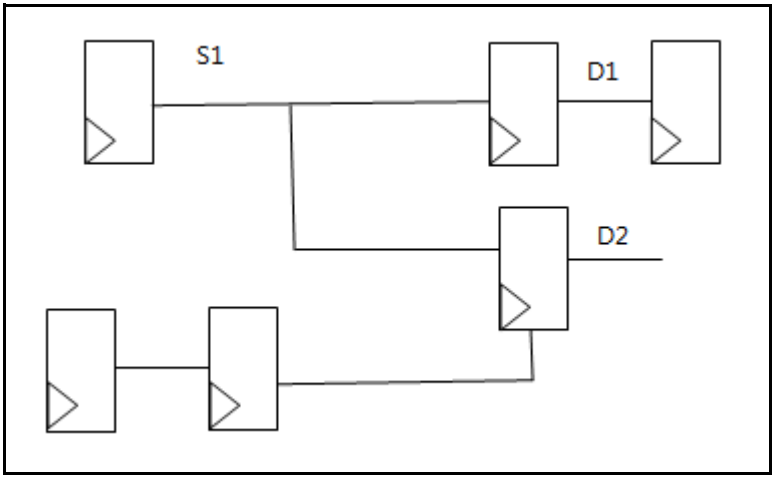


FIGURE 135. Scenario for the Ac_coherency06 Rule Exception

In the above figure, the `S1` source is synchronized by using *Conventional Multi-Flop Synchronization Scheme* at `D1`. However, it is synchronized by using enable-based synchronization at `D2`.

- If a source goes to multiple destinations within the same synchronizing cell.

  Such scenario is shown in the following figure:

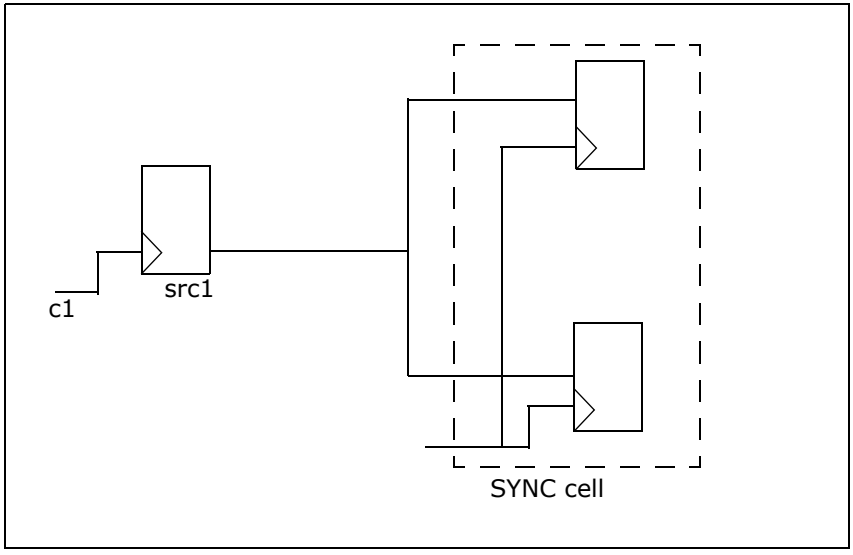


FIGURE 136. Scenario for the Ac_coherency06 Rule Exception

Parameter(s)

- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *num_flops*: Default value is 2. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock propagates from an input pin of a sequential library cell if a

combinational timing arc is specified from that pin to any output pin of the cell.

- *ignore_num_rtl_buf_invs*: Default value is `many`. Set this parameter to `one` to allow one buffer and inverter. Other possible values are `two` and `none`.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *allow_half_sync*: Default value is `yes`. Set this parameter to `no` to not treat half synchronizers as valid synchronizers.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *allow_enabled_multiflop*: Default value is `no`. Set this parameter to `yes` to consider enabled flip-flops as destination or synchronizer flip-flops in conventional multi-flop synchronization scheme. Other possible value is `same_enable`.
- *report_instance_pin*: Default value is `no`. Set this parameter to `yes` to report the name of instance pin of a netlist design. Other possible values are `flop`, `latch`, `bbox`, `seqCell`, and `all`.

- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *coherency_check_type*: Default value is `control`. Set this parameter to `reset` to check coherency issues of reset path signals only.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *allow_combo_logic* (Optional): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.
- *clock* (Optional): Use this constraint to specify clock signals.
- *ip_block* (Optional): Use this constraint to specify IP blocks in your design.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *sync_cell*: (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears for a source *<sig-name>* that is synchronized at two places in the same clock domain:

[AcCoh6_1] [WARNING] Source <type> '<source-name>' is synchronized <destination-count> (at '<destination-list>') in the same destination domain

The arguments of the above message are explained below:

Argument	Description
<type>	Refers to a source type. It can be `flop`, `latch`, `library-cell`, `port`, or `black-box`.
<source-name>	Refers to the name of the source signal. By default, source net name is reported. However, if the *report_inst_for_netlist* parameter is set to `yes`, source instance name is reported for netlist designs. To report the hierarchical pin name of the source instance, set the *report_instance_pin* parameter to yes.
<destination-count>	Refers to the count of total destinations sampling the reported source. Each bus bit is counted separately.
<destination-list>	Refers to the name of destinations sampling the source.

Potential Issues

This violation appears if a source is synchronized multiple times in the same domain.

Consequences of Not Fixing

Not fixing this violation may result in:

- Data coherency issues if synchronized destinations driven by the same source converge.
- A higher gate count as this is an excessive synchronization.

Such designs are not reusable as there is a great risk of chip failure due to

potential convergence problem.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the rule-based spreadsheet.

   To open this spreadsheet, right-click on the rule-name header in the *Results* pane, and select the *Spreadsheet Viewer* option from the shortcut menu.

   For details on this spreadsheet, see *Rule-based spreadsheet - Ac_coherency06.csv*.
2. Click on a violation in the rule-based spreadsheet to open the message-based spreadsheet attached to each source. For details on this spreadsheet, see *Message-based spreadsheet - Ac_coherency06_<num>.csv*.
3. Start debugging domain-wise using the schematic that will show the source being synchronized at multiple destinations of the same domain.

   Alternatively, select the path of a particular destination and you can see the source getting synchronized to particular destinations selectively.
4. To fix the violation, synchronize the signal once and then distribute it.

Example Code and/or Schematic

Consider the following rule-based spreadsheet of the *Ac_coherency06* rule:

A	B	C	D	E	F
ID	SOURCE	DEST. COUNT	DEST. DOMAIN	DEST. CLOCK TAG	WAIVED
8	test.t	2	1	MEMCLK	No

FIGURE 137. Rule-Based Spreadsheet of the Ac_coherency06 Rule

In the above spreadsheet, click on the link in the *ID* column to view the message-based spreadsheet of this violation.

The following figure shows the message-based spreadsheet in this case:

A Schematic	B Type	C Signal Name	D Clock Names	E Tag Names	G Synchronization Method
1	Source flop	test.t	test.clk1	SYSCLK	-
2	Destination flop	test.dest	test.clk2	MEMCLK	Qualifier defined on destination
3	Destination flop	test.a_synccell.out	test.clk2	MEMCLK	Synchronizing Cell

FIGURE 138. Message-Based Spreadsheet of the Ac_coherency06 Rule

To view the source flip-flop and its clock path in the schematic, click on the link in the *ID* column of the source and open the schematic. Following is the schematic in this case:

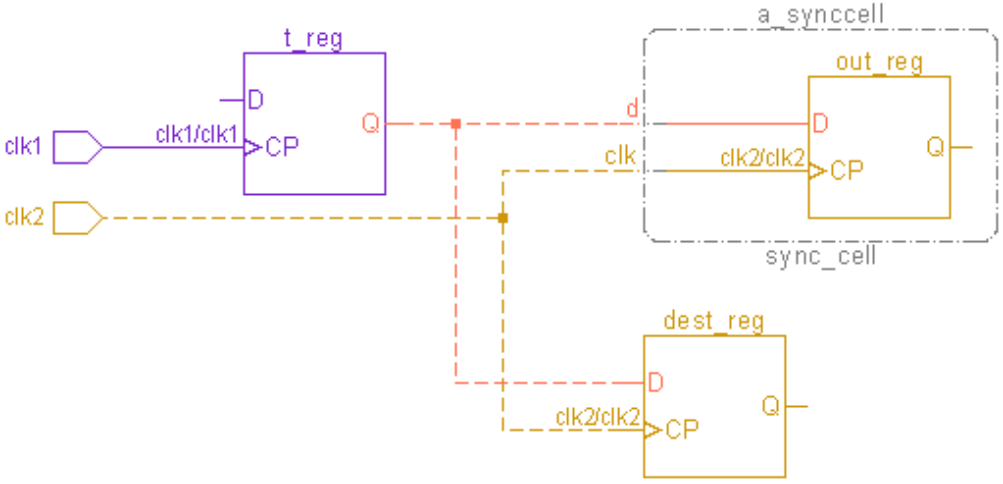


FIGURE 139. Schematic of the Ac_coherency06 Rule Violation

Schematic Details

The *Ac_coherency06* rule highlights the following details in schematic:

- Source flip-flop and its clock path
- Destination flip-flops of a one domain and their clock paths
- Crossing path
- Path of the synchronizers of the highlighted destinations

Default Severity Label

Warning

Rule Group

SYNCHRONIZATION

Reports and Related Files

- Rule-based spreadsheet - Ac_coherency06.csv

  Each row of this spreadsheet shows one violation per source per destination domain. See *Figure 137*.

  The details of columns of this spreadsheet are described below:

Column	Description
ID	Specifies a unique ID for a violation.
SOURCE	Specifies the name of the source signal that is synchronized multiple times in the same domain.
DEST.COUNT	Specifies the number of destination signals in the crossing.
DEST.DOMAIN	Specifies the domain of the destination signals.
DEST.CLOCK TAG	
WAIVED	Specifies if the reported violation is waived

- Message-based spreadsheet - *Ac_coherency06_<num>.csv*

  This spreadsheet shows details of destinations of the same domain (one destination per row). See *Figure 138*.

  The details of columns of this spreadsheet are described below:

Column	Description
Schematic	Shows a link to the schematic
Type	Specifies the type of source or destination in the crossing.
Signal Name	Specifies the name of source or destination signal in the crossing.
Clock Names	Specifies the name of the clock domain.
Tag Names	Specifies the clock tag name.
Internal Clock Domain Tag	Specifies a unique tag number generated for a clock net connected to a sequential element or a black box. For details, see Using the *Using the Clock Domain Tag*.
Synchronization Method	Specifies the scheme used to synchronize the crossing.

Ac_repeater01

Reports invalid repeater insertion in a design

When to Use

Use this rule to check if *Repeaters* are correctly inserted in a design.

Prerequisites

Specify *Repeaters* in the design by using the *repeater* constraint.

Description

The details of the *Ac_repeater01* rule are covered under the following topics:

- *Reason for the Ac_repeater01 Rule Violation*
- *Cases of Invalid Repeaters*
- *Rule Exceptions*

Reason for the Ac_repeater01 Rule Violation

The *Ac_repeater01* rule reports a violation if *Repeaters* are not correctly placed between the flip-flop paths of source and destination.

Note the following points:

- There is no limit on the number of repeater instances in the repeater chain.
- No synchronization checks are performed for repeater crossings.
- No filtering mechanism, other than waivers, is available to remove the *Ac_repeater01* rule violation.

Cases of Invalid Repeaters

Consider the following figure:

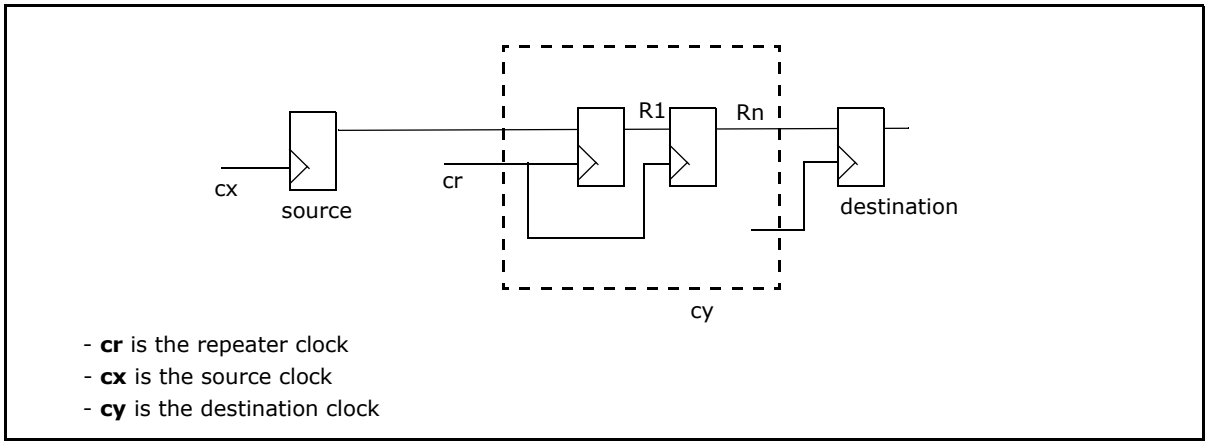


FIGURE 140. Example of Repeaters in a Design

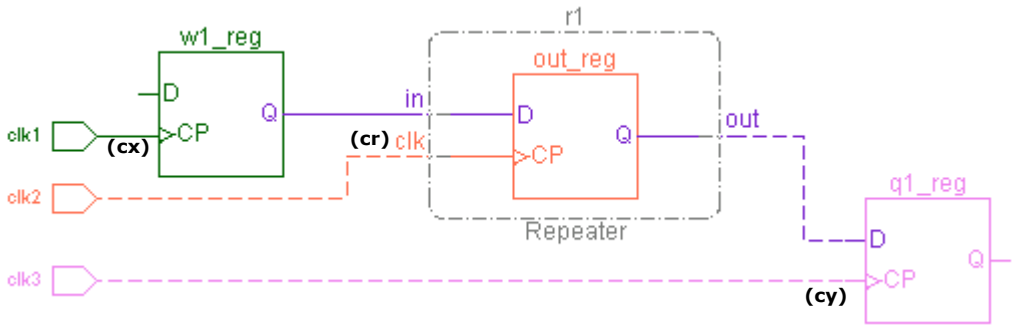
In the above figure, the repeater insertion is considered as invalid if any of the following conditions hold true:

- *Repeater Clock Coming From a Different Top-Level Source Clock*
- *Repeater Clock is Asynchronous To Source and/or Destination Clock*
- *Same Top-Level Source Clock for Source, Destination, and Repeater Clock*
- *Chain of Repeater Flip-Flops Clocked by Different Clocks*

Repeater Clock Coming From a Different Top-Level Source Clock

If `cr` is coming from a different top-level source clock other than the top-level source clocks of both `cx` and `cy`, but their domains are same, repeater insertion is considered as invalid.

This scenario is shown in the following figure:




```
// SGDC file
current_design top
clock -name clk1 -domain A
clock -name clk2 -domain A
clock -name clk3 -domain A
repeater -names "Repea*"
```

FIGURE 141.

Repeater Clock is Asynchronous To Source and/or Destination Clock

If `cr` is asynchronous to `cx` or `cy` or both, repeater insertion is considered as invalid.

The following figure shows the scenario in which the clock of the repeater chain (`r1` and `r2`) is asynchronous to both source and destination flip-flops:

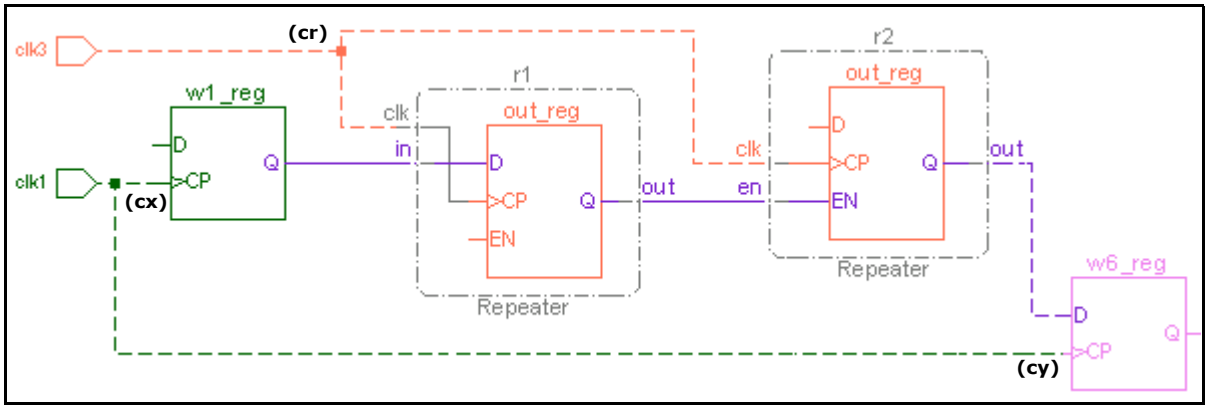
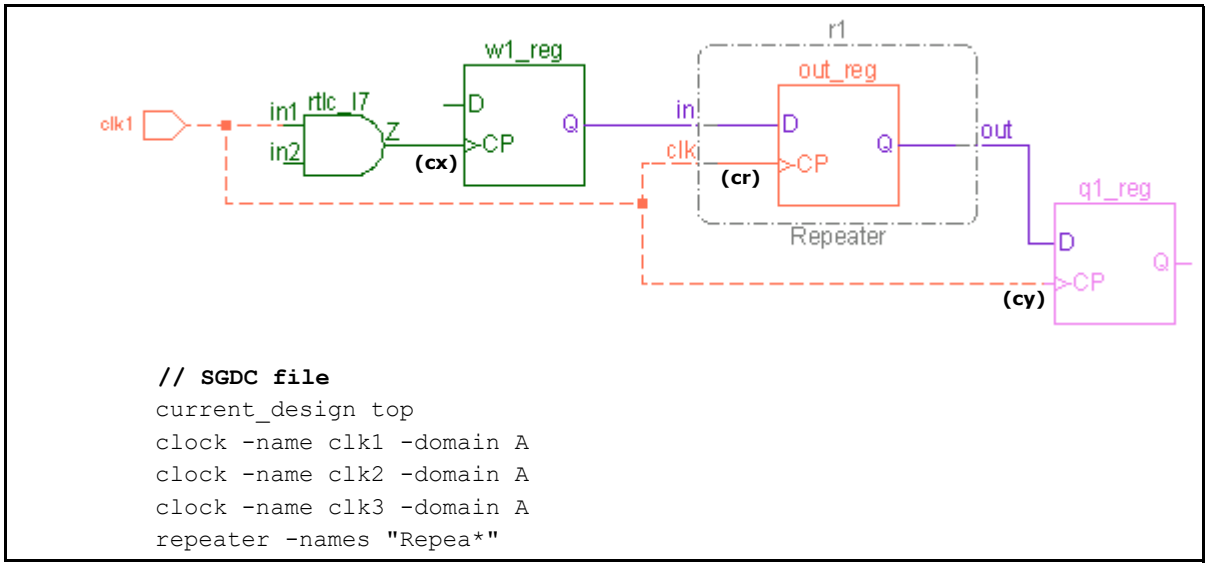


FIGURE 142.

Same Top-Level Source Clock for Source, Destination, and Repeater Clock

If `cx`, `cy`, and `cr` have the same top-level source clock but these clocks use different clock gating or divider logic, repeater insertion is considered as invalid.

For example, in the following figure, different clock logic is used between repeater and source:




```
// SGDC file
current_design top
clock -name clk1 -domain A
clock -name clk2 -domain A
clock -name clk3 -domain A
repeater -names "Repea*"
```

FIGURE 143.

Chain of Repeater Flip-Flops Clocked by Different Clocks

If the chain of repeater flip-flops is clocked by different clocks (multiple `cr` in this case), repeater insertion is considered as invalid.

This scenario is shown in the following figure:

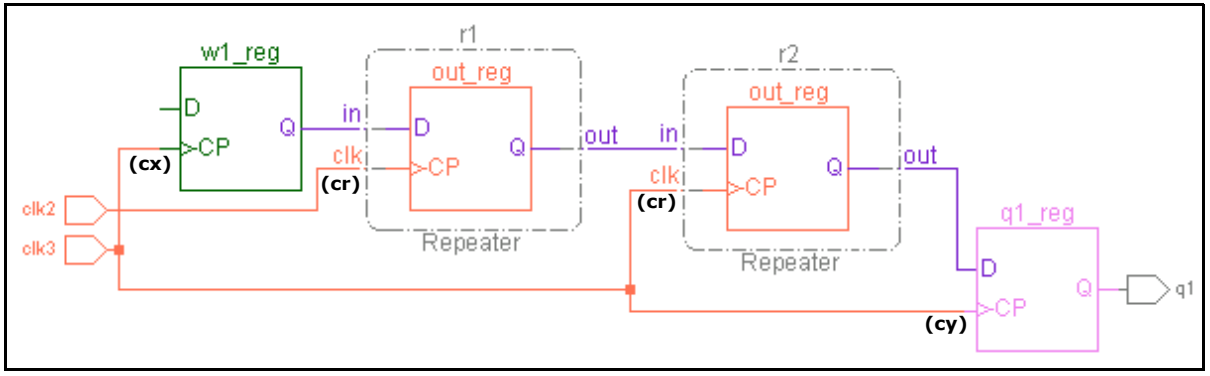


FIGURE 144. Repeaters clocked by different clocks

In the above figure, the `r1` and `r2` repeaters are clocked by different clock domains.

- If the repeater type is different in the repeater chain. That is, if repeater instances are in a different hierarchy or their parent modules are different.

  See *Example Code and/or Schematic*.

- If combinational logic is present between the source flip-flop and repeater chain, destination flip-flop and repeater chain, or between repeaters in the repeater chain.

  The following figure shows the scenario in which combinational logic is present between the destination flip-flop and repeater chain:

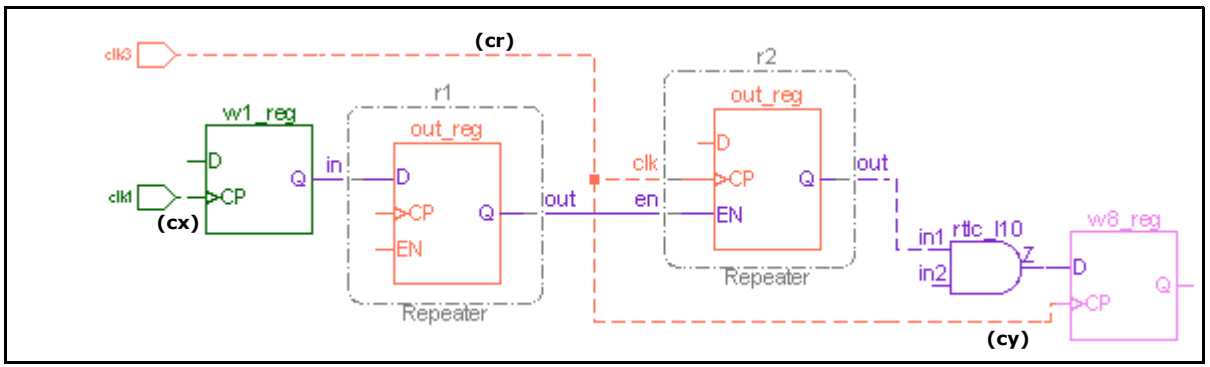


FIGURE 145.

Rule Exceptions

The *Ac_repeater01* rule does not report any violation if the repeater chain ends up on a top-level output port (specified by the *abstract_port*/*output* constraints), black box, or a hanging net.

Parameter(s)

- *allow_combo_logic_repeater*: Default value is `no`. Set this parameter to `yes` to allow combinational logic between source/destination and *Repeaters*.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.

Constraint(s)

repeater (Mandatory): Use this constraint to specify *Repeaters* in a design.

Messages and Suggested Fix

Message 1

The following message appears in case of an invalid repeater insertion:

```
[ERROR] Invalid repeater insertion: destination <dest-type>
<dest-name>, clocked by '<dest-clock>'. Reason: '<reason>'
[Total Sources: '<total-sources>']
```

The arguments of the above message are explained below:

Argument	Description
<dest-type>	Specifies the type of destination. The types can be *flop*, *latch*, or *library-cell*.
<dest-name>	Specifies the name of the output net of the hierarchical source. In case of an unknown library cell, this column shows the input net name. When the *report_inst_for_netlist* parameter is set to *yes*, it shows hierarchical pin name.
<dest-clock>	Specifies the hierarchical net/pin name of the destination clock.
<reason>	Specifies the reason for incorrect repeater insertion. See *Table 1*.
<total-sources>	Specifies the total number of source signals reaching the destination.

The following table describes different failure reasons reported by the *Ac_repeater01* rule:

TABLE 1 Failure Reasons Reported by the Ac_repeater01 rule

S. No.	Failure Reason	Cause for the failure reason (in reference with *Figure 140*)
1	Different synchronous clock used for repeater	If **cr** is coming from a different top-level source other than the top-level sources of both **cx** and **cy**, but their domains are same. See *Figure 141*.
2	Asynchronous clock used between repeater and destination	If **cr** is asynchronous to **cy**.
3	Asynchronous clock used between repeater and source	If **cr** is asynchronous to **cx**.
4	Repeater clock is asynchronous to both source and destination	If **cr** is asynchronous to both **cx** and **cy**. See *Figure 142*.
5	Different clock logic used between repeater and destination	If **cx**, **cy**, and **cr** have the same top-level source but **cy** and **cr** use different: • Clock-gating schemes (one of the scheme does not have gating or they have different CGC cells). • Clock dividers. • Clock net ignoring buffers and inverters.
6	Different clock logic used between repeater and source	Same as above. See *Figure 143*.
7	Repeaters with different clocks	If the chain of repeater slip-flops is clocked by different clocks (multiple **cr**). See *Figure 144*.

TABLE 1 Failure Reasons Reported by the Ac_repeater01 rule

S. No.	Failure Reason	Cause for the failure reason (in reference with *Figure 140*)
8	Different type of repeaters used	If the repeater type is different in the chain of repeater flip-flops. See *Example Code and/or Schematic*.
9	Combo logic present	If combinational logic is present between: • Source and repeater. • Repeater and destination. • Repeaters in the chain of repeater flip-flops. See *Figure 145*.

Potential Issues

This violation appears when the design contains invalid *Repeaters*. For details, see *Cases of Invalid Repeaters*.

Consequences of Not Fixing

If you do not fix this violation, the design may not meet the desired timing requirements.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. Open the *Rule-Based Spreadsheet* of this rule by right-clicking on the rule summary in the *Message* pane and selecting the *Open Spreadsheet* option.
2. From this spreadsheet, click on the violation to debug.

   This displays the *Message-Based Spreadsheet* showing details of the violation selected in the *Rule-Based Spreadsheet*.
3. Analyze the failure reasons shown in the *Message-Based Spreadsheet* and check the *Incremental Schematic* of the violation to see the cause of failure.

4. Based on the failure reason, perform appropriate actions described below:
   - ❒ For the failure reasons from 1 to 7 shown in *Table 1*, synchronize the clocks of source-repeater-destination chain from the same top-level clock.
   - ❒ For the failure reason 8 shown in *Table 1*, use repeater instantiation from the same repeater module.
   - ❒ For the failure reason 8 shown in *Table 1*, use the *allow_combo_logic_repeater* parameter to allow combinational logic between source/destination and repeater instances.

     If combinational logic is present between the repeater chain, modify the design to remove such logic between the repeater chain.

Message 2

The following informational message appears when the repeater insertion is valid:

```
[INFO] Valid repeater insertion: destination <dest-type> <dest-name>, clocked by '<dest-clock>' [Total Sources: '<total-sources>']
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Example 1 - Invalid Repeater Insertion

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(input d,clk1,clk2,clk3,output q1,q2);
 reg w1, w2, w3,w4, q1,q2;
 always@(posedge clk3)
     w1<=d;
 Repeater1 r1(w1, clk3, w2);
 Repeater2 r2(w2, clk3, w3);
 always@(posedge clk3)
     q1<=w3;
 endmodule

 module Repeater1(input in, clk, output reg out);
 always@(posedge clk)
     out<=in;
 endmodule

 module Repeater2(input in, clk, output reg out);
 always@(posedge clk)
     out<=in;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1
clock -name clk2
clock -name clk3
repeater -names "Repea*"
```

For the above example, the *Ac_repeater01* rule reports a violation indicating different repeater types used.

The following figure shows the rule-based spreadsheet that appears when you right-click on the rule summary in the *Message* pane and select the *Open Spreadsheet* option:

A	B	C	D	E	F	G
ID	TYPE	DESTINATION	DESTINATION CLOCK(s)	REASON	TOTAL SOURCES	WAIVED
9	flop	top.q1	top.clk3	Different type of repeaters used	1	No

FIGURE 146.

The above spreadsheet shows the summary of violations of the Ac_repeater01 rule. To know details of the above violation, click 9 in the ID column to open the message-based spreadsheet, as shown in the following figure:

A	B	C	D	E	F
ID	Signal Type	Signal Name	Clocks	Repeater Names (Repeater clocks)	Failure Reason
1	Destination flop	'top.q1'	top.clk3	-	-
1	Source flop	'top.w1'	top.clk3	top.w3 (top.clk3) top.w2 (top.clk3)	Different type of repeaters used

FIGURE 147.

The following figure shows the schematic of the above violation that indicates the usage of different repeater types:

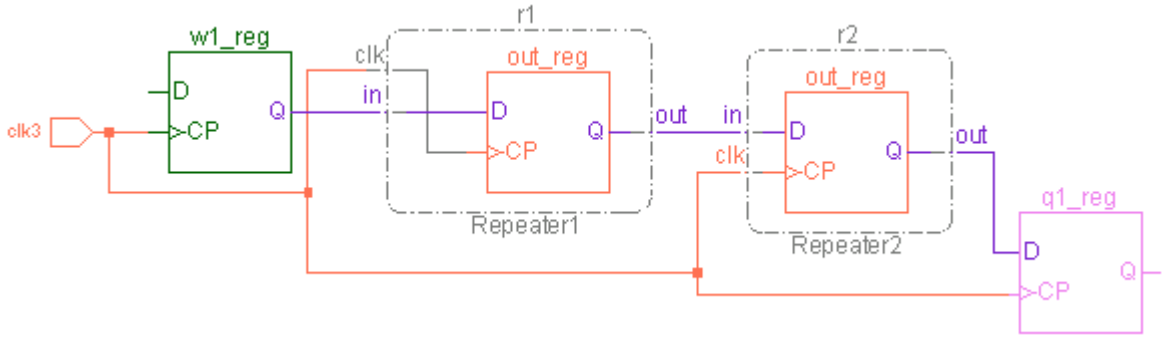


FIGURE 148.

To fix this violation, use instantiation from the same repeater module for the repeater chain. For details, see *Example 2 - Valid Repeater Insertion*.

Example 2 - Valid Repeater Insertion

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(input d,clk1,clk2,clk3,output q1,q2);
 reg w1, w2, w3,w4, q1,q2;
 always@(posedge clk3)
     w1<=d;
 Repeater1 r1(w1, clk3, w2);
 Repeater1 r2(w2, clk3, w3);
 always@(posedge clk3)
     q1<=w3;
endmodule

module Repeater1(input in, clk, output reg out);
 always@(posedge clk)
     out<=in;
endmodule

module Repeater2(input in, clk, output reg out);
 always@(posedge clk)
     out<=in;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1
clock -name clk2
clock -name clk3
repeater -names "Repe*"
```

For the above example, the *Ac_repeater01* rule reports an informational message indicating valid repeater insertion in a design.

The following figure shows the *Message-Based Spreadsheet* generated in this case:

A	B	C	D	E	F
ID	Signal Type	Signal Name	Clocks	Repeater Names (Repeater clocks)	Failure Reason
1	Destination flop	'top.q1'	top.clk3	-	-
1	Source flop	'top.w1'	top.clk3	top.w2 (top.clk3) top.w3 (top.clk3)	-

FIGURE 149.

In the above spreadsheet, no failure reason is reported as this is the case of a valid repeater insertion. The following figure shows the schematic in this case:

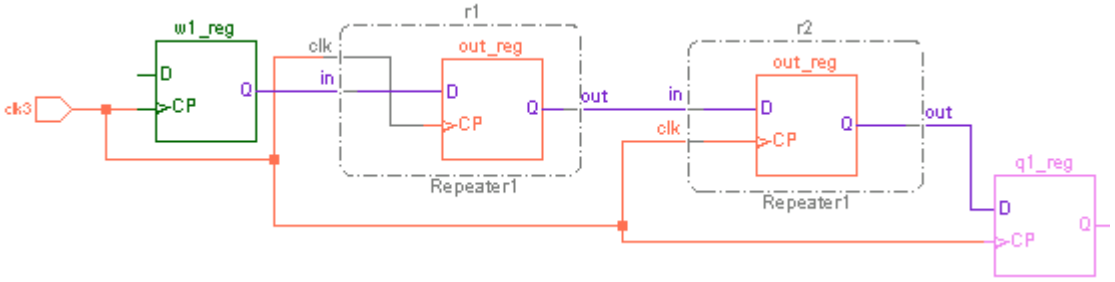


FIGURE 150. Valid repeater insertion

Default Severity Label

Error

Report and/or Related Files

The *Ac_repeater01* rule generates the *Rule-Based Spreadsheet*, *Message-Based Spreadsheet*, and *Rule-Based CSV*.

Rule-Based Spreadsheet

This spreadsheet shows the summary of all the violations reported by the *Ac_repeater01* rule. The following figure shows the example of the rule-based spreadsheet:

A	B	C	D	E	F	G
ID	TYPE	DESTINATION	DESTINATION CLOCK(s)	REASON	TOTAL SOURCES	WAIVED
17	flop	top.w6	top.clk1	Repeater clock is asynchronous to both source and destination	5	No
1C	flop	top.w7	top.clk2	Repeater clock is asynchronous to both source and destination	5	No
21	flop	top.w8	top.clk3	Combo logic present	5	No
25	flop	top.w10	top.clk1	Repeater clock is asynchronous to both source and destination	4	No
29	flop	top.w11	top.clk3	Combo logic present	4	No

FIGURE 151.

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
TYPE	Specifies the type of destination. The types can be *flop*, latch, or *library-cell*.
DESTINATION	Specifies the name of the output net of the hierarchical source. In case of an unknown library cell, this column shows the input net name. When the *report_inst_for_netlist* parameter is set to *yes*, it shows hierarchical pin name.
DESTINATION CLOCK(S)	Specifies the hierarchical net/pin name of the destination clock.
REASON	Specifies the reason for incorrect repeater insertion.
TOTAL SOURCES	Specifies the total number of source signals reaching the destination.
WAIVED	Specifies if the reported violation is waived.

Message-Based Spreadsheet

This spreadsheet of the *Ac_repeater01* rule shows the details of the violation selected in the *Rule-Based Spreadsheet* of this rule.

The following figure shows the example of a message-based spreadsheet of the *Ac_repeater01* rule:

ID	Signal Type	Signal Name	Clocks	Repeater Names (Repeater Clocks)	Failure Reason
1	**Destination flop**	**'top.w6'**	**top.clk1**	-	-
1	Source flop	'top.w1'	top.clk1	top.q2 (top.clk3) top.q1 (top.clk3)	Repeater clock is asynchronous to both source and destination
2	Source flop	'top.w2'	top.clk2	top.q2 (top.clk3) top.q1 (top.clk3)	Repeater clock is asynchronous to both source and destination
3	Source flop	'top.w3'	top.clk1	top.q2 (top.clk3)	Repeater clock is asynchronous to both source and destination
4	Source flop	'top.w4'	top.clk2	top.q2 (top.clk3)	Repeater clock is asynchronous to both source and destination
5	Source flop	'top.w5'	top.clk3	top.q2 (top.clk3)	Asynchronous clock used between repeater and destination

FIGURE 152.

The first row in the above spreadsheet shows the destination instance and the remaining rows shows all the sources of that destination.

The following table describes the details of the columns of the above spreadsheet:

Column Name	Description
Signal Type	Specifies the type as *Destination flop* or *Source flop*.
Signal Name	Specifies the source or destination name.
Clocks	Specifies the source or destination clocks.

Column Name	Description
Repeater Name (Repeater Clocks)	Specifies the name of repeaters and their corresponding clocks.
Failure Reason	Specifies the reason for invalid repeater insertion.

Rule-Based CSV

The *Ac_repeater_rule.csv* file shows the following consolidated information

Source name, source clocks, destination name, destination clocks, repeater names (repeater clocks), and failure reason

The following example shows the contents of a rule-based CSV file:

```
ID,Source Name,Source Clocks,Destination Name,Destination
Clocks,Repeater Names (Repeater clocks),Failure Reason

1,'top.w1',"top.clk3",'top.q1',"top.clk3"," top.w3
(top.clk3)\n top.w2 (top.clk3)\n ","Different type of
repeaters used"
```

Clock_sync05

Reports primary inputs sampled by multiple clock domains

When to Use

Use this rule to check if a primary input port is sampled by multiple clock domains (MSD).

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By setting the *use_inferred_clocks* parameter to `yes` to enable auto-generation of clock signals.
- By using a combination of both the above methods.

Description

The *Clock_sync05* rule reports primary inputs that are sampled by more than one of the following elements clocked by different clock domains:

- D inputs, load enable, set/reset pins of sequential elements
- A non-clock pin of a black box
- A primary output port

This **rule skips combinational logic** between primary inputs and the above-specified elements.

Example of an MSD

The following figure shows an example of an MSD:

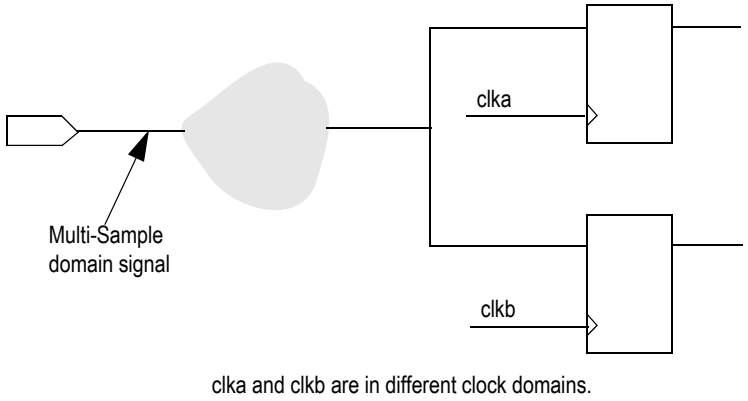


FIGURE 153. Example of MSD

NOTE: *Transparent latches (enabled latch) are also considered as combinational elements.*

This rule reports any two different domain flip-flops sampled by a port. You can expand the schematic to view other sampled flip-flops.

Rule Exceptions

The *Clock_sync05* rule does not report a violation if the primary input signal:

- Is defined by using the *input* or *abstract_port* constraint.
- Drives a quasi-static flip-flop.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.

- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *dump_detailed_info*: Default value is `none`. Set this parameter to a supported value to enable the rule to include detailed information in the generated rule/message-based spreadsheet.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

The following message appears at the location where a primary input *<port-name>* is sampled by two destination instances or ports of different clock domains:

```
[WARNING] Primary input signal '<port-name>' is sampled by
multiple clock-domains, clock '<clk1-name>' (at '<type>'
'<name1>') and clock '<clk2-name>' (at '<type>' <name2>')
```

The arguments of the above message are explained below:

Argument	Description
<port-name>	Primary input signal
<type>	flip-flop, latch, library-cell pin, black box pin, or output port

Argument	Description
<clk1-name> and *<clk2-name>*	Destination domain clocks where the input signal is being sampled
<name1> and *<name2>*	• Names of output nets of the corresponding flip-flop/ latch. <instance-name> in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is <inst-output-name>. • Names of input pin for black box and library-cells • Name of an output port

Potential Issues

This violation appears if your design contains a primary input port that drives multiple sequential elements, black box input pins, or primary output ports from different domains.

Consequences of Not Fixing

As input port belongs to a single clock domain, therefore, one of the destination instances latching the input port introduces metastability in a block/design.

If you do not fix this issue, design undergoes through metastability resulting in chip failure.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message.

The following figure illustrates a sample schematic for this rule:

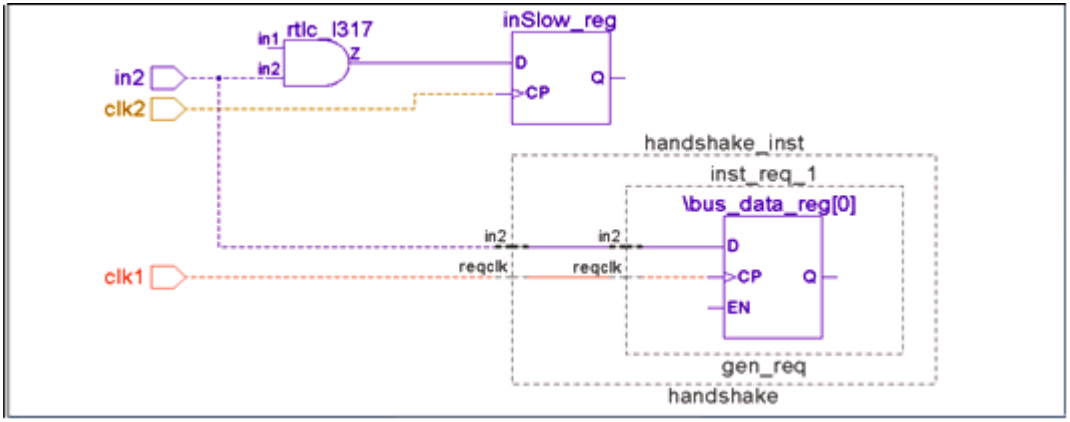


FIGURE 154. Schematic of the Clock_sync05 Rule Violation

In the schematic, you will see that the input is driving flip-flops from multiple clock domains.

To find the root cause of the problem, perform the following steps:

1. Enable *Show Case Analysis* in the schematic and check if *set_case_analysis* constraint is missing on any of the paths that should be blocked.
2. If the port is static, you may specify the *quasi_static* constraint to remove the violations.
3. If the input port does not belong to any of the domain by which it is being sampled, make sure that synchronizers are present to synchronize such input port on all the paths. Otherwise, it is better to first latch the input port in the domain it belongs to and then distribute it to synchronizers.
4. You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

The following figure illustrates a sample schematic of a violation reported by this rule:

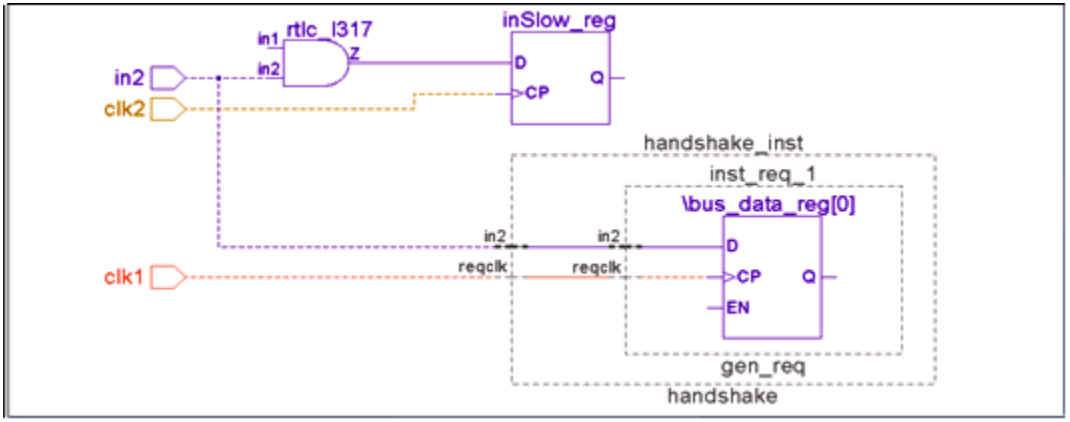


FIGURE 155. Schematic of the Clock_sync05 Rule Violation

To fix this violation, perform appropriate actions depending upon different situations, as described below:

- If `in2` belongs to the `clk1` domain, add a synchronizer of the `clk1` domain for the `inSlow_reg` flip-flop.
- If `in2` belongs to the `clk2` domain, add a synchronizer of the `clk2` domain for `\bus_data_reg[0]`.

Schematic Details

The *Clock_sync05* rule highlights the following paths in different colors in schematic:

- Path from primary input to one of its sampling flip-flops/latches for each clock domain
- Path from its clock source to the clock pin of that sampling flip-flop/latch

Default Severity Label

Warning

Rule Group

`SYNCHRONIZATION`

Reports and Related Files

No report and related file.

Ac_crossing01

Generates the crossing matrix spreadsheet

When to Use

Use this rule to view a summary of crossings per pair of clocks along with their clock domains.

Prerequisites

Specify the following information before running this rule:

- Use the `Advanced_CDC` and `adv_checker` license features.
- Enable the `Ac_sync_group` rules to get corresponding information by respective rules.
- Specify the following command in a project file to run this rule:

  ```
  set_goal_option addrules {Ac_crossing01}
  ```

  By default, this rule is switched off.

Description

The *Ac_crossing01* rule generates *The CrossingMatrix Spreadsheet* that shows a summary of crossings per pair of clocks along with their clock domains.

Parameter(s)

- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.

Messages and Suggested Fix

The following message appears to indicate that the crossing matrix spreadsheet is generated:

```
[INFO] CrossingMatrix spreadsheet generated for design
'<design-name>'
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Consider the design shown in the following modular schematic:

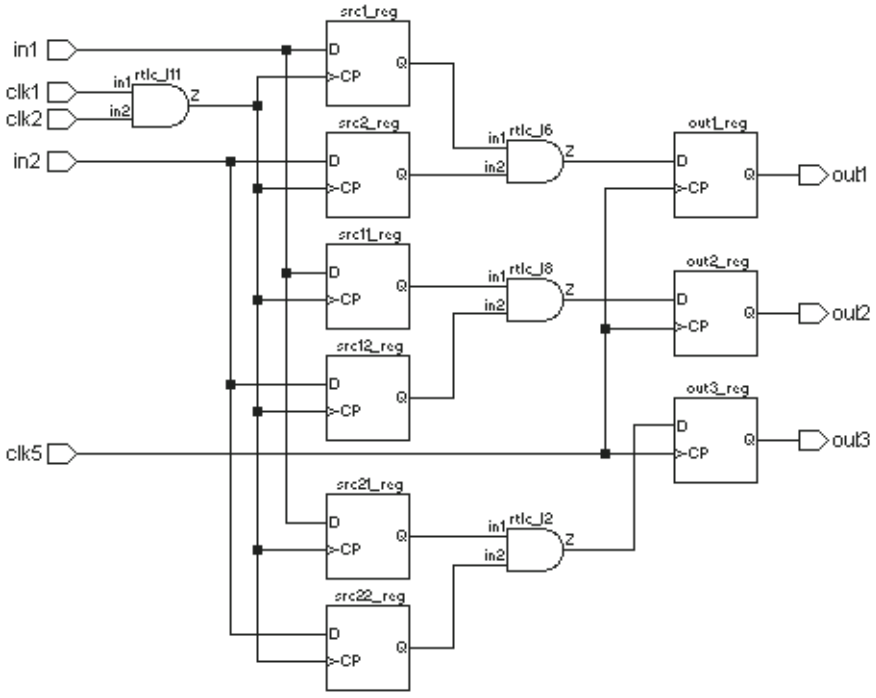


FIGURE 156. Design on which the Ac_crossing01 Rule is being Run

When you run the *Ac_crossing01* rule on the above design, the *CrossingMatrix.csv* spreadsheet is generated. Now based upon the rules enabled, such as `Ac_sync_group` rules, different clock pair information is shown in the spreadsheet.

Sample Spreadsheet generated by Ac_sync_group rules

The following figure shows the crossing matrix spreadsheet generated when the `Ac_sync_group` is enabled:

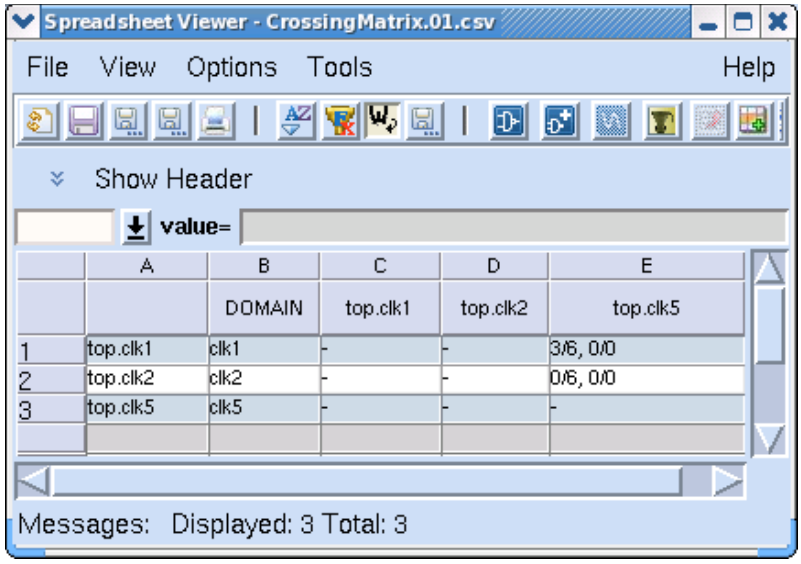


FIGURE 157. Crossing Matrix Spreadsheet

The above spreadsheet shows information per clock crossing pairs in the following format:

<a>/<b>, <c>/<d>

Where:

- *<a>* is the total number of *Ac_unsync01* and *Ac_unsync02* messages for the given clock pair.
- *<b>* is the total number of crossings for this clock pair for all the sources involved in the crossing.
- *<c>* is the total number of *Ac_sync01* and *Ac_sync02* violation messages for the given clock pair.
- *<d>* is the total number of crossings for this clock pair for all the sources involved in the crossing.

Description of the above format for each clock pair in this example is described below:

- For the clock pair `clk1-clk5`

- ❒ Three violations of the *Ac_unsync01* rule are reported in this case, and all sources are considered between the `clk1-clk5` clock pair.

  Therefore, the value of *<a>* and *<b>* is `3` and `6`, respectively.

- ❒ No *Ac_sync01*/*Ac_sync02* violation is reported in this case.

  Therefore, the value of *<c>* and *<d>* is `0`.

- For the `clk2-clk5` clock-pair

  No violation is reported for this clock pair and all sources are considered. Therefore, the value of *<a>* and *<b>* is `0` and `6`, respectively.

Default Severity Label

Info

Rule Group

ADV_CLOCKS

Reports and Related Files

CrossingMatrix.csv

This spreadsheet is generated in the spyglass_reports/clock-reset/Ac_crossing directory.

It shows clock-related details, and provides an indication of overall SpyGlass CDC solution risk in a design, crossings congestion area, and setup issues causing unusual crossing distribution.

Information per clock crossing pairs appears in the following format in the spreadsheet:

<a>/*<b>*, *<c>*/*<d>*

where,

- *<a>* is the count of *Ac_unsync01*/*Ac_unsync01* messages.
- *<b>* is the count of total unsynchronized crossings that exist for the clock crossing pair.
- *<c>* is the count of *Ac_sync01*/*Ac_sync02* messages.
- *<d>* is the count of total synchronized crossings that exist for a clock crossing pair.

NOTE: *<b> and <d> may be different from <a> and <c> in case multiple clocks are reaching to crossing instances.*

For details, see *Sample Spreadsheet generated by Ac_sync_group rules*.

Clock_sync03

Reports converging signals.

The *Clock_sync03* rule runs the *Clock_sync03a* and *Clock_sync03b* rules.

Clock_sync03b

Reports convergence of signals from different domains

NOTE: *The Clock_sync03b rule will be deprecated in a future release. The rule is not included in CDC GuideWare goals now and do not perform checks until specifically included in the user-defined goal options. In this case, the rule performs the checks and SpyGlass includes a deprecation message in both the spyglass.out and spyglass.log files.*

When to Use

Use this rule to check convergence of different domain signals.

NOTE: *It is recommended to use the Ac_conv03 rule instead of this rule.*

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By setting the *use_inferred_clocks* parameter to `yes` to enable auto-generation of clock signals
- By using a combination of both the above methods

Description

The *Clock_sync03b* rule reports convergence of signals that come from different domains.

Such signals may be:

- Synchronized in different domains by using *Conventional Multi-Flop Synchronization Scheme* or *Synchronizing Cell Synchronization Scheme*
- Unsynchronized
- Singular flip-flop outputs

In case of convergence of signals other than singular flip-flop outputs, this rule reports a violation if one of the following conditions is true:

- If signals are coming from the same source domain but different destination domains
- If signals are coming from different source domains but same destination domain

- If signals are coming from different source and destination domains

You must avoid all such convergences irrespective of whether signals are gray-encoded or not.

The following figure shows an example of signals from different domains converging together.

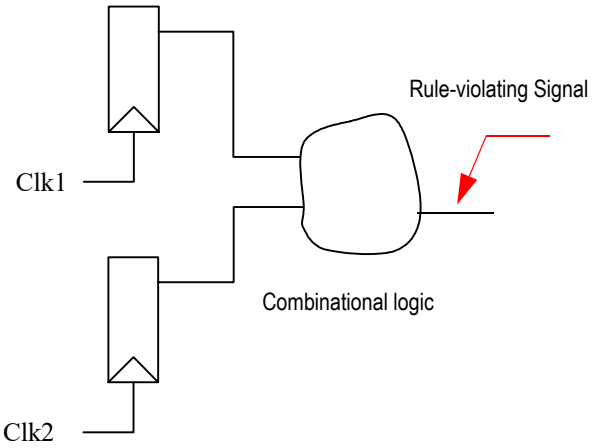


FIGURE 158. Convergence of Signals from Different Source Domain

The following figure shows an example of signals coming from two different sources that may or may not be synchronized and/or may converge after layers of sequential logic.

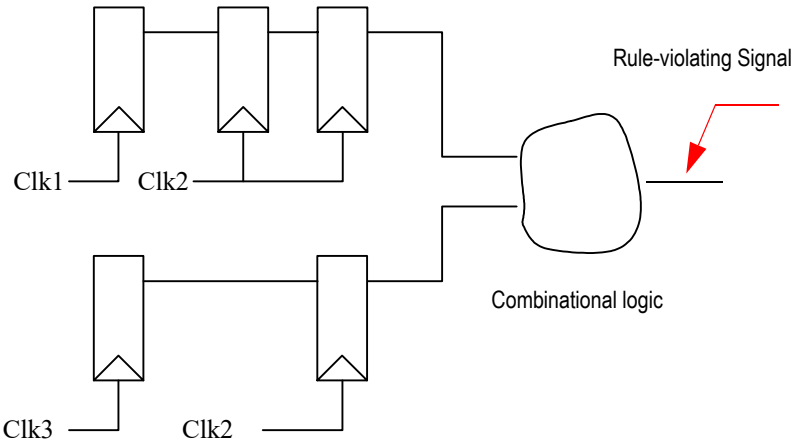


FIGURE 159. Convergence of Different Sources that may or may not be Synchronized

Guidelines for Using Parameters with the Clock_sync03b Rule

Using certain combination of parameters with this rule can make this rule run time and memory intensive. The following guidelines can help in reducing run time:

- If there are multiple convergences on the same path, then by default, this rule reports convergence on the last gate in the path that covers all the converging signals.

  The following figure shows an example of convergence of multiple signals.

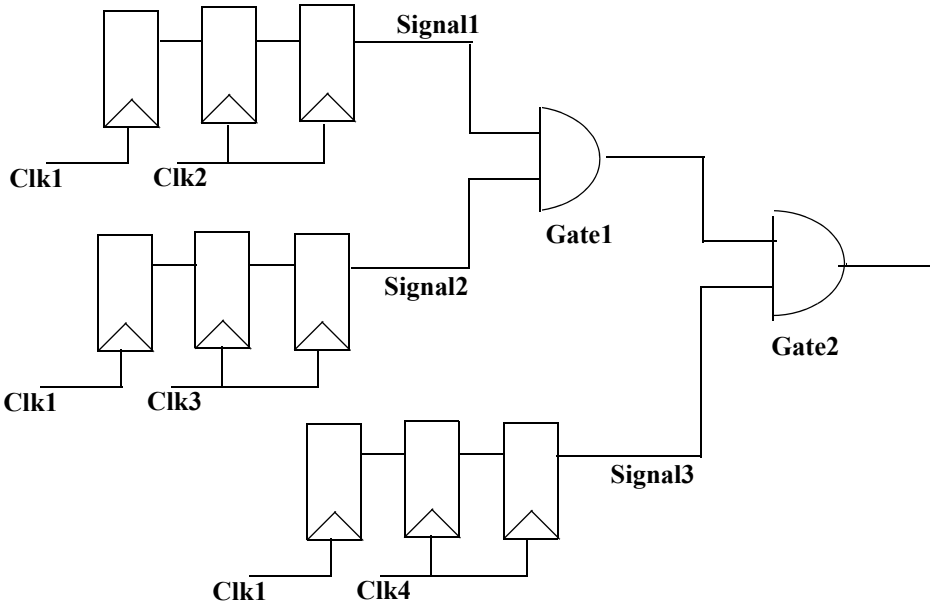


FIGURE 160. Multiple Convergences on Same Path

In the above example, by default, the *Clock_sync03b* rule reports convergence only on `Gate2`, which covers convergence of all signals: `Signal1`, `Signal2`, and `Signal3`.

However, if the *all_convergence_paths* parameter is set to `yes`, the rule will report both `Gate1` and `Gate2`, where `Signal1` and `Signal2` are reported at `Gate1`. These signals are also covered in convergence at `Gate2`. Therefore, it is recommended not to use the *all_convergence_paths* parameter as its usage may result in noise and increase run time and memory.

- Specifying a high value to the *reconvergence_stages* parameter may result in increases run time. In such case, ensure that the value of this parameter is set appropriately.
- To reduce run time, set the *show_reconv_paths* parameter to `no`. This avoids schematic data generation for convergence paths. In this case, only converging signals and the gate where they are converging is highlighted.

- Do not specify the *check_bus_bit_convergence* parameter with the *Clock_Sync03b* rule as this may result in inaccurate results.

Rule Exceptions

The *Clock_sync03b* rule has the following exceptions:

- It does not report convergences of vector signals.

  This rule reports convergence only if a signal reaches on a MUX select pin along with signals reaching one or more MUX input pins.
- It does not report convergence of synchronized signals at sequential cells.
- It does not report a violation when synchronizers are a part of the same FIFO memory.

Parameter(s)

- *reconvergence_stages*: Default value is `0`. Set this parameter to a positive integer value to specify the maximum number of flip-flops allowed in the fan-out path of a synchronizing signal.
- *no_convergence_check*: Default value is `NULL`. Specify net names that should not be checked for convergence.
- *report_conv_type*: Default value is `sync`. Set this parameter to `all` to report convergence from synchronized signals, unsynchronized signals, and standalone flip-flops. Other possible values are `sync`, `unsync`, and `nocross`.

NOTE: *This rule reports violation if an input port defined in a different domain converges with other signals.*

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a

MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *all_convergence_paths*: Default value is `no`. Set this parameter to `yes` to report all convergence paths.
- *show_reconv_paths*: Default value is `yes`. Set this parameter to `no` to highlight only converging signals and the gate where the signals are converging. This reduces runtime.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *allow_enabled_multiflop*: Default value is `no`. Set this parameter to `yes` to consider enabled flip-flops as destination or synchronizer flip-flops in conventional multi-flop synchronization scheme. Other possible value is `same_enable`.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *ip_block* (Optional): Use this constraint to specify IP blocks in your design.

- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *no_convergence_check* (Optional): Use this constraint to specify nets that should not be checked for convergence.

NOTE: *If you specify nets by using the no_convergence_check constraint as well as the no_convergence_check parameter, SpyGlass considers the nets specified by both the constraint and parameter.*

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at a location where two or more signals converge on a net *<net-name>*:

[WARNING] <obj-type> '<sig-name>' converge on '<net-name>'.

The arguments of the above message are explained below:

Argument	Description
<obj-type>	`Signals` in case of RTL designs. `Instances` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `Signals`
<sig-name>	Comma-separated list of signal names in case of RTL designs. Comma-separated list of instances in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is list of signal names
<inst-name>	*<out-net-name>* of converging instance in case of RTL designs. *<inst-name>* of converging instance in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is *<out-net-name>*

Potential Issues

This violation appears when different domain signals converge.

Consequences of Not Fixing

Convergence of synchronizers from different domains can result in data coherency issues and may cause chip failure.

How to Debug and Fix

Convergence of synchronizers from the different domains can cause data coherency. View the *Incremental Schematic* of the violation message to analyze the path of convergence of synchronizers.

Based on the information viewed in the schematic, perform appropriate actions, as described below:

- If convergence is being reported for signals that are in an IP, specify the *ip_block* constraint for the IP if you do not want to report within the IP.
- If convergence is for static signals, use the *cdc_false_path* or *quasi_static* constraints.
- If the path of synchronizers confirms that synchronizers cannot functionally control the converging net at the same time, the violation can be waived.
- If some of the intermediate nets in the path confirm exclusivity between synchronizers, you can specify such nets by using the *no_convergence_check* parameter.

To check clock domains of different source signals, right-click on the source net in the *Incremental Schematic* window and select the *Show Debug Data->Clock-reset* option from the shortcut menu. This option is enabled when you set the *enable_debug_data* parameter to `yes`.

In some cases, convergence of synchronizers from different domains may be required. In such cases, you should modify your design to ensure that there is no glitch or any coherency issue.

Example Code and/or Schematic

Consider the following schematic of a violation reported by the *Clock_sync03b* rule:

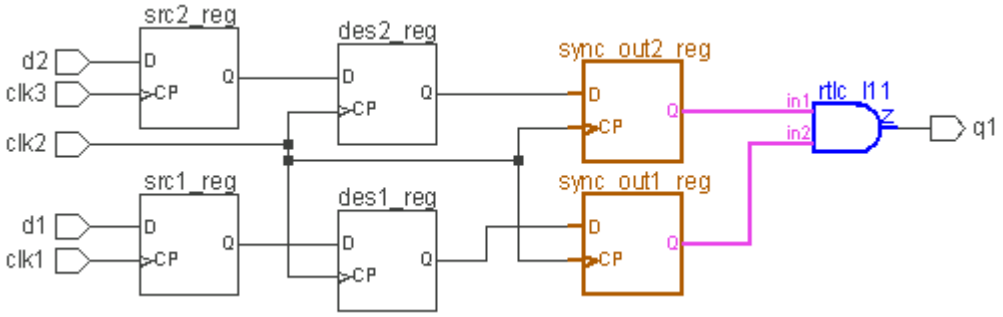


FIGURE 161. Schematic of the Clock_sync03b Rule Violation

In the above example, two different domain source signals, `src1` and `src2` that are synchronized in the same domain by using *Conventional Multi-Flop Synchronization Scheme* are converging on an AND gate.

To fix this violation, remove the convergence.

Convergence between signals coming from different domains is a serious issue. You should analyze your design carefully for such cases to check why such convergence is required, and avoid such convergences if possible.

Schematic Details

The *Clock_sync03b* rule highlights the following information in different colors in the schematic:

- Path from each destination flip-flop to the converging gate
- Instance on which convergence is taking place

If the *show_reconv_paths* parameter is set to `no`, the rule highlights only converging signals and the gate where they are converging. It does not show the complete path.

Default Severity Label

Warning

Rule Group

`SYNCHRONIZATION`

Reports and Related Files

None

Clock_sync06

Reports primary outputs driven by multiple clock domain flip-flops or latches

When to Use

Use this rule to check if a primary output port is driven by multiple clock domains.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By setting the *use_inferred_clocks* parameter to `yes` to enable auto-generation of clock signals.
- By using a combination of both the above methods.

Description

The *Clock_sync06* rule reports primary outputs that are driven by more than one of the following elements clocked by different clock domains:

- D inputs or load enable inputs of sequential elements
- A non-clock pin of a black box
- A primary output port

The following figure shows an example of MTD:

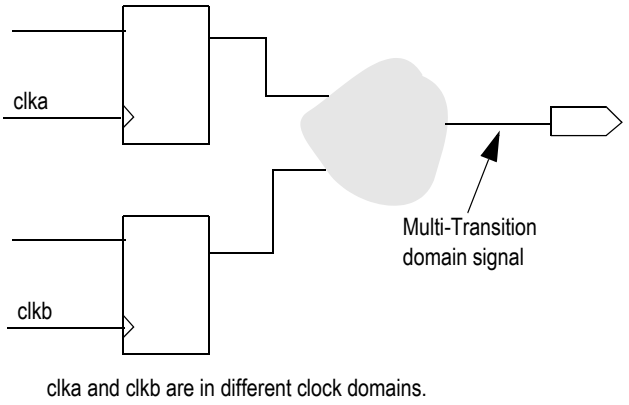


FIGURE 162. Example of MTD

This rule reports any two different domain flip-flops driving a port. You can expand the schematic to view other flip-flops driving the port.

Rule Exceptions

The *Clock_sync06* rule does not report a violation if an output port:

- Is already defined by using the *output* constraint.
- Is driven by a quasi-static flip-flop.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

- *dump_detailed_info*: Default value is `none`. Set this parameter to a supported value to enable the rule to include detailed information in the generated rule/message-based spreadsheet.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

The following message appears at the location where a primary output *<port-name>* is driven by instances of different clock domains:

```
[WARNING] Primary output signal '<port-name>' is in the fan-out cone of multiple clock-domains, clock '<clk1-name>' (at <type> '<name1>') and clock '<clk2-name>' (at <type> '<name2>')
```

The arguments of the above message are explained below:

Argument	Description
<port-name>	Primary output signal
<type>	flip-flop, latch, library-cell pin, black box pin, or input port
<clk1-name> and *<clk2-name>*	Different domain clocks driving the output signal
<name1> and *<name2>*	• Names of output nets of the corresponding flip-flop/ latch. <instance-name> in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is <inst-output-name>. • Names of output pin for black box and library-cells • Name of an input port

Potential Issues

This violation appears if your design contains a primary output port that is driven by multiple sequential elements, black box input pins, or primary output ports from different domains.

Consequences of Not Fixing

If you do not fix this violation, metastability conditions may arise in your design resulting in chip failure.

Metastability conditions arise because an output port that is fed by multiple clock domains eventually feeds a destination that is outside the block boundary. As this port is driven by multiple clock domains, the destination sampling such output port goes metastable.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message.

The following illustrates a sample schematic for this rule:

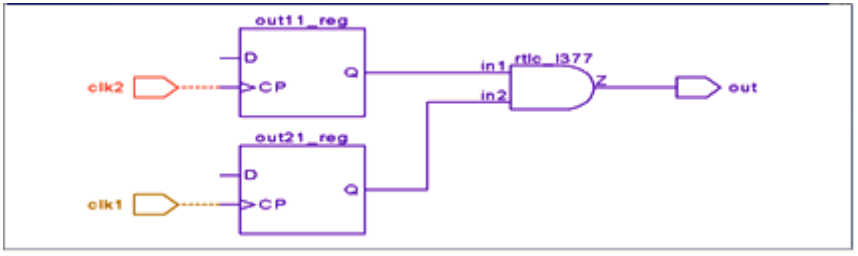


FIGURE 163. Schematic of the Clock_sync06 Rule Violation

In the schematic, you will see that the output is being driven by flip-flops from multiple clock domains.

To find the root cause of the problem, perform the following steps:

1. Enable *Show Case Analysis* in the schematic and check if *set_case_analysis* constraint is missing on any of the paths that should be blocked.
2. If the port is static, you may specify *quasi_static* constraint to remove the violations.

3. If the output port does not belong to any of the domain by which it is being fed, make sure that synchronizers are present to synchronize such asynchronous signals before they reach the output port.
4. You can also view case analysis settings along with the violation of this rule.

To fix this problem, synchronize signals from separate domains into one domain and then use that domain to drive the output port. In cases where this is not possible, that is there is a need to multiplex signals heavily onto limited pins, disable/waive this rule.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

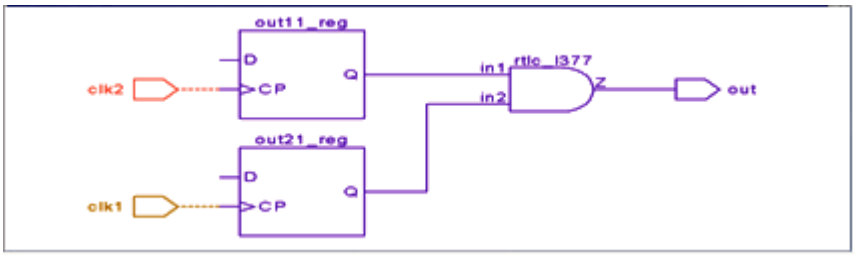


FIGURE 164. Schematic of the Clock_sync06 Rule Violation

To fix the above violation, either synchronize `out21_reg` into the `clk2` domain or synchronize `out11_reg` into the `clk1` domain before convergence at the `AND` gate.

Schematic Details

The *Clock_sync06* rule highlights the following paths in different colors in schematic:

- Path from one of its source flip-flops/latches to the primary output for each clock domain
- Path from its clock source to the clock pin of that source flip-flop/latch

Default Severity Label

Warning

Rule Group

SYNCHRONIZATION

Reports and Related Files

No report and related file

Clock_sync08a

Reports multi-flop synchronized bus-bits where double flip-flop output bits belong to the same bus

When to Use

Use this rule to review synchronization used for bus signals.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint.
- By setting the *use_inferred_clocks* parameter to `yes` to enable auto-generation of clock signals.
- By using a combination of both the above methods.

Description

The *Clock_sync08a* rule reports bus-bits that are synchronized by using the *Conventional Multi-Flop Synchronization Scheme*, and the output signals of the destination flip-flop and synchronizing flip-flops are bits of the same bus or are part of FIFO read/write pointers.

NOTE: *Please note the following:*

- *Do not separately synchronize each bit by using standard control synchronization techniques, such as Conventional Multi-Flop Synchronization Scheme.*

Parameter(s)

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *allow_enabled_multiflop*: Default value is `no`. Set this parameter to `yes` to consider enabled flip-flops as destination or synchronizer flip-flops in conventional multi-flop synchronization scheme. Other possible value is `same_enable`.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to automatically generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears when a bit of the bus *<bus-name>* is

synchronized by using a non-recommended technique:

```
[CSync8a_1] [WARNING] Bus '<bus-name>' is synchronized using
"conventional multi-flop for metastability technique"
(Destination: '<dest-name>')
```

NOTE: *For RTL designs, <dest-name> is the name of the output net of the corresponding flip-flop. For netlist designs, if the report_inst_for_netlist parameter is set to yes, <dest-name> is the name of the flip-flop instance. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains a bus that is either a FIFO pointer or it synchronized by using *Conventional Multi-Flop Synchronization Scheme* where the destination flip-flops and all the synchronizer flip-flops are a part of the same bus.

Consequences of Not Fixing

If you do not fix this violation, buses are not properly synchronized. This may result in data coherency issues.

In general, buses that are synchronized by using *Conventional Multi-Flop Synchronization Scheme*, where destination flip-flops and all the synchronizer flip-flops are a part of the same bus, are FIFO pointers. Such cases are intentional and do not result in any issue in the design.

How to Debug and Fix

To debug and fix this violation, view the incremental schematic of the violation and analyze the reported issue.

If the reported issue is intentional, waive the message. Else, use a common enable technique to transfer data or gray-code the bus.

Example Code and/or Schematic

The following figure illustrates a scenario in which this rule reports a violation:

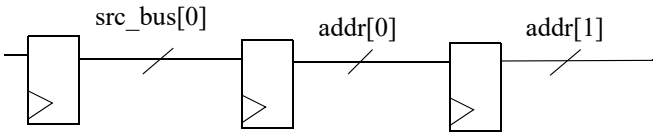


FIGURE 165. Clock_sync08a Rule Violating Scenario

In the above case, the bus-bits are synchronized by using the *Conventional Multi-Flop Synchronization Scheme*, and the output signals of the destination flip-flop and synchronizing flip-flops are bits of the same bus.

Schematic Details

The *Clock_sync08a* rule highlights the following paths in the schematic:

- Path from the source (flip-flop, black box, or primary input) to the destination (flip-flop, black box, or primary output).
- Path from its clock source to the clock pin of the source flip-flop/black box. The source flip-flop/black box will be highlighted in the same color as the source clock.
- Path from its clock source to the clock pin of the destination flip-flop/black box. The destination flip-flop/black box will be highlighted in the same color as the destination clock.
- The synchronizer (Conventional Multi-Flop) for the clock crossings found to be synchronized.

Default Severity Label

Warning

Rule Group

SYNCHRONIZATION

Reports and Related Files

Clock_sync08a.csv

Clock_sync09

Reports signals that are synchronized more than once in the same clock domain

When to Use

Use this rule to detect signals that are synchronized more than once in the same clock domain.

NOTE: *It is recommended to use the Ac_coherency06 rule instead of this rule.*

Description

The *Clock_sync09* rule reports signals at clock domain crossings that are synchronized more than once in the same clock domain by using the *Conventional Multi-Flop Synchronization Scheme* or *Qualifier Synchronization Scheme Using qualifier -crossing*.

The following figure shows such a scenario:

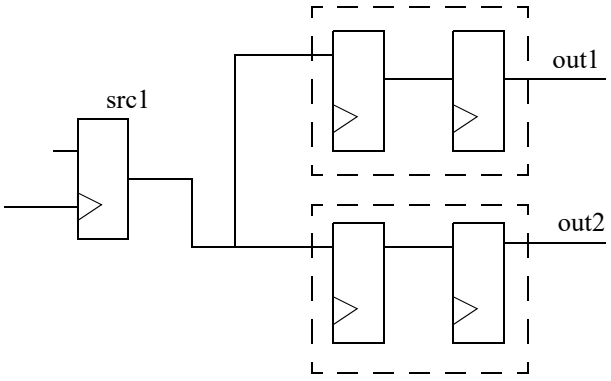


FIGURE 166. Scenario of the Clock_sync09 Rule Violation

In the above figure, `src1` is synchronized multiple times in the same domain by using the *Conventional Multi-Flop Synchronization Scheme*.

This rule reports any two destinations on which a source is synchronized in the same domain. Set the *report_all_sync* parameter to `yes` to report all destinations with respect to a source.

Rule Exceptions

The *Clock_sync09* does not a report violation in the following cases:

- When both synchronizers are a part of same FIFO memory
- When there is a common qualifier for both synchronizers of type *Synchronized Enable Synchronization Scheme*, *AND Gate Synchronization Scheme*, *Glitch Protection Cell Synchronization Scheme*, *MUX-Select Sync (Without Recirculation) Synchronization Scheme*, *Clock-Gating Cell Synchronization Scheme*
- When a common user-defined qualifier is used to synchronize for both destinations

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_all_sync*: Default value is `no`. Set this parameter to `yes` to generate a spreadsheet displaying all destinations where a source is being synchronized.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *allow_enabled_multiflop*: Default value is `no`. Set this parameter to `yes` to consider enabled flip-flops as destination or synchronizer flip-flops in conventional multi-flop synchronization scheme. Other possible value is `same_enable`.

- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears for a source *<sig-name>* that is synchronized at two places in the same clock domain:

```
[CSync9_1] [WARNING] Source <type> <obj-type> "<inst-name>" is synchronized at least twice (at <type> "<name1>" and <type> "<name2>")
```

The arguments of the above message are explained below:

Argument	Description
<type>	Specifies the synchronization element type, which can be flip-flop or black box or port
<obj-type>	`output` in case of RTL designs. `instance` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `output`
<inst-name>	*<inst-out-net-name>* in case of RTL designs. *<inst-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs
<name1> and *<name2>*	Specifies the names of the synchronization elements. **NOTE:** For RTL designs, <name1> and <name2> are names of the output nets of the corresponding flip-flops. For netlist designs, if the *report_inst_for_netlist* parameter is set to yes, these are the names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.

Potential Issues

This violation appears if a source is synchronized multiple times in the same domain.

Consequences of Not Fixing

Not fixing this violation may result in:

- Data coherency issues if synchronized destinations driven by the same source converge.
- A higher gate count as this is an excessive synchronization.

Such designs are not reusable as there is a great risk of chip failure due to potential convergence problem.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.

   The following figure illustrates a sample schematic for this rule:

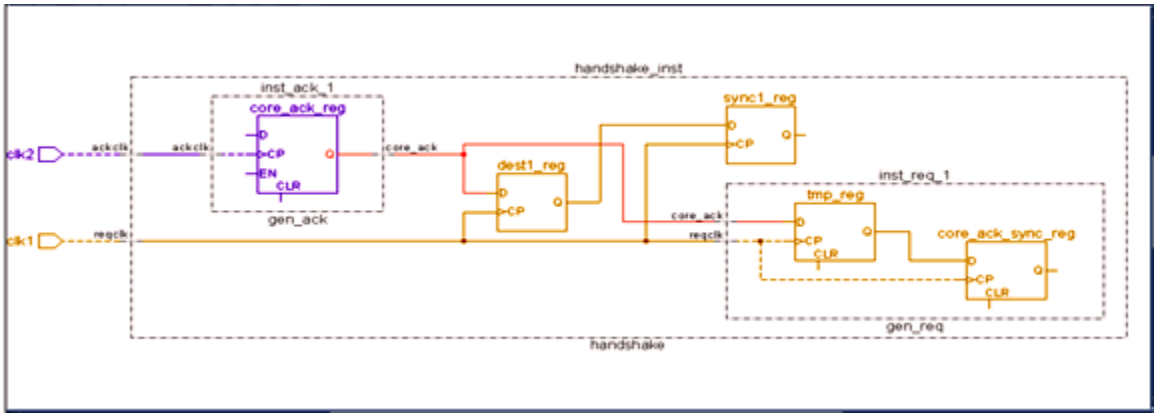


FIGURE 167. Schematic of the Clock_sync09 Rule Violation

In the schematic, you will see the same source signal feeding to two different synchronizers, where synchronizers are in the same domain.

2. If one of the synchronized lines is blocked, unused, hanging, or static, you may waive the violation by using the *cdc_reduce_pessimism* parameter, or *cdc_false_path* or *quasi_static* constraints.
3. To fix this problem, synchronize the signal once and then distribute it.

Example Code and/or Schematic

Consider the following schematic of a violation reported by the *Clock_sync09* rule:

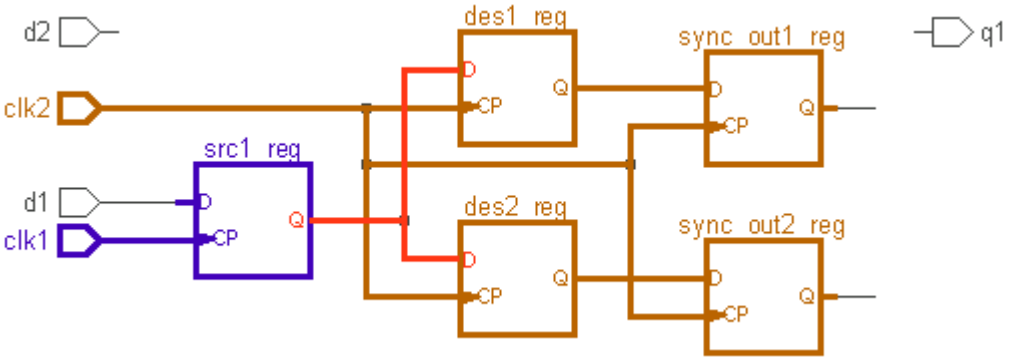


FIGURE 168. Schematic of the Clock_sync09 Rule Violation

In the above example, the `src1` source signal is getting synchronized in the same domain at multiple points, `des1` and `des2`, by using *Conventional Multi-Flop Synchronization Scheme*.

To fix this violation, synchronize the `src1` source only once and then distribute it at multiple places.

Schematic Details

The *Clock_sync09* rule highlights the following paths in schematic:

- Path from a source (flip-flop, black box, or primary input) to a destination (flip-flop, black box, or primary output). The source flip-flop/black box is highlighted in the same color as the source clock.
- Path from a clock source to a clock pin of a source flip-flop/black box. The destination flip-flop/black box is highlighted in the same color as the destination clock.
- The synchronizer for the clock crossings found to be synchronized

Default Severity Label

Warning

Rule Group

`SYNCHRONIZATION`

Reports and Related Files

No report and related file

Ac_cdc01

Checks data loss on clock domain crossings

The *Ac_cdc01* rule group runs the *Ac_cdc01a*, *Ac_cdc01b*, and *Ac_cdc01c* rules.

Ac_cdc01a

Checks data loss for multi flop or sync cell or qualifier synchronized clock domain crossings

When to Use

Use this rule to identify clock domain crossings that are prone to data loss.

Prerequisites

Following are the prerequisites:

- Use the *Advanced_CDC* and *adv_checker* license.
- Specify a clock period for at least one clock in an SGDC file.

  When you specify a clock period for at least one clock, SpyGlass assigns the default period 10 to the clocks for which no clock period is defined.

Description

The details of the *Ac_cdc01a* rule are covered under the following topics:

- *Checks Performed by the Ac_cdc01a Rule*
- *Types of Synchronized Crossings Checked by the Ac_cdc01a Rule*
- *Rule Exceptions*

Checks Performed by the Ac_cdc01a Rule

The *Ac_cdc01a* rule checks the following by default:

- Fast-to-slow crossings only, and
- Same period clocks.

To check for slow-to-fast crossings, set the *fa_verify_slow_to_fast* parameter.

Consider the following scenario in which data passes from fast-to-slow clock domain crossing:

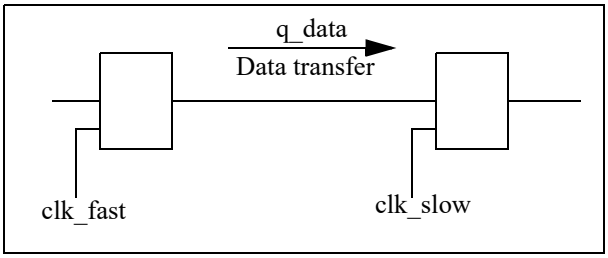


FIGURE 169. Data Passing from Fast-to-Slow Clock Domain Crossing

In the above scenario, there is a high possibility of data loss, as shown in the following figure:

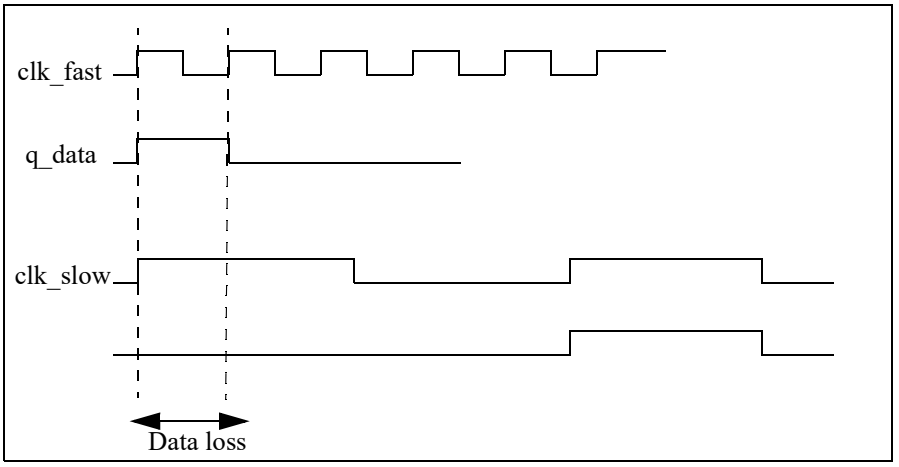


FIGURE 170. Data Loss from Fast-to-Slow Clock Domain Crossing

To avoid data loss in such cases, use an extender to extend source data for at least a full cycle of the destination clock.

The following figure shows how this issue can be fixed:

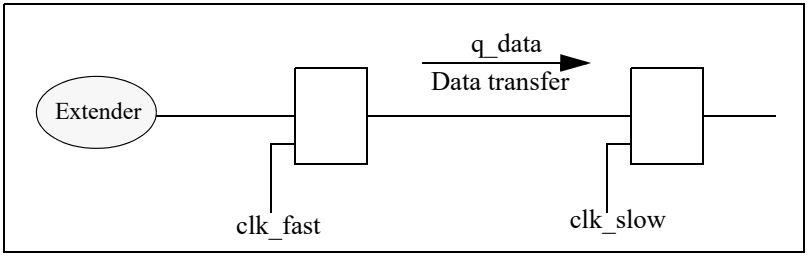


FIGURE 171. Fix the Data Loss Issue

Types of Synchronized Crossings Checked by the Ac_cdc01a Rule

The *Ac_cdc01a* rule checks for crossings that are synchronized by using any of the following techniques:

- *Conventional Multi-Flop Synchronization Scheme*
- *Qualifier Synchronization Scheme Using qualifier -crossing*
- *Synchronizing Cell Synchronization Scheme*

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with reference to the source data to capture it properly. By default, the check is performed by assuming a setup and hold margin of one clock edge. The required margin can be controlled by using the *fa_holdmargin* parameter.

Rule Exceptions

The *Ac_cdc01a* rule does not check for read/write pointers of automatically-inferred FIFOs in a design. Currently, it checks for user-defined FIFO, library cell based FIFOs, and DW FIFOs.

Parameter(s)

- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *num_flops*: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops

required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.

- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_dump_hybrid*: Default value is `partial`. Set this parameter to `all` to generate SVA for all the types of assertions (pass, fail, partially-proved). The other possible value are `pass`, `fail`, `+fail`, and `none`.
- *fa_holdmargin*: Default value is `1`. Set this parameter to `0` to control hold margins between data and clock.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *cdc_dump_assertions*: Default value is "". Set this parameter to `sva` to generate SystemVerilog Assertions (SVA) corresponding to the rules and the design assumptions specified in an SGDC file.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.

- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *fa_verify_slow_to_fast*: Default is 100. By default, data loss is checked only for fast-to-slow crossings. When this parameter is set to a specific number, data loss is also checked for slow-to-fast crossings where the percentage of the slow clock-period with reference to the fast clock-period is greater than this number.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *cdc_bus_compress*: Default value is `Ac_glitch03`, `Ac_cdc01`. Set this value to one or more values mentioned in *Possible values of the cdc_bus_compress parameter* to control the number of bits of vector signals for rule checking.

- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *clock* (Mandatory): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis*: (Optional): Use this constraint to specify case-analysis signals.
- *sync_cell*: (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *monitor_time* (Optional): Use to specify the design initialization time frames during simulation. The rest of the simulation time is considered as the design's functional time.
- *meta_design_hier* (Optional): Use to specify the test bench name and design instance name in the SGDC file.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

Message 1

The following message appears for the `FAILED` or `Others (Constraints-Conflict)` status for a clock-domain crossing between a fast and a slow clock:

[AcCdc1a_1] [ERROR] `Fast('<clk1-name>') to slow('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: <FAILED | Others (Constraints-Conflict)>`

The arguments of the above message are explained below:

TABLE 2 Arguments of Ac_cdc01a

Argument	Description
<clk1-name>	Source clock name
<clk2-name>	Destination clock name
<flop1-name>	Source clock domain flip-flop
<flop2-name>	Destination clock domain flip-flop

NOTE: *For RTL designs,* `<flop1-name>` *and* `<flop2-name>` *are names of the output nets of the corresponding flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to yes,* `<flop1-name>` *and* `<flop2-name>` *are names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains an active destination clock edge that does not arrive with a sufficient margin with respect to source data.

Consequences of Not Fixing

If you do not fix this violation, data transferred from a source clock to a destination clock may not be captured properly.

How to Debug and Fix

Open the *Waveform Viewer* window corresponding to the message, and check the marker that appears on the waveform.

This rule requires a minimum setup and hold margin of clock edges (specified by parameter *fa_holdmargin*) with respect to each source data change. By default, the value of this parameter is `one`, which indicates that for each source data change, a minimum setup and hold margin of one clock edge is required. If you know that your source data does not need to hold this long, you may change the value of the `fa_holdmargin` parameter to 0.

For example, consider the following *Waveform Viewer* window:

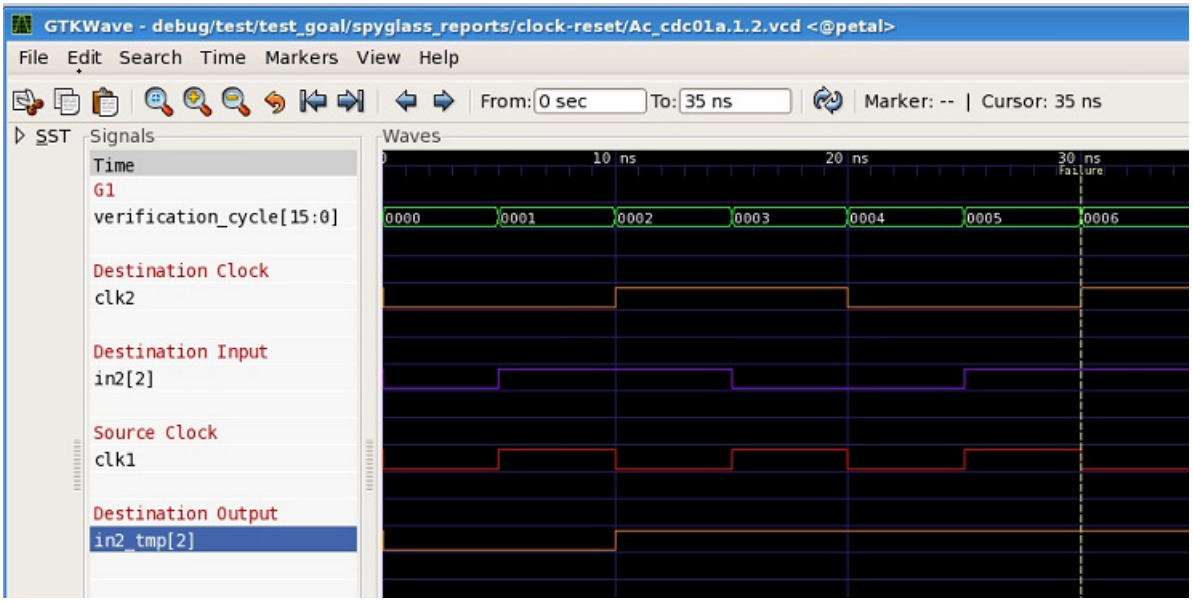


FIGURE 172. The Waveform Viewer Window

In the above waveform, the source data, `in2_tmp[2]`, does not hold for one cycle of destination clock (with the default value of *fa_holdmargin* set to `1`), and therefore causing a fail to appear. Data loss can be seen from the output net, `in2[2]`, which fails to capture the source data, `in2_tmp[2]`.

This rule checks for the same frequency clocks as well (provided that the `period` field of the clock is also specified).

Reasons for Failure

Some of the reasons that may cause false failures are as follows:

- Presence of a potential reset/clear signal causing such violation.
- In this case, provide the reset/clear in the constraint file as a reset.
- The setup (*clock*, *reset*, *set_case_analysis*, *input* constraints) is not correct and complete.
- In this case, use *Formal Setup Rules* to check for the correctness of the setup.
- The initial state values in the waveform viewer are not correct.

Provide correct initial state in the constraints file or provide a VCD file from which an initial state can be loaded.

Message 2

The following message appears for the partially-proved status for a clock-domain crossing between a fast and a slow clock:

```
[AcCdc1a_2] [WARNING] Fast('<clk1-name>') to slow('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check:Partially-Proved
```

For information on arguments of the above message, see *Table 2*.

Potential Issues

This violation appears if your design contains an active destination clock edge that does not arrive with a sufficient margin with respect to source data.

Consequences of Not Fixing

If you do not fix this violation, data transferred from a source clock to a destination clock may not be captured properly.

How to Debug and Fix

The partially-proved status appears when SpyGlass is not able to conclude (falsify or prove) data hold check in the given amount of time. In this case, try the following options to enable SpyGlass do the analysis:

- Increase assertion run-time by using the *fa_atime* parameter.
- Use incremental analysis approach by using the *fa_propfile* parameter.
- Use the *fa_abstract* parameter that applies abstraction technique to reduce complex verification problem into simpler and solvable problem.

Message 3

The following message appears for the passed status for a clock-domain crossing between a fast and a slow clock:

```
[AcCdc1a_3] [INFO] Fast('<clk1-name>') to slow('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check:Passed
```

For information on arguments of the above message, see *Table 2*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational message and does not require debugging.

By default, SpyGlass does not show data hold checks that passed the verification. To view them, set the *fa_msgmode* parameter to `all` or `pass`.

Message 4

The following message appears for the `FAILED` or `Others (Constraints-Conflict)` status a clock-domain crossing between a slow and a fast clock:

[AcCdc1a_4] [ERROR] `Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: <FAILED | Others (Constraints-Conflict)>`

`For information on arguments of the above message, see` *Table 2*.

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 5

The following message appears for the partially-proved status for a clock-domain crossing between a slow and a fast clock:

[AcCdc1a_5] [WARNING] `Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check:Partially-Proved`

`For information on arguments of the above message, see` *Table 2*.

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 6

The following message appears for the passed status for a clock-domain crossing between a slow and a fast clock:

```
[AcCdc1a_6] [WARNING] Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check:Passed
```

For information on arguments of the above message, see *Table 2*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable.

How to Debug and Fix

See *How to Debug and Fix*.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

```
// test.v

module top(input src_clk, des_clk, INPUT, output reg OUTPUT);
reg SRC_Q, SYNC_Q;
always @(posedge src_clk)
   SRC_Q <= INPUT;
always @(posedge des_clk)
   begin
   SYNC_Q <= SRC_Q;
   OUTPUT <= SYNC_Q;
   end
endmodule


// constr.sgdc

current_design top
clock -name src_clk -period 10
clock -name des_clk -period 20
```

For the above example, the *Ac_cdc01a* rule reports a violation for the fast-to-slow clock domain crossing from `SRC_Q` to `SYNC_Q`. The following spreadsheet shows the details of this violation:

A	B	C	D	E	F	G	H
ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	STATUS	TYPE	WAIVED
5	top.SRC_Q	top.src_clk	top.SYNC_Q	top.des_clk	FAILED	Fast->Slow	No

FIGURE 173. Spreadsheet generated by the Ac_cdc01a rule

Following is the *Waveform Viewer* showing the cause of this violation:

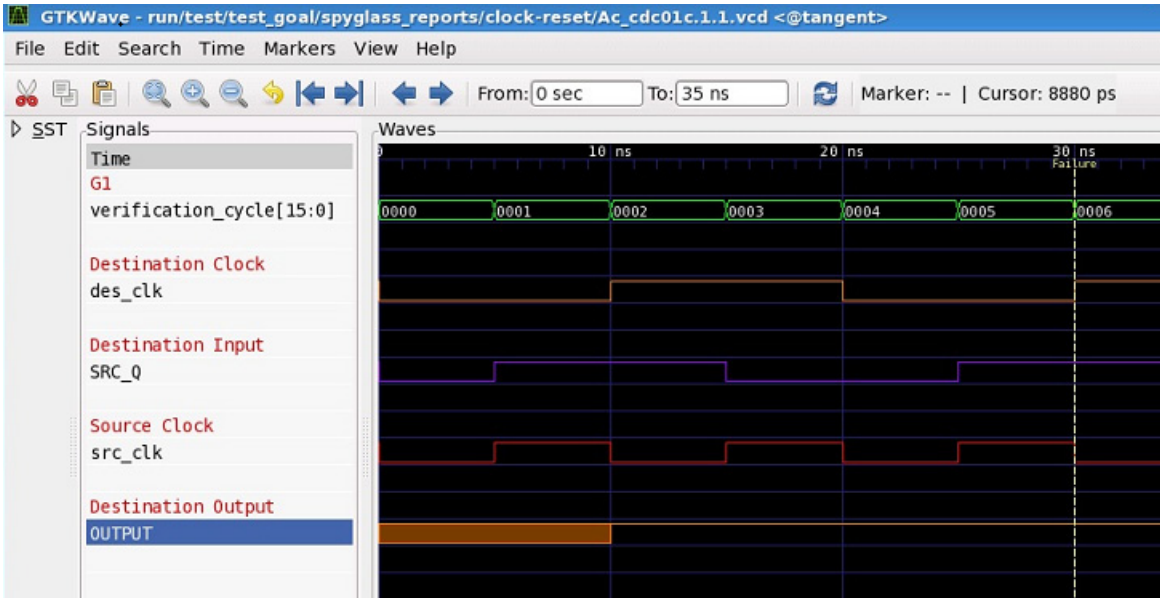


FIGURE 174. Waveform Viewer showing the cause for Ac_cdc01 violation

The following figure shows the schematic of this violation:

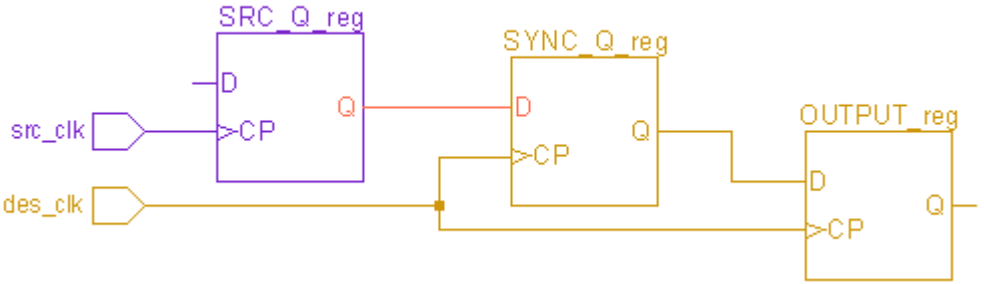


FIGURE 175. Schematic of the Ac_cdc01a rule

Back Annotations

- HDL: Show a slow clock domain register in the HDL
- Schematic: This rule highlights the following details in the schematic:

- ❒ Path from the source object to the destination object
- ❒ Path from its clock source to the clock pin of the source object. The source object will be highlighted in the same color as the source clock.
- ❒ Path from its clock source to the clock pin of the destination object. The destination object will be highlighted in the same color as the destination clock.
- ❒ The synchronizer for the clock crossings found to be synchronized

Rule Severity

The rule severity varies according to the assertion status as follows:

- `FAILED`: Error
- `Partially-Proved`: Warning
- `PASSED`: Info
- `Others(Constraints-Conflict)`: Error

Rule Group

`ADV_CLOCKS`

Report and Related Files

- Ac_cdc01a.csv: All violations of the Ac_cdc01a rule are generated in this spreadsheet file. See *Figure 173*.
- Ac_cdc01a.<ID>.OverConstrainInfo: This file contains details of conflicting constraints. For details, see *Overconstrain Info File*.

Ac_cdc01b

Checks data loss for crossings synchronized by a technique other than multi flop, sync cell, or qualifier synchronization scheme

When to Use

Use this rule to identify clock domain crossings that are prone to data loss.

Prerequisites

Following are the prerequisites:

- Use the *Advanced_CDC* and *adv_checker* license.
- Specify a clock period for at least one clock in an SGDC file.

  When you specify a clock period for at least one clock, SpyGlass assigns the default period 10 to the clocks for which no clock period is defined.

Description

The *Ac_cdc01b* rule checks the following by default:

- Fast-to-slow crossings only, and
- Same period clocks.

To include slow-to-fast crossings, set the *fa_verify_slow_to_fast* parameter.

Consider the following scenario in which data passes from fast-to-slow clock domain crossing:

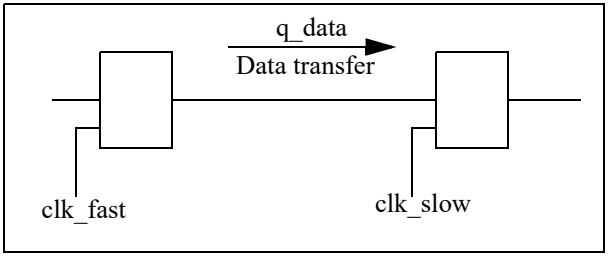


FIGURE 176. Data Passing from Fast-to-Slow Clock Domain Crossing

In the above scenario, there is a high possibility of data loss, as shown in the following figure:

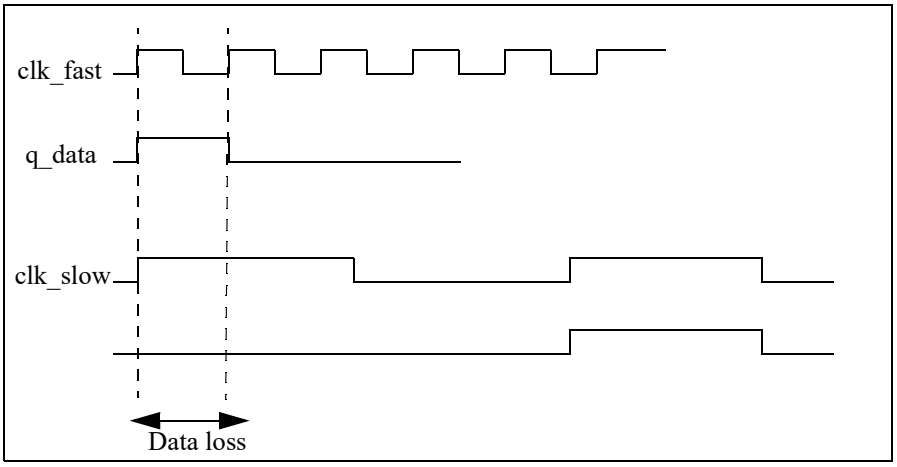


FIGURE 177. Data Loss from Fast-to-Slow Clock Domain Crossing

To avoid data loss in such cases, use an extender to extend source data for at least a full cycle of the destination clock.

The following figure shows how this issue can be fixed:

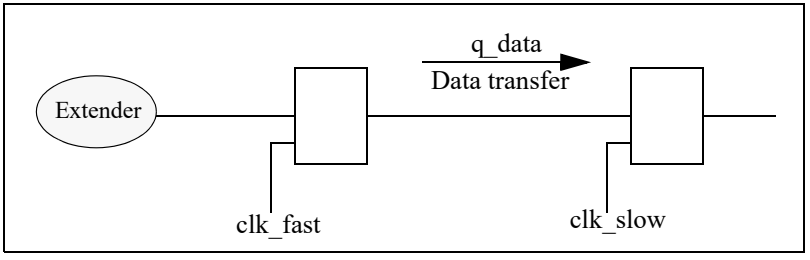


FIGURE 178. Fix the Data Loss Issue

Types of Crossings Checked by the Ac_cdc01b Rule

The *Ac_cdc01b* rule checks crossings that are **NOT** synchronized using any of the following techniques:

- *Conventional Multi-Flop Synchronization Scheme*
- *Qualifier Synchronization Scheme Using qualifier -crossing*

- *Synchronizing Cell Synchronization Scheme*

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with reference to the source data to capture it properly. By default, the check is performed by assuming a setup and hold margin of one clock edge. The required margin can be controlled by using the *fa_holdmargin* parameter.

Parameter(s)

- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *fa_holdmargin*: Default value is `1`. Set this parameter to `0` to control hold margins between data and clock.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_mux_dest_domain*: Default value is `none`. Set this parameter to `none` to turn off this parameter. Other possible values are `mux`, `enable`, `gp`, `cg`, `and`, `all`, `yes`, and `no`.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Synchronized Enable Synchronization Scheme*.
- *fa_dump_hybrid*: Default value is `partial`. Set this parameter to `all` to generate SVA for all the types of assertions (pass, fail, partially-proved). The other possible value are `pass`, `fail`, `+fail`, and `none`.
- *cdc_dump_assertions*: Default value is "". Set this parameter to `sva` to generate SystemVerilog Assertions (SVA) corresponding to the rules and the design assumptions specified in an SGDC file.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.

- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *fa_verify_slow_to_fast*: Default is 100. By default, data loss is checked only for fast-to-slow crossings. When this parameter is set to a specific number, data loss is also checked for slow-to-fast crossings where the percentage of the slow clock-period with reference to the fast clock-period is greater than this number.
- *synchronize_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.

- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *cdc_bus_compress*: Default value is `Ac_glitch03, Ac_cdc01`. Set this value to one or more values mentioned in *Possible values of the cdc_bus_compress parameter* to control the number of bits of vector signals for rule checking.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *monitor_time* (Optional): Use to specify the design initialization time frames during simulation. The rest of the simulation time is considered as the design's functional time.
- *meta_design_hier* (Optional): Use to specify the test bench name and design instance name in the SGDC file.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

Message 1

The following message appears for the `FAILED` or `Others (Constraints-Conflict)` status of a clock-domain crossing between a fast and a slow clock:

[AcCdc1b_1] [ERROR] Fast ('<clk1-name>') to slow ('<clk2-name>') clock crossing ('<flop1-name>' to '<flop2-name>') detected. Data hold check: <FAILED | Others (Constraints-Conflict)>

For information on arguments of the above message, see *Table 2*.

Potential Issues

This violation appears if your design contains an active destination clock edge that does not arrive with a sufficient margin with respect to the source data.

Consequences of Not Fixing

If you do not fix this violation, data transferred from a source clock to a destination clock may not be captured properly.

How to Debug and Fix

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with respect to the source data in order to capture it properly.

For details on debugging, see *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 2

The following message appears to indicate the partially-proved status of a clock-domain crossing between a fast and a slow clock:

[AcCdc1b_2] [WARNING] Fast ('<clk1-name>') to slow ('<clk2-name>') clock crossing ('<flop1-name>' to '<flop2-name>') detected. Data hold check:Partially-Proved

For information on arguments of the above message, see *Table 2*.

Potential Issues

See *Potential Issues* of the *Ac_cdc01a* rule.

Consequences of Not Fixing

See *Consequences of Not Fixing* of the *Ac_cdc01a* rule.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 3

The following message appears to indicate the passed status of a clock-domain crossing between a fast and a slow clock:

[AcCdc1b_3] [INFO] `Fast ('<clk1-name>') to slow ('<clk2-name>') clock crossing ('<flop1-name>' to '<flop2-name>') detected. Data hold check:PASSED`

For information on arguments of the above message, see *Table 2*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 4

The following message appears for the `FAILED` or `Others (Constraints-Conflict)` status a clock-domain crossing between a slow and a fast clock:

[AcCdc1b_4] [ERROR] `Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: <FAILED | Others (Constraints-Conflict)>`

For information on arguments of the above message, see *Table 2*.

Potential Issues

This violation appears if your design contains an active destination clock edge that does not arrive with a sufficient margin with respect to the source data.

Consequences of Not Fixing

If you do not fix this violation, data transferred from a source clock to a destination clock may not be captured properly.

How to Debug and Fix

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with respect to the source data in order to capture it properly.

For details on debugging, see *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 5

The following message appears to indicate the partially-proved status of a clock-domain crossing between a slow and a fast clock:

```
[AcCdc1b_5] [WARNING] Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: Partially-Proved
```

For information on arguments of the above message, see *Table 2*.

Potential Issues

See *Potential Issues* of the *Ac_cdc01a* rule.

Consequences of Not Fixing

See *Consequences of Not Fixing* of the *Ac_cdc01a* rule.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 6

The following message appears to indicate the passed status of a clock-domain crossing between a slow and a fast clock:

```
[AcCdc1b_6] [WARNING] Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: Passed
```

For information on arguments of the above message, see *Table 2*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

```
// test.v
   module top(input src_clk, des_clk, INPUT, ENABLE, output reg OUTPUT);
   reg SRC_Q, ENABLE_SYNC1, ENABLE_SYNC2, ENABLE_OUT;
   always @(posedge src_clk)
      begin
        SRC_Q <= INPUT;
        ENABLE_OUT <= ENABLE;
      end
   always @(posedge des_clk)
      begin
        ENABLE_SYNC1 <= ENABLE_OUT;
        ENABLE_SYNC2 <= ENABLE_SYNC1;
      end
   always @(posedge des_clk)
      if (ENABLE_SYNC2)
         OUTPUT <= SRC_Q;
   endmodule


// constr.sgdc
   current_design top
   clock -name src_clk -period 10
   clock -name des_clk -period 20
```

For the above example, the *Ac_cdc01b* rule reports a violation for fast-to-slow clock domain crossing between `SRC_Q` and `OUTPUT`. The following spreadsheet shows the details of this violation:

A	B	C	D	E	F	G	H
ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	STATUS	TYPE	WAIVED
6	top.SRC_Q	top.src_clk	top.OUTPUT	top.des_clk	FAILED	Fast->Slow	No

FIGURE 179. Spreadsheet generated by the Ac_cdc01b rule

Following is the *Waveform Viewer* showing the cause of this violation:

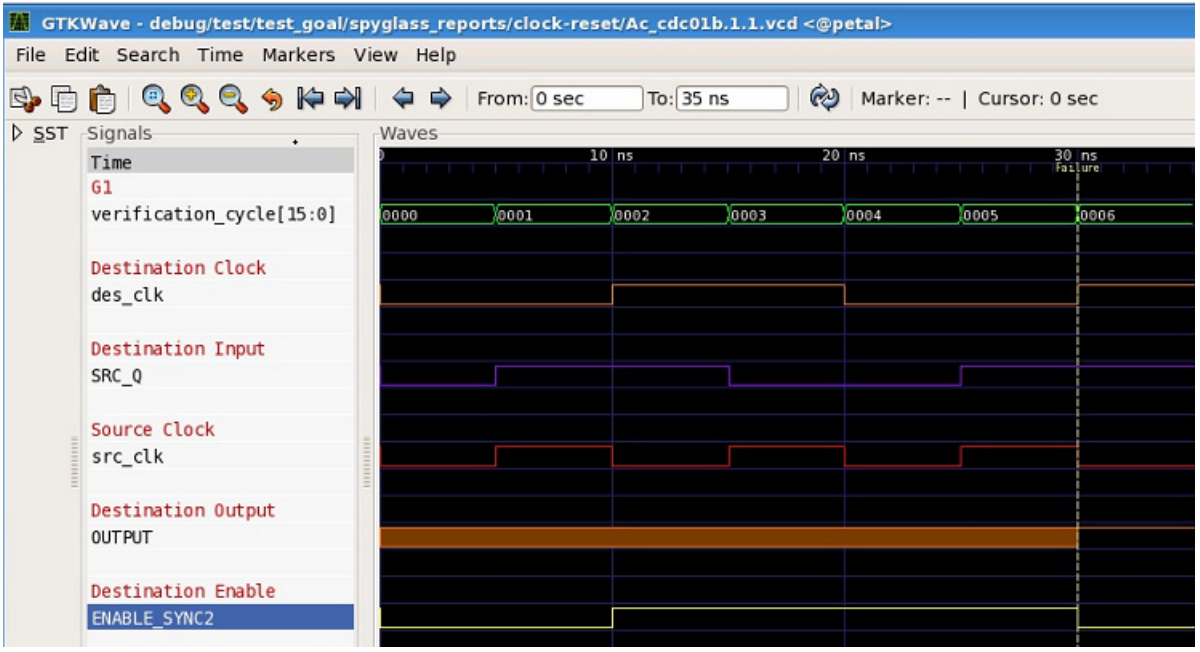


FIGURE 180. Waveform Viewer showing the cause for Ac_cdc01b violation

The following figure shows the schematic of this violation:

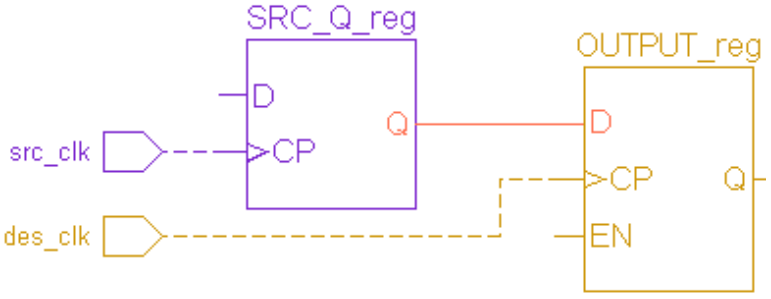


FIGURE 181. Schematic of the Ac_cdc01b rule

Back annotations

- In HDL, this rule highlights the slow clock domain register.
- In the schematic, this rule highlights:
  - The path from the source object to the destination object.
  - The path from its clock source to the clock pin of the source object. The source object will be highlighted in the same color as the source clock.
  - The path from its clock source to the clock pin of the destination object. The destination object will be highlighted in the same color as the destination clock.
  - The synchronizer for the clock crossings found to be synchronized.

Default Severity Label

The rule severity varies according to the assertion status as follows:

- `FAILED`: Error
- `Partially-Proved`: Warning
- `PASSED`: Info
- `Others(Constraints-Conflict)`: Error

Rule Group

`ADV_CLOCKS`

Reports and Related Files

- Ac_cdc01b.csv. See *Figure 179*.
- Ac_cdc01b.<ID>.OverConstrainInfo: This file contains details of conflicting constraints. For details, see *Overconstrain Info File*.

Ac_cdc01c

Checks data loss for unsynchronized clock domain crossings

When to Use

Use this rule to identify clock domain crossings that are prone to data loss.

Prerequisites

Following are the prerequisites:

- Use the *Advanced_CDC* and *adv_checker* license.
- Specify a clock period for at least one clock in an SGDC file.

  When you specify a clock period for at least one clock, SpyGlass assigns the default period 10 to the clocks for which no clock period is defined.

Description

The *Ac_cdc01c* rule checks crossings that are unsynchronized. By default, the type of crossings checked are:

- Fast-to-slow crossings only, and
- Same period clocks.

To include slow-to-fast crossings, set the *fa_verify_slow_to_fast* parameter.

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with reference to the source data to capture it properly. By default, the check is performed by assuming a setup and hold margin of one clock edge. The required margin can be controlled by using the *fa_holdmargin* parameter.

Parameter(s)

- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *num_flops*: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.

- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_dump_hybrid*: Default value is `partial`. Set this parameter to `all` to generate SVA for all the types of assertions (pass, fail, partially-proved). The other possible value are `pass`, `fail`, `+fail`, and `none`.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *fa_holdmargin*: Default value is `1`. Set this parameter to `0` to control hold margins between data and clock.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *enable_mux_dest_domain*: Default value is `none`. Set this parameter to `none` to turn off this parameter. Other possible values are `mux`, `enable`, `gp`, `cg`, `and`, `all`, `yes`, and `no`.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Synchronized Enable Synchronization Scheme*.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *cdc_dump_assertions*: Default value is "". Set this parameter to `sva` to generate SystemVerilog Assertions (SVA) corresponding to the rules and the design assumptions specified in an SGDC file.

- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *fa_verify_slow_to_fast*: Default is 100. By default, data loss is checked only for fast-to-slow crossings. When this parameter is set to a specific number, data loss is also checked for slow-to-fast crossings where the percentage of the slow clock-period with reference to the fast clock-period is greater than this number.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.

- *cdc_bus_compress*: Default value is `Ac_glitch03, Ac_cdc01`. Set this value to one or more values mentioned in *Possible values of the cdc_bus_compress parameter* to control the number of bits of vector signals for rule checking.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *monitor_time* (Optional): Use to specify the design initialization time frames during simulation. The rest of the simulation time is considered as the design's functional time.
- *meta_design_hier* (Optional): Use to specify the test bench name and design instance name in the SGDC file.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

Message 1

The following message appears for the `FAILED` or `Others (Constraints-Conflict)` status for a clock-domain crossing between a fast and a slow clock:

[AcCdc1c_1] [ERROR] `Fast('<clk1-name>') to slow('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: <FAILED | Others (Constraints-Conflict)>`

`For information on arguments of the above message, see` *Table 2*.

Potential Issues

This violation appears if your design contains an active destination clock

edge that does not arrive with a sufficient margin with respect to the source data.

Consequences of Not Fixing

If you do not fix this violation, data transferred from a source clock to a destination clock may not be captured properly.

How to Debug and Fix

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with respect to the source data in order to capture it properly.

For details on debugging, see *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 2

The following message appears to indicate the partially-proved status of a clock-domain crossing between a fast and a slow clock:

```
[AcCdc1c_2] [WARNING] Fast('<clk1-name>') to slow('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: Partially-Proved
```

`For information on arguments of the above message, see Table 2.`

Potential Issues

See *Potential Issues* of the *Ac_cdc01a* rule.

Consequences of Not Fixing

See *Consequences of Not Fixing* of the *Ac_cdc01a* rule.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 3

The following message appears to indicate the passed status of a clock-domain crossing between a fast and a slow clock:

```
[AcCdc1c_3] [INFO] Fast ('<clk1-name>') to slow ('<clk2-name>')
```

clock crossing ('<flop1-name>' to '<flop2-name>') detected. Data hold check: Partially-Proved

For information on arguments of the above message, see *Table 2*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 4

The following message appears for the FAILED or Others (Constraints-Conflict) status for a clock-domain crossing between a slow and a fast clock:

[AcCdc1c_4] [ERROR] Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: <Failed | Others (Constraints-Conflict)>

For information on arguments of the above message, see *Table 2*.

Potential Issues

This violation appears if your design contains an active destination clock edge that does not arrive with a sufficient margin with respect to the source data.

Consequences of Not Fixing

If you do not fix this violation, data transferred from a source clock to a destination clock may not be captured properly.

How to Debug and Fix

Corresponding to each source data transition (including sources modeled using abstract ports), an active destination clock edge should always arrive with sufficient margin with respect to the source data in order to capture it

properly.

For details on debugging, see *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 5

The following message appears to indicate the partially-proved status of a clock-domain crossing between a slow and a fast clock:

[AcCdc1c_5] [WARNING] Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: Partially-Proved

For information on arguments of the above message, see *Table 2*.

Potential Issues

See *Potential Issues* of the *Ac_cdc01a* rule.

Consequences of Not Fixing

See *Consequences of Not Fixing* of the *Ac_cdc01a* rule.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Message 6

The following message appears to indicate the passed status of a clock-domain crossing between a slow and a fast clock:

[AcCdc1c_6] [INFO] Slow('<clk1-name>') to fast('<clk2-name>') clock crossing('<flop1-name>' to '<flop2-name>') detected. Data hold check: Passed

For information on arguments of the above message, see *Table 2*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc01a* rule.

Example Code and/or Schematic

Example 1

Consider the following files specified for SpyGlass analysis:

```
// test.v

   module top(input src_clk, des_clk, INPUT, output reg OUTPUT);
   reg SRC_Q;

   always @(posedge src_clk)
      SRC_Q <= INPUT;

   always @(posedge des_clk)
      OUTPUT <= SRC_Q;
   endmodule


// constr.sgdc

   current_design top

   clock -name src_clk -period 10
   clock -name des_clk -period 20
```

For the above example, the *Ac_cdc01c* rule reports a violation for fast-to-slow clock domain crossing between `SRC_Q` and `OUTPUT`. The following spreadsheet shows the details of this violation:

A	B	C	D	E	F	G	H
ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	STATUS	TYPE	WAIVED
5	top.SRC_Q	top.src_clk	top.OUTPUT	top.des_clk	FAILED	Fast->Slow	No

FIGURE 182. Spreadsheet generated by the Ac_cdc01c rule

Following is the *Waveform Viewer* showing the cause of this violation:

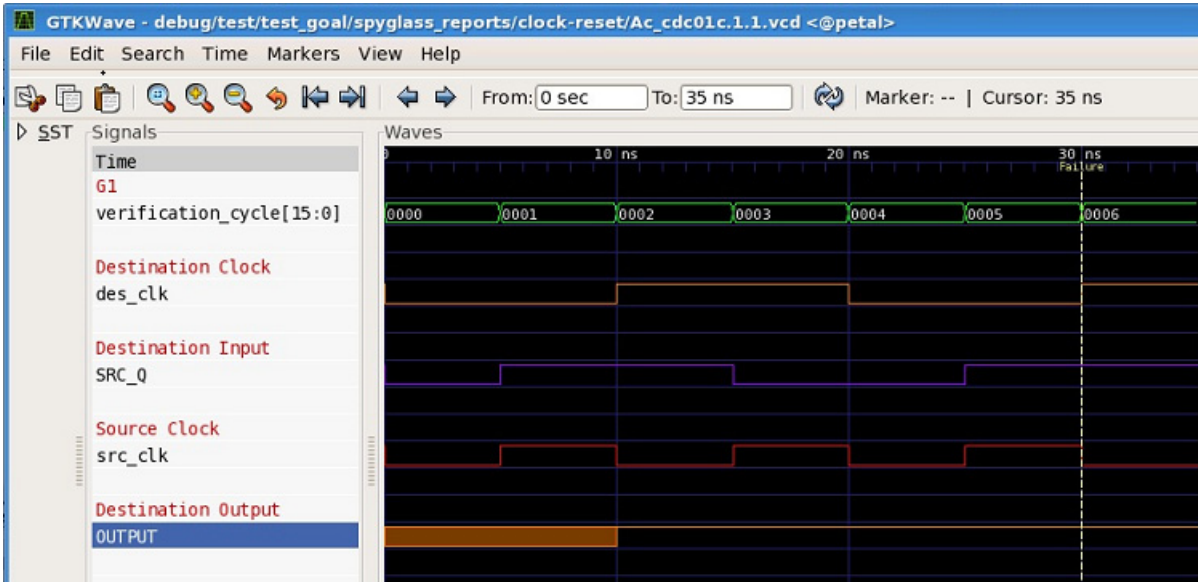


FIGURE 183. Waveform Viewer showing the cause for Ac_cdc01c violation

The following figure shows the schematic of this violation:

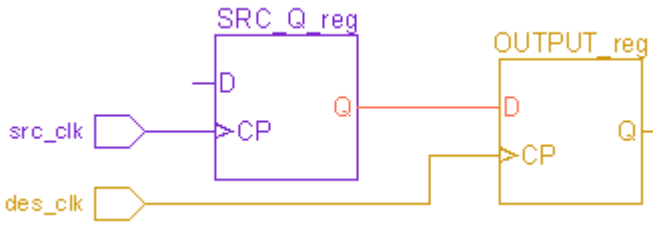


FIGURE 184. Schematic of the Ac_cdc01c rule

Back annotations

- In HDL, this rule highlights the slow clock domain register
- In the schematic, this rule highlights the following information:
  - Path from the source object to the destination object

- ❒ Path from its clock source to the clock pin of the source object. The source object will be highlighted in the same color as the source clock.
- ❒ Path from its clock source to the clock pin of the destination object. The destination object will be highlighted in the same color as the destination clock.

Example 2

Consider the following files specified for SpyGlass analysis:

```
# test.v
  module abc(clk1, clk2, in1, in2, out);
  input clk1, clk2, in1, in2;
  output out;
  wire r3;
  reg out;
  assign r3 = in1 & in2;
  always@(posedge clk2)
     out = r3;
  endmodule

# constraints.clock
  current_design abc
  clock -name abc.clk1 -value rtz -period 30 -edge 0 15
  clock -name abc.clk2 -value rtz -period 20 -edge 0 10
  clock -tag clk3 -value rtz -period 10 -edge 0 5
  abstract_port -ports abc.in1 -clock abc.clk1
  abstract_port -ports abc.in2 -clock clk3
```

In the above example, the source of the data is an input port.

Due to the *abstract_port* constraints on the input ports `in1` (slower clock `abc.clk1`) and `in2` (faster virtual clock `clk3`), the data is travelling from the fast clock to the slow clock, thereby resulting in the possibility of data loss. Therefore, the *Ac_cdc01c* rule reports the following violation:

```
Fast('clk3') to slow('abc.clk2') clock crossing (from 'in2' to
'abc.out') detected. Data hold check:FAILED,test.v, 11
```

Default Severity Label

The rule severity varies according to the assertion status as follows:

- `FAILED`: Error
- `Partially-Proved`: Warning
- `PASSED`: Info
- `Others(Constraints-Conflict)`: Error

Rule Group

`ADV_CLOCKS`

Report and Related Files

- All violations of the *Ac_cdc01c* rule are stored in the spreadsheet file, Ac_cdc01c.csv. See *Figure 182*.
- Ac_cdc01c.<ID>.OverConstrainInfo: This file contains details of conflicting constraints. For details, see *Overconstrain Info File*.

Ac_cdc08

Reports control-bus clock domain crossings which do not follow gray encoding

When to Use

Use this rule to detect control buses that cross a synchronized clock-domain crossing, but do not follow gray encoding.

Prerequisites

The *Ac_cdc08* rule works only with the `Advanced_CDC` and `adv_checker` license features.

Description

The *Ac_cdc08* rule reports synchronized control-bus clock domain crossings that do not follow gray encoding.

The *Ac_cdc08* rule checks only those multi-bit control buses that are synchronized based on the following synchronization schemes:

- The *Conventional Multi-Flop Synchronization Scheme*
- The *Synchronizing Cell Synchronization Scheme* provided you have specified the synchronizer cells by using the *synchronize_cells* parameter (for cells whose output pin is connected to a scalar net) or by using the *synchronize_data_cells* parameter (for cells for which an output pin is connected to a vector net).
- *Qualifier Synchronization Scheme Using qualifier -crossing*

Parameter(s)

- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *fa_abstract*: Default value is *Ac_handshake01*, *Ac_glitch03*. Set the value of this parameter to *Ac_cdc08* to enable abstraction for this rule. Other possible values are `all`, `none` or a list of any of rules:

  Ac_handshake01, *Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, *Ac_fifo01*, and

Ac_glitch03.

- *fa_atsrc*: Default value is `no`. Set this parameter to `yes` to check for gray-encoding at an output of a source instance.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.

Constraint(s)

- *clock* (Mandatory): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

Message 1

The following message reports the `FAILED` or `Others (Constraints-Conflict)` status when a control bus crosses a synchronized clock-domain crossing but does not follow gray encoding:

`[AcCdc8_1] [ERROR]` `Control bus '<bus-name>' is crossing clock domain. Gray encoding check: <FAILED | Others (Constraints-Conflict)>`

Potential Issues

This violation appears if a control bus crossing a clock-domain crossing

does not follow gray encoding.

Consequences of Not Fixing

Gray-encoding is useful for data correlation. If you do not use gray encoding, data may be lost.

How to Debug and Fix

Open the *Waveform Viewer* window corresponding to the message, and check the marker that appears on the waveform.

This marker is positioned at a point where more than one bits are found changing in the same cycle, thereby violating gray encoding. Therefore, this specific transition is a *witness* to the failure.

For example, consider the following *Waveform Viewer* window:

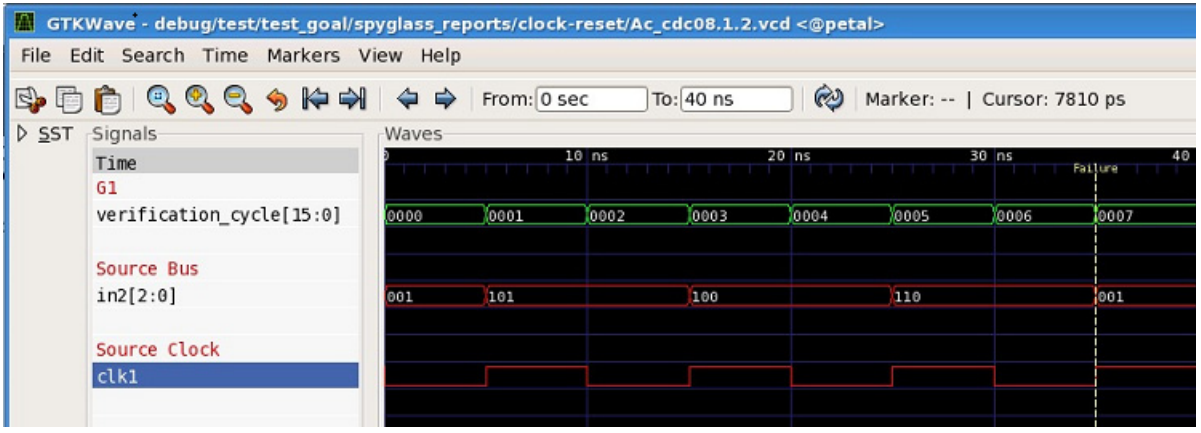


FIGURE 185. The Waveform Viewer Window

In the above waveform, the marker appears at a point where the `in2` signal transitions from 6 to 1 (110 -> 001), thereby violating gray encoding protocol.

Reasons for Failure

Some of the reasons that may cause false failures are as follows:

- Presence of a potential reset/clear signal causing such violation.

- In this case, provide the reset/clear in the constraint file as a reset.
- The setup (clocks, resets, set_case_analysis, input constraints) is not correct and complete.
- In this case, use *Formal Setup Rules* to check for the correctness of the setup.
- The initial state values in the waveform viewer are not correct.

  In this case, provide a correct initial state in the constraints file or provide a VCD file from which an initial state can be loaded.

Message 2

The following message reports the `Partially-Proved` status when a control bus crosses a synchronized clock-domain crossing but does not follow gray encoding:

`[AcCdc8_2] [WARNING]` `Control bus '<bus-name>' is crossing clock domain. Gray encoding check: Partially-Proved`

Potential Issues

This violation appears if a control bus crossing a clock-domain crossing does not follow gray encoding.

Consequences of Not Fixing

Gray-encoding is useful for data correlation. If you do not use gray encoding, data may be lost.

How to Debug and Fix

The `Partially-Proved` status appears when SpyGlass is not able to conclude (falsify or prove) gray encoding check in the given amount of time.

In this case, you need to help the tool complete the analysis. You may try following options for better results:

- Increase assertion run-time by using the *fa_atime* parameter.
- Use incremental analysis approach by using the *fa_propfile* parameter.

- Use the *fa_abstract* parameter that applies abstraction technique to reduce complex verification problem into simpler and solvable problem.

Message 3

The following message reports the `PASSED` status when a control bus crosses a synchronized clock-domain crossing but does not follow gray encoding:

`[AcCdc8_3] [INFO] Control bus '<bus-name>' is crossing clock domain. Gray encoding check: PASSED`

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational message and does not require debugging.

By default, SpyGlass does not show bus crossing clock domains that passed the verification. To view them, set the *fa_msgmode* parameter to `all` or `pass`.

Example Code and/or Schematic

Consider the following scenario in which this rule reports a violation:

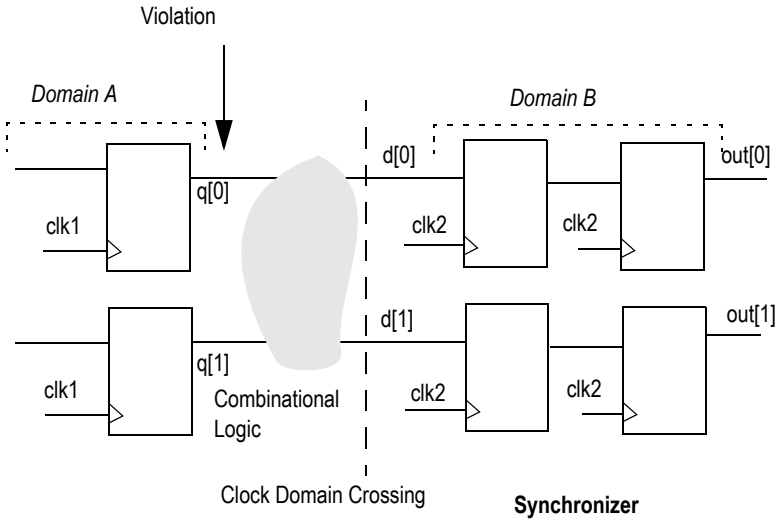


FIGURE 186. Scenario of the Ac_cdc08 Rule Violation

In the above scenario, the Ac_cdc08 rule reports a violation if the output of source flip-flops are not gray-encoded.

To fix this violation, provide a gray-encoding logic at the source of the flip-flops, as shown in the following figure:

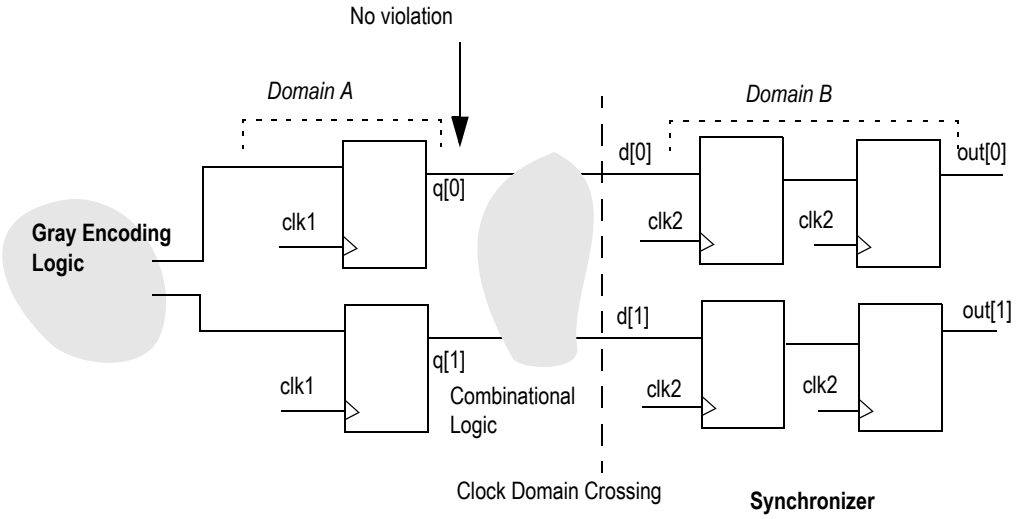


FIGURE 187. Gray-Encoding Logic at the Source of the Flip-Flops

Back annotations

- HDL: Shows a bus in the source code
- Schematic: Following details are highlighted in the schematic:
  - Path from the source object to the destination object.
  - Path from its clock source to the clock pin of the source object. The source object will be highlighted in the same color as the source clock.
  - Path from its clock source to the clock pin of the destination object. The destination object will be highlighted in the same color as the destination clock.
  - The synchronizer for the clock crossings found to be synchronized.
- Waveform: Shows the control bus (input to the destination flip-flops or output of the source flip-flops) and the source clock.

Default Severity Label

The rule severity varies according to the assertion status as follows:

- `FAILED`: Error

- `Partially-Proved`: Warning
- `PASSED`: Info
- `Others(Constraints-Conflict)`: Error

Rule Group

`ADV_CLOCKS`

Report and Related File

Ac_cdc08.<ID>.OverConstrainInfo: This file contains details of conflicting constraints. For details, see *Overconstrain Info File*.

Ac_clockperiod03

Reports a set of correlated clocks for which design cycle time is greater than threshold value

When to Use

Use this rule to detect correlated clocks for which design cycle time exceeds the specified threshold value.

Prerequisites

Enable this rule by using the following command in a project file:

```
set_goal_option addrules Ac_clockperiod03
```

By default, this rule is switched off.

Description

The *Ac_clockperiod03* rule reports correlated clocks for which design cycle is greater than the specified threshold value.

Correlated clocks are clocks that interact with one another and are present in the fan-in cone of a particular assertion.

Parameter(s)

- *fa_verif_cycles*: Default value is `1024`. Set the value of this parameter to a positive integer value (greater than 2 and less than 65535) to specify a maximum number of verification cycles for a clock.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for correlated clocks for which design cycle is greater than a threshold value:

[WARNING] Design cycle for clocks '<list>' exceeds the threshold. Refer file: '<file-name>' for details

The arguments of the above message are explained below:

Argument	Description
<list>	List of clocks whose design cycle is greater than the threshold value
<file-name>	Name of the spreadsheet file

Potential Issues

This violation appears if there are some assertions with a fan-in cone that contain huge asynchronous clocks.

Consequences of Not Fixing

If you do not fix this violation, design cycle computation process, and functional analysis process becomes slow. In such cases, the following may happen:

- Concluding rate of assertions go down with more assertions being reported as *Partially Proved*.
- If the design cycle time exceeds the threshold value, all assertions are reported under the `Others(Internal-Error)` category.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the *Spreadsheet Viewer* for this rule.
2. In the spreadsheet, view the list of correlated clocks for which design cycle is greater than a threshold value specified using the *fa_verif_cycles* parameter.
3. Specify periods of clocks in such a way so that they are multiples of each other. Try to ensure that the collective design cycle is low.
4. For better formal results, it is recommended to decrease the value of the `fa_verif_cycles` parameter and maintain clock period values such that the collective design cycle stays lower than the threshold value.

Example Code and/or Schematic

Consider the following `clock` constraint specifications in an SGDC file:

```
clock -name clk1 -period 1
clock -name clk2 -period 4
clock -name clk3 -period 3000
```

In the above example, the *Ac_clockperiod03* rule does not report any violation if an assertion contains the `clk1` and `clk2` clocks in its fan-in.

However, this rule reports a violation if an assertion contains the `clk1` and `clk3` clocks in its fan-in and if the `fa_verif_cycle` parameter is set to its default value,`1024`.

Default Severity Label

Warning

Rule Group

`ADV_CLOCKS`

Report and Related File

Ac_clockperiod03.csv: This file contains details of correlated clocks and their design cycle.

Ac_conv01

Reports signals from the same domain that are synchronized in the same destination domain and converge after any number of sequential elements

When to Use

Use this rule to check sequential convergences of same-domain signals synchronized in the same destination domain.

Description

The details of the *Ac_conv01* rule are covered under the following topics:

- *Reason for the Ac_conv01 Rule Violation*
- *Features of the Ac_conv01 Rule*
- *Handling MUXes by the Ac_conv01 Rule*
- *Rule Exceptions*

Reason for the *Ac_conv01* Rule Violation

The *Ac_conv01* rule reports a violation for two cases, *Case 1* and *Case 2*.

Case 1

The *Ac_conv01* rule reports same source signals that converge after satisfying the following conditions:

- Signals from the same source domain are synchronized in the same destination domain.
- Signals are synchronized by using the *Conventional Multi-Flop Synchronization Scheme*, *Synchronizing Cell Synchronization Scheme*, or *Qualifier Synchronization Scheme Using qualifier -crossing*.
- Synchronized signals converge after any number of sequential elements on some net of a design.

The same source signals are further analyzed to determine whether a specific signal in the fan-in cone is driving these signals or the same signal is getting synchronized multiple times. You can control the depth of this fan-in cone through the *conv_src_seq_depth* parameter.

The following figure shows the example of convergence of signals coming from the same clock domains:

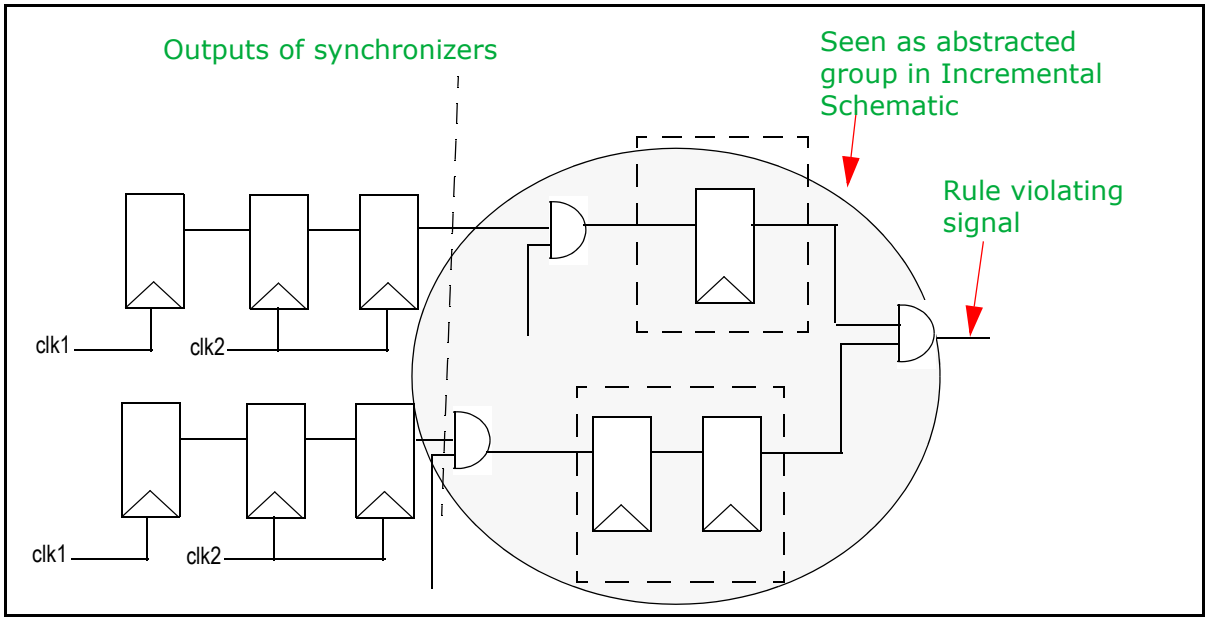


FIGURE 188. Convergence of Signals from Same Source Domain Converging after Sequential Elements

Case 2

The *Ac_conv01* rule also reports a violation when a signal diverges and then converges and after diverging, it is synchronized by the same-domain synchronizers on the divergent paths.

The following figure shows such scenario:

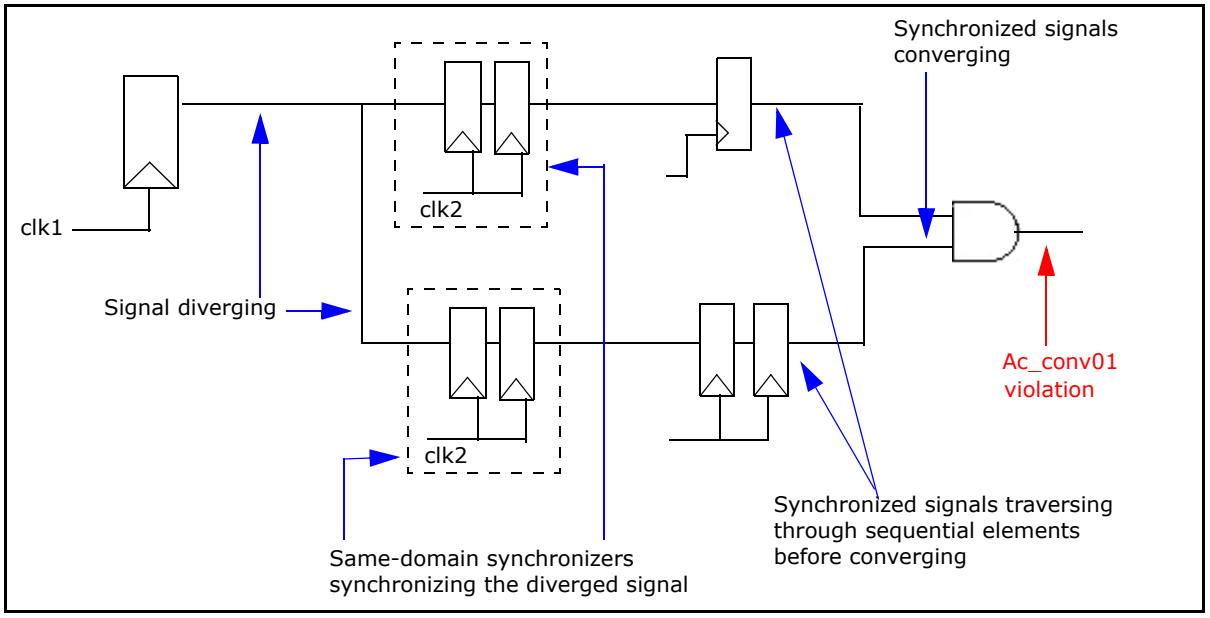


FIGURE 189. Example- The Ac_conv01 Rule Violation

The Ac_conv01 rule reports sequential convergence for bus-merged destination names. But the schematic and depth is calculated for any one representative bit of this bus-merged destination name. However, it may happen that the representative bit has the depth 0 and a non-representative bit has a non-zero depth due to which it is considered as sequential convergence. Therefore, it might seem as a combinational convergence from depth/schematic, which is reported in sequential convergence.

Features of the Ac_conv01 Rule

Following are the salient features of this rule:

- It reports only one violation if the same set of signals converge in more than one path.
- It allows flip-flops in the output cone synchronizing signals.
- When convergence occurs at multiple points in a path, it reports a violation at a point that has maximum number of converging signals.

- It checks for both scalar and bus synchronized signals.
- It reports violations on the closest point, as shown in the following figure:

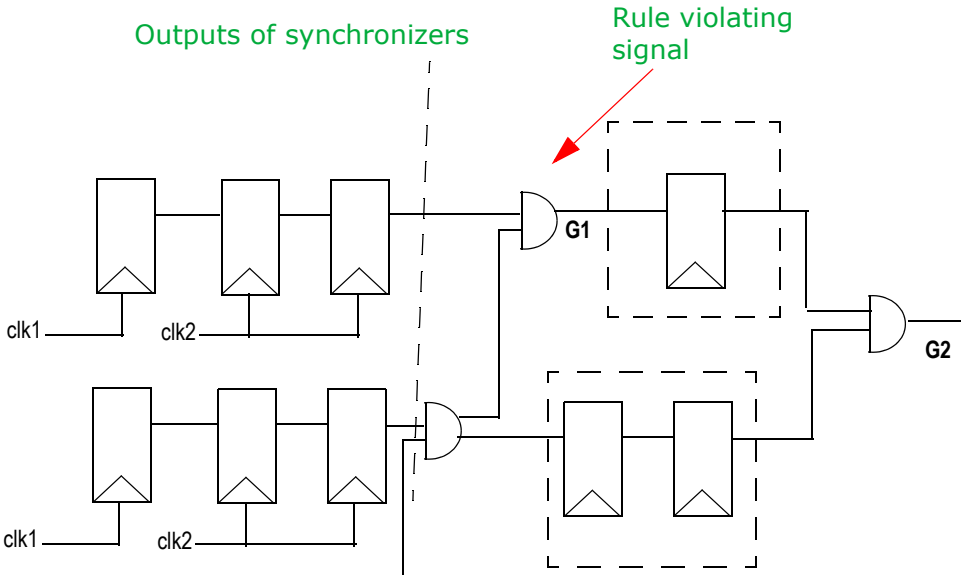


FIGURE 190. Ac_conv01 Rule Violation Reported on the Closest Point

In the above example, the *Ac_conv01* rule reports violation at `G1`, which is closest to the converging signals.

NOTE: *If same convergence is seen on two nets, this rule reports convergence on a net closest to the converging signal.*

- It reports a violation on internal nets if the output cone of that net is blocked due to *set_case_analysis* or design power/ground.

Handling MUXes by the Ac_conv01 Rule

On encountering a mux, this rule propagates convergences of all the data input and control inputs. For example, in the following figure, the

convergence on the mux is propagated to the AND gate:

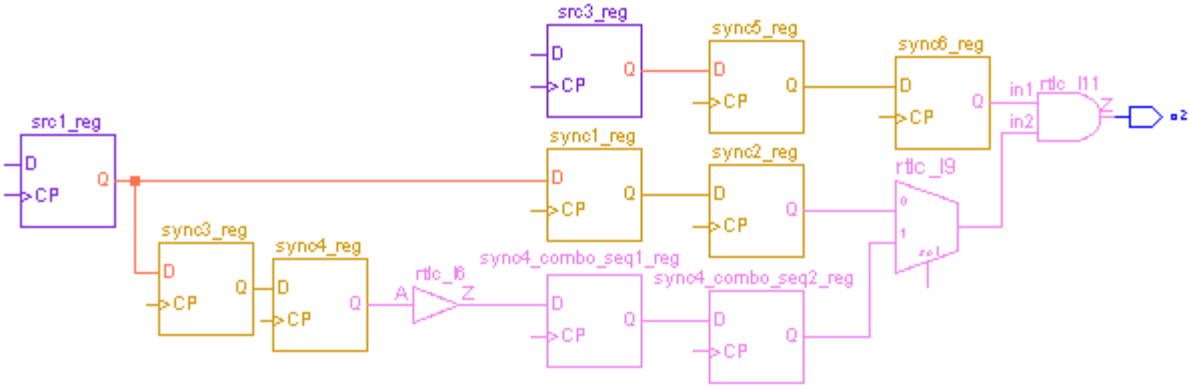


FIGURE 191.

In the above scenario, the *Ac_conv01* rule reports a violation at the AND gate for the `sync2`, `sync4`, and `sync6` synchronizers.

Rule Exceptions

This rule has the following exceptions:

- By default, this rule ignores convergence of clock domain crossing signals specified by the *cdc_false_path* constraint.
- This rule does not report convergence if two data crossings having same control line are converging.

  For example, this rule does not report any convergence in the following scenario:

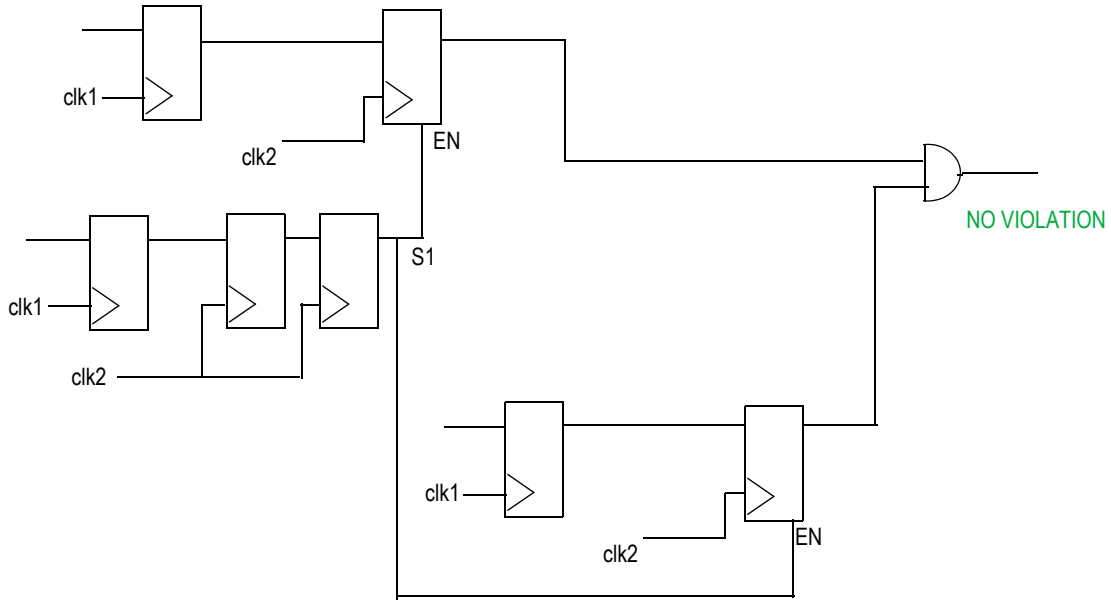


FIGURE 192. Ac_conv01 Rule Exception - No Violation on Any Convergence

However, if the two data crossings have a different control line, this rule reports convergence for the control signals, as shown in the following figure:

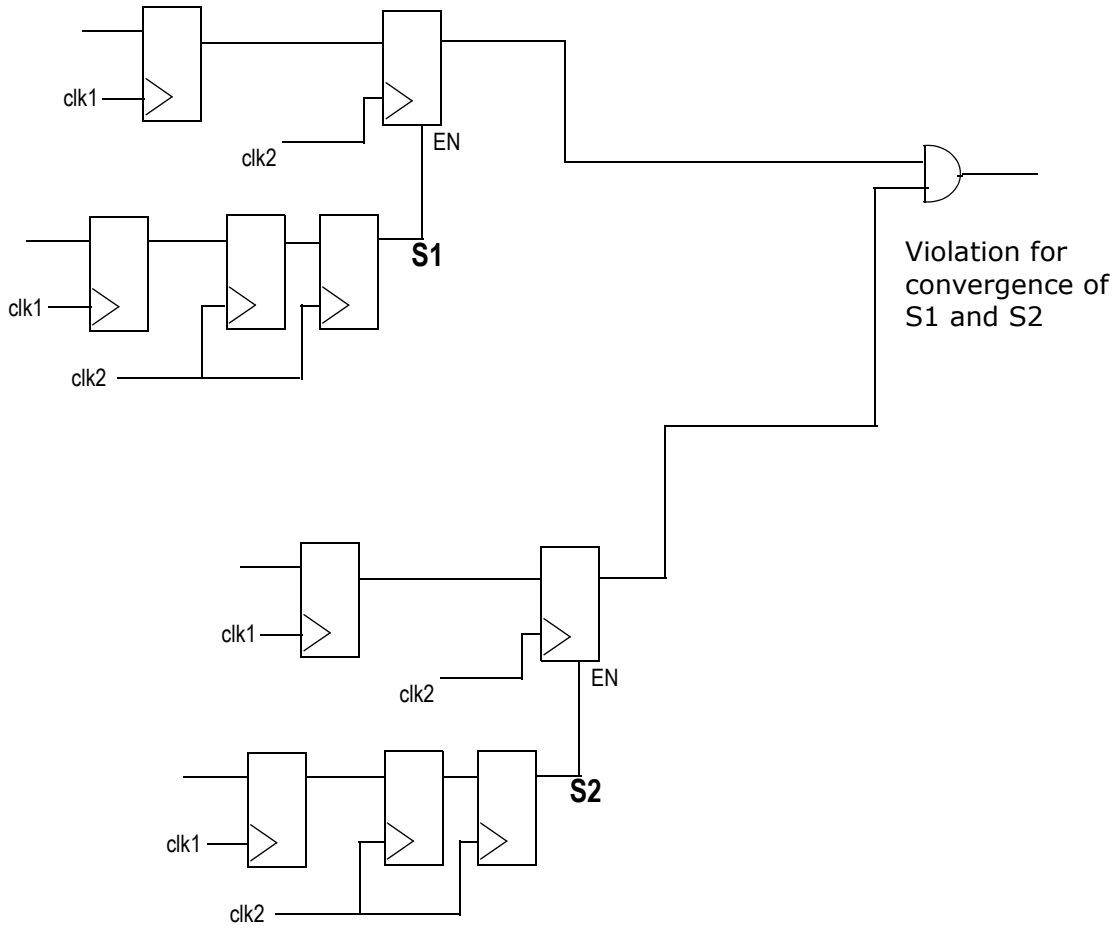


FIGURE 193. Ac_conv01 Rule Violation on Convergence of Control Signals

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *no_convergence_check*: Default value is `NULL`, which means that this rule checks all nets for convergence. Specify net names that should not be checked for convergence.

- *convergence_stop_at_mux*: Default value is `no`. Set this parameter to `yes` to stop propagation of relevant signals whenever an RTL mux is encountered.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *conv_sync_as_src*: Default value is `no`. Set this parameter to `yes` to check convergence for synchronizers that are also used as a source in other crossings.
- *conv_clock_reset_path*: Default value is `no`. Set this parameter to yes to enable convergence detection of the synchronizer propagation through the clock and reset pin.
- *conv_sync_seq_depth*: Default value is 0. Specify a positive integer value to set the sequential depth to be considered for propagation of synchronizers.
- *conv_sync_seq_depth_opt*: Default value is no. Set this parameter to yes to improve the runtime performance of the *Ac_conv01* rule when the *conv_sync_seq_depth* parameter is set to 1.
- *conv_src_seq_depth*: Default value is -1. Specify a positive integer value to set the sequential depth to be skipped while detecting the common net of the source of synchronizers. Other possible value is 0.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to `stop_conv_at_seq_lib` to stop synchronizer propagation across sequential library cell. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *show_source_in_spreadsheet*: Default value is `yes`. Set this parameter to `no` to generate a link from the spreadsheet of the *Ac_conv01*, *Ac_conv02*, or *Ac_conv03* rules to the message-based spreadsheet of *The Ac_sync_group Rules* showing the source of synchronizers.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *delayed_ptr_fifo*: Default value is `no`. Set this parameter to `yes` when the read/write pointers are delayed and the multiplexer inside the memory is one-hot or implemented using gates.
- *enable_and_sync*: Default value is `no`. Set this parameter to `yes` to enable the *AND Gate Synchronization Scheme*.
- *enable_mux_dest_domain*: Default value is `none`. Set this parameter to `none` to turn off this parameter. Other possible values are `mux`, `enable`, `gp`, `cg`, `and`, `all`, `yes`, and `no`.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Synchronized Enable Synchronization Scheme*.
- *enable_sync_cell*: Default value is Default value is `NULL`. Set this parameter to a list of synchronizer cells.
- *glitch_protect_cell*: Default value is `NULL`. Specify a comma or space-separated list of glitch protection cell names for the *Glitch Protection Cell Synchronization Scheme*.
- *ignore_nets_clock_path_file_name*: Default value is *ignore_nets_clock_path.txt*. Specify a file containing hierarchical names of

nets (one name per line) so that SpyGlass halts clock propagation along the path when any of these nets is encountered.

- *ignore_num_rtl_buf_invs*: Default value is `many`. Set this parameter to `one` to allow one buffer and inverter. Other possible values are `two` and `none`.
- *num_flops*: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.
- *one_cross_per_dest*: Default value is `yes`. Set this parameter to `no` to report all unsynchronized clock crossings for a destination.
- *strict_double_flop*: Default value is `no`. Set this parameter to `yes` to mark clock crossings as synchronized.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *compute_num_convergences*: Default value is 1. The maximum possible value for `compute_num_convergences` is 10. Set this parameter to any integer to specify the number of convergences to be computed for the same set of synchronizers.
- *coherency_check_type*: Default value is `control`. Set this parameter to `reset` to check convergence issues on control crossings of reset paths only.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *cdc_filter_coherency* (Optional): Use this constraint to specify points at or beyond which no convergence of signals should be reported.
- *no_convergence_check* (Optional): Use this constraint to specify nets that should not be checked for convergence.

NOTE: *If you specify nets by using the no_convergence_check constraint as well as the no_convergence_check parameter, SpyGlass considers the nets specified by both the constraint and parameter.*

- *cdc_attribute* (Optional): Use this constraint to specify mutually exclusive and unrelated signals such that convergence-related violations are suppressed for such signals.

NOTE: *The cdc_attribute constraint is the preferred constraint over the cdc_filter_coherency and the no_convergence_check constraints to specify unrelated signals.*

- *clock* (Optional): Use this constraint to specify clock signals.
- *ip_block* (Optional): Use this constraint to specify IP blocks in your design.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *num_flops* (Optional): Use this constraint to specify the minimum number of flip-flops required in a synchronizer chain.
- *output_not_used* (Optional): Use this constraint to specify a primary output port.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.
- *sync_cell* (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.

- *allow_combo_logic* (Optional): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.

Messages and Suggested Fix

Message 1

The following message appears at the location where same domain signals converge:

`[WARNING] [ACCV1_1] <num> synchronizers <sig-names> converge on <gate-type> '<gate-name>'`

The arguments of the above message are explained below:

Argument	Description
<num>	Number of synchronized signals
<sig-names>	Comma separated list of synchronized destination output net names. When the design is a netlist design and the *report_inst_for_netlist* parameter is set to yes, the rule reports a comma-separated list of synchronized destination output instance names.
<gate-type>	Gate type, such as `MUX`, `combinational gate`, `flop`, `latch`, `sequential library-cell`, or `black-box`
<gate-name>	Name of the gate output on which convergence occurs. When the *report_inst_for_netlist* parameter is set to `yes`, the instance pin is reported.

Potential Issues

This violation appears when multiple same domain synchronized signals converge. The convergence may also occur after a sequential logic.

Consequences of Not Fixing

Convergence between synchronized signals may lead to data coherency issues and may cause chip failure.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. Open *Rule-based spreadsheet - Ac_conv01.csv* to view a summary of all the violations of this rule.

   Each row in this spreadsheet summarizes one violation.
2. In the rule-based spreadsheet, click on the *ID* column of the violation to be debugged. The message-based spreadsheet appears showing the violation details.

   For details on this spreadsheet, see *Message-based spreadsheet - ac_conv_<num>.csv*.
3. View the schematic of the violation by clicking on the link in the *Schematic* column of the message-based spreadsheet.
4. Based on the information in the spreadsheet and the schematic, perform appropriate actions, as described below:
   - ❒ If convergence is reported for signals that are in an IP and you do not want violations to be reported within the IP, specify that IP in the *ip_block* constraint.
   - ❒ If convergence is for static signals, use the *cdc_false_path* or *quasi_static* constraints.
   - ❒ If the path of synchronizers confirms that synchronizers cannot functionally control the converging net at the same time, waive the violation.
   - ❒ If some of the intermediate nets in the path confirm exclusivity between the synchronizers, specify such nets to the *no_convergence_check* parameter.
   - ❒ If convergence is happening at a mux, you may use the *convergence_stop_at_mux* parameter to stop propagation of convergences beyond the mux output.
   - ❒ If you want to stop propagation of convergences beyond a particular net, pin, or instance, specify such net, pin, or instance to the `-stop_points` argument of the *cdc_filter_coherency* constraint.
   - ❒ If you do not want convergence to be reported at a particular net, pin, or instance, specify such net, pin, or instance to the `-conv_gates` argument of the *cdc_filter_coherency* constraint.
   - ❒ If you do not want convergence to be reported for certain set of signals, pins, or instances, which are either destination or sources, specify such set of signals, pins, or instances to the `-unrelated` argument of the *cdc_filter_coherency* constraint.

Message 2

The following message appears at the location where same domain signals converge:

```
[WARNING] [AcCv1_1_coherency01] <num> synchronizers <sig-names>
converge on <gate-type> '<gate-name>' (same source divergence)
```

Potential Issues

This violation appears when a signal diverges, gets synchronized by the same domain synchronizers in the divergent paths, traverses through a sequential logic, and finally converges.

Consequences of Not Fixing

Such convergence may cause data coherency issues that may cause chip failure.

How to Debug and Fix

See *How to Debug and Fix*.

Example Code and/or Schematic

Example 1 - Same Source domain Convergence (Without Any Common Diverging Source Net)

Consider the following schematic of a violation reported by the *Ac_conv01* rule:

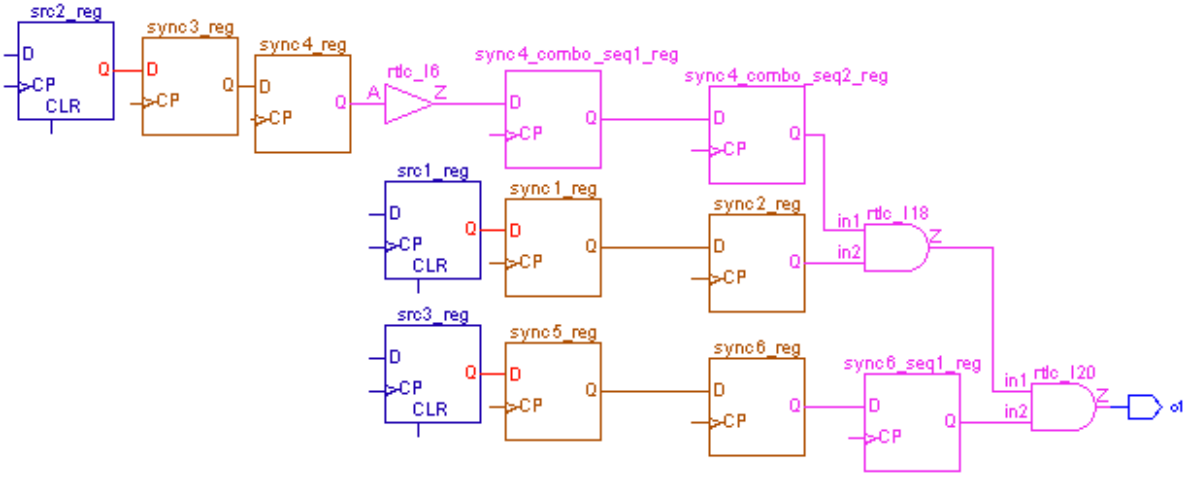


FIGURE 194. Schematic of the Ac_conv01 Rule Violation

In the above example, three same domain synchronizers, `sync1`, `sync3`, and `sync5` synchronized in the same domain by the *Conventional Multi-Flop Synchronization Scheme* are converging on an `AND` gate after the sequential logic.

The *Rule-based spreadsheet - Ac_conv01.csv* and *Message-based spreadsheet - ac_conv_<num>.csv* for this violation is shown in the following figures:

A	B	C	D	E	F	G	H
ID	ONVERGING GAT	GATE TYPE	ER OF SYNCHRON	OURCE DIVERGEN	CALAR SOURC	AX. DEP	MIN. D
D	conv1.o1	combinational gate	3	no	yes	2	0

FIGURE 195. Rule-Based Spreadsheet of the Ac_conv01 Rule

A	B	C	D	E	F	G	
Schematic	Type	Signal Nam	equential Dep	Source(s)	estination Clock	ource Clock(	File
1	Converging Gate	conv1.o1	-	-	-	-	conv.v:
2	Destination flop	conv1.sync 1	0	conv1.src1	conv1.c3	conv1.c1	conv.v:
3	Destination flop	conv1.sync 3	2	conv1.src2	conv1.c3	conv1.c1	conv.v:
4	Destination flop	conv1.sync 5	1	conv1.src3	conv1.c3	conv1.c1	conv.v:

FIGURE 196. Message-Based Spreadsheet of the Ac_conv01 Rule

To fix this violation, remove the convergence.

Schematic Details

The *Ac_conv01* rule highlights the paths from the converging signals to the signal on which they converge. Different colors are used to highlight the following:

- The net on which convergence is reported
- The paths from the output of synchronizers to the net on which convergence is reported
- Synchronizers
- Crossing paths from sources to destination
- Sources of synchronizers.
- Same source reconvergence, that is the path from diverging net till source of the synchronizer (optional)

Example 2 - Same Source Divergence

Consider the following rule-based spreadsheet of the *Ac_conv01* rule:

A	B	C	D	E	F	G	H
ID	ONVERGING GAT	GATE TYPE	ER OF SYNCHRON	OURCE DIVERGEN	CALAR SOURC	AX. DEP	IN. DEPT
C	conv1.o1	combinational gate	2	direct	yes	2	0
F	conv1.o2	combinational gate	2	sequential	yes	2	1

FIGURE 197. Same Source Divergence - Rule-Based Spreadsheet (Ac_conv01)

NOTE: *For information on the columns of the above spreadsheet, see Rule-based spreadsheet - Ac_conv01.csv.*

In the above spreadsheet, the first row shows details of the violation related to same source divergence. Click the *ID* column of this row to open the message-based spreadsheet of this violation. The following figure shows the message-based spreadsheet:

A	B	C	D	E	F	G
Schematic	Type	Signal Name	Sequential Depth	Source(s)	stination Cloc	Source Clock
4	Converging Gate	conv1.o2	-	-	-	-
5	Destination flop	conv1.sync3	2	conv1.src1	conv1.c3	conv1.c1
6	Destination flop	conv1.sync5	1	conv1.src3	conv1.c3	conv1.c1

FIGURE 198. Same Source Divergence - Message-Based Spreadsheet (Ac_conv01)

NOTE: *For information on the columns of the above spreadsheet, see Message-based spreadsheet - ac_conv_<num>.csv.*

The following figure shows the schematic of the violation shown in the above spreadsheet:

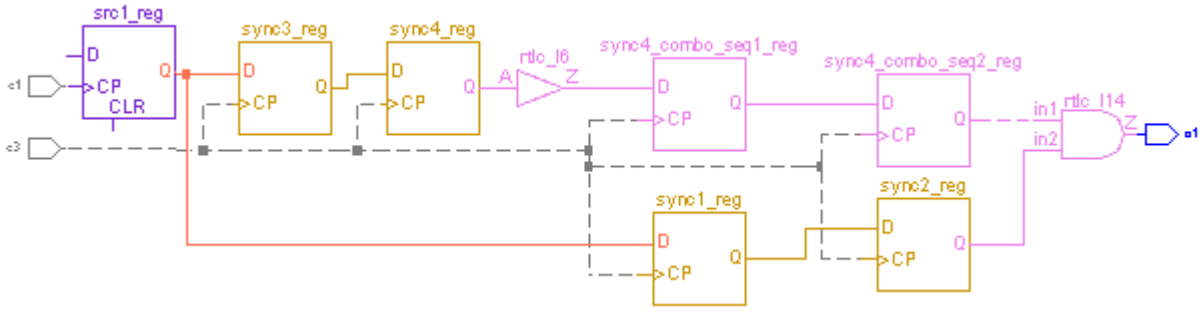


FIGURE 199. Same Source Divergence - Schematic (Ac_conv01)

The above schematic shows the divergence and then convergence of the `src1` source that is synchronized in the same domain.

Example 3 - Common Net Driving Multiple Sources

In the spreadsheet shown in *Figure 197*, the second row shows the details of the violation indicating a common net detected from a sequential fan-in cone. This net drives multiple sources of converging synchronizers.

The following figure shows the message-based spreadsheet of this violation:

A	B	C	D	E	F	G	H	I
Schematic	Type	Signal Name	Sequential Depth	Source(s)	Diverging Net(s)	Destination Clock(s)	Source Clock(s)	File:Line
4	Converging Gate	conv1.o2	-	-	-	-	-	conv.v:33
5	Destination flop	conv1.sync3	2	conv1.src1	conv1.commonSrc	conv1.c3	conv1.c1	conv.v:22
6	Destination flop	conv1.sync5	1	conv1.src3	conv1.commonSrc	conv1.c3	conv1.c1	conv.v:23

FIGURE 200. Message-Based Spreadsheet- The Ac_conv01 rule

NOTE: *For information on the columns of the above spreadsheet, see Message-based*

spreadsheet - ac_conv_<num>.csv.

The following figure shows the schematic of the above violation in which a common net is driving multiple sources:

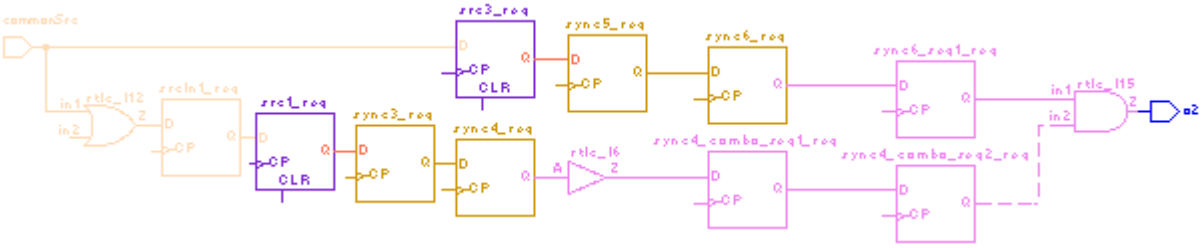


FIGURE 201.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Report and Related Files

The Ac_conv01 rule generates the *Rule-based spreadsheet - Ac_conv01.csv*, *Message-based spreadsheet - ac_conv_<num>.csv*, and *Ac_conv_detail.rpt*.

Rule-based spreadsheet - Ac_conv01.csv

This spreadsheet shows all the *Ac_conv01* rule violations in separate rows.

To open this spreadsheet, right-click on the rule-name header in the *Results* pane, and select the *Spreadsheet Viewer* option from the shortcut menu.

NOTE: *This spreadsheet is generated in the spyglass_reports/clock-reset/ directory.*

Figure 197 shows this spreadsheet. The details of each column of this spreadsheet are described below:

TABLE 3 Columns in a Rule-Based Spreadsheet (Ac_conv01)

Column	Description
ID	Specifies a unique ID for a violation.
CONVERGING GATE	Specifies the name of the converging gate.
GATE TYPE	Specifies the type of converging gate.
NUMBER OF SYNCHRONIZERS	Specifies the number of synchronizers present in the path of the converging gate.
SOURCE DIVERGENCE	Specifies if converging synchronizers are driven by a common source net. Possible values reported in this column are: • Direct: At least one source of synchronized crossing is driving multiple synchronizers. See *Example 2 - Same Source Divergence*. • Combinational: At least one common net is detected from combinational fan-in cone traversal, which drives multiple sources of converging synchronizers. • Sequential: At least one common net is detected from sequential fan-in cone traversal, which drives multiple sources of converging synchronizers. • no: No common net exists
SCALAR SOURCE	Specifies *Yes* or *No* indicating if the source of all the synchronized crossings is scalar or not. For example, in *Figure 194*, *yes* is reported as src1, src2, and src3 are scalar sources.
MAX. DEPTH	Specifies the maximum synchronizer depth from any synchronizer to a converging point. This is an integer value. For example, in *Figure 194*, the maximum number of sequential elements present between `sync4` and `o1` are 2.
MIN. DEPTH	Specifies the minimum synchronizer depth from any synchronizer to a converging point. This is an integer value.
AVG DEPTH	Specifies the average depth from a source to a converging point. This is a float value. For example, in *Figure 194*, Total depth is 3 (2+0+1) and number of sources is 3. Therefore, the average depth shown in the spreadsheet in *Figure 197* is 1.

Column	Description
SAME DEPTH CONVERGENCE	Specifies *yes* when multiple synchronizers exist with the same depth in convergence. Else, the value No is reported. For example, in *Figure 194*, *no* is reported as the paths have different depths: 0,1, and 2.
WAIVED	Specifies if the reported violation is waived.

NOTE: *If you run the Ac_conv01 rule in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the corresponding Message-based spreadsheet - ac_conv_<num>.csv. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.*

Message-based spreadsheet - ac_conv_<num>.csv

This spreadsheet shows details of the selected violation.

To open this spreadsheet, click on the link in the *ID* column of the rule-based spreadsheet. Alternatively, double-click on the violation of this rule from GUI.

NOTE: *This spreadsheet is generated in the spyglass_reports/clock-reset/Ac_conv01/ directory.*

Figure 198 shows this spreadsheet. In this spreadsheet, there is a separate row for each converging destination signal.

The details of each column of this spreadsheet are described below:

TABLE 4 Columns in a Message-Based Spreadsheet (Ac_conv01)

Column	Description
Schematic	Shows a link for schematic.
Type	Specifies any of the following types: • Converging Gate • Destination flop • Destination latch • Destination library-cell • Destination black-box • Destination port The first row is for a converging gate.
Signal Name	Specifies the name of the signal.
Sequential Depth	Specifies the depth from a synchronizer till the converging gate.
Source(s)	Specifies the source of the signal.
Diverging Net(s)	(Optional) Specifies the name of the common net that drives multiple sources. For details, see *Example 3 - Common Net Driving Multiple Sources*. Note that this column appears only in case of a common net driving multiple sources.
Destination Clock(s)	Specifies the destination clock of the signal.
Source Clock(s)	Specifies the source clocks of the signal.
File:Line	Specifies the design file name and line number where the information pertaining to each row in this spreadsheet is present

NOTE: *In the message-based spreadsheet, some column values are not applicable for the row of type "Converging Gate". The cells in such columns have the value "-".*

Ac_conv_detail.rpt

The Ac_conv_detail report contains details of all the violations detected by the Ac_conv01, Ac_conv02, and Ac_conv03 rules. The report is not generated by default. Specify `set_option report Ac_conv_detail` in the project file to generate this report.

The report consists of following sections:

- Section A: Lists the sequential convergence of same-domain signals synchronized in the destination domain (Ac_conv01)
- Section B: Lists the combinational convergence of same-domain signals synchronized in the same destination domain (Ac_conv02)
- Section C: Convergence of different-domain signals synchronized in the same destination domain(Ac_conv03)

The following figure shows a sample Ac_conv_detail report.

A. Sequential convergence of same-domain signals synchronized in the same destination domain(Ac_conv01)

S. No.	Converging Gate	Signal Name	Source Name	Destination Clock	Source Clock	File:Line
1.	block2.w4	block2.c1_d1	"block2.c1_s1"	"block2.clk2"	"block2.clk1"	conv.v:11
		block2.inactives_c1_c2	"-"	"block2.clk2"	"block2.clk1"	conv.v:1
2.	block2.w3	block2.c1_d1	"block2.c1_s1"	"block2.clk2"	"block2.clk1"	conv.v:11
		block2.ss_c1_c2	"-"	"block2.clk2"	"block2.clk1"	conv.v:1

B. Combinational convergence of same-domain signals synchronized in the same destination domain(Ac_conv02)

S. No.	Converging Gate	Signal Name	Source Name	Destination Clock	Source Clock	File:Line
1.	block2.w6	block2.c1_d1	"block2.c1_s1"	"block2.clk2"	"block2.clk1"	conv.v:11
		block2.s_c1_c2	"-"	"block2.clk2"	"block2.clk1"	conv.v:1
2.	block2.w2	block2.c1_d1	"block2.c1_s1"	"block2.clk2"	"block2.clk1"	conv.v:11
		block2.s1_c1_c2	"-"	"block2.clk2"	"block2.clk1"	conv.v:1
3.	block2.w7	block2.s0_c1_c2	"-"	"block2.clk2"	"block2.clk1"	conv.v:1
		block2.s1_c1_c2	"-"	"block2.clk2"	"block2.clk1"	conv.v:1
4.	block2.w	block2.c1_d1	"block2.c1_s1"	"block2.clk2"	"block2.clk1"	conv.v:11
		block2.c2_d1	"block2.c2_s1"	"block2.clk2"	"block2.clk1"	conv.v:14

C. Convergence of different-domain signals synchronized in the same destination domain(Ac_conv03)

This section is empty because either rule Ac_conv03 has not been run or no violation is found for the rule.

FIGURE 202. Sample Ac_conv_detail Report

Ac_conv02

Reports same-domain signals that are synchronized in the same destination domain and converge before sequential elements.

When to Use

Use this rule to check combinational convergences of the same domain signals synchronized in the same destination domain.

Prerequisites

Following are the prerequisites for running this rule:

- Specify clock signals in any of the following ways:
  - By using the *clock* constraint.
  - By setting the *use_inferred_clocks* parameter to `yes` to enable auto-generation of clock signals.
  - By using a combination of both the above methods.
- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The details of the *Ac_conv02* rule are covered under the following topics:

- *Reasons for the Ac_conv02 Rule Violation*
- *Checking the Gray Encoding of Converging Signals*
- *Features of the Ac_conv02 Rule*
- *Handling MUXes by the Ac_conv02 Rule*

Reasons for the Ac_conv02 Rule Violation

The *Ac_conv02* rule reports same source signals that converge after satisfying the following properties:

- Signals from the same source domain are synchronized in the same destination domain.
- The signals are synchronized using *Conventional Multi-Flop Synchronization Scheme*, *Synchronizing Cell Synchronization Scheme*, or *Qualifier Synchronization Scheme Using qualifier -crossing*.

- Synchronized signals converge before encountering a sequential element (a flip-flop, latch, or sequential library cell).

The same source signals are further analyzed to determine whether a specific signal in the fan-in cone is driving these signals or the same signal is getting synchronized multiple times. You can control the depth of this fan-in cone through the *conv_src_seq_depth* parameter.

The following figure shows an example of convergence of signals coming from same source clock domains.

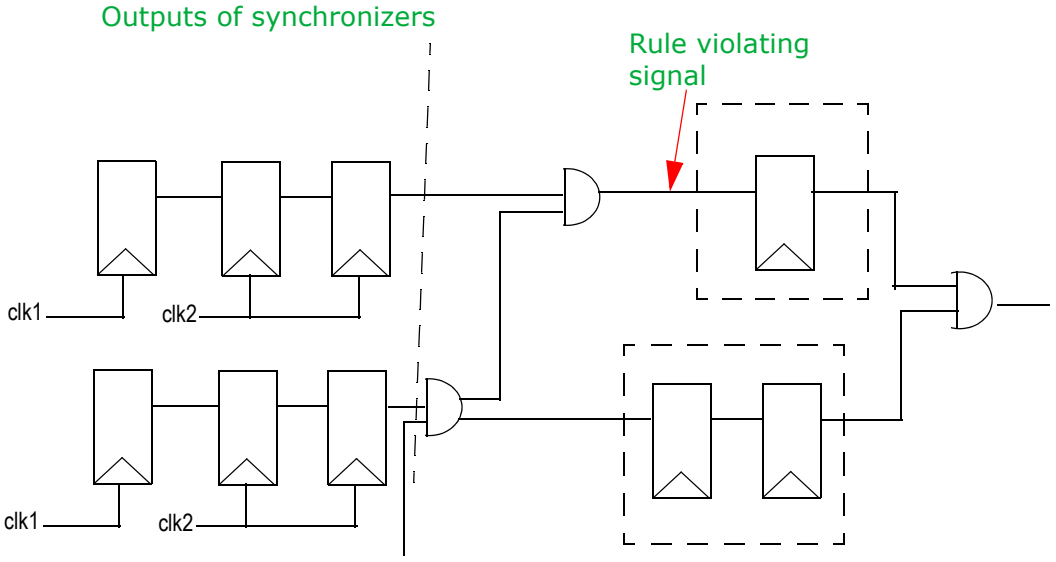


FIGURE 203. Convergence of Signals from Same Source Domain Converging after Sequential Elements

The *Ac_conv02* rule also reports a violation when a signal diverges and then converges. After diverging, the signal is synchronized by the same-domain synchronizers on divergent paths. In this case, the synchronized signal from each divergent path does not encounter any sequential element (flip-flop, latch, or sequential library cell) before converging.

The following figure shows such scenario of the *Ac_conv02* rule violation:

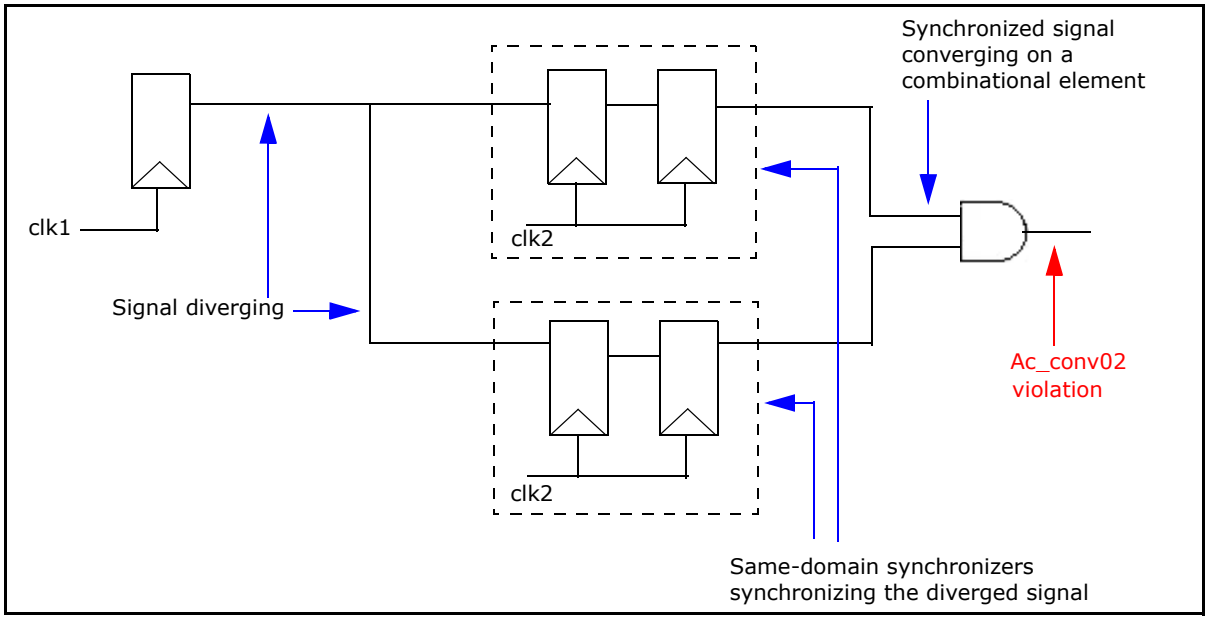


FIGURE 204. Example- The Ac_conv02 Rule Violation

Checking the Gray Encoding of Converging Signals

The *Ac_conv02* rule functionally verifies if converging signals are gray-encoded. A set of signals are gray-encoded if at most one of the signal change values between two consecutive clock cycles. The result of this check can be any one of the following:

- `PASSED`: Indicates that signals are gray-encoded.
- `FAILED`: Indicates that signals are violating gray-encoding.
- `PP`: Indicates that the gray-encoding property was partially proved. In this case, SpyGlass is not able to prove or falsify the property in the specified time.
- `DISABLED`: Indicates that functional check for gray-encoding could not be performed.

  Following are some of the possible reasons for which the functional check is performed for gray encoding:

- Non-availability of advanced SpyGlass CDC solution license
- Presence of multi-top design
- Over-constraining
- *fa_msgmode* parameter is set to none
- Converging nets have two or more domains converging in their fan-in
- Destination is a synchronizer black box instance

Specialized Gray Encoding Check on FIFO

Previously, to check gray encoding, usually the following two steps were used:

1. Run structural rules: Previously, you used to run structural rules (such as the cdc_verify_struct goal), analyze the violations, and then apply the recommended constraint (such as, the *cdc_filter_coherency* constraint for Ac_conv02) to suppress the violations.
2. Run formal rules: Then, you used to run the formal rules (such as the, cdc_verify goal) with the above created structural constraints, to perform necessary formal checks.

For example, consider the below schematic where bus sync[0:3] merges with sync_1. In this case, the Ac_conv02 rule reports convergence on (sync[0:3],sync_1) in structural run.

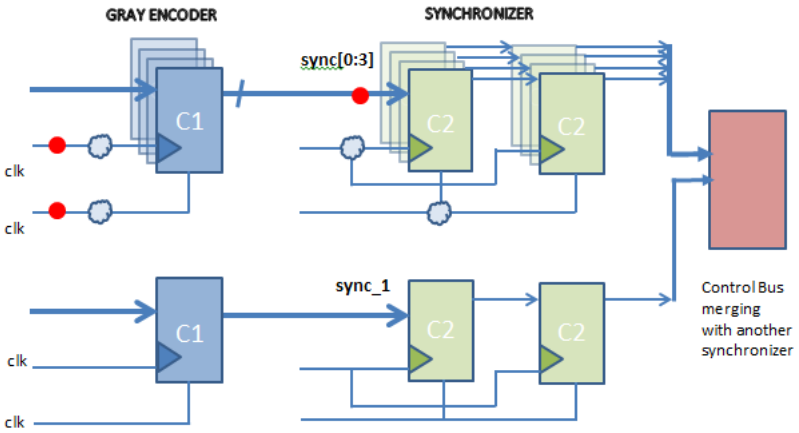


FIGURE 205.

In the above case, after running the structural rules, you could:

- Check gray encoding on (sync[0:3],sync_1). For this, you run the formal rules.
- Check gray encoding on sync[0:3] as well - To check this, you used to specify the *gray_signals* constraint on sync[0:3], which was read by the Ac_conv05 rule and the signal was checked for gray encoding.

The Ac_conv02 rule also supports specialized gray encoding checks. You can now use the `cdc_gen_unrelated_coherency` parameter to generate an SGDC file, which contains the *cdc_filter_coherency* and the *gray_signals* constraints. The generated file, as shown below, contains constraints for Ac_conv02 violations that have at least one vector bus along with either one or more scalar signal or one or more vector buses:

```
# Convergence Point <ConvergencePointNameInViolation>

current_design <design top name>
cdc_filter_coherency -unrelated <list of synchronizers
converging in Ac_conv02>
gray_signals -name <vector-bus-in-above-constraint-if-
applicable>
gray_signals -name <next-vector-bus-in-above-constraint-if-
```

applicable>

….

For the scenario described above, the following SGDC file is generated when the cdc_gen_unrelated_coherency parameter is set to yes:

```
cdc_filter_coherency -unrelated sync[0:3] sync_1
gray_signals -name sync[0:3]
```

You can comment or uncomment the above constraints and use it in the next formal run. In the subsequent run, the Ac_conv02 rule violations are suppressed due to the -unrelated argument and the Ac_conv05 rule checks for gray encoding on the signals specified with the *gray_signals* constraint.

If you want to check both (sync[0:3],sync_1) and sync[0:3] in the above-mentioned schematic, comment the corresponding cdc_filter_coherency -unrelated constraint in the generated file.

When you set the cdc_gen_unrelated_coherency parameter to yes, the *Ac_conv02Setup01* rule reports the following info message for the generated SGDC file:

```
SGDC file generated for design: <designRootName> using
candidate combinational convergence
```

You can click the info message to load the sgdc file in RTL and constraint file viewer. You can then edit the file and use it in the next run.

Features of the Ac_conv02 Rule

Following are the features of this rule:

- It reports only one violation if same set of signals converge in more than one path.
- When the convergence happens at multiple points in a path, this rule reports a violation at a point that has most number of signals converging.
- It ignores convergence of clock domain crossing signals specified by the *cdc_false_path* constraint
- It checks both scalar and bus signals
- It does not allow flip-flops in the output cone of synchronizing signals. It reports only combinational convergences.

- It reports a violation on internal nets if the output cone of that net is blocked due to *set_case_analysis* or design power/ground.

Handling MUXes by the Ac_conv02 Rule

The *Ac_conv02* rule treats muxes in a special way. On encountering RTL muxes, this rule propagates convergences of only one data input and all control inputs. If several data inputs have convergences, one of them is selected randomly.

The rule uses this scheme to minimize the number of false convergences reported by this rule.

Note that convergences from different data inputs of the mux are not viable functionally because control inputs select one of them at a time.

Parameter(s)

- *no_convergence_check*: Default value is `NULL`. Specify net names that should not be checked for convergence.
- *convergence_stop_at_mux*: Default value is `no`. Set this parameter to `yes` to stop the propagation of relevant signals whenever an RTL mux is encountered.
- *conv_sync_as_src*: Default value is `no`. Set this parameter to `yes` to check convergence for synchronizers that are also used as a source in other crossings.
- *conv_src_seq_depth*: Default value is -1. Specify a positive integer value to set the sequential depth to be skipped while detecting the common net of the source of synchronizers. Other possible value is 0.
- *conv_clock_reset_path*: Default value is `no`. Set this parameter to yes to enable convergence detection of the synchronizer propagation through the clock and reset pin.
- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_dump_hybrid*: Default value is `partial`. Set this parameter to `all` to generate SVA for all the types of assertions (pass, fail, partially-proved). The other possible value are `pass`, `fail`, `+fail`, and `none`.

- *fa_grayhold*: Default value is `no`. Set this parameter to `yes` to checks for gray-encoding at the output of a destination instance with respect to a destination clock.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *fa_atime*: Default value is `20`. Set this parameter to a positive integer value to specify a time that the tool should take to perform functional analysis per assertion.
- *fa_abstract*: Default value is *Ac_handshake01*, *Ac_glitch03*. Set the value of this parameter to *Ac_cdc08* to enable abstraction for this rule. Other possible values are `all`, `none` or a list of any of rules:

  Ac_handshake01, *Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, *Ac_fifo01*, and *Ac_glitch03*.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *cdc_dump_assertions*: Default value is "". Set this parameter to `sva` to generate SystemVerilog Assertions (SVA) corresponding to the rules and the design assumptions specified in an SGDC file.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *show_source_in_spreadsheet*: Default value is `yes`. Set this parameter to `no` to generate a link from the spreadsheet of the *Ac_conv01*, *Ac_conv02*, or *Ac_conv03* rules to the message-based spreadsheet of *The Ac_sync_group Rules* showing the source of synchronizers.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *allow_half_sync*: Default value is `yes`. Set this parameter to `no` to not treat half synchronizers as valid synchronizers.
- *clock_gate_cell*: Default value is `NULL`. Set the value of this parameter to a comma or space-separated list of clock-gating cell names for the *Clock-Gating Cell Synchronization Scheme*.
- *enable_and_sync*: Default value is `no`. Set this parameter to `yes` to enable the *AND Gate Synchronization Scheme*.

- *enable_mux_dest_domain*: Default value is `none`. Set this parameter to `none` to turn off this parameter. Other possible values are `mux`, `enable`, `gp`, `cg`, `and`, `all`, `yes`, and `no`.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Synchronized Enable Synchronization Scheme*.
- *enable_sync_cell*: Default value is Default value is `NULL`. Set this parameter to a list of synchronizer cells.
- *glitch_protect_cell*: Default value is `NULL`. Specify a comma or space-separated list of glitch protection cell names for the *Glitch Protection Cell Synchronization Scheme*.
- *ignore_nets_clock_path_file_name*: Default value is *ignore_nets_clock_path.txt*. Specify a file containing hierarchical names of nets (one name per line) so that SpyGlass halts clock propagation along the path when any of these nets is encountered.
- *num_flops*: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.
- *one_cross_per_dest*: Default value is `yes`. Set this parameter to `no` to report all unsynchronized clock crossings for a destination.
- *strict_double_flop*: Default value is `no`. Set this parameter to `yes` to mark clock crossings as synchronized.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *compute_num_convergences*: Default value is 1. The maximum possible value for compute_num_convergences is 10. Set this parameter to any integer to specify the number of convergences to be computed for the same set of synchronizers.
- *coherency_check_type*: Default value is `control`. Set this parameter to `reset` to check convergence issues on control crossings of reset paths only.
- *cdc_gen_unrelated_coherency*: Default value is `no`. Set this parameter to `yes` to generate the `unrelated_coherent_signals.sgdc` file.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *conv_all_mux_data_pins*: Default value is `no`. Set the value of the parameter to `yes` to enable synchronizer propagation through all data pins of muxes for convergence detection.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *cdc_filter_coherency* (Optional): Use this constraint to specify points at or beyond which no convergence of signals should be reported.
- *no_convergence_check* (Optional): Use this constraint to specify nets that should not be checked for convergence.

NOTE: *If you specify nets by using the no_convergence_check constraint as well as the no_convergence_check parameter, SpyGlass considers the nets specified by both the constraint and parameter.*

- *cdc_attribute* (Optional): Use this constraint to specify mutually exclusive and unrelated signals such that convergence-related violations are suppressed for such signals.

NOTE: *The cdc_attribute constraint is the preferred constraint over the cdc_filter_coherency and the no_convergence_check constraints to specify*

unrelated signals.

- *clock* (Optional): Use this constraint to specify clock signals.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *monitor_time* (Optional): Use to specify the design initialization time frames during simulation. The rest of the simulation time is considered as the design's functional time.
- *meta_design_hier* (Optional): Use to specify the test bench name and design instance name in the SGDC file.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *num_flops* (Optional): Use this constraint to specify the minimum number of flip-flops required in a synchronizer chain.
- *output_not_used* (Optional): Use this constraint to specify a primary output port.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.
- *sync_cell* (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *allow_combo_logic* (Optional): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.
- *breakpoint* (Optional): Use this constraint to specify breakpoints in a design.
- *define_tag* (Optional): Use this constraint to define a named condition for application of certain stimulus at the top port or an internal node.
- *ip_block* (Optional): Use this constraint to specify IP blocks in a design.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

- *watchpoint* (Optional): Use this constraint to specify watch points in a design.

Messages and Suggested Fix

Message 1

The following message appears for the `FAILED` or `Others (Constraints-Conflict)` status at the location where signals converge:

```
[AcCv2_1] [ERROR] <num> synchronizers <sig-names> converge on <gate-type> '<gate-name>'. Gray encoding check: '<FAILED | Others (Constraints-Conflict)>'
```

The arguments of the above message are explained below:

TABLE 5 Argument details of the Ac_conv02 rule

Argument	Description
<num>	Number of synchronized signals
<sig-names>	Comma separated list of synchronized destination output net names. When the design is a netlist design and the *report_inst_for_netlist* parameter is set to yes, the rule reports a comma-separated list of synchronized destination output instance names.
<gate-type>	Gate type, such as `MUX`, `combinational gate`, `flop`, `latch`, `sequential library-cell`, or `black-box`
<gate-name>	Name of the gate output on which convergence occurs. When the *report_inst_for_netlist* parameter is set to `yes`, the instance pin is reported.

Potential Issues

This violation appears when multiple same domain synchronized signals converge.

Consequences of Not Fixing

If you do not gray encode converging signals, there may be data coherency issues and may cause chip failure.

How to Debug and Fix

To debug and fix the violation, perform the following steps:

1. Open the rule-based spreadsheet.

   For details on this spreadsheet, see *Rule-based spreadsheet - Ac_conv02.csv*.

2. In the rule-based spreadsheet, click on the *ID* column of the violation to be debugged. The message-based spreadsheet appears showing the violation details.

   For details on this spreadsheet, see *Message-based spreadsheet - ac_conv_<num>.csv*.

3. View the schematic of the violation by clicking on the link in the *Schematic* column of the message-based spreadsheet.

   In the schematic, analyze the path of convergence of synchronizers.

4. Based on the information in the spreadsheet and the schematic, perform appropriate actions, as described below:

   - ❒ If convergence is reported for signals that are in an IP and you do not want violations to be reported within the IP, specify that IP in the *ip_block* constraint.
   - ❒ If convergence is for static signals, use the *cdc_false_path* or *quasi_static* constraints.
   - ❒ If the path of synchronizers confirms that synchronizers cannot functionally control the converging net at the same time, waive the violation.
   - ❒ If some of the intermediate nets in the path confirm exclusivity between the synchronizers, specify such nets to the *no_convergence_check* parameter.
   - ❒ If convergence is happening at a mux, you may use the *convergence_stop_at_mux* parameter to stop propagation of convergences beyond the mux output.
   - ❒ If you want to stop propagation of convergences beyond a particular net, pin, or instance, specify such net, pin, or instance to the `-stop_points` argument of the *cdc_filter_coherency* constraint.

- If you do not want convergence to be reported at a particular net, pin, or instance, specify such net, pin, or instance to the `-conv_gates` argument of the *cdc_filter_coherency* constraint.
- If you do not want convergence to be reported for certain set of signals, pins, or instances, which are either destination or sources, specify such set of signals, pins, or instances to the `-unrelated` argument of the *cdc_filter_coherency* constraint.

FAILED Status

Open the *Waveform Viewer* window corresponding to the message, and check the marker that appears on the waveform. This marker is positioned at a transition where at least two of the nets that drive the converging signals are changing at the same time. Therefore, this specific transition is a *witness* to the failure.

In such cases:

- Analyze the logic that drives the converging nets to check for the presence of an explicit gray encoder or an FSM that guarantees that no two converging nets change at the same time.
- Analyze the logic behind the convergence point. If this logic functionally ensures that all the converging signals cannot control the convergence point at the same time, the violation can be waved.

Following are some of the reasons that may cause false failures:

- Check for the potential reset/clear signal causing such violation. Provide the reset/clear in the constraint file as a reset.
- Check if the setup (*clock*, *reset*, *set_case_analysis*, *input* constraints) is correct and complete. Use the *Formal Setup Rules* to check for the correctness of the setup.
- Check the initial state values in the *Waveform Viewer* window. If the values are not correct, provide correct initial state in the constraints file or provide a VCD file from which an initial state can be loaded.
- View the *Waveform Viewer* window. In this window, you will notice that at least two nets that drive the converging double-flop synchronizers will be changing at the same time during a clock cycle. Analyze the logic behind the convergence point. If this logic functionally ensures that all

the converging signals cannot control the convergence point at the same time, the violation can be waved.

Convergence between separately synchronized signals can lead to data-coherency issues. It is recommended to use a common synchronizer for all signals that are converging or verify that two or more converging signals do not control the net on which it is converging at the same time. If signals pass the gray-encoding check, it means that the destination domain will never operate on a value not produced in the source domain. This must be the case for fifo pointers so that the fifo operates correctly.

For details, see *Performing Functional Analysis in SpyGlass CDC*.

Message 2

The following message appears for the FAILED or Others (Constraints-Conflict) status at the location where signals diverge and then converge:

```
[AcCv2_1_coherency02] [ERROR] <num> synchronizers <sig-names>
converge on <gate-type> '<gate-name>' (same source divergence).
Gray encoding check: '<status>'
```

Potential Issues

This violation appears when a signal diverges, gets synchronized by the same-domain synchronizers on the divergent paths, and then finally the signal converge on a combinational element.

Consequences of Not Fixing

If you do not gray encode converging signals, there may be data coherency issues that may cause chip failure.

How to Debug and Fix

See *How to Debug and Fix*.

Message 3

The following message appears for the `Partially-Proved` status at the location where signals converge:

```
[AcCv2_3] [WARNING] <num> synchronizers <sig-names> converge on
<gate-type> '<gate-name>'. Gray encoding check: 'Partially-
Proved'
```

For information on the arguments of the above message, see *Table 5*.

Potential Issues

This violation appears when multiple same domain synchronized signals converge.

Consequences of Not Fixing

If you do not gray encode converging signals, there may be data coherency issues and may cause chip failure.

How to Debug and Fix

See *How to Debug and Fix*.

In addition, for the Partially Proved status, perform the following actions:

- Increase assertion run-time by using *fa_atime* parameter.
- Use the incremental analysis approach by using the *fa_propfile* parameter.
- Use the *fa_abstract* parameter to apply abstraction technique to reduce complex verification problem into simpler and solvable problem.

Message 4

The following message appears for the Partially-Proved status at the location where signals diverge and then converge:

```
[AcCv2_3_coherency02] [WARNING] <num> synchronizers <sig-names>
converge on <gate-type> '<gate-name>' (same source divergence).
Gray encoding check: 'Partially-Proved'
```

Potential Issues

This violation appears when a signal diverges, gets synchronized by the same-domain synchronizers on the divergent paths, and then finally the signal converge on a combinational element.

Consequences of Not Fixing

If you do not gray encode converging signals, there may be data coherency issues and may cause chip failure.

How to Debug and Fix

See *How to Debug and Fix*.

In addition, for the Partially Proved status, perform the following actions:

- Increase assertion run-time by using *fa_atime* parameter.
- Use the incremental analysis approach by using the *fa_propfile* parameter.
- Use the *fa_abstract* parameter to apply abstraction technique to reduce complex verification problem into simpler and solvable problem.

Message 5

The following message appears for the `Passed` or `DISABLED` status at the location where signals converge:

```
[AcCv2_4] [WARNING/INFO] <num> synchronizers <sig-names>
converge on <gate-type> '<gate-name>'. Gray encoding check:
'<Passed | DISABLED>'
```

For information on the arguments of the above message, see *Table 5*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix*.

If the status is DISABLED, check if any of the following situations is resulting in blocking the rule to perform functional check:

- In case of non-availability of advanced SpyGlass CDC solution license, check the *Ac_license01* rule violation, if any.
- In case of more than one top specified, check the *Ac_multitop01* rule violation, if any.
- In case of over-constraining, check the *Ac_sanity04* rule violation, if any.
- Check if the *fa_msgmode* parameter is set to `none`.

If the status is PASSED, no debugging is required.

Example Code and/or Schematic

Example 1

Consider the following schematic of a violation reported by the *Ac_conv02* rule:

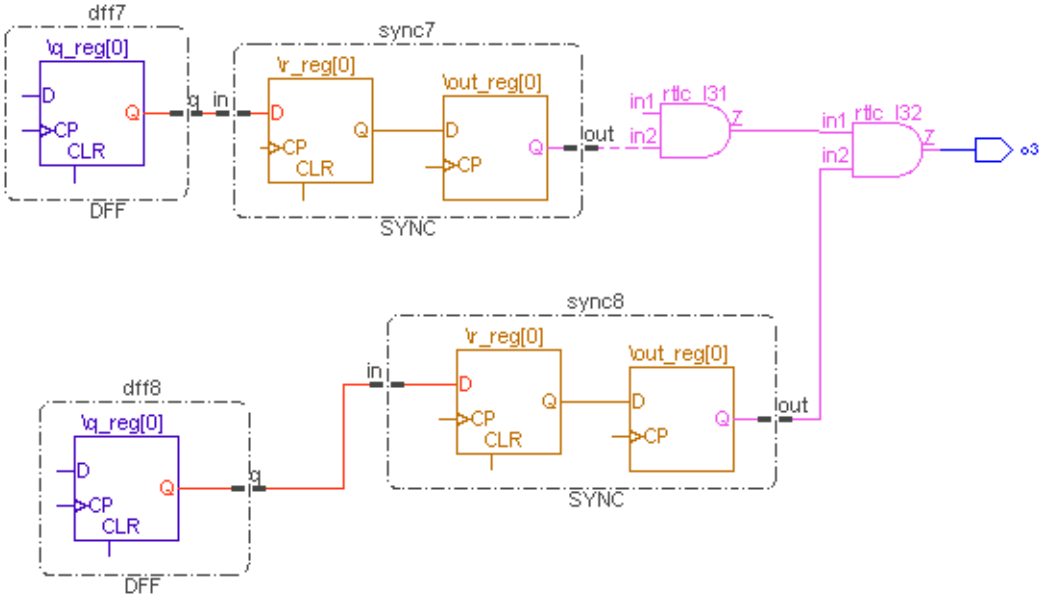


FIGURE 206. Schematic of the Ac_conv02 Rule Violation

In the above example, two same domain synchronizers, `sync7` and `sync8`, which are synchronized in the same domain by using *Conventional Multi-Flop*

Synchronization Scheme, are converging on the AND gate and are not gray-encoded.

To fix this violation, check if this convergence is expected. If it is expected, add a gray-encoder for the sources. However, if it is not expected, remove the convergence.

Schematic Details

The *Ac_conv02* rule highlights the paths from the converging signals to the signal on which they converge. Different colors are used to highlight the following:

- The net on which convergence is reported
- The paths from the output of synchronizers to the net on which convergence is reported
- Synchronizers
- Crossing paths from sources to destination
- Sources of synchronizers.
- Same source reconvergence, that is the path from diverging net till source of the synchronizer (optional)

Example 2 - Same Source Divergence

Consider the following rule-based spreadsheet of the *Ac_conv02* rule:

A	B	C	D	E	F	G
ID	CONVERGING GATE	GATE TYPE	OF SYNCHF	STATUS	SOURCE DIVERGENC	SCALAR SOURCE
C	conv1.o2	combinational gate	2	FAILED	sequential	yes
F	conv1.o1	combinational gate	2	FAILED	direct	yes

FIGURE 207.

In the above spreadsheet, the second row shows details of the violation related to same source divergence. Click the ID column of this row to open the message-based spreadsheet of this violation. The following figure shows the message-based spreadsheet:

A	B	C	D	E	F	G
Schematic	Type	Signal Name	Source(s)	Diverging Net(s	estination Clock(	Source Clock(
4	Converging Gate	conv1.o1	-	-	-	-
5	Destination flop	conv1.sync1	conv1.src1	conv1.src1	conv1.c3	conv1.c1
6	Destination flop	conv1.sync3	conv1.src1	conv1.src1	conv1.c3	conv1.c1

FIGURE 208.

The following figure shows the schematic of the violation shown in the above spreadsheet:

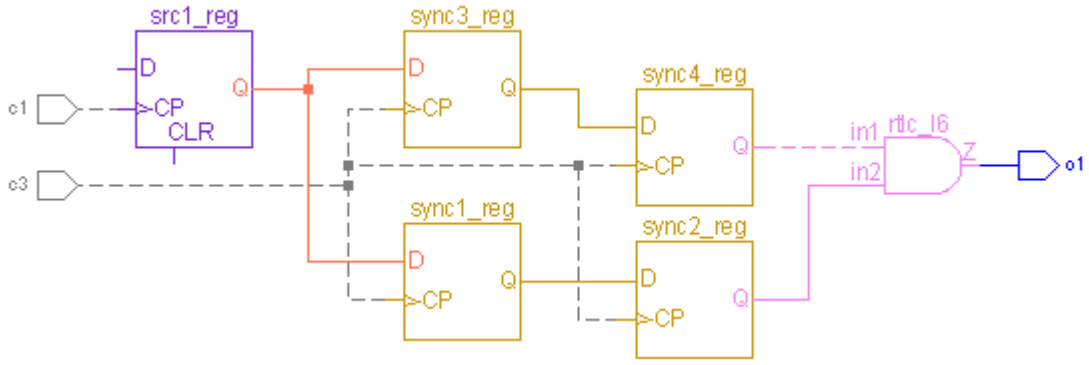


FIGURE 209.

The above schematic shows the divergence and then convergence of the `src1` source that is synchronized in the same domain.

Example 3 - Common Net Driving Multiple Sources

In the spreadsheet shown in *Figure 207*, the first row shows the details of the violation indicating a common net detected from a sequential fan-in cone. This net drives multiple sources of converging synchronizers.

The following figure shows the message-based spreadsheet of this violation:

A	B	C	D	E	F	G
Schematic	Type	Signal Nam	Source(s)	Diverging Net(s	estination Clock	ource Clock(
1	Converging Gate	conv1.o2	-	-	-	-
2	Destination flop	conv1.sync 3	conv1.src1	conv1.commonS rc	conv1.c3	conv1.c1
3	Destination flop	conv1.sync 5	conv1.src3	conv1.commonS rc	conv1.c3	conv1.c1

FIGURE 210.

The following figure shows the schematic of the above violation in which a common net is driving multiple sources:

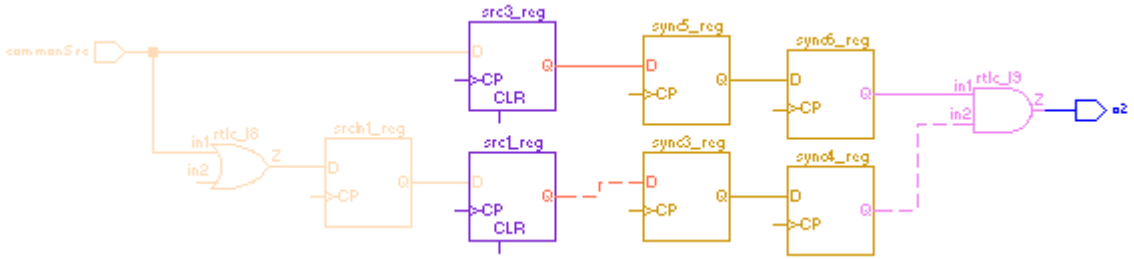


FIGURE 211.

Default Severity Label

The rule severity varies according to the assertion status as follows:

- `FAILED`: Error
- `Partially-Proved/DISABLED`: Warning
- `PASSED`: Info
- `Others(Constraints-Conflict)`: Error

Rule Group

`ADV_CLOCKS`

Reports and Related Files

- Rule-based spreadsheet - Ac_conv02.csv

  This spreadsheet shows all the *Ac_conv02* rule violations in separate rows. It is generated in the *spyglass_reports/clock-reset/* directory.

  To open this spreadsheet, right-click on the rule-name header in the *Results* pane, and select the *Spreadsheet Viewer* option from the shortcut menu.

  Figure 207 shows an example of the rule-based spreadsheet of this rule.

  The details of each column of this spreadsheet are described below:

Column	Description
ID	Specifies a unique ID for a violation.
CONVERGING GATE	Specifies the name of the converging gate.
GATE TYPE	Specifies the type of converging gate.
NUMBER OF SYNCHRONIZERS	Specifies the number of synchronizers present in the path of the converging gate.
STATUS	Specifies the gray-encoding status: *FAILED*, *Partially-Proved*, *PASSED/DISABLED*, or *Others(Constraints-Conflict)*
SOURCE DIVERGENCE	Specifies if converging synchronizers are driven by a common source net. Possible values reported in this column are: • *Direct*: At least one source of synchronized crossing is driving multiple synchronizers. • *Combinational*: At least one common net is detected from combinational fan-in cone traversal, which drives multiple sources of converging synchronizers. • *Sequential*: At least one common net is detected from sequential fan-in cone traversal, which drives multiple sources of converging synchronizers. • *No*: No common net exists
SCALAR SOURCE	Specifies *Yes* or *No* indicating if the source of all the synchronized crossings is scalar or not.
WAIVED	Specifies if the reported violation is waived.

NOTE: *If you run the Ac_conv02 rule in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the*

corresponding Message-based spreadsheet - ac_conv_<num>.csv. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.

NOTE:

- Message-based spreadsheet - ac_conv_<num>.csv

  This spreadsheet shows details of the selected violation. It is generated in the *spyglass_reports/clock-reset/Ac_conv02/* directory.

  To open this spreadsheet, click on the link in the *ID* column of the rule-based spreadsheet. Alternatively, double-click on the violation of this rule from GUI.

  Figure 208 shows an example of the message-based spreadsheet of this rule.

  In this spreadsheet, there is a separate row for each converging destination signal.

  The details of each column are described below:

Column	Description
Schematic	Shows a link for schematic.
Type	Specifies any of the following types: • Converging Gate • Destination flop • Destination latch • Destination library-cell • Destination black-box • Destination port The first row is for a converging gate.
Signal Name	Specifies the name of the signal.
Source(s)	Specifies the source of the signal.
Diverging Net(s)	(Optional) Specifies the name of the common net that drives multiple sources. For details, see *Example 3 - Common Net Driving Multiple Sources*. Note that this column appears only in case of a common source driving multiple nets.
Destination Clock	Specifies the destination clock of the signal.

Column	Description
Source Clock(s)	Specifies the source clocks of the signal.
File:Line	Specifies the design file name and line number where the information pertaining to each row in this spreadsheet is present.

NOTE: *In the message-based spreadsheet, some column values are not applicable for the row of type "Converging Gate". The cells in such columns have the value "-".*

- Ac_conv02.<ID>.OverConstrainInfo: This file contains details of conflicting constraints. For details, see *Overconstrain Info File*.
- Ac_conv_detail.rpt: This report contains details of all the violations detected by the Ac_conv01, Ac_conv02, and Ac_conv03 rules. For details, see *Ac_conv_detail.rpt*.

Ac_conv02Setup01

Setup rule for Ac_conv02

When to Use

This rule runs by default.

Description

This rule generates an info message about the `unrelated_coherent_signals.sgdc` file generated for the violations of the Ac_conv02 rule.

Parameter(s)

- *cdc_gen_unrelated_coherency*: Default value is `no`. Set this parameter to `yes` to generate the `unrelated_coherent_signals.sgdc` file.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears when a SGDC file is generated for any converging signal:

`SGDC file generated for design: '<design_name>' using candidate combinational convergence"`

Potential Issues

NA

Consequences of Not Fixing

NA

How to Debug and Fix

NA

Default Severity Label

Info

Reports and Related Files

The `unrelated_coherent_signals.sgdc` file is generated for the *Ac_conv02* rule violations.

Ac_conv03

Checks different domain signals synchronized in the same destination domain and are converging

When to Use

Use this rule to check convergences of different domain signals that are synchronized in the same domain.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By using the automatically generated clock signals after setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The details of the *Ac_conv03* rule are covered under the following topics:

- *Reasons for the Ac_conv03 Rule Violation*
- *Features of the Ac_conv03 Rule*
- *Handling MUXes by the Ac_conv03 Rule*

Reasons for the Ac_conv03 Rule Violation

The *Ac_conv03* rule reports different domain signals that converge after satisfying the following conditions:

- Signals are synchronized by using any of the following schemes:
  - *Conventional Multi-Flop Synchronization Scheme*
  - *Synchronizing Cell Synchronization Scheme*
  - *Qualifier Synchronization Scheme*
- Signals from different source domains are synchronized in the same destination domain.
- Synchronized signals converge before a sequential element on some net of the design.

  Use the *conv03_report_seq_conv* parameter to enable reporting of

convergences after sequential elements.

NOTE: *If same convergence is seen on two nets, this rule reports convergence on the net closest to the converging signal.*

The following figure shows an example of convergence of signals coming from different source clock domains.

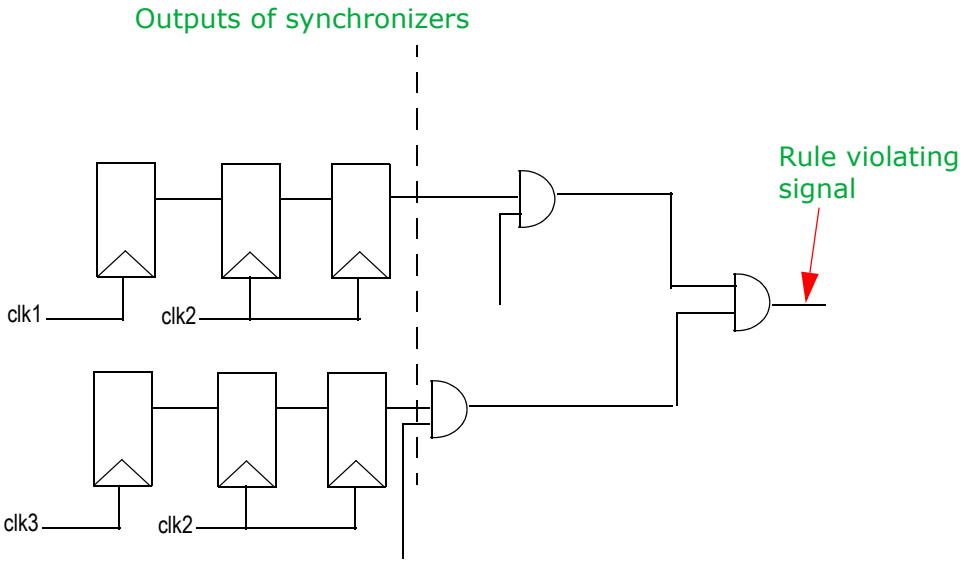


FIGURE 212. Convergence of Signals from Different Source Domain Elements

NOTE: *The Ac_conv03 rule reports sequential convergence (under the conv03_report_seq_conv parameter setting) for bus-merged destination names. But the schematic and depth is calculated for any one representative bit of this bus-merged destination name. However, it may happen that the representative bit has the depth 0 and a non-representative bit has a non-zero depth due to which it is considered as sequential convergence. Therefore, it might seem as a combinational convergence from depth/schematic, which is reported in sequential convergence.*

Features of the Ac_conv03 Rule

Various features of the *Ac_conv03* rule are as follows:

- It reports only one violation for different paths.

- It performs convergence check for synchronizers that are in the same destination domain but have sources from different domains.

  However, this rule stops checking beyond the gate where a different domain signal converges. The following figure illustrates this scenario:

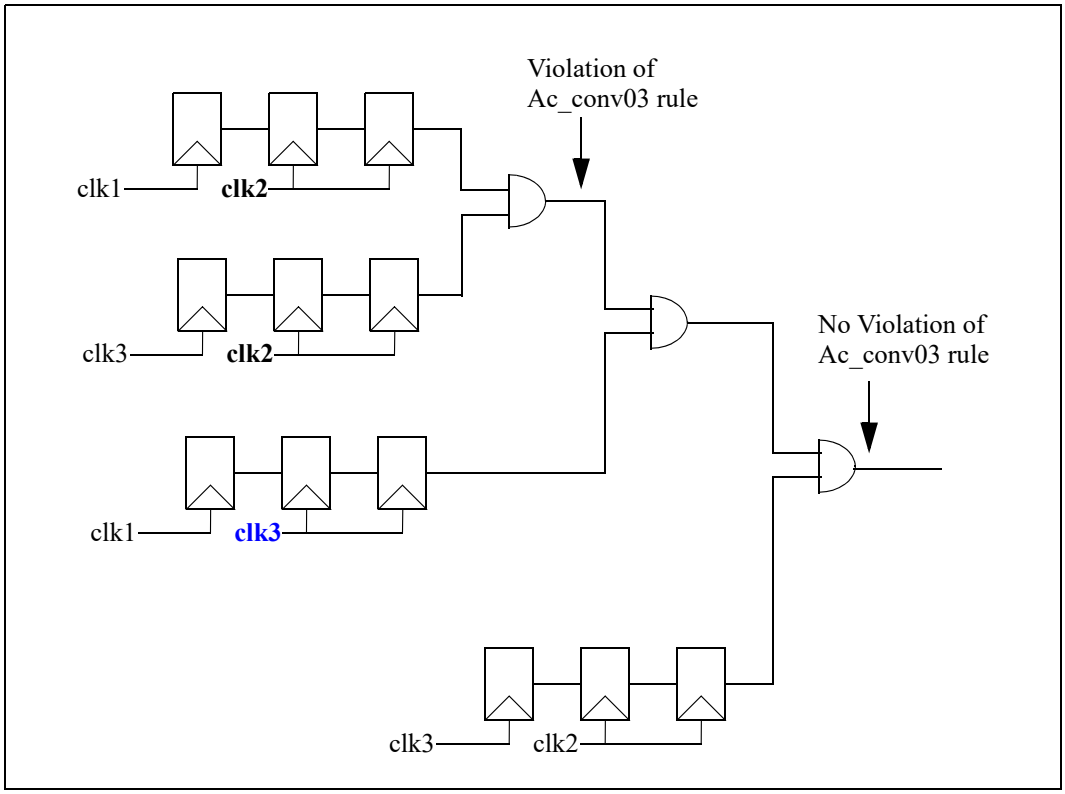


FIGURE 213. No Rule-Checking Beyond the Gate Where a Different Domain Signal Converges

- It ignores convergence of clock domain crossing signals specified by the *cdc_false_path* constraint.
- It checks bus signals.
- It reports violations on the closest point as shown below:

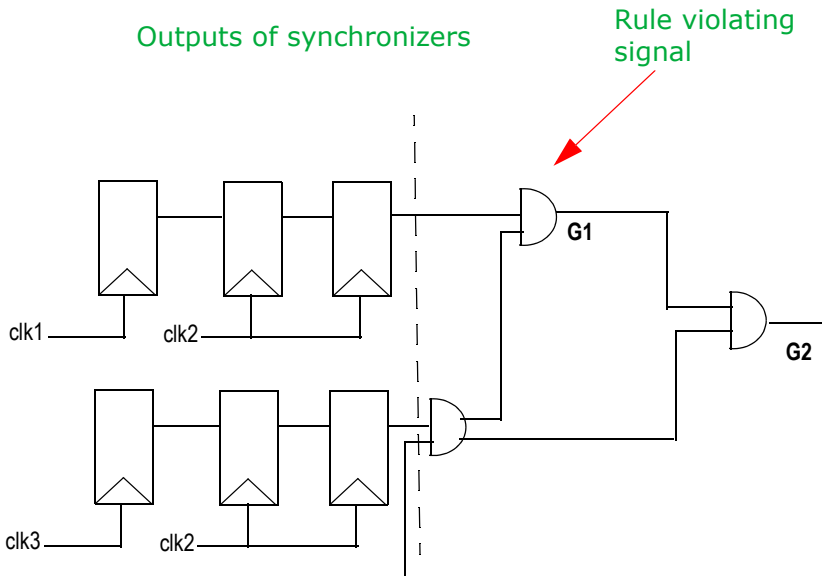


FIGURE 214. Ac_conv03 Rule Violation on the Closest Point

In the above figure, the *Ac_conv03* rule reports a violation on `G1`, which is closest to the converging signals.

- If two data crossings have a different control line, this rule reports convergence for control signals if the *conv03_report_seq_conv* parameter is set.

  The following figure illustrates this scenario.

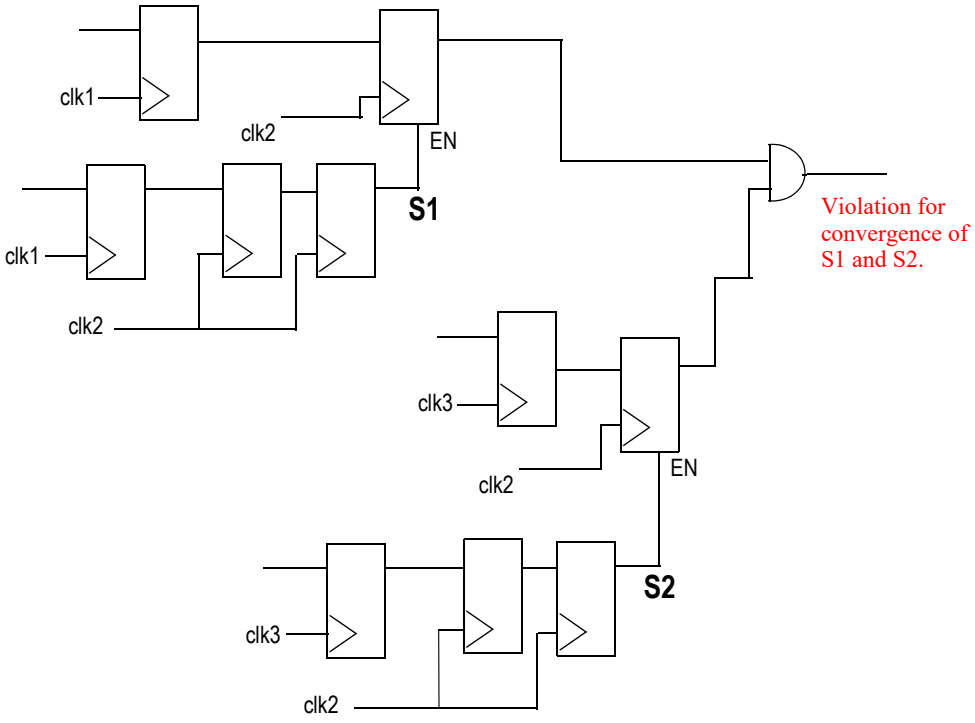


FIGURE 215. Ac_conv03 Rule Violation on the Convergence of Control Signals

- It reports a violation on internal nets if the output cone of that net is blocked due to *set_case_analysis* or design power/ground.

Handling MUXes by the Ac_conv03 Rule

On encountering a mux, this rule propagates convergences of all the data input and control inputs. For example, in the following figure, the convergence on the mux is propagated to the AND gate:

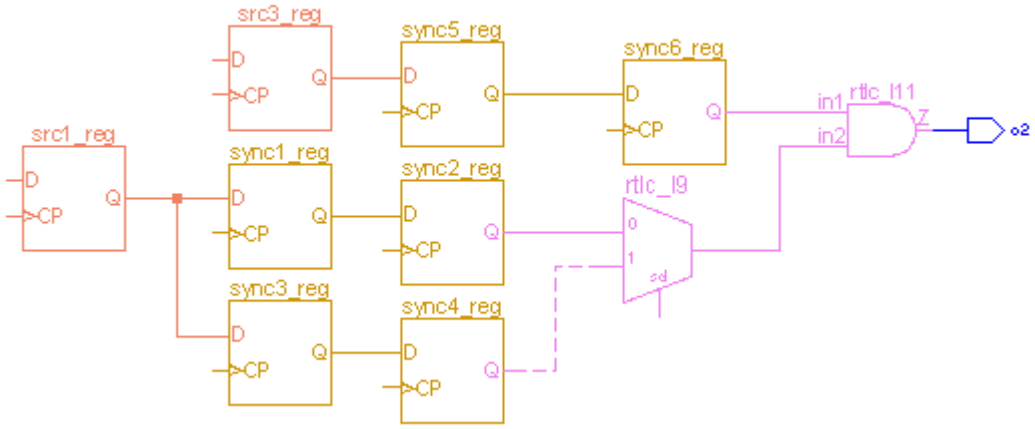


FIGURE 216.

In the above scenario, the *Ac_conv03* rule reports a violation at the AND gate for the `sync2`, `sync4`, and `sync6` synchronizers.

Parameter(s)

- *conv03_report_seq_conv*: Default value is `no`. Set this parameter to `yes` to enable this rule to propagate synchronizers past sequential elements.
- *conv_sync_as_src*: Default value is `no`. Set this parameter to `yes` to check convergence for synchronizers that are also used as a source in other crossings.
- *conv_clock_reset_path*: Default value is `no`. Set this parameter to yes to enable convergence detection of the synchronizer propagation through the clock and reset pin.
- *convergence_stop_at_mux*: Default value is `no`. Set this parameter to `yes` to stop propagation of relevant signals whenever an RTL MUX is encountered.
- *no_convergence_check*: Default value is `NULL`. Specify net names that should not be checked for convergence.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to `stop_conv_at_seq_lib` to stop synchronizer propagation across sequential library cell. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *show_source_in_spreadsheet*: Default value is `yes`. Set this parameter to `no` to generate a link from the spreadsheet of the *Ac_conv01*, *Ac_conv02*, or *Ac_conv03* rules to the message-based spreadsheet of *The Ac_sync_group Rules* showing the source of synchronizers.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *allow_half_sync*: Default value is `yes`. Set this parameter to `no` to not treat half synchronizers as valid synchronizers.
- *clock_gate_cell*: Default value is `NULL`. Set the value of this parameter to a comma or space-separated list of clock-gating cell names for the *Clock-Gating Cell Synchronization Scheme*.

- *delayed_ptr_fifo*: Default value is `no`. Set this parameter to `yes` when the read/write pointers are delayed and the multiplexer inside the memory is one-hot or implemented using gates.
- *enable_and_sync*: Default value is `no`. Set this parameter to `yes` to enable the *AND Gate Synchronization Scheme*.
- *enable_mux_dest_domain*: Default value is `none`. Set this parameter to `none` to turn off this parameter. Other possible values are `mux`, `enable`, `gp`, `cg`, `and`, `all`, `yes`, and `no`.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *enable_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Synchronized Enable Synchronization Scheme*.
- *enable_sync_cell*: Default value is Default value is `NULL`. Set this parameter to a list of synchronizer cells.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *glitch_protect_cell*: Default value is `NULL`. Specify a comma or space-separated list of glitch protection cell names for the *Glitch Protection Cell Synchronization Scheme*.
- *ignore_nets_clock_path_file_name*: Default value is *ignore_nets_clock_path.txt*. Specify a file containing hierarchical names of nets (one name per line) so that SpyGlass halts clock propagation along the path when any of these nets is encountered.
- *ignore_num_rtl_buf_invs*: Default value is `many`. Set this parameter to `one` to allow one buffer and inverter. Other possible values are `two` and `none`.
- *num_flops*: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.
- *one_cross_per_dest*: Default value is `yes`. Set this parameter to `no` to report all unsynchronized clock crossings for a destination.

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *strict_double_flop*: Default value is `no`. Set this parameter to yes to mark clock crossings as synchronized.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is NULL. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the *Synchronizing Cell Synchronization Scheme*.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *compute_num_convergences*: Default value is 1. The maximum possible value for compute_num_convergences is 10. Set this parameter to any integer to specify the number of convergences to be computed for the same set of synchronizers.
- *coherency_check_type*: Default value is `control`. Set this parameter to `reset` to check convergence issues on control crossings of reset paths only.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *cdc_filter_coherency* (Optional): Use this constraint to specify points at or beyond which no convergence of signals should be reported.

- *no_convergence_check* (Optional): Use this constraint to specify nets that should not be checked for convergence.

NOTE: *If you specify nets by using the no_convergence_check constraint as well as the no_convergence_check parameter, SpyGlass considers the nets specified by both the constraint and parameter.*

- *cdc_attribute* (Optional): Use this constraint to specify mutually exclusive and unrelated signals such that convergence-related violations are suppressed for such signals.

NOTE: *The cdc_attribute constraint is the preferred constraint over the cdc_filter_coherency and the no_convergence_check constraints to specify unrelated signals.*

- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *ip_block* (Optional): Use this constraint to specify IP blocks in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *num_flops* (Optional): Use this constraint to specify the minimum number of flip-flops required in a synchronizer chain.
- *output_not_used* (Optional): Use this constraint to specify a primary output port.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.
- *sync_cell* (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *allow_combo_logic* (Optional): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.
- *quasi_static* (Optional): Use this constraint to specify signals for which the value is predominantly static.

Messages and Suggested Fix

The following message appears at the location where two or more signals converge:

`[AcCv3_1] [WARNING] <num> synchronizers <sig-names> converge on <gate-type> '<gate-name>'. '`

The arguments of the above message are explained below:

Argument	Description
<num>	Number of synchronized signals
<sig-names>	Comma separated list of synchronized destination output net names. When the design is a netlist design and the *report_inst_for_netlist* parameter is set to yes, the rule reports a comma-separated list of synchronized destination output instance names.
<gate-type>	Gate type, such as `MUX`, `combinational gate`, `flop`, `latch`, `sequential library-cell`, or `black-box`
<gate-name>	Name of the gate output on which convergence occurs. When the *report_inst_for_netlist* parameter is set to `yes`, the instance pin is reported.

Potential Issues

This violation appears when signals from different clock domains are synchronized in the same clock domain and they converge.

Consequences of Not Fixing

If you do not fix this violation, convergence of synchronizers from different domains can cause data coherency. This may result in chip failure.

How to Debug and Fix

To debug and fix the violation, perform the following steps:

1. Open the rule-based spreadsheet.

   For details on this spreadsheet, see *Rule-based spreadsheet -*

Ac_conv03.csv.

2. In the rule-based spreadsheet, click on the *ID* column of the violation to be debugged. The message-based spreadsheet appears showing the violation details.

   For details on this spreadsheet, see *Message-based spreadsheet - ac_conv_<num>.csv*.

3. View the schematic of the violation by clicking on the link in the *Schematic* column of the message-based spreadsheet.

   In the schematic, analyze the path of convergence of synchronizers.

4. Based on the information in the spreadsheet and the schematic, perform appropriate actions, as described below:

   - ❒ If convergence is reported for signals that are in an IP and you do not want violations to be reported within the IP, specify that IP in the *ip_block* constraint.
   - ❒ If convergence is for static signals, use the *cdc_false_path* or *quasi_static* constraints.
   - ❒ If the path of synchronizers confirms that synchronizers cannot functionally control the converging net at the same time, waive the violation.
   - ❒ If some of the intermediate nets in the path confirm exclusivity between the synchronizers, specify such nets to the *no_convergence_check* parameter.
   - ❒ If convergence is happening at a mux, you may use the *convergence_stop_at_mux* parameter to stop propagation of convergences beyond the mux output.
   - ❒ If you want to stop propagation of convergences beyond a particular net, pin, or instance, specify such net, pin, or instance to the `-stop_points` argument of the *cdc_filter_coherency* constraint.
   - ❒ If you do not want convergence to be reported at a particular net, pin, or instance, specify such net, pin, or instance to the `-conv_gates` argument of the *cdc_filter_coherency* constraint.
   - ❒ If you do not want convergence to be reported for certain set of signals, pins, or instances, which are either destination or sources, specify such set of signals, pins, or instances to the `-unrelated` argument of the *cdc_filter_coherency* constraint.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

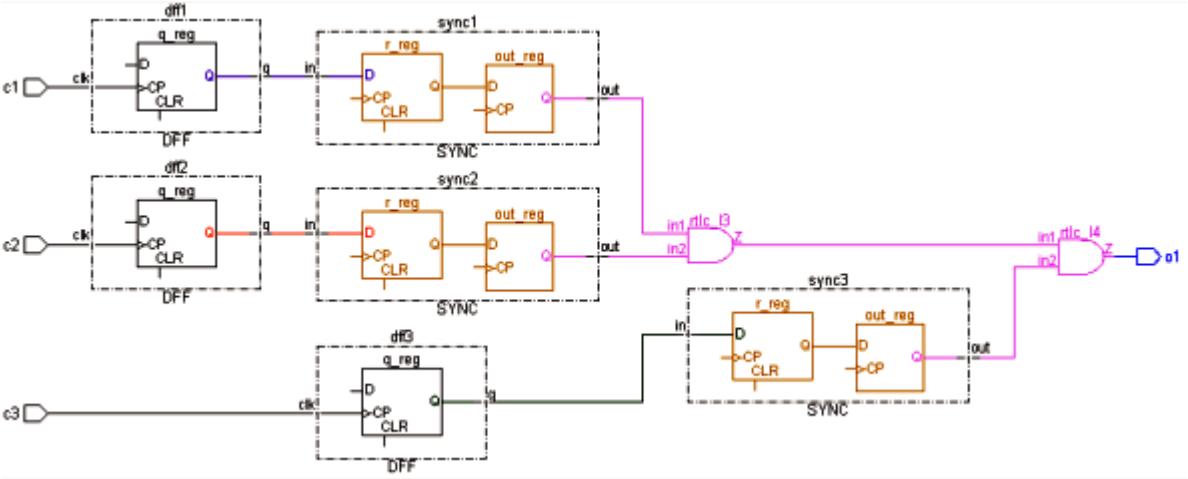


FIGURE 217. Schematic of the Ac_conv03 Rule Violation

In the above example, three different domain source signals, `dff1.q_reg`, `dff2.q_reg`, and `dff3.q_reg`, which are synchronized in the same domain by using *Conventional Multi-Flop Synchronization Scheme* are converging on the AND gate. Different domain source are highlighted in different colors.

Schematic Details

The schematic displays the paths from the converging signals to the signal on which they converge.

Different colors are used to highlight the following information:

- Each signal converging from a particular domain

  Signals from the same domain are shown in the same color. If signals are converging from different domains, unique colors are used to differentiate them.

  Maximum 16 different colors are supported for this purpose. Beyond this limit, signals from different domains are highlighted in a default color.

This default color is also used if the domain of a signal cannot be ascertained.

- Synchronizers
- Signals driving the synchronizers
- Paths from the output of synchronizers to the net on which convergence is reported
- Net on which convergence is reported

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

- Rule-based spreadsheet - Ac_conv03.csv

  This spreadsheet shows all the *Ac_conv03* rule violations in separate rows.

  To open this spreadsheet, right-click on the rule-name header in the *Results* pane, and select the *Spreadsheet Viewer* option from the shortcut menu.

NOTE: *This spreadsheet is generated in the spyglass_reports/clock-reset/ directory.*

The following figure shows an example of the rule-based spreadsheet of this rule:

A	B	C	D	E	F	G	H	I
ID	NVERGING G/	GATE TYPE	F SYNCH	CALAR SOUR(	AX. DEP	MIN. DEPTH	AVG. DEPTI	SAME DEPTH CON
E	test1.o1	combinational gate	3	no	1	0	0.33	yes
1A	test2.o1	combinational gate	3	no	1	0	0.33	yes

FIGURE 218. Rule-based spreadsheet of the Ac_conv03 rule

The details of each column of the above spreadsheet are described below:

Column	Description
ID	Specifies a unique ID for a violation.
CONVERGING GATE	Specifies the name of the converging gate.
GATE TYPE	Specifies the type of converging gate.
NUMBER OF SYNCHRONIZERS	Specifies the number of synchronizers present in the path of the converging gate.
SCALAR SOURCE	Specifies *Yes* or *No* indicating if the source of all the synchronized crossings is scalar or not.
MAX. DEPTH	Specifies the maximum synchronizer depth from any synchronizer to a converging point. This is an integer value.
MIN. DEPTH	Specifies the minimum synchronizer depth from any synchronizer to a converging point. This is an integer value.
AVG DEPTH	Specifies the average depth from a source to a converging point. This is a float value.
SAME DEPTH CONVERGENCE	Specifies *yes* when multiple synchronizers exist with the same depth in convergence. Else, the value No is reported.
WAIVED	Specifies if the reported violation is waived.

NOTE: *If you run the Ac_conv03 rule in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the corresponding Message-based spreadsheet - ac_conv_<num>.csv. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.*

- Message-based spreadsheet - ac_conv_<num>.csv

  This spreadsheet shows details of the selected violation.

  To open this spreadsheet, click on the link in the ID column of the rule-based spreadsheet. Alternatively, double-click on the violation of this rule from GUI.

NOTE: *This spreadsheet is generated in the spyglass_reports/clock-reset/Ac_conv03/ directory.*

The following figure shows an example of the message-based spreadsheet for this rule:

A	B	C	D	E	F	G
Schematic	Type	Signal Name	quential De	Source(s)	estination Clock(	Source Clock(s
4	Converging Gate	test1.o1	-	-	-	-
5	Destination flop	test1.sync1.r[0]	1	test1.dff1.q[0]	test1.c4	test1.c2
6	Destination flop	test1.sync2.r[0]	0	test1.dff2.q[0]	test1.c4	test1.c2
7	Destination flop	test1.sync3.r[0]	0	test1.dff3.q[0]	test1.c4	test1.c3

FIGURE 219. Message-based spreadsheet of the Ac_conv03 rule

In the above spreadsheet, there is a separate row for each converging destination signal.

The details of each column are described below:

Column	Description
Schematic	Shows a link for schematic.
Type	Specifies any of the following types: • Converging Gate • Destination flop • Destination latch • Destination library-cell • Destination black-box • Destination port The first row is for a converging gate.
Signal Name	Specifies the name of the signal.
Sequential Depth	Specifies the depth from a synchronizer till the converging gate.
Source(s)	Specifies the source of the signal.
Destination Clock(s)	Specifies the destination clock of the signal.
Source Clocks	Specifies the name of source clocks for each signal.
File: Line	Specifies the design file name and line number where the information pertaining to each row in this spreadsheet is present

NOTE: *In the message-based spreadsheet, some column values are not applicable for the row of type "Converging Gate". The cells in such columns have the value "-".*

- Ac_conv_detail.rpt: This report contains details of all the violations detected by the Ac_conv01, Ac_conv02, and Ac_conv03 rules. For details, see *Ac_conv_detail.rpt*.

Ac_conv04

Checks all the control-bus clock domain crossings that neither converge nor follow gray encoding

When to Use

Use this rule to check for coherency issues in the synchronized clock domain crossings to control bus signals.

Description

The *Ac_conv04* rule reports a violation in the following cases:

- Synchronized control-bus clock domain crossings do not follow gray encoding

  Only those synchronized control-bus clock domain crossings are checked by this rule that do not converge and therefore, are not reported by *Ac_conv01* and *Ac_conv02* rules.

  This rule checks only those multi bit control buses that are synchronized by any of the following synchronization schemes:

  - *Conventional Multi-Flop Synchronization Scheme*
  - *Synchronizing Cell Synchronization Scheme*, provided you have specified the synchronizer cells by using the *synchronize_cells* parameter (for cells whose output pin is connected to a scalar net) or by using the *synchronize_data_cells* parameter (for cells for which an output pin is connected to a vector net).
  - *Qualifier Synchronization Scheme Using qualifier -crossing*

  For example, consider the scenario shown in the following figure:

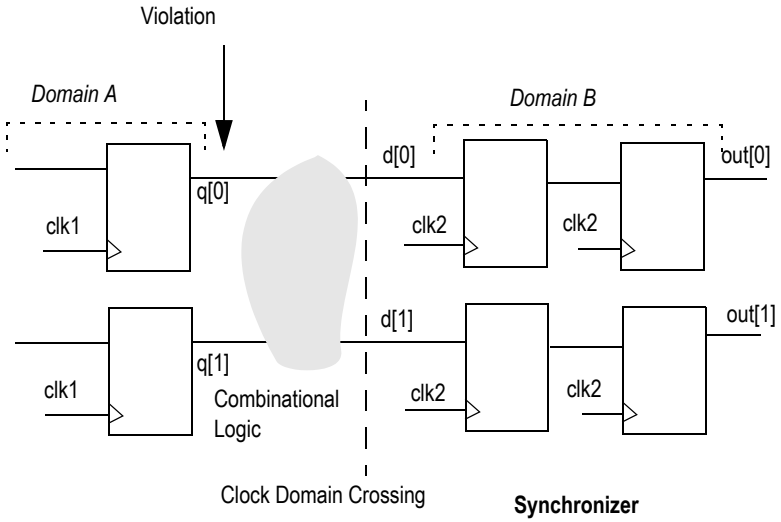


FIGURE 220. Scenario of *Ac_conv04* Rule Violation

In the above scenario, the *Ac_conv04* rule reports a violation if the input of destination flip-flops is not gray encoded.

- If different synchronization schemes or different enable signals are used for the source bus bits, as shown in the following figure:

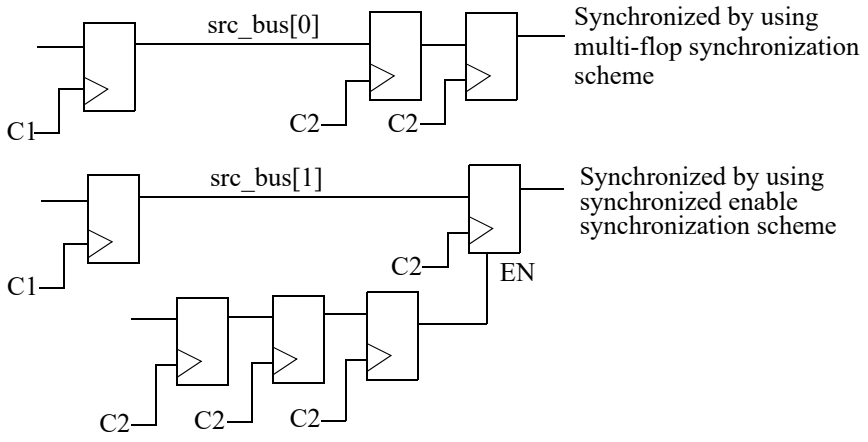


FIGURE 221. Scenario of *Ac_conv04* Rule Violation

- If the source bus bits do not have a common enable or select signals, as shown in the following figure:

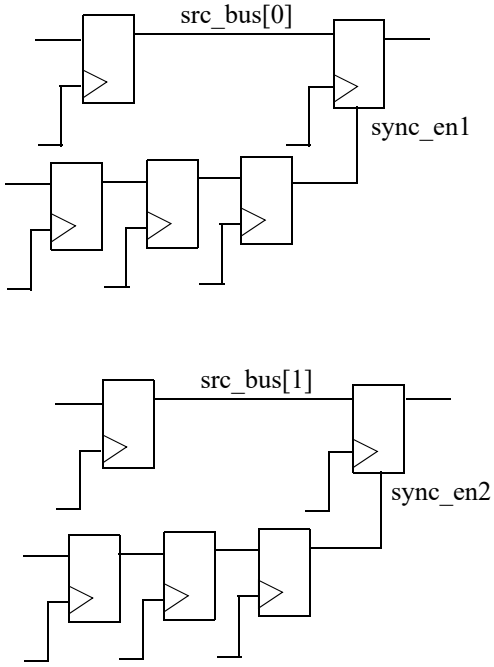


FIGURE 222. Scenario of *Ac_conv04* Rule Violation

The *Ac_conv04* rule reports a violation in the above case only if such case is not reported by the *Ac_conv01*, *Ac_conv02*, and *Ac_conv03* rules.

Rule Exceptions

The *Ac_conv04* rule does not check for the bus signals that get converged and are reported by the *Ac_conv01*, *Ac_conv02*, and *Ac_conv03* rules.

Parameter(s)

- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *fa_grayhold*: Default value is `no`. Set this parameter to `yes` to checks for gray-encoding at the output of a destination instance with respect to a destination clock.

- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *num_flops*: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *synchronize_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for scalar source domain signals for the *Synchronizing Cell Synchronization Scheme*.
- *synchronize_data_cells*: Default value is `NULL`. Specify a list of cells to this parameter that are considered as valid synchronizers for source domain vector signals for the Synchronizing Cell Synchronization Scheme.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *cdc_filter_coherency* (Optional): Use this constraint to specify points at or beyond which no convergence of signals should be reported.
- *clock* (Optional): Use this constraint to specify clocks in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *sync_cell* (Optional): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *cdc_attribute* (Optional): Use this constraint to specify mutually exclusive and unrelated signals such that convergence-related violations are suppressed for such signals.

Messages and Suggested Fix

Message 1

The following message reports the `FAILED` or `Others (Constraints-Conflict)` status for a control bus that crosses a synchronized clock-domain crossing but does not follow gray encoding:

[AcCv4_1] [ERROR] `Control destination bus <bus-name> has a clock domain crossing. Gray encoding check : <FAILED | Others (Constraints-Conflict)>`

Potential Issues

This violation appears if a control bus that crosses a clock-domain crossing is not gray encoded.

Consequences of Not Fixing

If you do not fix this violation, the data in the crossing may be lost.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc08* rule.

Message 2

The following message reports the `Partially-Proved` status for a control bus that crosses a synchronized clock-domain crossing but does not follow gray encoding:

```
[AcCv4_2] [WARNING] Control destination bus <bus-name> has a
clock domain crossing. Gray encoding check : Partially-Proved
```

Potential Issues

This violation appears if a control bus that crosses a clock-domain crossing is not gray encoded.

Consequences of Not Fixing

If you do not fix this violation, the data in the crossing may be lost.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc08* rule.

Message 3

The following message reports the `Passed` status for a control bus that crosses a synchronized clock-domain crossing but does not follow gray encoding:

```
[AcCv4_4] [INFO] Control destination bus <bus-name> has a clock
domain crossing. Gray encoding check : Passed
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc08* rule.

Message 4

The following message appears if at least two bits of the source bus are not uniformly synchronized to the destination bus:

`[AcCv4_5] [WARNING] At least two bits (<bit-num1>, <bit-num2>) of source bus "<source-bus>" are not uniformly synchronized to destination bus "<destination-bus>". Reason: <reason>`

The arguments of the above message are explained below:

Argument	Description
<bit-num1>	Name of one of the bit of the source bus
<bit-num2>	Name of one of the bit of the source bus
<source-bus>	Name of the source bus
<destination-bus>	Name of the destination bus
<reason>	Specifies any of the following reasons: • Different synchronization schemes used • Different enable signal used

Potential Issues

This violation appears when your design contains any of the following bus:

- A bus for which different bits are synchronized by different synchronization schemes.
- A bus for which different bits use different enable or select signals.

Consequences of Not Fixing

If you do not fix this violation, the bus bits may change at the same time. This may result in data coherency problem that can cause chip failure.

How to Debug and Fix

Based on the reason reported in the violation message, perform appropriate actions to debug the violation, as described below:

- If bus bits are synchronized by different synchronization schemes

  Action: To debug such cases, view the *Incremental Schematic* of the violation message.

  The following figure displays a sample schematic of this message:

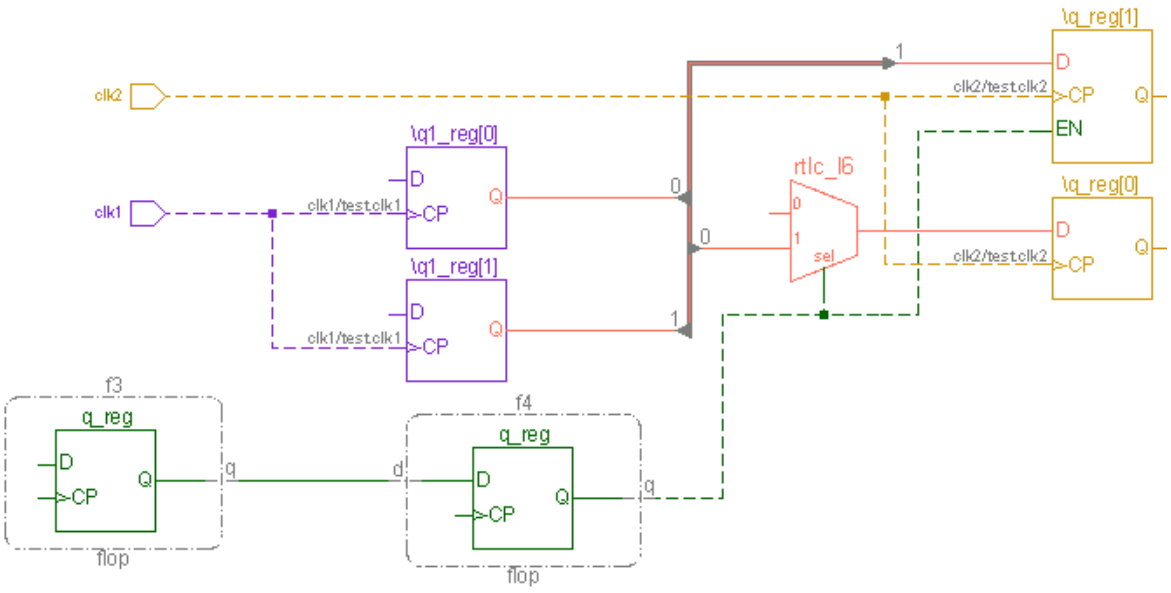


FIGURE 223. Bus Bits Synchronized by Different Synchronization Schemes

In the above schematic, notice that bits of the `test.q` bus are synchronized by different synchronization schemes.

You can also view case analysis settings along with the violations of this rule.

- If the bus bits use different enable or select signals.

  Action: To debug this message, view the *Incremental Schematic* of the violation message.

  The following figure displays a schematic for this violation:

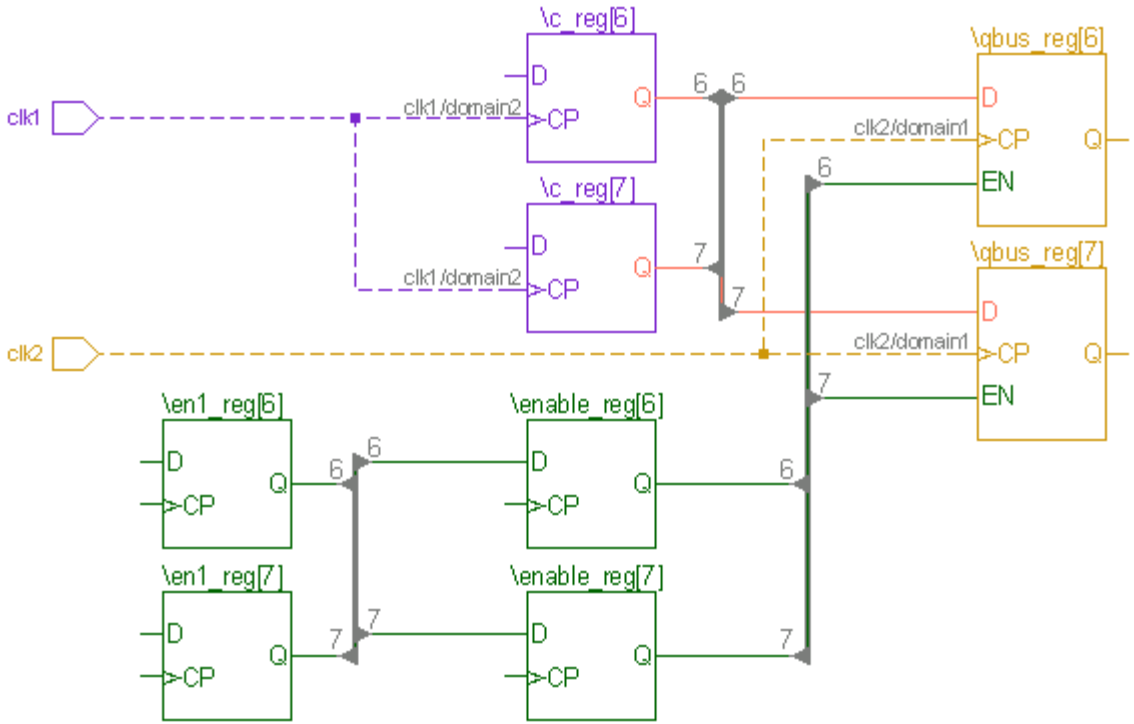


FIGURE 224. Bus Bits Using Different Enable or Select Signals

In the above schematic, notice that different synchronized enable signals are being used for different bus-bits.

You can also view case analysis settings along with the violations of this rule.

Message 5

The following message appears for the `Passed` or `DISABLED` status at the location where signals converge:

```
[ACCv4_5] [WARNING/INFO] Control destination bus <bus-name> has
a clock domain crossing. Gray encoding check : '<Passed |
DISABLED>'
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix*.

If the status is DISABLED, check if any of the following situations is resulting in blocking the rule to perform functional check:

- In case of non-availability of advanced SpyGlass CDC solution license, check the *Ac_license01* rule violation, if any.
- In case of more than one top specified, check the *Ac_multitop01* rule violation, if any.
- In case of over-constraining, check the *Ac_sanity04* rule violation, if any.
- Check if the `fa_msgmode` parameter is set to `none`.

If the status is PASSED, no debugging is required.

Example Code and/or Schematic

See *How to Debug and Fix* of this rule.

Rule Group

`ADV_CLOCKS`

Default Severity Label

The rule severity varies according to the assertion status as follows:

- `FAILED`: Error
- `Partially-Proved/DISABLED`: Warning
- `PASSED`: Info
- `Others(Constraints-Conflict)`: Error

Reports and Related Files

The Advanced CDC Report

Ac_conv05

Performs gray-encoding checks on signals

When to Use

Use this rule to perform gray-encoding checks on the signals specified by the *gray_signals* constraint.

Prerequisites

Use the *gray_signals* constraint to specify the signals that should be checked for gray encoding.

Description

The *Ac_conv05* rule reports the status of gray encoding check performed on the signals specified by the *gray_signals* constraint.

Parameter(s)

- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *gray_signals* (Mandatory): Use this constraint to specify signals for which gray encoding checks should be performed.
- *clock* (Optional): Use this constraint to specify clocks in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.

Messages and Suggested Fix

Message 1

The following message appears to shows the `FAILED` or `OTHERS (Constraint-Conflict)` status of the gray encoding check:

```
[AcCv5_1] [ERROR] Signals '<signal-name>' checked for gray coding. Status: <FAILED | OTHERS (Constraint-Conflict)>
```

Potential Issues

None

Consequences of Not Fixing

If you do not fix this violation, there may be data coherency issues and may cause chip failure.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc08* rule.

Message 2

The following message appears to shows the `Partially-Proved` status

of the gray encoding check:

```
[AcCv5_2] [WARNING] Signals '<signal-name>' checked for gray coding. Status: Partially-Proved
```

Potential Issues

None

Consequences of Not Fixing

If you do not fix this violation, there may be data coherency issues and may cause chip failure.

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc08* rule.

Message 3

The following message appears to shows the `Passed` status of the gray encoding check:

```
[AcCv5_3] [INFO] Signals '<signal-name>' checked for gray coding. Status: Passed
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

See *How to Debug and Fix* of the *Ac_cdc08* rule.

Example Code and/or Schematic

See *Example Code and/or Schematic*.

Rule Group

ADV_CLOCKS

Default Severity Label

Error | Warning | Info

Ac_datahold01a

Checks the functional synchronization of synchronized data crossings

When to Use

Use this rule to verify the correctness of synchronized data clock domain crossings in a design.

Prerequisites

The *Ac_datahold01a* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The *Ac_datahold01a* rule reports synchronized data clock domain crossings where data can be unstable when the enable is active.

NOTE: *This rule checks synchronized data crossings detected by the Ac_sync01 and Ac_sync02 rules.*

This rule checks data crossings synchronized by all synchronization schemes except the *Clock-Gating Cell Synchronization Scheme* because crossings synchronized by this scheme is checked by other SpyGlass CDC rules.

NOTE: *This rule is switched off by default.*

For additional details, refer the *Using the Hybrid CDC Flow* section.

Rule Exceptions

It does not perform functional verification if the source of a crossing is a library cell without functional view.

Parameter(s)

- *cdc_qualifier_depth*: Default value is -1. Specify a positive integer value indicating the depth of sequential logic till which a qualifier should be searched.
- *cdc_qualifier_depth_start*: Default value is `num_flop`. Set this parameter to `sync_chain` so that the last flip-flop of the synchronization chain is the starting point beyond which a qualifier should be searched. Other possible value is `dest`.

- *cdc_effective_bus_verif*: Default value is `none`. Set this parameter to `Ac_datahold01a` to generate a single property for all the bits of destination sources with the same destination enable expression.
- *fa_abstract*: Default value is *Ac_handshake01*, *Ac_glitch03*. Set the value of this parameter to *Ac_cdc08* to enable abstraction for this rule. Other possible values are `all`, `none` or a list of any of rules:

  Ac_handshake01, *Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, *Ac_fifo01*, and *Ac_glitch03*.
- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_dump_hybrid*: Default value is `partial`. Set this parameter to `all` to generate SVA for all the types of assertions (pass, fail, partially-proved). The other possible values are `pass`, `fail`, `+fail`, and `none`.
- *fa_hide_complex_expr*: Default value is `yes`. Set this parameter to `no` to generate the actual enable expression instead of the <complex enable expression> string.
- *fa_holdmargin_window*: Default value is 0. Set this parameter to 1 to control setup and hold margins of one clock edge with respect to each data change.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *cdc_dump_assertions*: Default value is "". Set this parameter to `sva` to generate SystemVerilog Assertions (SVA) corresponding to the rules and the design assumptions specified in an SGDC file.

- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *report_instance_pin*: Default value is `no`. Set this parameter to `yes` to report the name of instance pin of a netlist design. Other possible values are `flop`, `latch`, `bbox`, `seqCell`, and `all`.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer, same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *auto_detect_datahold01_enable*: Default value is `yes`. Set this parameter to `no` to reuse the enable expressions from the Ac_sync rule.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *clock* (Mandatory): Use this constraint to specify clock signals.
- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *monitor_time* (Optional): Use to specify the design initialization time frames during simulation. The rest of the simulation time is considered as the design's functional time.

- *meta_design_hier* (Optional): Use to specify the test bench name and design instance name in the SGDC file.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.

Messages and Suggested Fix

This rule reports the following message:

```
Synchronized crossing: destination <type1> '<name1>', clocked by '<clock-name1>', source <type2> '<name2>', clocked by '<clock-name2>'. Data-enable sequencing check: <status>
```

The arguments of the above message are explained below:

Argument	Description
<type1>	The values can be flop, latch, library-cell, black box, or primary output
<name1>	Destination net name for RTL designs. Destination instance name for netlist designs, if the *report_inst_for_netlist* parameter is set to yes. To report the hierarchical pin name of the destination instance, set the *report_instance_pin* parameter to yes.
<clock-name1>	Destination clock name
<type2>	The values can be flop, latch, library-cell, black box, or primary output
<name2>	Source net name in case of RTL designs. Source instance name in case of netlist designs, if the *report_inst_for_netlist* parameter is set to yes. Otherwise, it is source net name.
<clock-name2>	Source clock name
<status>	Specifies the assertion status that can be any of the following: • PASSED • FAILED • Partially-Proved • Other (Constraints-conflict) • DISABLED

Potential Issues

This violation appears if your design contains a source that is not synchronized correctly by a set of qualifiers and destination domain flip-flops.

Consequences of Not Fixing

If you do not fix this violation, data change may not be captured properly.

How to Debug and Fix

To debug the violation of this rule, perform appropriate actions based on the status of violation messages, as described below:

- If the status of data enable sequencing check is PASSED (Info)

  This is an informational message and does not require debugging.

  By default, SpyGlass does not show data enable sequencing checks that passed the verification. To view them, set the *fa_msgmode* parameter to `all` or `pass`.

- If the status is FAILED (Error)

  Open the *Waveform Viewer* window corresponding to the message, and check the marker that appears on the waveform. This marker is positioned at a transition where enable is high at a point when data is still unstable.

  The *fa_holdmargin_window* parameter is used to enforce a margin between data stabilization and enable activation. By default, its value is zero, which implies that data can change at the same time as enable activation. If you are sure that your data needs one clock cycle to stabilize and then enable should be activated, change the value of the `fa_holdmargin_window` parameter to `1`.

  For example, consider the following *Waveform Viewer* window:

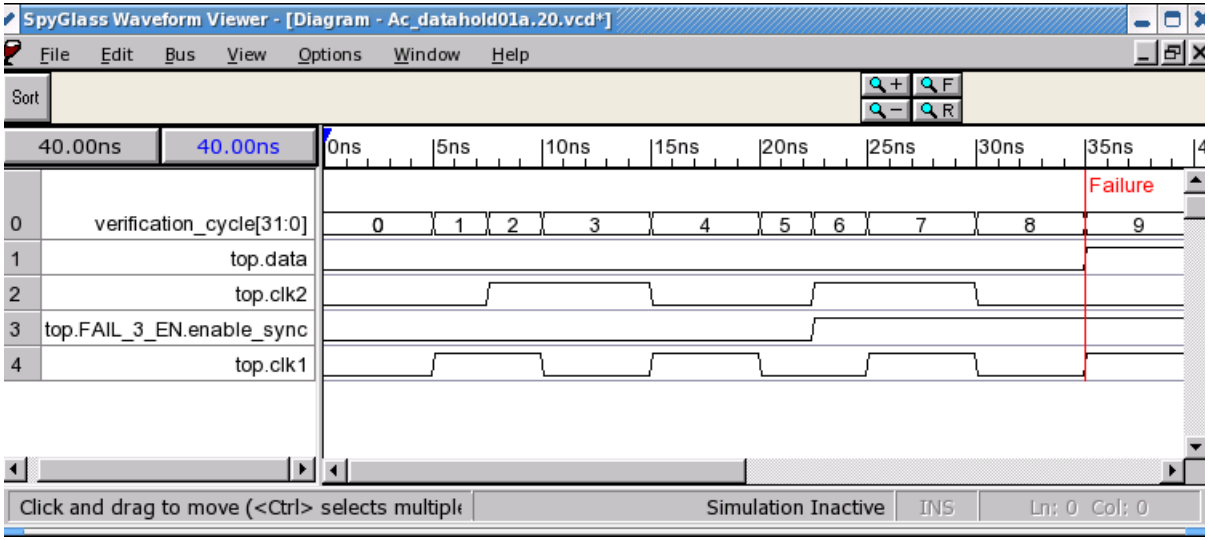


FIGURE 225. The Waveform Viewer Window

In the above waveform, the enable and data change at the same cycle (when the value of `fa_holdmargin_window` is set to `1`), and therefore causing a fail to appear.

Some of the reasons that may cause false failures are as follows:

- ❒ Presence of potential reset/clear signal causing such violation.

  In this case, provide the reset/clear in the constraint file as a reset.

- ❒ The setup (*clock*, *reset*, *set_case_analysis*, and *input* constraints) is not correct and complete.

  In this case, use *Formal Setup Rules* to check for the correctness of the setup.

- ❒ The initial state values in the waveform viewer are not correct.

  In this case, provide correct initial state in the constraints file or provide a VCD file from which an initial state can be loaded.

- ■ If the status is Partially-Proved - Partially Proved (Warning)

  This happens when SpyGlass is not able to conclude (falsify or prove) data enable sequencing check in the given amount of time. In this case, you need to help the tool complete the analysis. You may try following

options for better results:

- ❒ Increase assertion run-time by using the *fa_atime* parameter.
- ❒ Use incremental analysis approach by using the *fa_propfile* parameter.
- ❒ Use the *fa_flopcount*, *fa_seqdepth*, and *fa_scope* parameters to reduce complex verification problem into simpler and solvable problem.

■ If the status is DISABLED (Warning or Info)

SpyGlass reports the status as DISABLED under the Info or Warning severity in the following cases:

- ❒ If a crossing checked by the *Ac_sync01* or *Ac_sync02* rule is reported to be synchronized by the *Clock-Gating Cell Synchronization Scheme*:
  - ◆ Such crossings are not considered for functional checking.
  - ◆ Such crossings are reported as DISABLED under the Info severity by the *Ac_datahold01a* rule.
  - ◆ The reason for the DISABLED status is reported as the synchronization method, as shown in the following example:

```
Synchronized crossing: destination <type1>
'<name1>', clocked by '<clock-name1>', source
<type2> '<name2>', clocked by '<clock-name2>'.
Data-enable sequencing check: DISABLED (<reason>)
```

  Where, *<reason>* can be `Clock Gate Synchronization (auto-detected clock gating)`.

- ❒ If the *Ac_datahold01a* rule cannot check crossing reported by the *Ac_sync01* or *Ac_sync02* rule functionally, the *Ac_datahold01a* rule reports such crossings as DISABLED under the Warning severity.

  A crossing cannot be checked functionally in this case because of the following reasons:
  - ◆ Source is an input port defined by the *abstract_port* constraint.
  - ◆ Source is a black box terminal.
  - ◆ Source clock is a virtual clock.
  - ◆ Source is a library-cell without functional view

  The reason for the DISABLED status in this case is reported in the following way:

```
Synchronized crossing: destination <type1>
```

```
'<name1>', clocked by '<clock-name1>', source
<type2> '<name2>', clocked by '<clock-name2>'.
Data-enable sequencing check: DISABLED (<reason>)
```

Where, *<reason>* can be `Virtual clock defined on source port`, `Source is black-box`, `abstract_port constraint on source port`, or `source is a library-cell without functional view`.

Also see *Performing Functional Analysis in SpyGlass CDC*.

Example Code and/or Schematic

Consider the following spreadsheet of the *Ac_datahold01a* rule violation that is reported when *fa_holdmargin_window* is set to `1`:

A	B	C	D	E
Schematic	Type	Signal Name	Clock Names	Status
11	Destination flop	top.PASS_5_CGC.q	top.clk2	FAILED
11	Source flop	top.data	top.clk1	FAILED

FIGURE 226. The Spreadsheet Generated by the Ac_datahold01a Rule

The above spreadsheet shows two rows to display details of source and destination in a crossing in which data is unstable.

To view the schematic of the above violation, click a cell in the *Schematic* column and click the button in the spreadsheet tool bar.

The following figure shows the schematic of this violation:

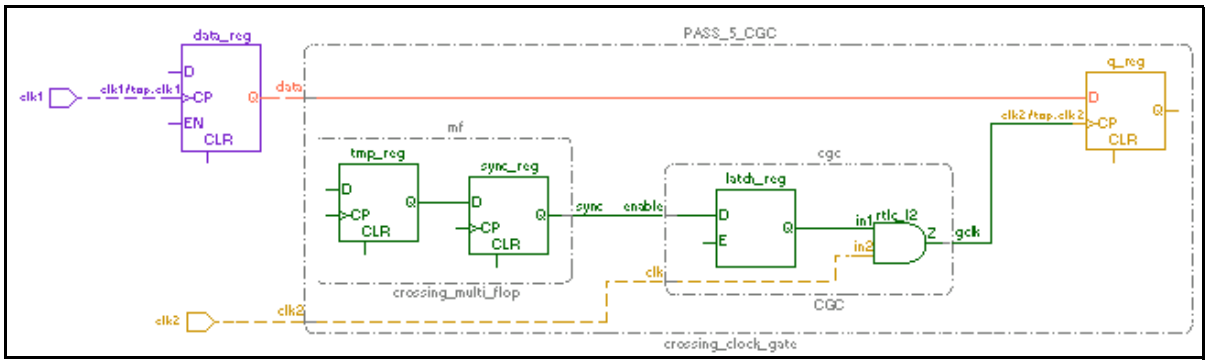


FIGURE 227. Schematic of the Ac_datahold01a Rule Violation

The above schematic shows the crossing at which data is unstable. To fix this violation, ensure that there is sufficient margin between data change and enable change.

Schematic Highlight

This rule highlights the following details in different colors in the schematic:

- Source clock and its instance.
- Destination clock and its instance.
- Clock domain crossing including any combinational logic.
- Synchronizer as identified by the *Ac_sync01* and *Ac_sync02* rules.

Default Severity Label

The rule severity varies according to the assertion status as follows:

- FAILED: Error
- Partially-Proved: Warning
- PASSED: Info
- Others(Constraints-Conflict): Error

Rule Group

ADV_CLOCKS

Reports and Related Files

- Ac_datahold01a.csv
- *Overconstrain Info File*

Clock_sync03a

Reports signals converging from the same source domain and are synchronized separately in the same destination domain

NOTE: *The Clock_sync03a rule will be deprecated in a future release. The rule is not included in CDC GuideWare goals now and do not perform checks until specifically included in the user-defined goal options. In this case, the rule performs the checks and SpyGlass includes a deprecation message in both the spyglass.out and spyglass.log files.*

When to Use

Use this rule to check convergence of synchronized signals of the same domain.

NOTE: *It is recommended to use the Ac_conv01 and Ac_conv02 rules instead of this rule.*

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By using the automatically-generated clock signals after setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_sync03a* rule reports converging signals coming from the same source domain and are synchronized in the same destination domain by using any of the following synchronization schemes:

- *Conventional Multi-Flop Synchronization Scheme*
- *Synchronizing Cell Synchronization Scheme*

The following figure shows convergence of signals coming from the same source clock domains.

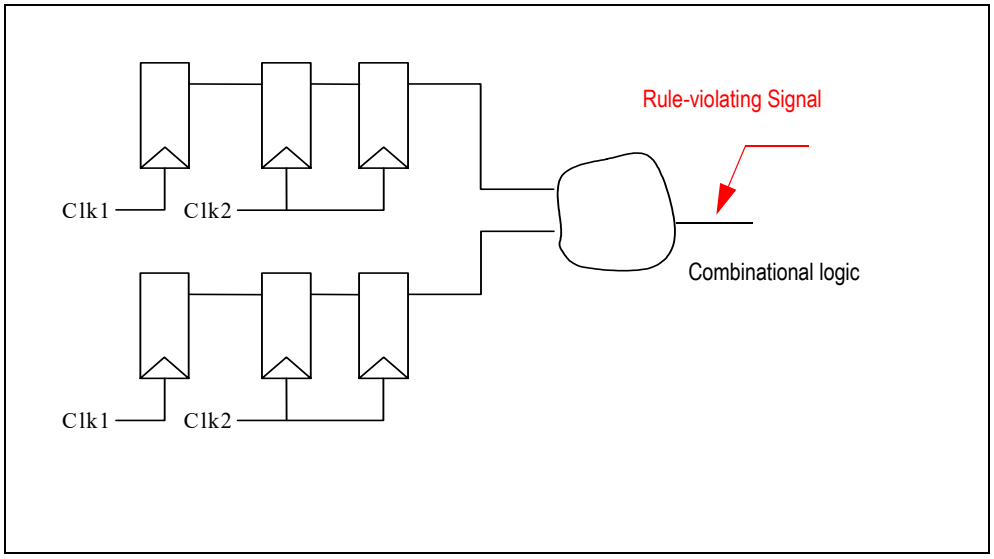


FIGURE 228. Convergence of Signals from Same Source Domain

In the above example, the *Clock_sync03a* rule reports convergence only if a signal reaches on the MUX select pin along with the signals that are reaching one or more MUX input pins.

Rule Exceptions

The *Clock_sync03a* rule has the following exceptions:

- It does not report a violation when synchronizers are a part of the same FIFO memory.
- It does not report convergence of synchronized signals at sequential cells.

Guidelines for Using Parameters for the Clock_sync03a Rule

The following guidelines can help in reducing run time:

- In case of multiple convergences on the same path, the *Clock_sync03a* rule reports convergence on the last gate in the path, which covers all the converging signals.

The following figure shows an example of convergence of multiple signals.

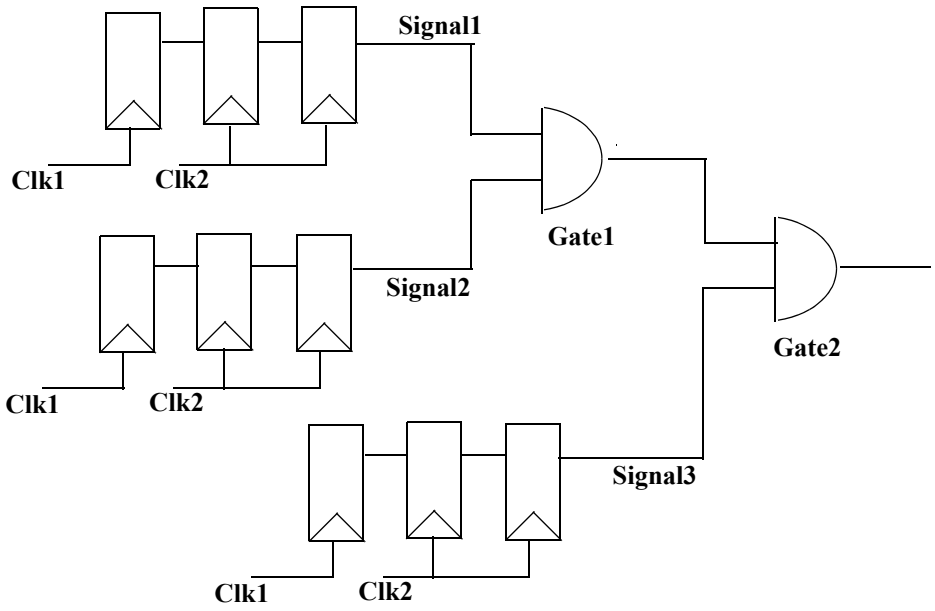


FIGURE 229. Multiple Convergences on Same Path

In the above example, the *Clock_sync03a* rule reports convergence only on `Gate2`, which covers convergence of all signals: `Signal1`, `Signal2`, and `Signal3`.

However, if you set the *all_convergence_paths* parameter to `yes`, this rule reports both `Gate1` and `Gate2`, where `Signal1` and `Signal2` are reported at `Gate1`. These signals are also covered in convergence at `Gate2`. Therefore, it is recommended not to use the *all_convergence_paths* parameter as this parameter may lead to noise and increase in run time and memory.

- High value for *reconvergence_stages* parameter may lead to increase in run time. In that case, check that the value for this parameter is set appropriately.
- You can avoid generation of schematic data for convergence paths to reduce run time. To do so, set the *show_reconv_paths* parameter to `no`.

In this case, only converging signals and the gate where they are converging is highlighted.

Parameter(s)

- *check_bus_bit_convergence*: Default value is `no`. Set this parameter to `yes` to consider bus signals.
- *reconvergence_stages*: Default value is `0`. Set this parameter to a positive integer value to specify the maximum number of flip-flops allowed in the fan-out path of a synchronizing signal.
- *no_convergence_check*: Default value is `NULL`. Specify net names that should not be checked for convergence.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *all_convergence_paths*: Default value is `no`. Set this parameter to `yes` to report all convergence paths.
- *show_reconv_paths*: Default value is `yes`. Set this parameter to `no` to highlight only converging signals and the gate where the signals are converging. This reduces runtime.
- *allow_enabled_multiflop*: Default value is `no`. Set this parameter to `yes` to consider enabled flip-flops as destination or synchronizer flip-flops in

conventional multi-flop synchronization scheme. Other possible value is `same_enable`.

- *enable_multiflop_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Conventional Multi-Flop Synchronization Scheme*.
- *fa_num_cores*: Default value is 0. Specify 2, 4, or 8 to specify the number of cores to be used by a multi-core engine.
- *reset_cross_seq*: Default value is `no`. Set this parameter to `yes` to report clock-domain crossings to the asynchronous reset pins of complex library cells.
- *fa_preprocess_engine*: Default value is `retime`. Set this parameter to `iso` to use the isomorphic reduction technique to optimize functional verification. Other possible values are `none` and `all`.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *no_convergence_check* (Optional): Use this constraint to specify nets that should not be checked for convergence.

NOTE: *If you specify nets by using the no_convergence_check constraint as well as the no_convergence_check parameter, SpyGlass considers the nets specified by both the constraint and parameter.*

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to suppress clock domain crossing checks for false paths.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears at the location where multiple signals converge on the instance *<inst-name>*:

[WARNING] <obj-type> '<sig-name>' converge on '<inst-name>': DISABLED

The arguments of the above message are explained below:

Argument	Description
<obj-type>	`Signals` in case of RTL designs. `Instances` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `Signals`.
<sig-name>	Comma-separated list of signal names in case of RTL designs. Comma-separated list of instances in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is list of signal names
<inst-name>	*<out-net-name>* of converging instance in case of RTL designs. *<inst-name>* of converging instance in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is *<out-net-name>*

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

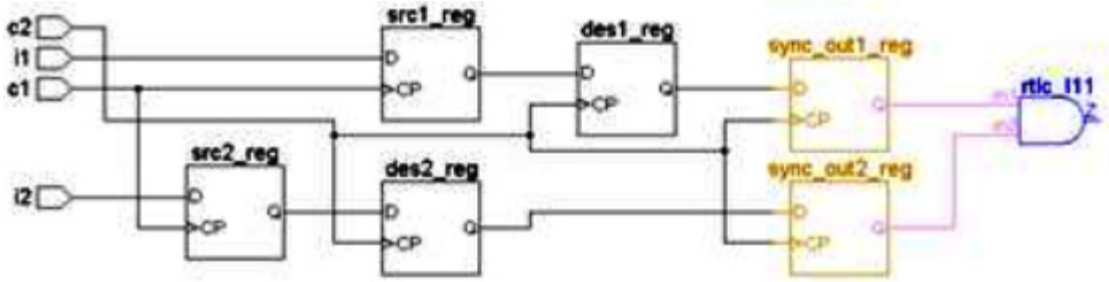


FIGURE 230. Schematic of the Clock_sync03a Rule Violation

In the above example, the `src1` and `src2` signals of the same source

domain are synchronized in the same destination domain by using the *Conventional Multi-Flop Synchronization Scheme*. These signals are converging on the AND gate.

Schematic Details

Different colors highlight each of the following details:

- Path from each destination flip-flop to the converging gate
- Instance on which convergence is taking place

If the *show_reconv_paths* parameter is set to `no`, the rule highlights only converging signals and the gate where they are converging. It does not show the complete path.

Default Severity Label

Warning

Reports and Related Files

No report or related file

Clock Glitch Checking Rules

Glitches on clock signals should be avoided. A glitch on a clock signal essentially renders a chip (or a section of a chip) to asynchronous behavior. A glitch-prone clock signal driving a flip-flop, memory or a latch may store incorrect and unstable D (or data) input of flip-flop, memory or a latch. The D (or Data) input cannot be guaranteed to be stable when the glitch on clock signal appears. Therefore, incorrect data can be stored causing chip functional failure.

Secondly, there is also a chance of violating set up and hold time of D (or data) input of a storage element causing metastability events. These events will cause intermittent chip functional failures (which in many cases are worse to diagnose and correct)

Thirdly, the glitches usually would vary with place and route of the design. Sometimes timing conditions and operating conditions (like process, voltage, and temperature) can mask out glitches. However, these same designs will not function correctly at different process, voltage, and temperature conditions. In addition, another layout ECO can re-introduce glitches on clocks. Therefore, it is best to detect and correct clock glitches at early stage of the design.

The SpyGlass CDC solution has the following rules for checking clock glitch conditions:

Rule	Reports
Ac_glitch01	Detects unsynchronized clock domain crossings subject to glitches due to glitch-prone MUX implementation
Ac_glitch02	Reports clock domain crossings subject to glitches due to re-converging source
Ac_glitch03	Reports clock domain crossings subject to glitches
Clock_glitch01	Enable signals that are gating clocks but are not the output of a flip-flop
Clock_glitch02	Clocks that are gated without latching their enable signal in the inactive half of the clock cycle
Clock_glitch03	MUXes where a clock signal is re-converging
Clock_glitch04	Flip-flop outputs that are converging on a flip-flop clock pin through combinational logic

Ac_glitch01

Reports unsynchronized clock domain crossings subject to glitches because of glitch-prone MUX implementations

When to Use

Use this rule to identify glitch-prone MUX implementations in clock domain crossing paths.

Prerequisites

Specify the `Advanced_CDC` and `adv_checker` license features.

Description

The *Ac_glitch01* rule reports unsynchronized clock domain crossings that are subject to glitches because of glitch-prone MUX implementations.

The following figure illustrates a MUX implementation that can generate glitches.

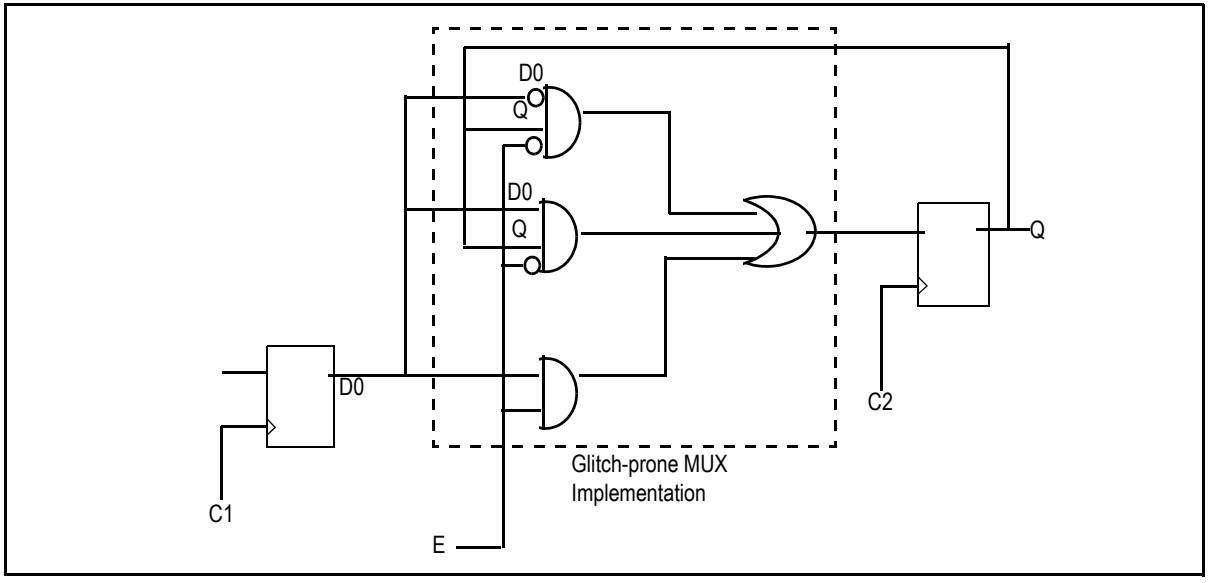


FIGURE 231. Example of glitch-prone MUX implementations

In the above figure, given the MUX inputs `D0`, `Q` and `E`, the *Ac_glitch01* rule

checks if there are structural paths from `D0` to the output of the implemented MUX (input of destination flip-flop) for both logic low and high value of enable (MUX select) pin. If only one such structural path exists, either for logic high or low value of the enable signal, the MUX is glitch-free. Else, glitches can occur and such cases are reported by this rule.

Rule Exception

This rule does not report a violation for a destination flip-flop that has an enable pin, unless it is tied to a constant value.

Parameter(s)

- *mux_search_depth*: Default value is `6`. Specify a positive integer value (greater than one) to specify a logic depth for implicit MUX detection.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.

Messages and Suggested Fix

The following message appears if a glitch may occur because of an unblocked race on a MUX logic:

```
[WARNING] Crossing from source '<source-name>' (clock '<clk1-name>') to destination '<dest-name>' (clock '<clk2-name>') uses glitch-prone MUX implementation with enable '<sig-name>'
```

The arguments of the above message are explained below:

Argument	Description
<source-name>	Source object name
<dest-name>	Destination object name
<clk1-name>	Source clock name
<clk2-name>	Destination clock name
<sig-name>	Enable signal of glitch-prone MUX

Potential Issues

This violation appears if your design contains MUX implementations in which data is reconverging.

Consequences of Not Fixing

If you do not fix this violation, unwanted glitches may occur in your design that can cause functional failure.

How to Debug and Fix

To debug this violation, perform the following steps:

1. Open the incremental schematic of the violation.
2. Check the path passing through a glitch-prone MUX logic in the schematic.
3. Ensure that the *set_case_analysis* constraint is set properly.

In general, optimization or technology mapping is the cause of the creation of a glitch-prone MUX during synthesis. In such situations, you can perform any of the following actions:

- Mark the MUX so that synthesis preserves the implementation and does not perform optimization on such MUXes.
- Use glitch-free MUX implementations to avoid glitches.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

// test.v

```
module flop(d,clk,q);
 input d, clk;
 output q;
  reg q;
  always @(posedge clk)
     q <= d;
endmodule

module top(data, en, clk1, clk2, out);
 input data, en, clk1, clk2;
 output out;
 wire en, d, w1, w2, w3, q_out;
 flop F1(data, clk1, d);
 assign w1 = !en & q_out & d;
 assign w2 = !en & q_out & !d;
 assign w3 = d & en;
 assign w4 = w1 | w2 | w3;
 flop F2(w4, clk2, q_out);
 assign out = q_out;
endmodule
```

// constr.sgdc

```
current_design top
clock -name "top.clk1" -value rtz
clock -name "top.clk2" -value rtz
```

For the above example, the *Ac_glitch01* rule reports a violation because a glitch-prone MUX implementation with the `top.en` enable is used by the crossing from the `top.F1.q` source to the `top.F2.q` destination.

The following figure shows the schematic of this violation:

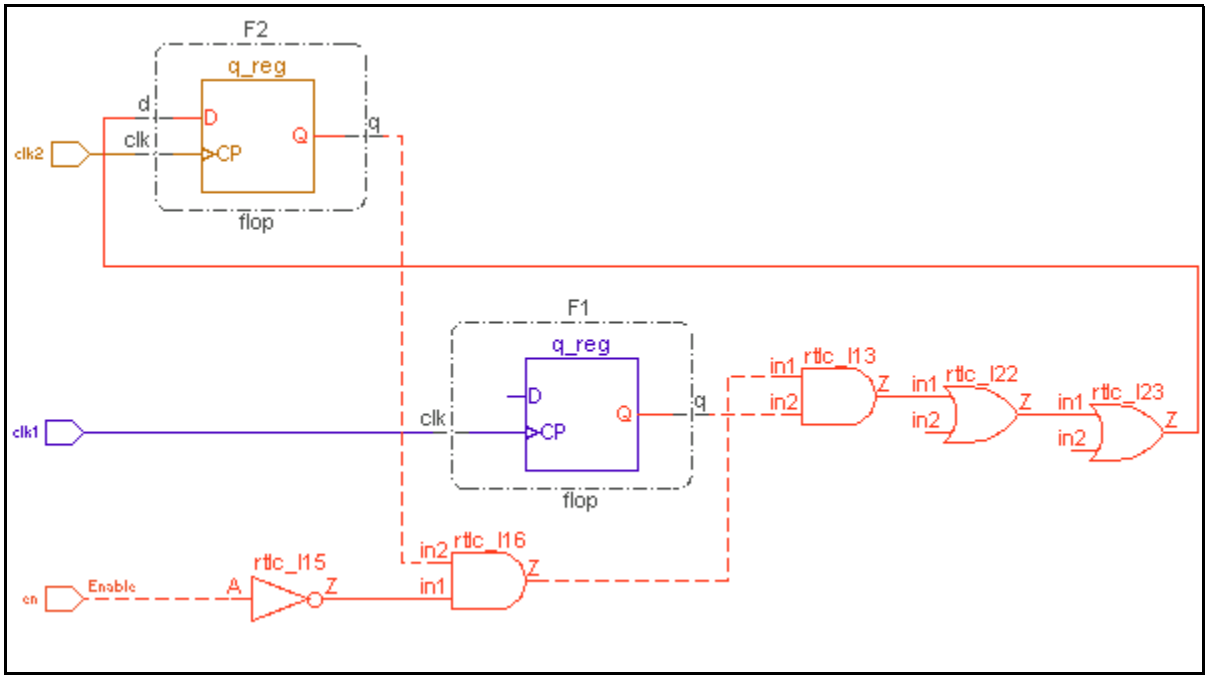


FIGURE 232. Schematic of the Ac_glitch01 Rule Violation

Schematic Details

The *Ac_glitch01* rule highlights the following paths in the schematic:

- Path from the source to the destination
- Path from the clock source to the clock pin of the source flip-flop/black box. The source flip-flop/black box will be highlighted in the same color as the source clock.
- Path from its clock source to the clock pin of the destination object. The destination will be highlighted in the same color as the destination clock.
- Enable path of the glitch-prone implemented MUX

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

Ac_glitch01.csv

Ac_glitch02

Reports clock domain crossings that are subject to glitches because of a reconverging source

When to Use

Use this rule to identify glitch-related issues in clock domain crossing paths.

Prerequisites

Specify the `Advanced_CDC` and `adv_checker` license features.

Description

The *Ac_glitch02* rule reports clock domain crossings in which a source reaches to the destination through multiple paths, as shown in the following figure.

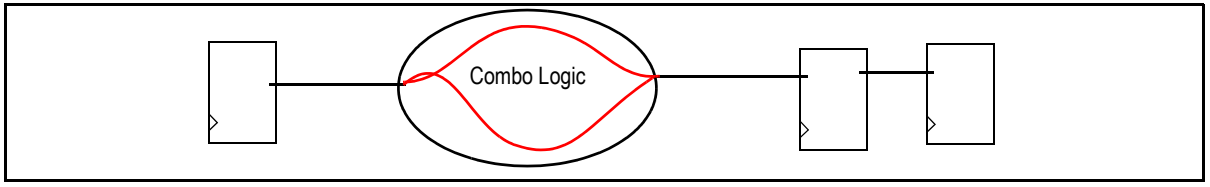


FIGURE 233. Ac_glitch02 Rule Violation

Crossings Checked by the *Ac_glitch02* Rule

By default, the *Ac_glitch02* rule checks the following crossings:

- Synchronized crossings from the *Conventional Multi-Flop Synchronization Scheme*, *Synchronizing Cell Synchronization Scheme*, or *Qualifier Synchronization Scheme Using qualifier -crossing* and the crossings have a combinational logic between the crossing path (the *allow_combo_logic* constraint is set)
- Unsynchronized crossings from *Conventional Multi-Flop Synchronization Scheme*, *Synchronizing Cell Synchronization Scheme*, or *Qualifier Synchronization Scheme Using qualifier -crossing* and these crossings are unsynchronized because of a combinational logic between the crossing path (*allow_combo_logic* constraint is not set)

NOTE: *The Ac_glitch03 rule reports all the cases reported by the Ac_glitch02 rule.*

Parameter(s)

- *check_single_source*: Default value is `yes`. Set this value to `no` to consider all sources of a destination for rule-checking.
- *glitch_check_type*: Default value is `sync_control`. Set this parameter to `unsync` to consider all unsynchronized crossings. Other possible value is `all`.
- *cdc_compatible*: Default value is `no`. Set this value to `yes` to make the rules mentioned in the *Used by* section dependant on the *Clock_sync** rules data rather than *The Ac_sync_group Rules* data.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *allow_combo_logic* (Optional): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.

Messages and Suggested Fix

The following message appears to indicate clock domain crossings that are subject to glitches:

```
[WARNING] Crossing from source '<source-name>' (clock
'<clk1-name>') to destination '<dest-name>' (clock
'<clk2-name>') may be subject to glitches
```

The arguments of the above message are explained below:

Argument	Description
<source-name>	Source object name
<dest-name>	Destination object name

Argument	Description
<clk1-name>	Source clock name
<clk2-name>	Destination clock name

Potential Issues

This violation appears if the source of a clock domain crossing has multiple paths to its destination, and these crossings are synchronized by any of the following synchronization schemes:

- *Conventional Multi-Flop Synchronization Scheme*
- *Synchronizing Cell Synchronization Scheme*
- *Qualifier Synchronization Scheme Using qualifier -crossing*

Consequences of Not Fixing

If you do not fix this violation, glitch-related issues may appear in the design.

How to Debug and Fix

To debug this violation, perform the following steps:

1. Open the incremental schematic of the violation.

   In the schematic, you will see two paths from an asynchronous source to the destination. This means that the delay in the first and the second path may be different. This can cause glitches for the destination.
2. Ensure that the *set_case_analysis* constraints have been specified properly to block one of the paths, if the paths are mutually exclusive.

   Else, fix the RTL so that there are no multiple paths from the same source to the destination to avoid glitches.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:

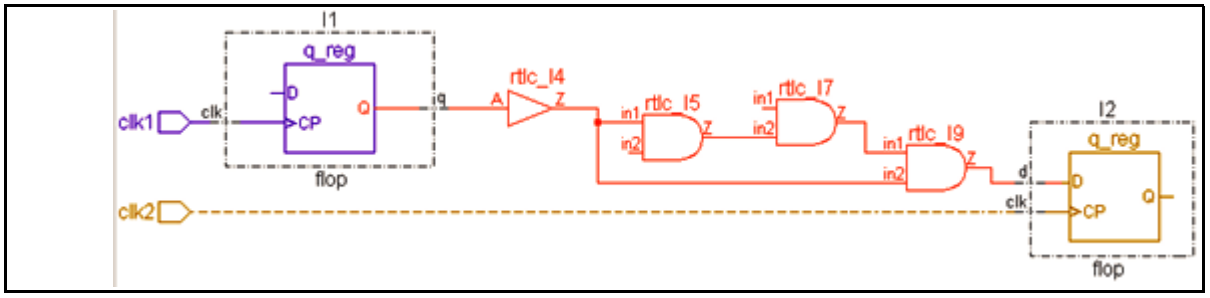


FIGURE 234. Schematic of the Ac_glitch02 Rule Violation

In the above example, the `I1.q_reg` source is reaching to the `I2.q_reg` destination flip-flop through multiple paths (through the `rtlc_I5` and `rtlc_I9` instances).

Therefore, the *Ac_glitch02* rule reports a violation.

Schematic Details

The *Ac_glitch02* rule highlights the following paths in the schematic:

- All reconverging paths from the source to the destination
- Path from the clock source to the clock pin of the source flip-flop/black box. The source flip-flop/black box is highlighted in the same color as the source clock.
- Path from its clock source to the clock pin of the destination object. The destination is highlighted in the same color as the destination clock.

Default Severity Label

Warning

Rule Group

`ADV_CLOCKS`

Reports and Related Files

Ac_glitch02.csv

Ac_glitch03

Reports clock domain crossings subject to glitches

When to Use

Use this rule to detect glitches at clock domain crossings.

Prerequisites

The *Ac_glitch03* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The details of the *Ac_glitch03* rule are covered in the following topics:

- *Reason for the Ac_glitch03 rule Violation*
- *Unsynchronized Crossing Reported by the Ac_glitch03 Rule*
- *Unate/Binate Analysis*
- *Checks Performed by the Ac_glitch03 Rule*
- *Status Reported by the Ac_glitch03 Rule Message*
- *Rule Exceptions*

NOTE: *The Ac_glitch03 rule reports all the cases reported by the Ac_glitch02 rule.*

For additional details, refer the *Using the Hybrid CDC Flow* section.

Reason for the Ac_glitch03 rule Violation

The *Ac_glitch03* rule checks glitch-prone combinational logic in the crossings synchronized by one of the following synchronizing schemes:

- *Conventional Multi-Flop Synchronization Scheme*
- *Synchronizing Cell Synchronization Scheme*
- *Qualifier Synchronization Scheme Using qualifier -crossing*

Also see *Unsynchronized Crossing Reported by the Ac_glitch03 Rule*.

The following figure shows the glitch-prone combinational logic:

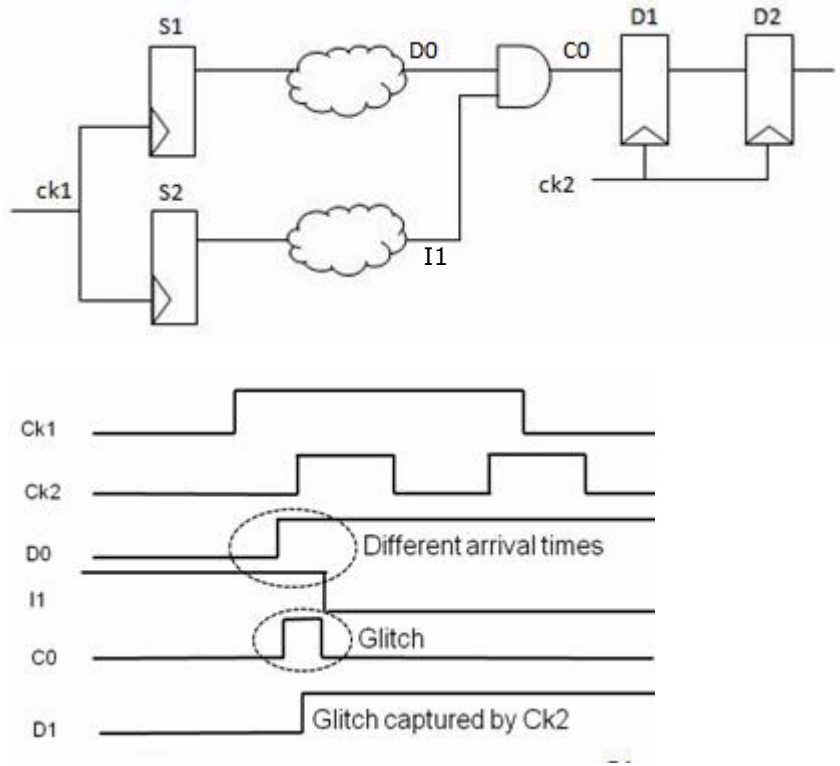


FIGURE 235. Glitch-Prone Combinational Logic Causing Functional Failure

In the above example, the glitch captured on `D1` propagates to `D2` and then to the downstream logic, potentially causing a functional failure.

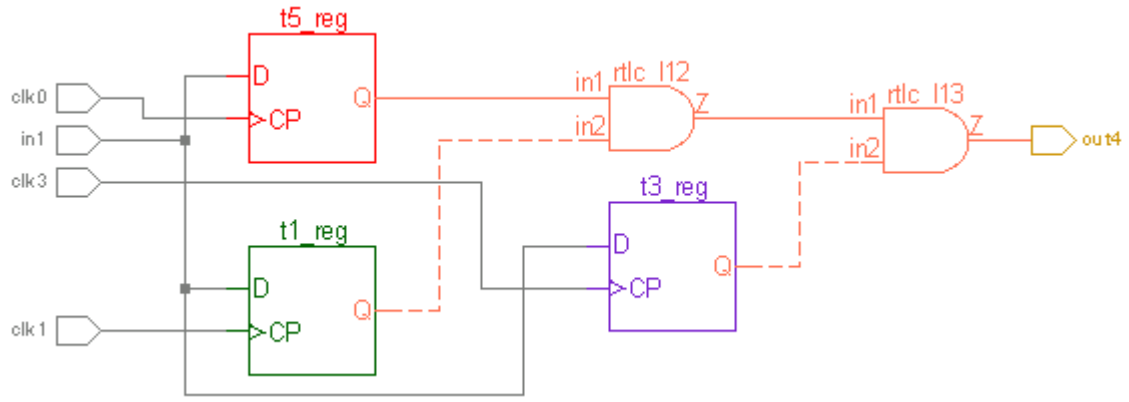
NOTE: *If a combinational logic is present in the crossing path and the allow_combo_logic* parameter *is set to* `no`*, such crossings are reported as unsynchronized and no glitch-related checks are performed in such cases. Set the allow_combo_logic parameter to* `yes` *to perform glitch-related checks on such crossings.*

Unsynchronized Crossing Reported by the Ac_glitch03 Rule

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.

The following schematic shows the example of such crossing reported by the *Ac_glitch03* rule:




```
// SGDC File
 clock -tag vck1 -domain d1
 output -name out4 -clock vck1
 signal_type -name out4 -type control

// Project File
 set_parameter cdc_reduce_pessimism +glitch_on_vck_port
```

FIGURE 236.

For the other types of unsynchronized crossings, glitch checking is performed when the *glitch_check_type* parameter is set to `unsync` or `all`.

Unate/Binate Analysis

You can configure the *Ac_glitch03* rule to report violations related with

same source reconvergence when the reconverging paths have different polarities or at least one path has an unknown polarity.

For details, see *no_unate_reconv*.

Checks Performed by the Ac_glitch03 Rule

This rule detects the glitches that may occur due to the following reasons:

- When sources are in different domains

  In such cases:

  - No gray-encoding check is performed.
  - The following reason is reported:

    Sources from different domains in fanin

- When destination domain signals are in the fan-in of a synchronizer

  In such cases, the destination signal of a crossing is driven by synchronous signals in addition to the sources. As a result:

  - No gray-encoding check is performed.
  - The following reason is reported:

    Signals from destination domain in fanin

NOTE: *Ports without any constraints are considered in the destination domain.*

- When a source reaches the destination through multiple paths

  In such cases, a source diverges and converges back before reaching to the destination. As a result:

  - No gray-encoding check is performed.
  - The following reason is reported:

    Source reconverges

- If none of the above cases is true, glitches occur when multiple same domain sources are toggling in a control crossing

  In such cases, the gray-encoding check is performed to ensure that at the most only one source is changing.

Status Reported by the Ac_glitch03 Rule Message

Based on the checks performed, this rule reports the following status in the

violation message:

Status	Description	Violation Severity
FAILED	This status appears when multiple sources toggle at the same time.	ERROR
Partially-Proved	This status appears if gray-encoding check is incomplete within the time specified by the fa_atime parameter	WARNING
PASSED	This status appears when at most one of the sources toggle at a time	INFO
Others	This status appears if gray-encoding check could not be performed because of conflicting constraints or some internal error	ERROR

Rule Exceptions

- This rule does not consider crossings that are filtered by using the *cdc_false_path* or *ip_block* constraint.
- This rule ignores the following signals in a crossing:
  - Signals specified by using the *quasi_static* constraint
  - Synchronous resets generated in the destination domain
  - Source signals specified by the *cdc_false_path* constraint.
- This rule does not report violations for the same source reconvergence on the input pin of a MUX. This is because, only one input pin is active at a time. This scenario is shown in the following figure:

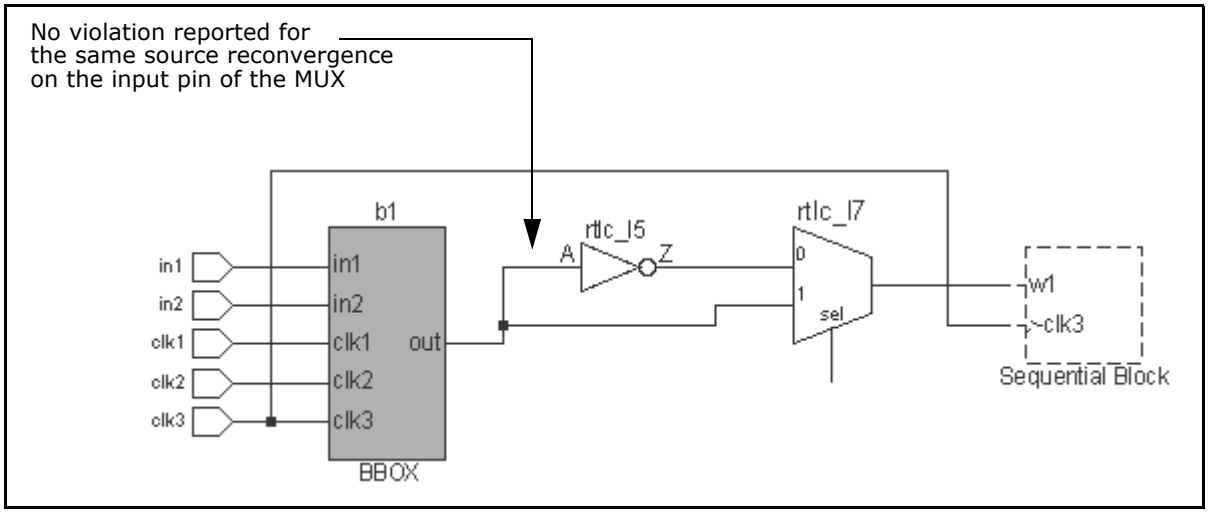


FIGURE 237. Example of the Ac_glitch03 Rule Exception

Parameter(s)

- *cdc_bus_compress*: Default value is `Ac_glitch03`. Set this parameter to `DeltaDelay02` to check all the bits of the source bus by the *DeltaDelay02* rule. For information on the other possible values, see *Possible values of the cdc_bus_compress parameter*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.
- *fa_abstract*: Default value is *Ac_handshake01*, *Ac_glitch03*. Set the value of this parameter to *Ac_cdc08* to enable abstraction for this rule. Other possible values are `all`, `none` or a list of any of rules:

  Ac_handshake01, *Ac_cdc08*, *Ac_conv02*, *Clock_sync03a*, *Ac_fifo01*, and

Ac_glitch03.

- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_msgmode*: Default value is `fail`, `pp`, `coverage`. Set this parameter to `all` to report all the types of assertions. Other possible values are `no_msg`, `audit`, and `none`.
- *glitch_on_sync_src*: Default value is `no`. Set this parameter to `yes` to consider synchronous sources present in the input cone of a destination signal for glitch checking.
- *glitch_on_unconstrained_src*: Default value is `no`. Set this parameter to `yes` to consider the unconstrained ports present in the input cone of destination signal for glitch checking.
- *glitch_check_type*: Default value is `sync_control`. Set this parameter to `unsync` to consider all unsynchronized crossings. Other possible value is `all`.
- *strict_sync_check*: Default value is `no`. Set this parameter to `yes` if scan flip-flops are present.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- `sta_based_clock_relationship`: Default value is `no`. Set this parameter to `yes` to compute domains based on the specification of the *sg_clock_group* constraint.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.

NOTE: *The Ac_glitch03 rule reports violations on blackbox input/output ports if the abstract_port constraint is defined with the* `-end` *argument together with the -sync_active or -sync_inactive arguments.*

- *cdc_attribute* (Optional): Use this constraint to specify mutually exclusive and unrelated signals such that convergence-related violations are suppressed for such signals.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.
- *signal_type* (Optional): Use this constraint to specify the signal type (control or data).
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

Message 1

The following message appears when the gray-encoding check is performed:

```
[AcG3_1] [ERROR] Glitch check performed on destination
<destination-type> '<pin/port-name>' clocked by '<clock-name>'
(<num-sources> source(s), <num-domains> domain(s)). Multi-
source toggling check :'<FAILED | Others (Constraints-Conflict)
| Others (Internal-Error)>'
```

The arguments of the above message are explained below:

TABLE 6 Argument details of the Ac_glitch03 rule

Argument	Description
<destination-type>	Specifies the destination type as flop, latch, library-cell, port, or black box
<pin/port-name>	Specifies the hierarchical name of an instance pin or instance output port
<clock-name>	Specifies the name of the clock driving the destination. In case of multiple clocks, all clock names are shown.
<num-sources>	Specifies total number of asynchronous sources
<num-domains>	Specifies total number of asynchronous domains

Potential Issues

This violation appears if multiple same domain source signals in a crossing toggle at the same time.

Consequences of Not Fixing

If you do not fix this violation, the design may contain glitches.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the *Rule-based spreadsheet*.

   This spreadsheet lists details of all violations of the *Ac_glitch03* rule.
2. From the rule-based spreadsheet, open the spreadsheet of a particular violation by clicking in the *ID* column.

   This step displays the *Message-based spreadsheet*.
3. Click the button to open the *Waveform Viewer* window.

In the *Waveform Viewer* window, check the marker that appears on the waveform. This marker is positioned at a point where more than one bits are found changing in the same cycle. As a result, the gray-encoding check is violated. Therefore, this specific transition is a "witness" to the failure.

For example, consider the following *Waveform Viewer* window:

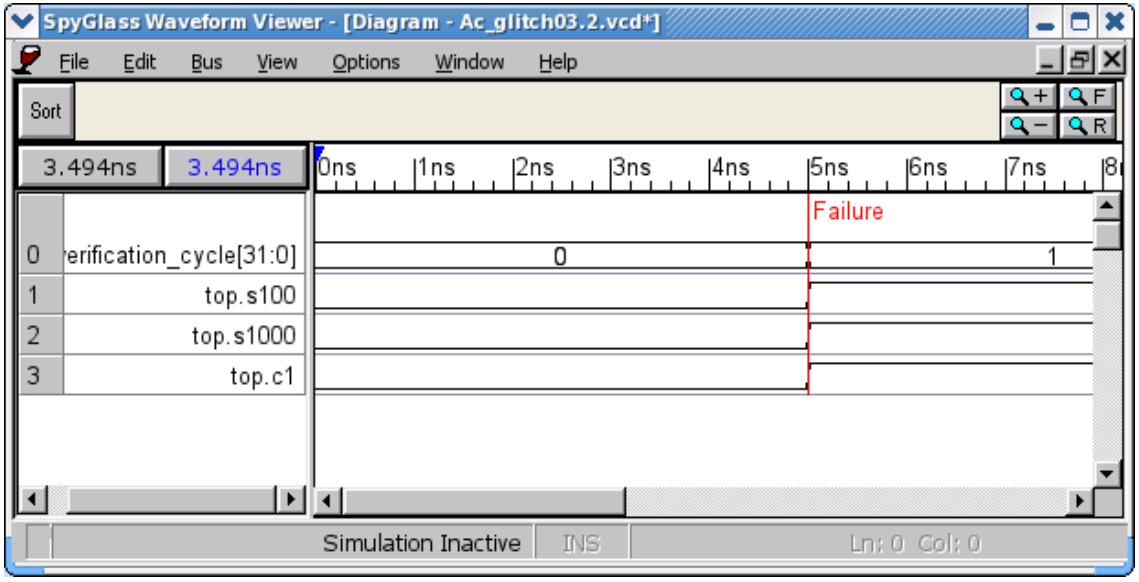


FIGURE 238. The Waveform Viewer Window

In the above window, the marker appears at a point where both asynchronous signals are moving from state 0 to state 1 simultaneously, thereby violating gray-encoding check.

Reasons for Failure

Following are some reasons that may cause false failures:

- Presence of a potential reset/clear signal. In this case, provide the reset/clear in the SGDC file.
- The setup (*clocks*, *resets*, *set_case_analysis*, and *input* constraints) is not correct and complete. In this case, use *Formal Setup Rules* to check the correctness of the setup.
- Initial state values shown in the *Waveform Viewer* window are not correct. In this case, provide a correct initial state in the SGDC file.

Message 2

The following message appears when the gray-encoding check is

performed:

```
[AcG3_2] [WARNING] Glitch check performed on destination
<destination-type> '<pin/port-name>' clocked by '<clock-name>'
(<num-sources> source(s), <num-domains> domain(s)). Multi-
source toggling check :'Partially-Proved'
```

For arguments details of the above message, see *Table 6*.

Potential Issues

This violation appears if multiple same domain source signals in a crossing toggle at the same time.

Consequences of Not Fixing

If you do not fix this violation, the design may contain glitches.

How to Debug and Fix

The *Ac_glitch03* rule reports the *Partially-Proved* status when SpyGlass is not able to conclude (falsify or prove) gray-encoding check in the given amount of time.

In such cases, perform the following actions to enable the tool to complete the analysis:

- Increase the assertion run-time by using the *fa_atime* parameter.
- Use incremental analysis approach by using the *fa_propfile* parameter.
- Use the *fa_abstract* parameter that applies abstraction technique to reduce complex verification problem into simpler and solvable problem. For details, see *Performing Functional Analysis in SpyGlass CDC*.

Message 3

The following message appears when the gray-encoding check is performed:

```
[AcG3_4] [INFO] Glitch check performed on destination
<destination-type> '<pin/port-name>' clocked by '<clock-name>'
(<num-sources> source(s), <num-domains> domain(s)). Multi-
source toggling check :'PASSED'
```

For arguments details of the above message, see *Table 6*.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Violation messages for which the status is reported as `PASSED` are informational messages, and do not require debugging.

By default, the message tree of such violations does not show the control crossing that passed the verification. To view such crossings, set the *fa_msgmode* parameter to `all` or `pass`.

Message 4

The following message appears when the gray-encoding check could not be performed:

[AcG3_3] [ERROR] Destination <destination-type> '<pin/port-name>' clocked by '<clock-name>' can glitch (<num-sources> source(s), <num-domains> domain(s)). Reason :'<reason(s)>'

The arguments of the above message are explained below:

Argument	Description
<destination-type>	Specifies the destination type as flop, latch, library-cell, port, or black box
<pin/port-name>	Specifies the hierarchical name of an instance pin or instance output port
<clock-name>	Specifies the name of the clock driving the destination. In case of multiple clocks, all clock names are shown.
<num-sources>	Specifies total number of asynchronous sources

Argument	Description
<num-domains>	Specifies total number of asynchronous domains
<reason>	Specifies one or a combination of the following reasons: • Sources from different domains in fanin • Source reconverges • Signals from destination domain in fanin • Source is port • Source is black-box • Sources from same domain in fanin • Presence of combinational logic specified through constraint • Unconstrained signal converges with source If the rule reports a combination of any of the above reasons, they are shown as a comma-separated list.

Potential Issues

This violation appears in the following cases:

- If source signals from different domains converge before reaching to the destination signal
- If synchronous signals and source signals together drive the destination signal
- If a source signal diverges and converges back before reaching to the destination
- If the source is a port
- If the source is a black box terminal
- If license is not available
- If multiple top-level modules are specified
- If the *fa_msgmode* parameter is set to `none`
- If source converges with an unconstrained primary port

Consequences of Not Fixing

If you do not fix this violation, the design may contain glitches.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the *Rule-based spreadsheet*.

   This spreadsheet lists details of all violations of the *Ac_glitch03* rule.

2. From the rule-based spreadsheet, open the spreadsheet of a particular violation by clicking in the *ID* column.

   This step displays the *Message-based spreadsheet*.

3. Based on the reason reported in the violation message, perform appropriate actions, as described below.

Actions Performed based on the Reason Reported

The following points describe the reported reasons and the corresponding actions to be performed:

- `Sources from different domains in fanin`

  This reason is reported if asynchronous sources from different domains are reaching to the destination.

  The following figure shows the schematic of such violation:

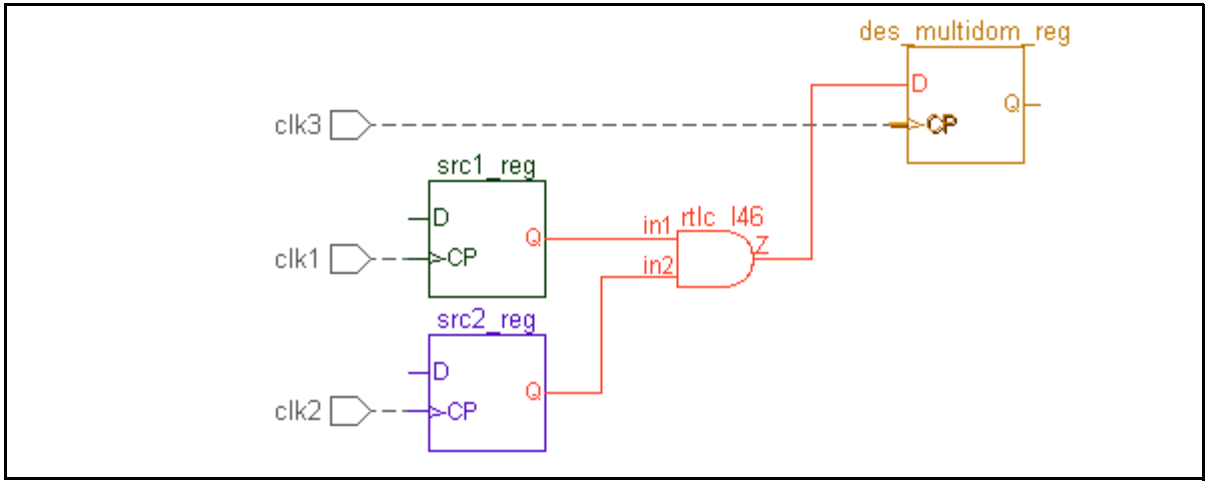


FIGURE 239. Asynchronous Sources From Different Domains Reaching Destination

Action: To fix such violations:

- ❒ Ensure that the setup of clocks is correct.
- ❒ Check if a source in other domain is quasi-static.

- `Signals from destination domain in fanin`

  This reason is reported if a synchronous source is present in the fan-in of the destination signal.

  The following figure shows the schematic of such violation:

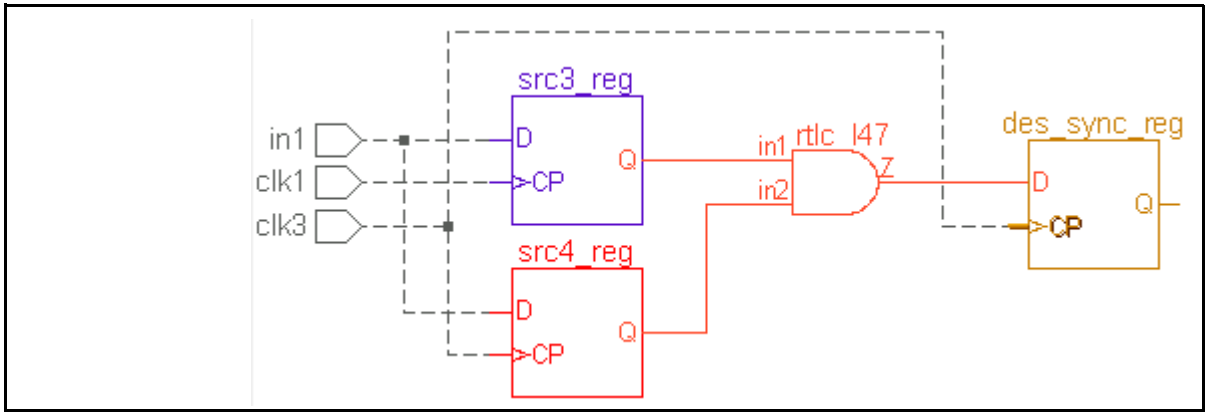


FIGURE 240. Presence of Synchronous Source in the Fan-in of Destination

Action: To fix such violations:

- ❒ Ensure that the setup for clocks is correct and check if synchronous source is quasi-static.
- ❒ Else, first synchronize the asynchronous source before it converges with synchronous sources.

- `Source reconverges`

  This reason is reported if an asynchronous source reaches its destination through multiple paths.

  To view the schematic of such a violation, click the *Reconverging_Sources_03.csv* tab of the message-based spreadsheet and click the  button.

  The following figure shows the schematic of such violation:

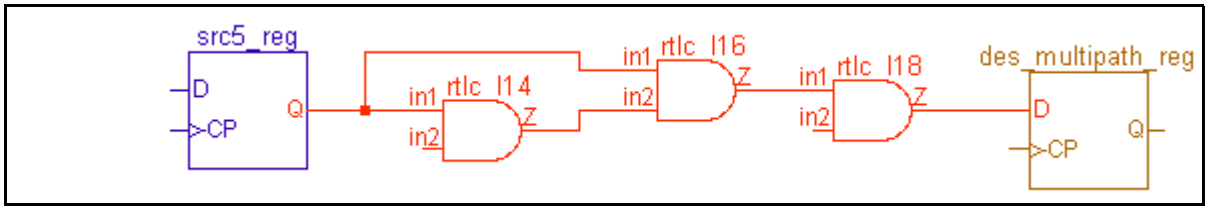


FIGURE 241. Asynchronous Source Reaching Destination through Multiple Paths

Action: To fix such violations:

- ❒ Modify the design so that the reported source reaches its destination through a single path.
- ❒ Ensure that timing delays of all paths from where source reaches the destination are same.

■ `Source is port` and `Source is black box`

This reason is reported if an asynchronous source is a port or a black box terminal.

The following figure shows the schematic of a violation in which the source is a black box terminal:

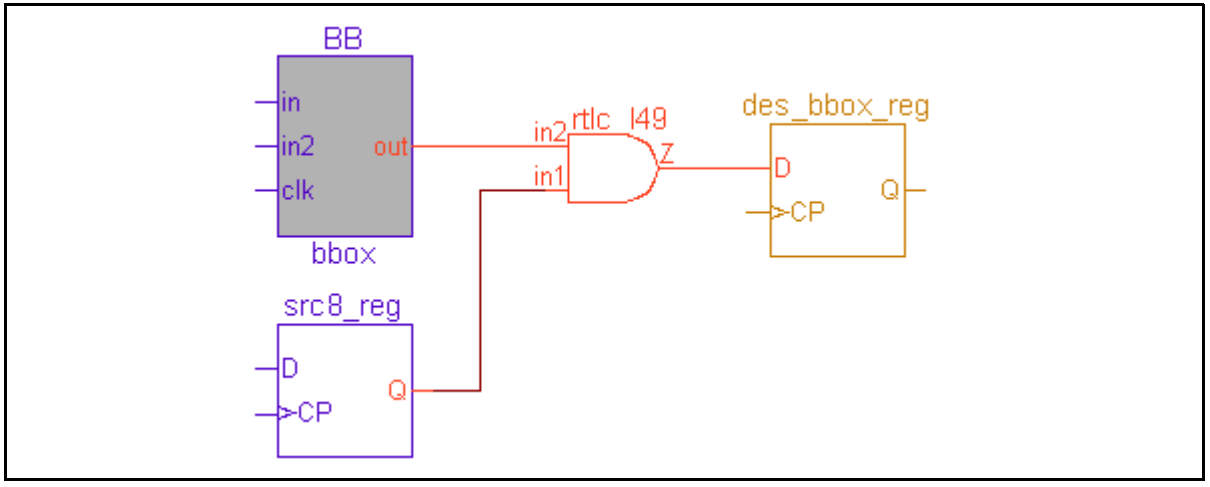


FIGURE 242. Asynchronous Source is a Black Box Terminal

In such cases, gray-encoding check cannot be performed as one of the

asynchronous sources is a port or a black box terminal.

- Unconstrained signal converges with source

  This reason is reported if an unconstrained port converge with a source as shown in figure.

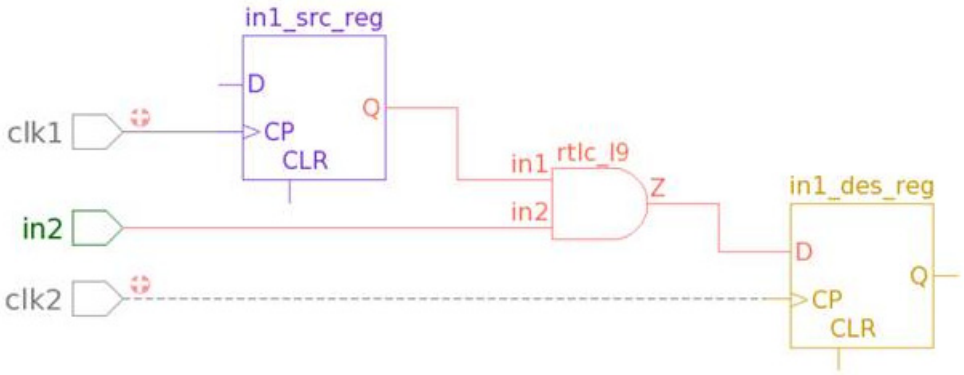


FIGURE 243. Unconstrained Signal Converges with Source

Action: To fix such violations, ensure that proper constraints, such as quasi_static, input, abstract_port, set_case_analysis are applied on the unconstrained signal.

Example code and/or Schematic

For example on *Message 1*, see *How to Debug and Fix*.

For example on *Message 4*, see *How to Debug and Fix*.

Schematic Details

The *Ac_glitch03* rule highlights path from all sources (asynchronous and synchronous) to destination.

Sources of the same domain appear in the same color.

Default Severity Label

Error | Warning | Info

Rule Group

VERIFY

Reports and Related Files

- *The Advanced CDC Report*
- *Overconstrain Info File*
- *The Glitch_detailed Report*
- The following spreadsheets:
  - Rule-based spreadsheet

    This spreadsheet appears when you click on the header of a violation. The following figure shows the rule-based spreadsheet:

A ID	B TYPE	C DEST.	D CLOCK(s	E GRAY-ENCODING CHECK	F REASON	G TAL SOURC	H TAL SOURCE DOM
13	flop	test.des_bbox	test.clk3	DISABLED	Source is black-box	2	1
14	flop	test.des_port	test.clk3	DISABLED	Source is port	2	1
17	flop	test.des_multipath	test.clk3	DISABLED	Source re converges	2	1
18	flop	test.des_multidom	test.clk3	DISABLED	Sources from different domains in fanin	2	2
1A	flop	test.des_pass	test.clk3	PASSED	N.A.	2	1
1B	flop	test.des_fail	test.clk3	FAILED	Functional Failure	2	1

FIGURE 244. Rule-Based Spreadsheet of the Ac_glitch03 Rule

NOTE: *If you run the Ac_glicth03 rule in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the corresponding Message-based spreadsheet. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.*

  - Message-based spreadsheet

    This spreadsheet appears when you click on a specific violation message of this rule. It contains three tabs, as shown in the following

figure:

A	B	C	D
Type	Source	Clock(s)	Internal Clock Domain Tag
flop	test.src10	test.clk1	0
flop	test.src11	test.clk1	0

FIGURE 245. Message-Based Spreadsheet of the Ac_glitch03 Rule

The following table describes the information under each tab of this spreadsheet:

Tab Name	Description
Asynchronous_Sources	Shows information of asynchronous signals in the fan-in of a destination The following columns are present under this tab: • Type: Specifies the object type of an asynchronous source signal, such as flip-flop, latch, black box, or primary input. • Source: Specifies the name of an asynchronous source signal. • Clock(s): Specifies the name of clocks reaching the source signal. • Internal Clock Domain Tag: Specifies a unique tag number generated for a clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.

Tab Name	Description
Synchronous_Sources	Shows information about destination-domain signals in the fan-in of a destination. The following columns are present under this tab: • Type: Specifies the object type of a synchronous source signal, such as flip-flop, latch, black box, or primary input. • Synchronous Signals: Specifies the name of a destination-domain signal. • Clock(s): Specifies the name of clocks reaching the synchronous source signal. • Internal Clock Domain Tag: Specifies a unique tag number generated for a clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.
Reconverging_Sources	Shows information about asynchronous sources that have multiple paths to a destination. From this tab, you can view the schematic that displays a source reaching the destination through multiple paths. The following columns are present under this tab: • ID: Displays a link to the schematic of the violation • Type: Specifies the object type of asynchronous source signal, such as flip-flop, latch, black box, or primary input. • Source: Specifies the name of asynchronous source signal. • Clock(s): Specifies the name of clocks reaching the source signal. • Internal Clock Domain Tag: Specifies a unique tag number generated for a clock net connected to a sequential element or a black box. For details, see *Using the Clock Domain Tag*.

For all the above three tabs, the header section of the message-based spreadsheet displays a destination name as a comment in the following format:

```
#Destination <type>: <pin/port-name>
```

Where:

- *<type>* specifies the destination type as `flop`, `latch`, `library-cell`, `port`, or `black box`.
- *<pin/port-name>* specifies the hierarchical name of an instance pin or instance output port.

Ac_glitch04

Reports glitches on synchronized data path crossings or unsynchronized crossings

When to Use

Use this rule to detect glitches in data path of clock domain crossings.

Prerequisites

The *Ac_glitch04* rule works only with the *Advanced_CDC* and *adv_checker* license features.

Description

The *Ac_glitch04* rule checks glitch-prone combinational logic on the synchronized data crossings or Unsynchronized crossings. This rule detects two types of glitches:

Glitch on Synchronized Data Path

Glitch on Unsynchronized Path

Glitch on Synchronized Data Path

A synchronized data path would always have a combo gate in its path because there will be some qualifier merging with the source at some point. User puts qualifier to control the meta-stability in the path. However, it might happen that because of the combo gate, glitch is blocked or not blocked by the qualifier. Consider the following example:

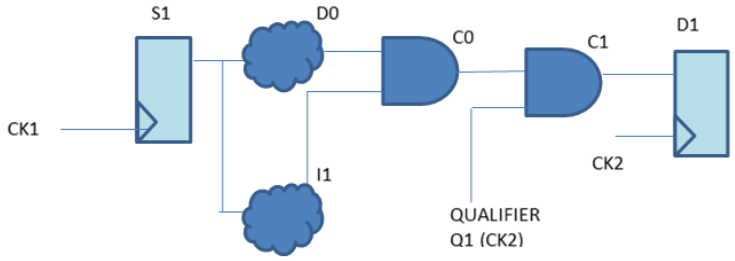


FIGURE 246.

In the above example the crossing from S1 to D1 is synchronized using the qualifier and hence there is no meta-stability issue in the path.

But the path from S1 to D1 has glitch issue. In the above example since S1 diverges and converges back at C0, glitch is produced at C0 because of different delay in path D0 and I1. Now this glitch propagates downstream which may cause functional failure. But the glitch can be blocked using the same qualifier (CK2). Hence the output of C1 can be made stable using the same qualifier and is a good value to pass to D1.

Glitch on Unsynchronized Path

In the above figure, when the blocking condition at C1 is not met due to any reason like Qualifier Q1 being replaced with an invalid qualifier or C1 gate being replaced by an invalid gate, then the glitch produced at C0 can reach the destination.

NOTE: *Ac_glitch03 also performs glitch check on unsync crossing when glitch_check_type parameter is set to unsync or all.*

This rule also performs *Unate/Binate Analysis*. You can configure the Ac_glitch04 rule to report violations related with same source reconvergence when the reconverging paths have different polarities or at least one path has an unknown polarity. For details, see *no_unate_reconv*.

Rule Exceptions

- This rule does not consider crossings that are filtered by using the *cdc_false_path* or *ip_block* constraint.
- This rule does not report violations for the same source reconvergence on the input pin of a MUX. This is because, only one input pin is active at a time.

Parameter(s)

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *check_multiclock_bbox*: Default value is `no`. Set this parameter to `yes` to show violations for the crossings in which a destination black box

receives multiple clocks but no SGDC constraint is defined on any of the black-box data pins receiving the clocks.

- *allow_combo_logic*: Default value is `no`. Set this parameter to `yes` to ignore combinational logic in the data transfer path between flip-flops at clock domain crossing.
- *enable_and_sync*: Default value is `no`. Set this parameter to `yes` to enable the *AND Gate Synchronization Scheme*.
- *enable_mux_sync*: Default value is `recirculation`. Set this parameter to an appropriate value to enable a particular synchronization scheme. Other possible values are `none`, `mux_select`, and `all`.
- *sync_reset*: Default value is `no`. Set this parameter to `yes` to allow at most one gate of an AND/NAND/OR/NOR type to be the synchronous reset for the flip-flop that follows.
- *glitch_protect_cell*: Default value is `NULL`. Specify a comma or space-separated list of glitch protection cell names for the *Glitch Protection Cell Synchronization Scheme*.
- *show_parent_module_in_spreadsheet*: Adds the PARENT_MODULE column in the rule-based spreadsheet of the supported rules.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *cdc_attribute* (Optional): Use this constraint to specify mutually exclusive and unrelated signals such that convergence-related violations are suppressed for such signals.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule-checking.
- *signal_type* (Optional): Use this constraint to specify the signal type (control or data).
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

Message 1

The rule reports the following message:

```
[WARNING | INFO] Destination <destination-type> '<pin/port-name>' clocked by '<clock-name>' can glitch (<num-sources> source(s), <num-domains> domain(s)). Reason :'<reason>'
```

The arguments of the above message are explained below:

TABLE 7 Argument details of the Ac_glitch04 rule

Argument	Description
<destination-type>	Specifies the destination type as flop, latch, library-cell, port, or black box
<pin/port-name>	Specifies the hierarchical name of an instance pin or instance output port
<clock-name>	Specifies the name of the clock driving the destination. In case of multiple clocks, all clock names are shown.
<num-sources>	Specifies total number of asynchronous sources
<num-domains>	Specifies total number of asynchronous domains
<reason>	Specified the reason for the violation. Can be either of Reconvergence with enable condition or Reconvergence without enable condition.

Potential Issues

This violation appears with the “Reconvergence without enable condition” reason if an asynchronous source reaches its destination through multiple paths.

The rule generates an Info message if the reason is “Reconvergence with enable condition”.

Consequences of Not Fixing

If you do not fix this violation, the design may contain glitches.

How to Debug and Fix

To debug the violation of this rule, open the message based spreadsheet and view the schematic.

To fix such violations:

- Modify the design so that the reported source reaches its destination through a single path.
- Ensure that valid qualifier blocks the source/glitch.

Example code and/or Schematic

For example on *Message 1*, see *How to Debug and Fix*.

Schematic Details

The *Ac_glitch04* rule highlights the re-converging path from particular source to a destination.

Default Severity Label

Warning | Info

Rule Group

VERIFY

Reports and Related Files

- *Overconstrain Info File*
- The following spreadsheets:
  - Rule-based spreadsheet

    This spreadsheet appears when you click on the header of a violation. The following figure shows the rule-based spreadsheet:

A ID	B TYPE	C DEST.	D CLOCK(s)	E REASON	F TOTAL SOURCES	G TOTAL SOURCE DOMAINS	H WAIVED
8	flop	top.d2	top.clk3	Reconvergence without enable condition	1	1	No
9	flop	top.d1	top.clk3	Reconvergence with enable condition	1	1	No

FIGURE 247. Rule-Based Spreadsheet of the Ac_glitch04 Rule

- Message-based spreadsheet

  This spreadsheet appears when you click on a specific violation message of this rule.

	A Schematic	B Type	C Signal Name	D Reconvergence	E Synchronization Reason	F Qualifier	G Clock(s)	H Internal Clock Domain Tag
1	1	Destination flop	top.d2	Yes	Unsync	-	top.clk3	0
2	1	Source flop	top.s1	Yes	Unsync	N.A.	top.clk1	1

FIGURE 248. Message-Based Spreadsheet of the Ac_glitch04 Rule

Clock_glitch01

Reports enable signals that are gating clocks but are not the output of a flip-flop

When to Use

Use this rule to detect glitches caused by a gating clock with enable signals that are not the output of a flip-flop.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By using the automatically-generated clock signals after setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_glitch01* rule reports enable signals that are gating clocks but are not the output of a flip-flop.

This rule checks for clock-gating done by using gates, such as AND gate, NAND gate, and NOR gate. It also checks enable signals in a latch-based gating.

The following figure explains the purpose of this rule.

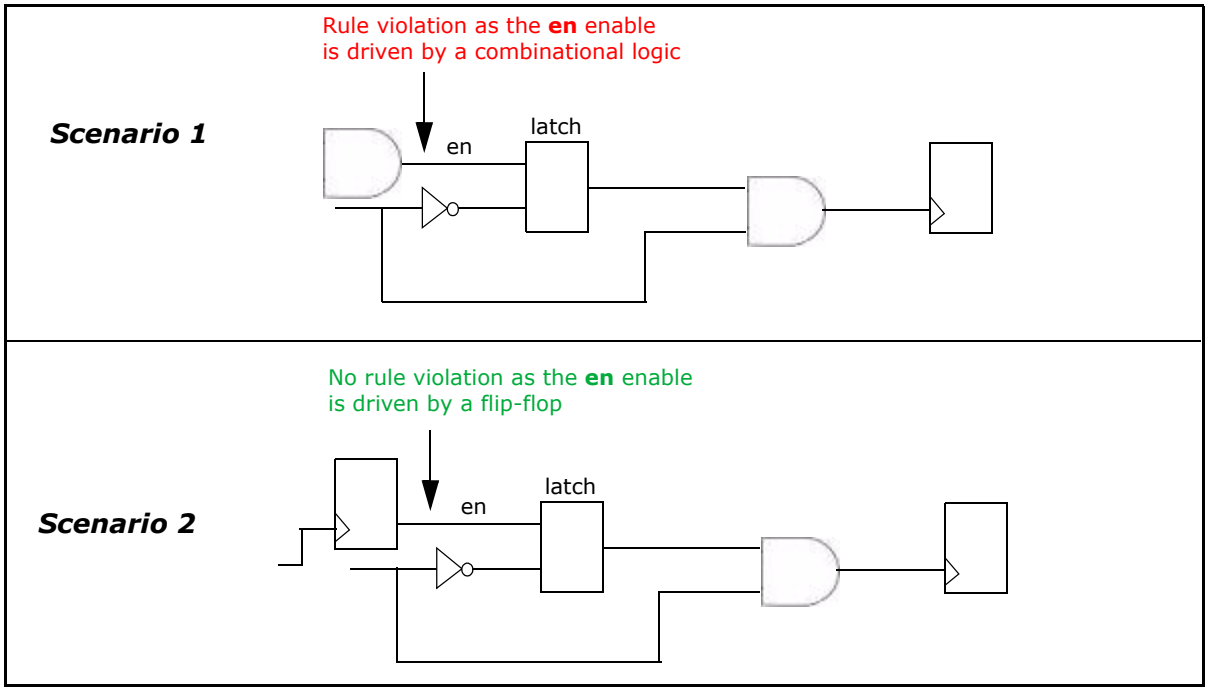


FIGURE 249. Clock_glitch01 examples

This rule ignores enable signals that are directly connected to input ports specified by the *input* constraint.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.
- *report_all_clockgate_enables*: Default value is `no`. Set this parameter to `yes` to enable the Clock_glitch01 rule to report all the enable nets that are directly merging with a clock signal.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for the enable signal *<en-name>* that is not registered:

```
[INFO] Enabled signal '<en-name>' should be a state signal
```

Potential Issues

This violation appears if an enable signal gating a clock is not the output of a flip-flop.

Consequences of Not Fixing

If you do not fix this violation, glitches may occur in the design.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. Open the incremental schematic of the violation of this rule.
2. Check the reported enable signal that is gating a clock but is not the output of a flip-flop.
3. Correct the logic by making the enable signal as the output of the flip-flop to ensure that no glitches occur on the gated clock signal.

Example Code and/or Schematic

Consider following schematic:

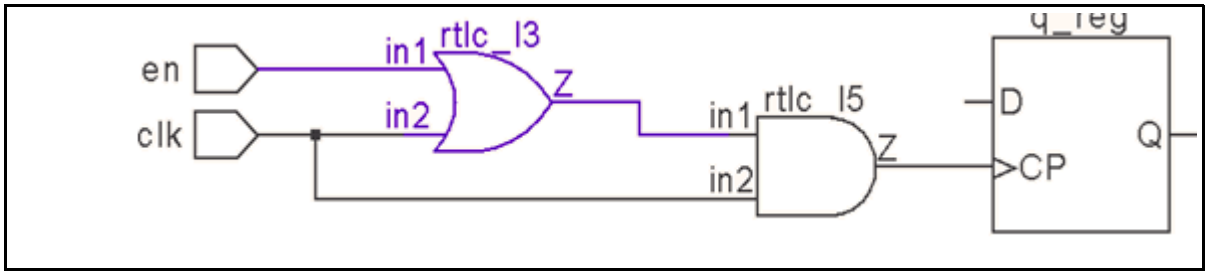


FIGURE 250. Schematic of the Clock_glitch01 Rule

In the above example, the enable signal `en` is not driven by a flip-flop. Therefore, the *Clock_glitch01* rule reports a violation.

Schematic Details

The *Clock_glitch01* rule highlights the clock-gating instance and the gated clock in the schematic.

Default Severity Label

Info

Rule Group

`VERIFY`

Reports and Related Files

No report or related file

Clock_glitch02

Reports clocks that are gated without latching their enable signal properly

When to Use

Use this rule to find clocks that are exposed to glitches, as the enable signal is not latched in the inactive half of clock cycle.

Description

The *Clock_glitch02* rule reports a violation if a clock signal is gated by using the normal AND/OR/NOR/NAND gate, and an enable signal is provided directly without latching the enable signal.

The gating enable signal should be latched with an inverted clock going to the enable pin of the latch. Latching the clock enable in the inactive half of the clock cycle ensures that clock-gating setup requirement is always met.

Rule Exceptions

This rule does not check if the gating logic is fed to a positive or negative edge triggered flip-flop. Such cases are caught by the *Ac_xclock01* rule.

For example, this rule does not report a violation in the following case because the gating logic is fed to a negative edge triggered flip-flop:

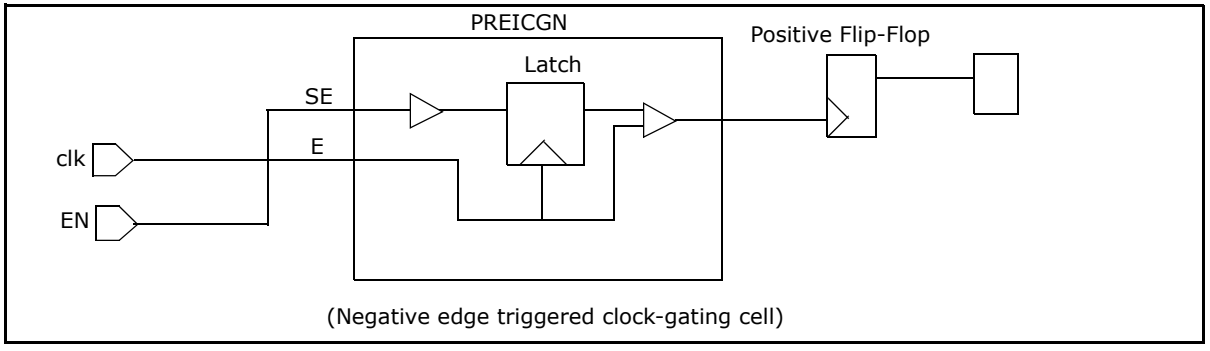


FIGURE 251. Clock_glitch02 Example

Design Impact

Functionality (bug escape), Portability, and re-use

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use the auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *report_all_clockgate_enables*: Default value is `no`. Set this parameter to `yes` to enable the Clock_glitch02 rule to report all the enable nets that are directly merging with a clock signal.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

Message 1

The following message appears for a clock signal *<clk-name>* whose enable signal *<en-name>* is not latched in the inactive half of the clock cycle:

```
[INFO] Gate the clock '<clk-name>' after latching the enable
'<en-name>' in the inactive half of the clock cycle
```

Potential Issues

This violation is reported if your design contains combinational gates in a clock path for which the other input is not latched.

Consequences of Not Fixing

Gated clocks may result in timing hazards, such as glitches that can lead to duty cycle distortion.

It is best to change the gating signal for a clock when the clock is inactive. If a gating signal is functionally changed while the clock is active, there is always a possibility of a glitch.

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

The following message appears for a clock signal *<clk-name>* whose enable signal *<en-name>* is latched in the inactive half of the clock cycle, but the latch input is tied to a constant:

```
[INFO] Gate the clock '<clk-name>' after latching the enable
'<en-name>' in the inactive half of the clock cycle (latch data
is tied to constant)
```

NOTE: *This violation is not reported if the clock_reduce_pessimism parameter is not set to check_enable_for_glitch.*

Potential Issues

This violation is reported if a latch input is tied to constant value.

Consequences of Not Fixing

Enable tied to constant is equivalent to not having an enable/gating logic.

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 3

The following message appears for a clock signal *<clk-name>* whose enable signal *<en-name>* is latched in the inactive half of the clock cycle, but the latch data is driven by hanging net:

```
[INFO] Gate the clock '<clk-name>' after latching the enable
'<en-name>' in the inactive half of the clock cycle (latch data
is undriven)
```

NOTE: *This violation is not reported if the clock_reduce_pessimism parameter is not set to check_enable_for_glitch.*

Potential Issues

This violation is reported if your design contains latch data that is undriven or coming from a hanging net.

Consequences of Not Fixing

Hanging nets cannot be controlled by the user. Therefore, they cannot behave as enable signals.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, check the clock-gating logic present in the fan-out of clock net.
3. You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

Consider the following code snippet in which this rule reports a violation:

```
module test(q,d,clk,en,in1);
  input en, clk,d,in1;
  output reg q;
  wire tclk, ten;
  assign ten = en & in1;
  assign tclk = (clk & ten);
  always@(posedge tclk)
    q <= d;
endmodule
```

The following figure shows the schematic for the above example:

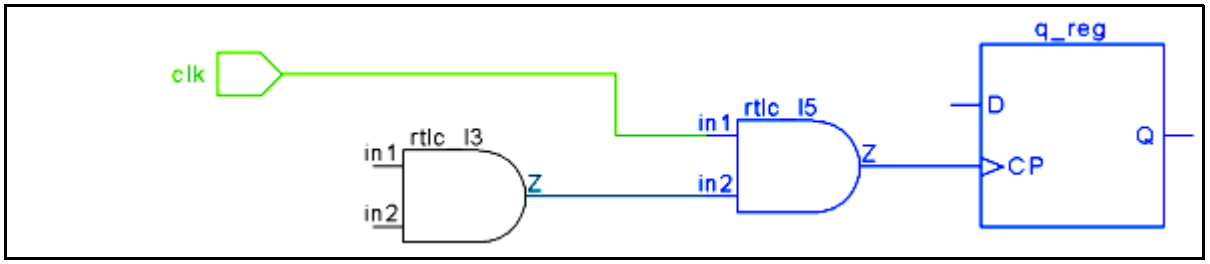


FIGURE 252. Schematic of the Clock_glitch02 Rule

To fix this violation, add the following clock-gating instance instead of `assign tclk = (clk & ten)`:

```
gating_cell CG1(.en(1'b0),.clk(clk1),.out(g_clk2));
module gating_cell(en,clk,out);
  input en, clk;
  output out;
  reg t1;
  always@(clk)
    if(!clk)
      t1 <= en;
  assign out = t1 & clk;
endmodule
```

Schematic Details

The *Clock_glitch02* rule highlights the following information in different colors in the schematic:

- Source clock signal
- Clock-gating signal
- Gating instance

Default Severity Label

Info

Rule Group

VERIFY

Reports and Related Files

Clock_glitch02.csv: This is the rule-based spreadsheet that contains details of all violations of this rule. The spreadsheet includes details of the clock, enable signal, and the latch input type.

Clock_glitch03

Reports clock signals that pass through a MUX and reconverge back on the same MUX

When to Use

Use this rule to detect cases in which a clock glitch can be created due to clock signals that re-converge back on the same MUX.

Description

The *Clock_glitch03* rule reports clock signals that reconverge back on the same MUX.

In such cases, outputs of a MUX after passing through a clock-pin of flip-flop/latches reconverge back on the same MUX. This results in creation of a glitch.

For example, this rule reports a violation in the following scenario:

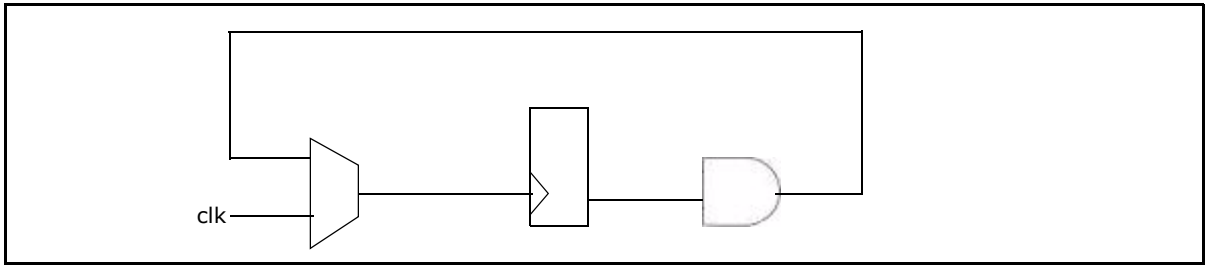


FIGURE 253. Clock_glitch03 Rule Violating Scenario

You should be cautious about clock path and MUX select transitions in such cases. It is recommended to review this thoroughly and avoid the usage, if possible.

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use the auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location where a MUX is being inferred when a clock signal *<sig-name>* re-converges on the MUX:

```
[INFO] Clock signal <sig-name> re-converges on mux
```

Potential Issues

This violation is reported if your design contains unconstrained MUX select signals in clock re-convergence paths.

Consequences of Not Fixing

If you do not fix this violation, glitches may get created in clock path.

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

In case more than one MUX is re-converging on the clock path, only one message appears with the count of re-converging MUXes:

```
[INFO] Clock signal <sig-name> re-converges on mux (<num-muxes>
```

muxes present in path)

where, *<num-muxes>* is the number of re-converging MUXes present in the path.

Potential Issues

This violation is reported if your design contains more than one MUX in the clock-path in which a clock signal re-converges after passing through flip-flops/latches.

Consequences of Not Fixing

If you do not fix this violation, glitches may get created in clock path.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message to check the MUX where the clock signal re-converges.

To find the signals that need to be constrained, perform the following steps:

1. Back-trace the MUX select-signal till it hits the input ports, black box output, or flip-flops.
2. Apply the *set_case_analysis* constraint on appropriate signals in SGDC file.

If signals or connected nets are already constrained, perform the following steps:

1. Enable *Show Case Analysis* in the *Incremental Schematic* window to see where constant propagation is blocked.
2. Apply correct constant value to the signals by using the *set_case_analysis* constraint.

To fix the issue, use glitch free MUXes.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:

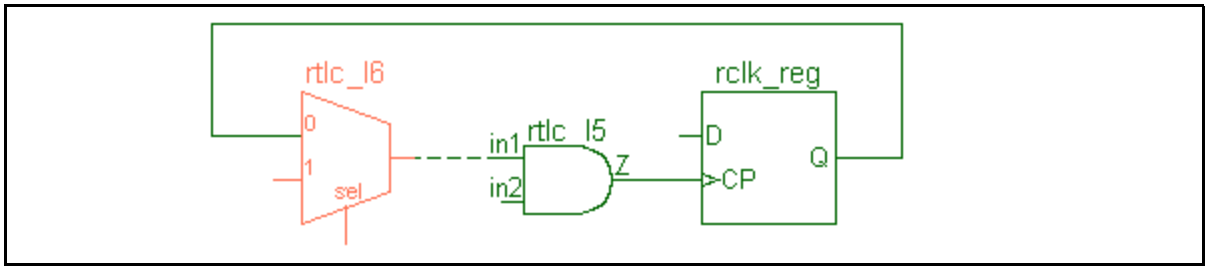


FIGURE 254. Schematic of Clock_glitch03 Rule

To fix the violation in the above case, constraint the select signal of the MUX to block the re-convergence path.

Schematic Details

The *Clock_glitch03* rule highlights a clock signal and a re-convergence feedback loop through a MUX in the schematic.

Default Severity Label

Info

Rule Group

VERIFY

Reports and Related Files

Clock_glitch03.csv: This is the rule-based spreadsheet that contains details of all violations of this rule. The spreadsheet includes details of the clock and the number of muxes.

Clock_glitch04

Reports flip-flops that converge on a clock pin of a flip-flop through a combinational logic

When to Use

Use this rule to detect the possibility of a glitch due to cases in which flip-flop outputs converge on a clock pin of another flip-flop through a combinational logic.

Description

The *Clock_glitch04* rule reports flip-flops that converge on a flip-flop clock pin through a combinational logic.

Such cases result in the creation of a clock glitch when a combinational logic is used to generate clocks.

The combinational logic can be driven by flip-flops directly.

The *Clock_glitch04* rule does not report a violation if the destination flop is not receiving a clock or the clock is constant.

This rule reports only one message if the same pair of flip-flops is converging at more than one clock pin.

Rule Exceptions

This rule ignores flip-flops converging through a MUX, as shown in the following figure:

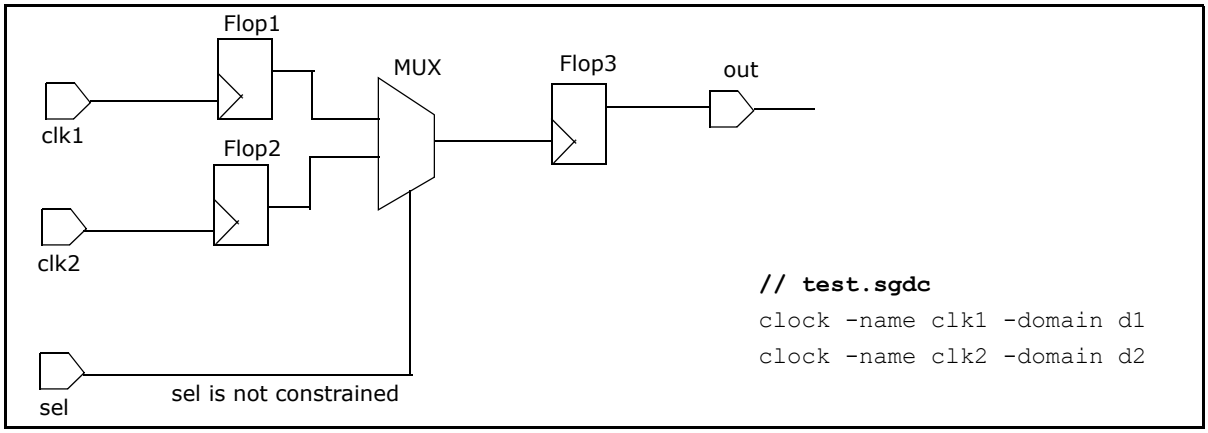


FIGURE 255. Clock_glitch04 Rule Example

Parameter(s)

report_inst_for_netlist: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when outputs of flip-flops *<flop1-name>* and *<flop2-name>* are converging to the clock pin of the flip-flop *<flop3-name>* through a combinational gate *<gate-name>*:

[INFO] Flop <flop1-name> and Flop <flop2-name> are converging through combinational logic(<gate-name>) to clock of Flop <flop3-name>

NOTE: *For RTL designs, <flop1-name>, <flop2-name> and <flop3-name> are names of the output nets of the corresponding flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to yes, they are the names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation is reported if output of flip-flops is feeding to the combinational logic, which in turn drives a generated clock.

Consequences of Not Fixing

If you do not fix this violation, glitches may occur on the generated clock path.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, review the logic displayed in the clock path.
3. Enable *Show Case Analysis* to check if the *set_case_analysis* constraints are missing in the SGDC file that makes the logic in the clock path glitch free.
4. To remove this violation, you should correct the logic to avoid glitches on clock signal.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

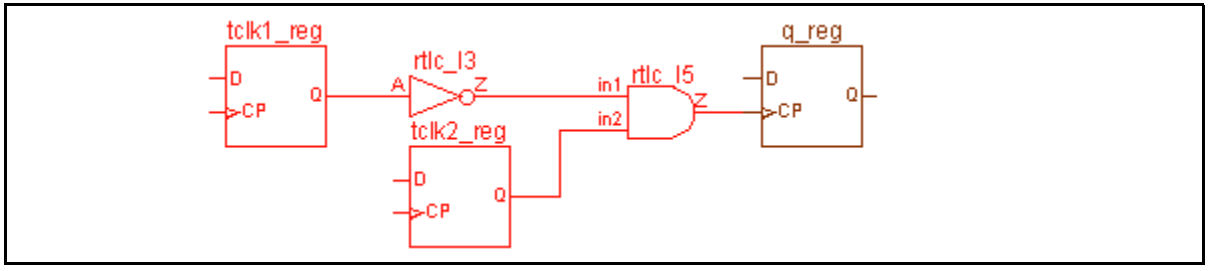


FIGURE 256. Schematic of the Clock_glitch04 Rule

To fix the violation in the above case, you should review the logic driving the `tclk1` and `tclk2` flip-flops so that no condition arises that results in glitches due to the AND gate.

Schematic Details

The *Clock_glitch04* rule highlights the following in different colors:

- Source flip-flops and the converging logic

- Destination flip-flop whose clock pin is driven by the converging instance output

Default Severity Label

Info

Rule Group

VERIFY

Report and Related File

Clock_glitch04.csv: This is the rule-based spreadsheet that contains details of all violations of this rule. The spreadsheet includes details of the signals and the converging flops.

Clock_glitch05

Flags asynchronous sources that converge with different domain clocks

When to Use

Use this rule to report asynchronous sources that converge with clocks of different domains.

Description

The *Clock_glitch05* rule reports asynchronous source signals, which gate the clock, but are not in the same domain as the clock signal. One violation per clock cone per asynchronous source is reported.

A clock cone is a net that directly drives either of the following:

- Clock pin of a sequential element
- Black box pin (without **assume_path** on that pin)
- Top-level port
- Hanging net

If a clock cone receives multiple asynchronous sources, the *Clock_glitch05* rule reports multiple violations.

NOTE: *The* `Clock_glitch05` *rule does not report hanging nets if the* `cdc_reduce_pessimism remove_redundant_logic` *parameter is specified.*

Prerequisites

Specify the following in the project file:

```
set_goal_option rules Clock_glitch05
```

Specify clock signals by:

- Using the **clock** or **generated_clock** constraint
- Setting the **use_inferred_clocks** parameter to `yes` to use automatically generated clock signals
- Using a combination of both the above methods

Parameter(s)

- *disable_seq_clock_prop*: Default value is `no`. Set this parameter to `yes` to disable propagation of clocks beyond flip-flops.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock-domain crossings involving black-box instances and clock-domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see the *Allowed Values of the cdc_reduce_pessimism Parameter* section in the *SpyGlass CDC Rules Reference* guide.
- *enable_generated_clocks*: Default value is `no`. Set this parameter to `yes` to enable SpyGlass to consider the **generated_clock** constraint.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *abstract_port* (Optional and applicable for virtual clocks): Use this constraint to define abstracted information for block ports.
- *input* (Optional and applicable for virtual clocks): Use this constraint to specify clock domain at input ports.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

- *generated_clock* (Optional): Use this constraint to specify generated/derived clocks.
- *quasi_static*: Use this constraint to specify signals that have a value which is predominantly static.

Messages and Suggested Fix

The following message is reported:

`[ERROR] Asynchronous source '<source-name>' converges with different domain clock(s) at '<converging-net>'`

Potential Issues

A clock signal is being gated by a source of domain different from the clock signal.

Consequences of Not Fixing

Gated clocks may result in timing hazards, such as glitches that can lead to duty cycle distortion.

How to Debug and Fix

Double-click the message to open the spreadsheet and/or view the incremental schematic. The spreadsheet contains the asynchronous source signal details and also all the clock signals that are reaching.

Review the logic to ensure that there are no glitches on the gated clock signal.

Example Code and Schematic

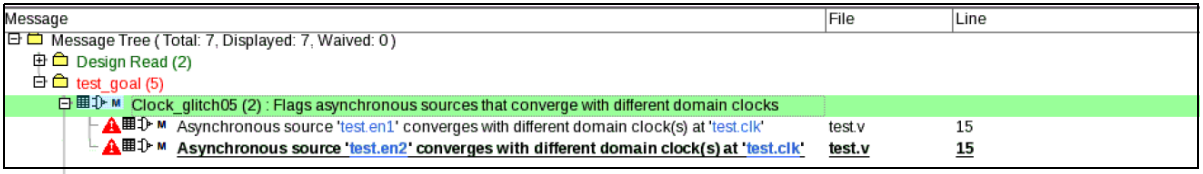
This example illustrates when the *Clock_glitch05* rule reports a violation message.

In the following design, the asynchronous source `en1` (domain: `clk1`) is merging with clocks of different domain `clk2` and `clk3`. Therefore, the *Clock_glitch05* rule reports a violation.

Similarly, the asynchronous source `en2` (domain: `clk2`) is merging with clock of different domain `clk1`. Therefore, the *Clock_glitch05* rule reports a violation.

```
module test(in, clk1, clk2, clk3, out);
input in, clk1, clk2, clk3;
output out;

reg en1, en2, des1, des2;
always @(posedge clk1)
    en1 <= in;

always @(posedge clk2)
    en2 <= in;

assign clk_merge = clk2 && clk3;

assign en = en1 && en2;
assign clk = clk_merge && en;

always@(posedge clk)
begin
    des1 <= in;
    des2 <= des1;
end

assign out = des2;

endmodule
```

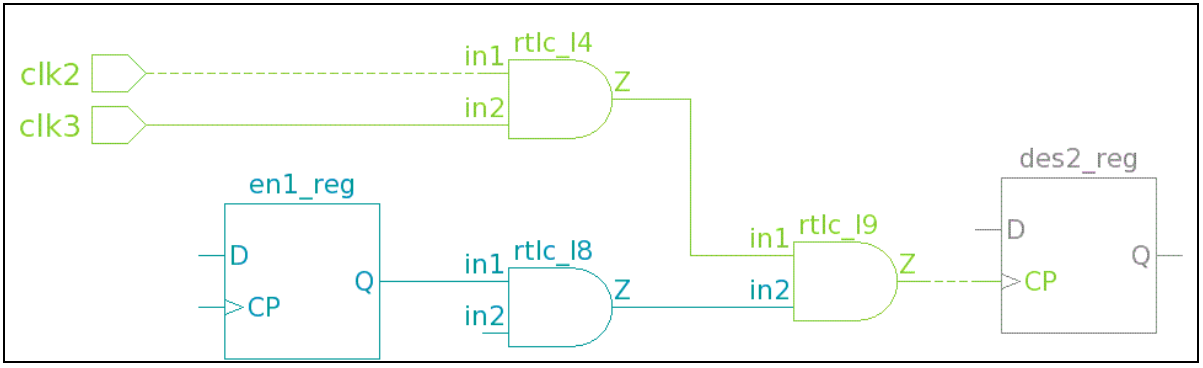
After running the *Clock_glitch05* rule, the following violation messages are generated:



Open the *Clock_glitch05* rule spreadsheet to view the source and converging net names. Alternatively, open the incremental schematic and a spreadsheet associated with each message.

	A ID	B SOURCE	C CONVERGING NET	D WAIVED
1	5	test.en1	test.clk	No
2	6	test.en2	test.clk	No

For the first violation message, the following schematic and spreadsheet is generated.



The spreadsheet is as follows:

	A Type	B Name	C Internal Clock Domain Tag
1	Asynchronous Source	test.en1	0
2	Clock	test.clk2	1
3	Clock	test.clk3	2

Default Severity Label

Error

Rule Group

VERIFY

Reports and Related Files

No report or related file

Clock Checking Rules

The SpyGlass CDC solution has the following rules for checking clock conditions:

Rule	Reports
Clock_check01	Latches, tristate gates, or XOR/XNOR gates found in a clock tree
Clock_check02	High fan-out clock nets not driven by specified placeholder cells
Clock_check03	Bus-bits used as clocks
Clock_check04	Mixed clock edges used in the design
Clock_check05	Deep ripple clock-dividers
Clock_check06a	Unexpected cells found in a clock tree
Clock_check06b	Cells that are instantiated in a clock tree and do not have consistent `threshold_voltage_group` attribute value set in the .lib files
Clock_converge01	Clocks whose multiple fan-outs converge
Clock_hier01	Reports clock-gating wrapper modules in clock-path
Clock_hier02	Reports combinational wrapper modules in clock-path
Clock_hier03	Reports combinational gates that do have valid wrapper modules in the clock-path
Ac_xclock01	Cases in which X values can be present on a clock path resulting in non-deterministic shifting during the scan operation.

Clock_check01

Reports unexpected cells, such as latches, tristate gates, or XOR/XNOR gates found in a clock tree.

When to Use

Use this rule to detect unexpected cells, such as latch, tristate, or XOR/XNOR gates in a clock tree.

Description

The *Clock_check01* rule reports unexpected cells, such as latches, tristate gates, or XOR/XNOR gates found in a clock tree.

This rule reports only first found instance of an unexpected cell even if that cell is instantiated in more than one clock-tree.

This rule reports such cells under informational message in the following cases:

- If an enable pin of a latch or a tristate gate is tied with a constant logic value. In such cases, the latch/tristate is permanently enabled or disabled.
- If one of the inputs of a XOR/XNOR gate is tied with a constant logic value. In such cases, the XOR/XNOR gate effectively works as buffer or inverter.

Rule Exceptions

This rule has the following exceptions:

- It does not report a violation for tristate gates in PAD cells.
- It ignores clocks defined by using the *clock* constraint. It automatically infers clocks and reports unexpected gates in the path of such clocks.

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *ignore_latches*: Default value is `yes`. Set this parameter to `no` to consider signals ending on the latch enable terminals.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following warning appears at the location where a clock signal of a flip-flop/latch *<inst-name>* is first used if a latch/tristate/XOR/XNOR gate is encountered in the tree of the clock signal:

[ClkC1_1] [WARNING] Unexpected <gate-type> gate (at <name>) in clock tree of flop/latch (output <obj-type> <inst-name>)

The arguments of the above message are explained below:

Argument	Description
<gate-type>	Can be latch, tristate, XOR, or XNOR.
<name>	The name of the latch, tristate signal, or the output of XOR/XNOR gate
<obj-type>	net in case of RTL designs. pin in case of netlist designs, if the *report_inst_for_netlist* parameter is set to yes. Otherwise, it is net
<inst-name>	*<flop/latch-output-net-name>* in case of RTL designs. *<flop/latch-inst-name>.<pin-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to yes. Otherwise, it is same as in case of RTL designs. The following informational message appears at the location where tristate/latch is permanently enabled/disabled or the input of XOR/XNOR is tied to active high/low

Potential Issues

This violation appears if a clock tree in your design contains cells, such as latches, tristates, or XOR/XNOR gates.

Consequences of Not Fixing

Such cells in a clock tree may block further propagation of clock or may

change the clock behavior.

As designs are very sensitive to clocks, the clock tree should contain only permitted cells.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, check the clock path containing unexpected gate.
3. Check if the violation message reports tristate/latch gate as *permanently enabled*. This implies that the gate is always enabled and acts like a buffer.
4. Check if the violation message reports XOR/XNOR gate as *acting as inverter* or *acting as buffer*. This implies that the gate acts like a buffer or an inverter.
5. Check if the violation message reports tristate/latch gate as *permanently disabled*. This implies that the gate is always blocked.
6. You can also view case analysis settings along with the violation of this rule.

Considering violations reported by this rule as a real error depends on your design methodology. For example, if you are using low power design, you will have latches in the clock tree. Disable the rule if it does not apply to your methodology.

Message 2

The following informational message appears at the location where a tristate/latch is permanently enabled/disabled or the input of XOR/XNOR is tied to active high/low:

[ClkC1_2] [INFO] `Unexpected <gate-type> (<state>) gate (at <name>) in clock tree of flop/latch (output <obj-type> <inst-name>)`

Where, *<state>* can be `acting as buffer/inverter` in case of XOR/XNOR gates or `permanently enabled/disabled` in case of tristate gates or latches.

Potential Issues

None

Consequences of Not Fixing

None

How to Debug and Fix

None

Example Code and/or Schematic

Consider the following example:

```
module test (clk1,clk2,in1,in2,out1,out);
input clk1,clk2,in1,in2;
output out,out1;
reg out,out1;
reg off1;
wire o_xor = clk1^ clk2;
wire o_xor2 = clk1^ o_xor;
always @ (posedge clk1)
begin
  off1 <=in1;
end
always @ (posedge o_xor)
  out <= off1;
always@(posedge o_xor2)
  out1 <= in2;
endmodule
```

For the above example, the *Clock_check01* rule reports a violation as an unexpected XOR gate, o_xor, is found in the clock tree.

Following is the schematic of a violation of this rule:

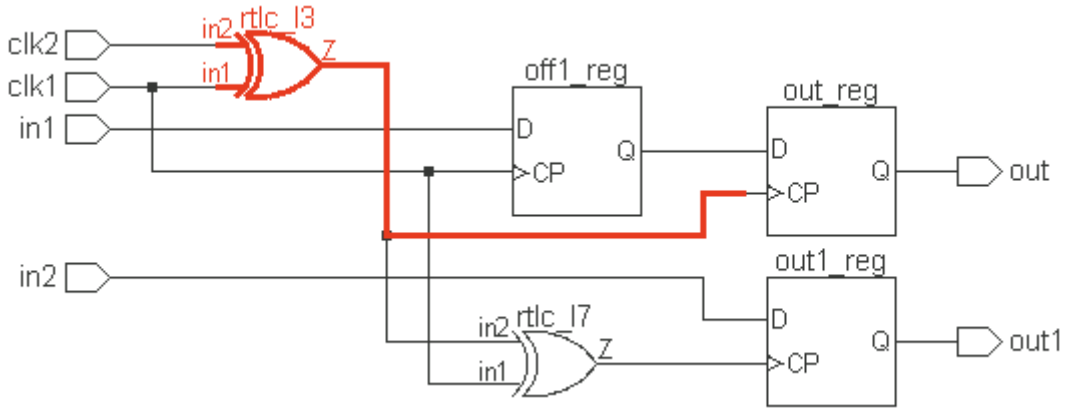


FIGURE 257. Schematic of the Clock_check01 Rule Violation

Schematic Details

The *Clock_check01* rule highlights unexpected gate and path from this unexpected gate to a clock pin of a flip-flop or a latch.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report and related file

Clock_check02

Reports high fan-out clock nets that are not driven by any of the specified placeholder cell

When to Use

Use this rule to analyze clock trees when certain fan-out nets have weak drive strength because of a high fan-out and missing placeholder cells.

Prerequisites

Specify the following information before running this rule:

- Specify the name of placeholder cells by using the *CTS_placeholder_cells* parameter.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By automatically inferring clocks by setting the *use_inferred_clocks* parameter to `yes`
  - By using a combination of both the above methods

Description

The *Clock_check02* rule reports clock nets that have a fan-out greater than the value specified by the *clock_fanout_max* rule parameter, and these clock nets are not driven by the instances of cells specified by the *CTS_placeholder_cells* parameter.

Parameter(s)

- *CTS_placeholder_cells*: Default value is `NULL`. Specify a comma or space-separated list of placeholder cells.
- *clock_fanout_max*: Default value is `24`. Specify a positive integer value to specify a maximum fan-out limit for clocks.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use automatically-generated clock information.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for the clock signal *<clk-name>* of the fan-out *<num>* that is more than the value specified by the *clock_fanout_max* parameter, and this clock signal is not driven by instances of any of the placeholder cells specified using the *CTS_placeholder_cells* parameter:

[WARNING] Clock “<clk-name>” drives <num> flops that exceeds maximum allowed limit '$clock_fanout_max' and is not driven by any placeholder cells

Potential Issues

This violation appears if the total number of sequential elements driven by a clock net exceeds the maximum limit specified by the *clock_fanout_max* parameter, and such clocks nets are not driven by any of the specified placeholder cells.

Consequences of Not Fixing

If you do not fix this violation, the high fan-out clock nets may have weak drive strength if they are not driven by placeholder cells.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, check the cell present in the fan-out of the clock net.

3. Check if you have specified the correct cell name in the *CTS_placeholder_cells* parameter.
4. Insert a CTS placeholder cell for this clock net.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v
module test (d,clk,q);
 input [3:0]d;
 input clk;
 output [3:0]q;
 reg [3:0]q;
 BBOX b1 (.A(clk),.B(),.Q(clk1));
 always @ (posedge clk1)
     q <= d;
endmodule

module top(d,clk,q);
 input [3:0]d;
 input clk;
 output [3:0]q;
 test abc(d,clk,q);
endmodule
```

```
// constr.sgdc
current_design top
clock -name top.abc.clk
assume_path -input A -output Q
            -name BBOX
```

```
set_parameter clock_fanout_max 3
set_parameter CTS_placeholder_cells top
```

For the above example, the *Clock_check02* rule reports a violation because the `top.abc.clk` clock drives four flip-flops that exceeds the maximum allowed limit of three, and this clock is not driven by the specified placeholder cell.

The following figure shows the schematic of this violation:

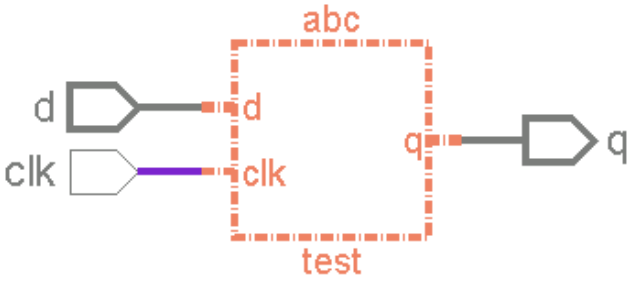


FIGURE 258. Schematic of the Clock_check02 Rule Violation

Schematic Details

The *Clock_check02* rule highlights the clock net that is not driven by the specified placeholder cell.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Clock_check03

Reports bus bits that are used as clocks

When to Use

Use this rule to identify bus-bits that are used as clocks.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to auto-generate clock signals
- By using a combination of both the above methods

Description

The *Clock_check03* rule reports bits of bus signals that are used as clocks or latch enables. It also reports bus signals present in the fan-in cone of the pin of a flip-flop or enable pin of a latch enable.

This rule reports only one violation per path even if there are multiple bus bits in the path.

Parameter(s)

- *show_derived_busclocks*: Default value is `no`. Set this parameter to `yes` to report bus-bit signals that are in derived clock path, in addition to bus-bit signals that are in the primary clock path.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears if the clock signal *<clk-name>* that is a bus-bit or has a bus-bit *<bit-name>* in its hierarchy, is first set:

```
[WARNING] Clock '<clk-name>' is a bus-bit or is connected to a bus-bit '<bit-name>'
```

Potential Issues

This violation appears if your design contains bus-bits that are used as clocks.

Consequences of Not Fixing

Some methodologies do not support bus-bits being used as clocks because that can break downstream tools.

How to Debug and Fix

Open the incremental schematic to view the violating clock signal.

If required, define clocks by using scalar signals with different naming convention, such as `CLK_0` and `CLK_1`.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

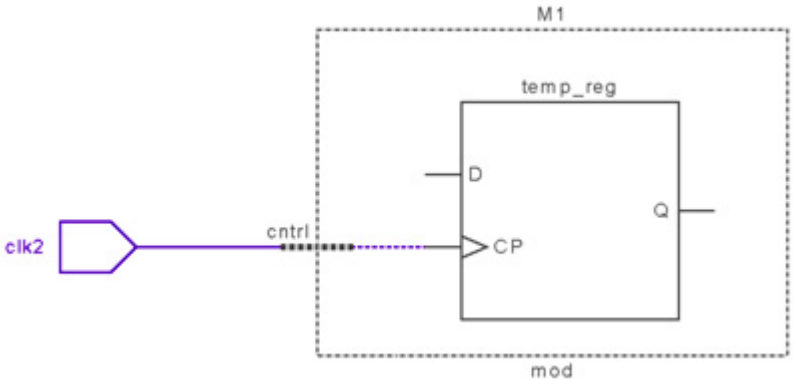


FIGURE 259. Schematic of the Clock_check03 Rule Violation

In the above example, the *Clock_check03* rule reports a violation because `CLK2` is connected to the bus-bit `cntrl[0]` that is used as a clock.

Schematic Highlight

The *Clock_check03* rule highlights the path from a bus-bit clock net to a flip-flop clock pin or a latch enable pin.

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

No report or related file

Clock_check04

Reports the usage of both the edges (positive and negative) of a clock

When to Use

Use this rule to detect clocks for which both the edges are used.

Description

The *Clock_check04* rule reports the usage of both edges of a clock.

Note the following points:

- This rule reports one violation per clock signal.
- This rule also reports in case of half synchronizers. You can waive off such messages, if required.

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *clock_edge*: Default value is `positive`. Set this parameter to `negative` to report clock descriptions for which a positive edge specified.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where the specified clock signal *<clk-name>* is used with an edge different from the recommended edge (`$clock_edge`):

`[WARNING]` `Recommended edge ($clock_edge) of clock '<clk-name>' not used`

Potential Issues

This violation appears if your design contains clocks that are used at both positive and negative edges.

Consequences of Not Fixing

`Designs using both clock edges are quite sensitive to clock duty cycles. Therefore, designs containing clocks for which both the edges are used are less portable.`

How to Debug and Fix

This rule reports a violation when the recommend edge of a clock is not used. You can view the Incremental Schematic of the violation message to check the clock path.

You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

Consider the following Verilog file:

```
// test.v
module test(input d, clk, output reg q);
always @(negedge clk)
  q <= d;
endmodule
```

For the above example, this rule reports a violation as a negative edge for the `clk` clock is used when the recommended edge is positive

Following is the schematic of this violation:

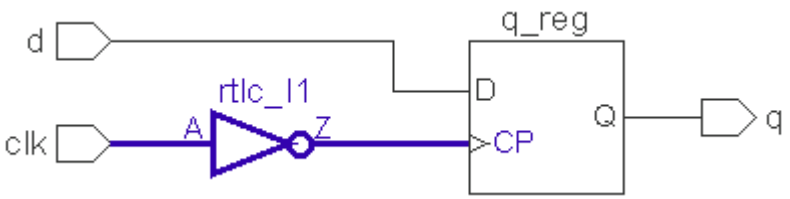


FIGURE 260. Schematic of the Clock_check04 Rule Violation

To fix this violation, use a single edge for the `clk` clock.

Schematic Details

The *Clock_check04* rule highlights the path from a clock net to the first flip-flop encountered that is triggered by a different clock edge.

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

No report related or file

Clock_check05

Reports deep clock divider chains

When to Use

Use this rule during the RTL or pre-layout phase to detect deep clock divider chains.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By automatically inferring clocks by setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_check05* rule reports ripple clock dividers that exceed the specified depth.

Rule Functioning

This rule functions in the following manner:

- This rule starts traversal from a clock source (primary input or a black box pin) and traverses the hierarchy to look for ripple clock dividers that exceed the specified depth.
- By default, a clock does not propagate across a latch enable. As a result, traversal stops when the clock signal reaches the latch enable and traversal continues on other paths.

  To let a clock divider chain to traverse through a latch enable, remove the `latch_en` value from the *clock_reduce_pessimism* parameter.

Parameter(s)

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a

MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.

- *clock_ripple_depth*: Default value is `2`. Set the value of this parameter to a positive integer value to specify the maximum allowed depth of ripple clock divider.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears at the location where the output of *<flop1-name>* is first assigned when *<flop1-name>* is the first flip-flop in a ripple clock divider chain that goes up to the flip-flop *<flop2-name>* (and beyond) where the chain is `$clock_ripple_depth` long:

[WARNING] `Deep ripple clock divider (flop <flop1-name> to flop <flop2-name> and beyond) of depth exceeding $clock_ripple_depth found`

NOTE: *For RTL designs, <flop1-name> and <flop2-name> are names of the output nets of*

the corresponding flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to yes, <flop1-name> and <flop2-name> are names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.

Potential Issues

This violation appears if your design contains deep clock divider chains.

Consequences of Not Fixing

If you do not fix this violation, it may introduce significant clock skew in the derived clocks with respect to the primary source clocks.

Clock skew in a design may cause timing issues in a design.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. Double-click on the rule violation.
2. Open the incremental schematic.
3. If the ripple depth is acceptable for the given design, specify this depth by using the *clock_ripple_depth* parameter.
4. Use a PLL, DLL, or some other clock generation structures in the design with well-controlled skew.

Example Code and/or Schematic

Consider the following file specified during SpyGlass analysis:

```
module test(input d1, d2, d3, clk, in, output q);
wire q1, q2, q3;
flop f1 (d1, clk, q1);
flop f2 (d2, q1, q2);
assign q3 = q2 & in;
flop f3 (d3, q3, q);
endmodule

module flop (input d, clk, output reg q);
always @(posedge clk)
  q <= d;
endmodule
```

Here, the default depth of ripple clock divider is 2. However, the depth of the clock divider chain is greater than two between the `q1` and `q2` flip-flops.

Therefore, the *Clock_check05* rule reports the following violation:

`Deep ripple clock divider (flop 'test.q1' to flop 'test.q2' and beyond) of depth exceeding 2 found`

The following figure shows the schematic of this violation:

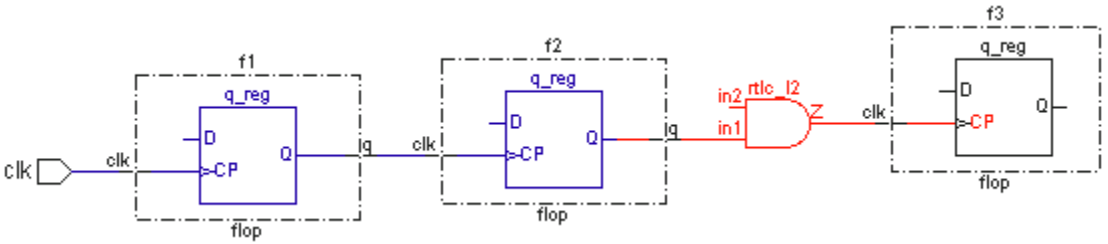


FIGURE 261. Schematic of the Clock_check05 Rule Violation

Schematic Details

The *Clock_check05* rule highlights the following information in different colors in the schematic:

- The clock divider path from the starting flip-flop up to the flip-flop where the depth exceeds the specified limit
- Output of the flip-flop exceeding the limit

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

- *The CDC Report*
- Clock_check05.csv

Clock_check06a

Reports unexpected cells found in a clock tree

When to Use

Use this rule during the RTL or pre-layout phase to detect unexpected cells in a clock tree.

Prerequisites

Specify the following information before running this rule:

- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By automatically inferring clocks by setting the *use_inferred_clocks* parameter to `yes`
  - By using a combination of both the above methods
- Specify either of *unexpected_ckcells_file* and *expected_ckcells_file* parameters.

  If you do not specify any of these parameters, SpyGlass does not report any cell as unexpected. However, if you specify both the parameters, the *unexpected_ckcells_file* parameter is ignored.

Description

The *Clock_check06a* rule reports unexpected cells in a clock tree.

Rule Exceptions

This rule ignores all SpyGlass generated buffers with names as `rtlc_*` and marks them as unexpected cells.

Parameter(s)

- *cdc_express*: Default values is `no`. Set this parameter to `peakmem` to reduce peak memory. Other possible value is `yes`.
- *unexpected_ckcells_file*: Default value is `NULL`. Specify a comma or space-separated list of files containing a list of cells that are not allowed in clock trees.

- *expected_ckcells_file*: Default value is NULL. Specify a comma or space-separated list of files containing a list of cells that are allowed in clock trees.
- *use_inferred_clocks*: Default value is no. Set this parameter to yes to use auto-generated clock information.
- *same_domain_at_gate*: Default value is no. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Mandatory): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where the instance *<inst-name>* of the cell *<cell-name>* (specified using the *unexpected_ckcells_file* parameter) is found in the clock tree of the clock *<clk-name>*:

[WARNING] Unexpected cell '<cell-name>' (Instance '<inst-name>') found in tree of clock '<clk-name>'

Potential Issues

This violation appears if the clock tree in your design contains any unexpected user-instantiated or synthesized cells.

Consequences of Not Fixing

If you do not fix this violation, the unexpected cells in the clock tree may cause timing issues or glitches.

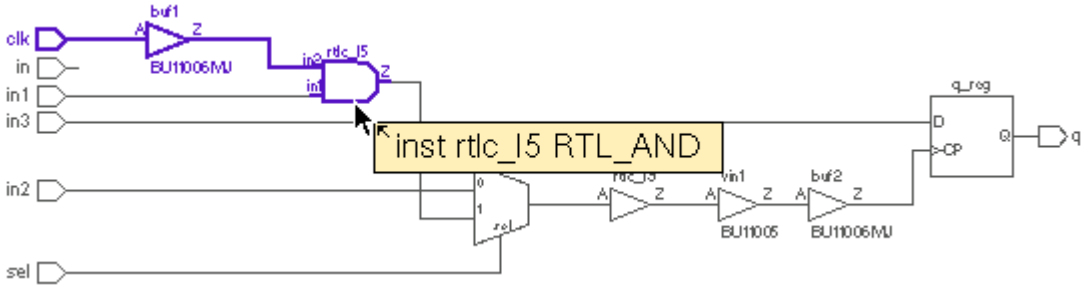
How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. Open incremental schematic of the violation.
2. Check if you intent to retain the reported cell in the clock tree.
3. If you want to retain the reported cell in the clock tree, specify that cell in the *expected_ckcells_file* parameter.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
expected_ckcells_file parameter set to file1.txt

# file1.txt
B.11006MJ
```

FIGURE 262. Schematic of the Clock_check06a Rule Violation

In the above example, the *Clock_check06a* rule reports the `RTL_AND` cell as the unexpected cell because this cell is not specified by the *expected_ckcells_file* parameter.

Schematic Details

The *Clock_check06a* rule highlights the path from the clock source to the unexpected cell instance in the schematic.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Clock_check06b

Reports the cells in a clock tree that do not have the same threshold_voltage_group attribute value

When to Use

Use this rule during the RTL or pre-layout phase to check.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By automatically inferring clocks by setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_check06b* rule reports cells that are instantiated in a clock tree and do not have consistent `threshold_voltage_group` attribute value set in the .lib files.

The *Clock_check06b* rule considers the first-found .lib cell instantiated in the clock tree as the reference and reports another cell instantiated in the clock tree if the `threshold_voltage_group` attribute value for the cell is different.

Parameter(s)

- *cdc_express*: Default values is `no`. Set this parameter to `peakmem` to reduce peak memory. Other possible value is `yes`.
- *same_threshold_all_cktree*: Default value is no. Set this parameter to yes to check for all clock trees together.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Mandatory): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location of the instance *<inst2-name>* of the .lib cell *<cell2-name>* found in the clock tree of the clock *<clk2-name>*, and the .lib cell does not have the same `threshold_voltage_group` attribute value as the first-found cell *<cell1-name>* (instantiated as *<inst1-name>* in the clock tree of the clock *<clk1-name>*:

[WARNING] Cell '<cell2-name>' (instance: '<inst2-name>', threshold value: '<value2>', clock: '<clk2-name>') has different threshold voltage value from the first cell '<cell1-name>' (instance: '<inst1-name>', threshold value: '<value1>', clock: '<clk1-name>')

Where clocks *<clk1-name>* and *<clk2-name>* are the same clocks if the *same_threshold_all_cktree* parameter is not set and can be same or different if the *same_threshold_all_cktree* parameter is set.

Potential Issues

This violation appears if library cells in a clock tree have different threshold voltage group value.

Consequences of Not Fixing

If you do not fix this violation, library cells with a different threshold voltage may trigger at different voltage levels and cause unexpected timing issues or blocking of clock paths.

How to Debug and Fix

To debug and fix this violation, perform following steps:

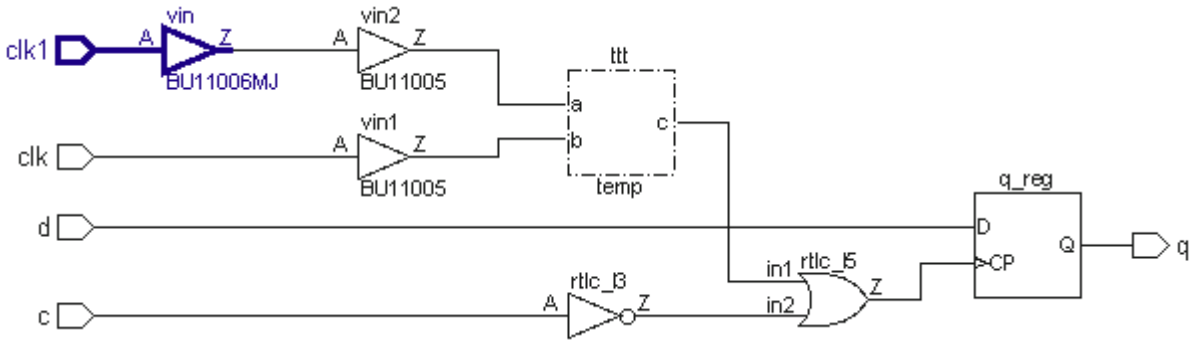
1. Double-click on the violation message, and check the cell highlighted in the RTL.

   The highlighted cell is the one that is used as the reference cell.
2. Check the other cells reported in the violation message.

   Both these cells should have a different threshold voltage group defined in the library file.
3. Correct the logic to ensure that all cells in a clock tree have the same `threshold_voltage_group` attribute.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
set_parameter use_inferred_clocks yes
read_file -type gateslib buffer_cells.lib
set_parameter run_cells_in_cktree_rules yes
set_parameter same_threshold_all_cktree yes
```

FIGURE 263. Schematic of the Clock_check06b Rule Violation

In the above example, the `BU11006MJ` cell has a different threshold voltage value from the first cell `BU11005`.

Therefore, the *Clock_check06b* rule reports a violation.

Schematic Details

The *Clock_check06b* rule highlights the path from the clock source to the

rule-violating cell instance in the schematic.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Clock_check10

Reports the clock signals that are used as non-clock signals

When to Use

Use this rule to identify the clock signals that are used as non-clock signals.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By automatically inferring clocks by setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_check10* rule reports a violation if a clock signal drives any of the following objects:

- Data pin of a flip-flop, latch, or library cell
- Control pin of a flip-flop, tristate, or library cell
- Reset pin of a flip-flop, latch, or library cell
- Black box with the *abstract_port* and *signal_in_domain* constraint
- Library cell inputs for which type (as `clock`, `control`, or `reset`) could not be inferred from the library. For example, the read address of a memory cell
- Primary port

For example, this rule reports a violation in the following scenario:

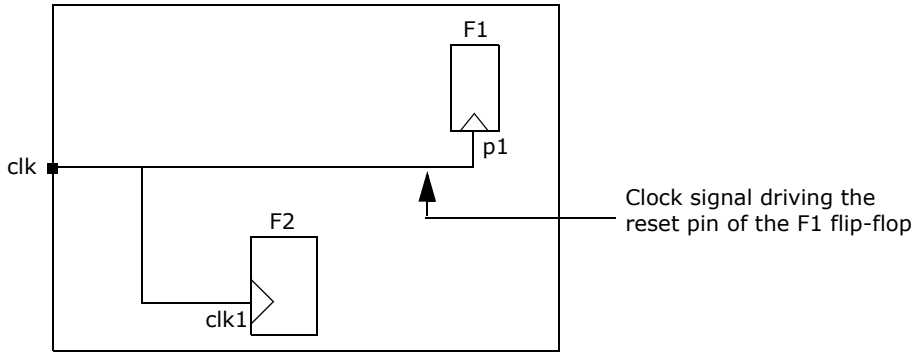


FIGURE 264. Clock_check10 Rule Violation

The *Clock_check10* rule supports multi-line highlighting to show the lines where it is used as a clock signal and as a non-clock signal.

Rule Exceptions

A flip-flop whose output is used as a clock signal is considered as a clock generator. Since clock generators have a feedback loop, the *Clock_check10* rule does not report violations for the data and control pins of such a flip-flops.

The following figure shows such a scenario:

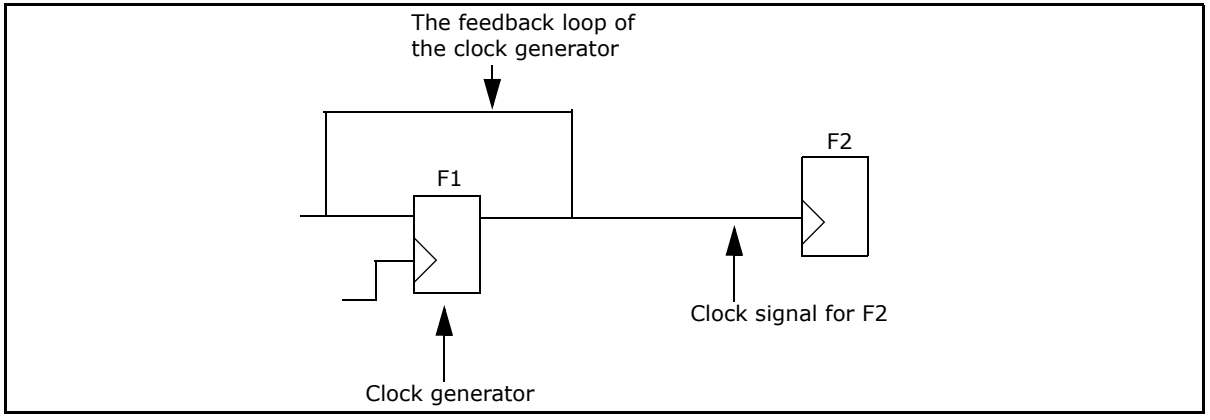


FIGURE 265. Clock_check10 Rule Exception

Parameter(s)

- *clock_usage*: Default value is `data,control,reset,bbox,others`. Specify a comma-separated list of values to specify signal types to be reported for non-clock usage. Possible types are `data`, `control`, `reset`, `port`, `bbox`, `others`, `derived`, and `all`.
- *report_detail*: Default is `all`. The parameter supports the *Clock_check10* and the *Setup_library01* rules. Set this parameter to a supported rule to report all the violations of the specified rule and a reduced set of violations of the other supported rule. Other possible value is `none`.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *enable_debug_data*: Default value is `no`. Set this parameter to `yes` to view debug information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Constraint(s)

- *clock* (Mandatory): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears when the *<clk-name>* clock signal

reaches the data of a flip-flop, latch, or library cells:

[WARNING] Clock signal '<clk-name>' (at <flop|latch|blackbox|library-cell> '<name1>') is reaching to data of <flop|latch|tristatelibrary-cell> <name2>

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name1>	For RTL designs, this argument is the output net name of the instance where the clock signal is used as a clock. For library cells which do not have functional attributes, such as memory cells, this argument is the name of the input pin. For netlist designs, if the *report_inst_for_netlist* parameter is set to yes, this argument is the instance name otherwise it is the same as in case of RTL designs.
<name2>	For RTL designs, this argument is the output net name of the instance where the clock signal is used as a data. For library cells which do not have functional attributes, such as memory cells, and the connected net is not an internally generated net, the corresponding net name is reported. Else, the cell pin name is reported. For netlist designs, if the *report_inst_for_netlist* parameter is set to yes, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if your design contains a clock signal that reaches the data input of a flip-flop, latch, or library cell.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug this violation, analyze the incremental schematic of the violation.

To fix this violation, avoid using clocks as non-clock signals.

Message 2

The following message appears when the *<clk-name>* clock signal reaches the control of a flip-flop, tristate, or library cell:

[WARNING] Clock signal '<clk-name>' (at <flop|latch|blackbox|library-cell> '<name1>') is reaching to control of <flop|library-cell|tristate> <name2>

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name1>	Refer to the *<name1>* description.
<name2>	For RTL designs, this argument is the output net name of the instance where the clock signal is used as a control. For library cells which do not have functional attributes, such as memory cells, and the connected net is not an internally generated net, the corresponding net name is reported. Else, the cell pin name is reported. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if your design contains a clock signal that reaches the control input of a flip-flop, tristate, or library cell.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug and fix this violation, analyze the incremental schematic of the violation. Check if you can avoid using clocks as non-clock signals.

Message 3

The following message appears when the *<clk-name>* clock signal reaches a primary port:

```
[WARNING] Clock signal '<clk-name>' (at
<flop|latch|blackbox|library-cell> '<name1>') is reaching to
port <port-name>
```

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name1>	Refer to the *<name1>* description.
<port-name>	Refers to the port name where the clock signal is arriving.

Potential Issues

This violation appears if your design contains a clock signal that reaches an output port.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug this violation, analyze the incremental schematic of the violation.

To fix this violation, avoid using clocks as non-clock signals.

Message 4

The following message appears when the *<clk-name>* clock signal reaches a black box instance at the *<pin-name>* pin:

```
[WARNING] Clock signal '<clk-name>' (at
<flop|latch|blackbox|library-cell> '<name1>') is reaching to
blackbox instance of <name2>
```

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name1>	Refer to the *<name1>* description.
<name2>	For RTL designs, this argument is the input net name of the black box instance where the clock signal has reached. If the connected net is an internally generated net, the corresponding black box pin name is reported. If the input net is not an internally generated net, the corresponding connected net name is reported. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if your design contains a clock signal that reaches a black box instance pin where the *abstract_port* and *signal_in_domain* constraints are defined.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug this violation, analyze the incremental schematic of the violation.

To fix this violation, avoid using clocks as non-clock signals.

Message 5

The following message appears when the *<clk-name>* clock signal reaches the reset of a flip-flop, latch, or library cell:

[WARNING] Clock signal '<clk-name>' (at <flop|latch|blackbox|library-cell> '<name1>') is reaching to reset of <flop|library-cell|latch|tristate> <name2>

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name1>	Refer to the *<name1>* description.
<name2>	For RTL designs, this argument is the output net name of the instance where the clock signal is used as a reset. For library cells which do not have functional attributes, such as memory cells, and the connected net is not an internally generated net, the corresponding net name is reported. Else, the cell pin name is reported. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if your design contains a clock signal that reaches the reset input of a flip-flop, latch, or library cell.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug this violation, analyze the incremental schematic of the violation.

To fix this violation, avoid using clocks as non-clock signals.

Message 6

The following message appears when the *<clk-name>* clock signal reaches the input pin of a library cell of type excluding clock, control, and reset:

```
[WARNING] Clock signal '<clk-name>' (at
<flop|latch|blackbox|library-cell> '<name1>') is reaching to
others of <library-cell> <name2>
```

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name1>	Refer to the *<name1>* description.
<name2>	For RTL designs, this argument is the output net name of the instance where the clock signal is used as others, such as read address of a memory cell. For library cells which do not have functional attributes, such as memory cells, and the connected net is not an internally generated net, the corresponding net name is reported. Else, the cell pin name is reported. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if your design contains a clock signal that reaches the library cell inputs for which type (as `clock`, `control`, or `reset`) could not be inferred from the library. For example, the read address of a memory cell.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug this violation, analyze the incremental schematic of the violation.

To fix this violation, avoid using clocks as non-clock signals.

Message 7

The following message appears when a clock signal *<clk-name>* has been used as a data signal, but not been used as a clock signal:

```
[WARNING] Clock signal '<clk-name>' (not used as clock) is
reaching to <flop|latch|blackbox|library-cell> <name2>
```

The arguments of the above message are explained below:

Argument	Description
<clk-name>	Hierarchical name of the clock signal
<name2>	For RTL designs, this argument is the output net name of the instance where the clock signal is used as data. For library cells which do not have functional attributes, such as memory cells, and the connected net is not an internally generated net, the corresponding net name is reported. Else, the cell pin name is reported. For netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, this argument is the instance name otherwise it is the same as in case of RTL designs.

Potential Issues

This violation appears if your design contains a clock signal that reaches the data input of a flip-flop, latch, or library cell.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug this violation, analyze the incremental schematic of the violation. To fix this violation, avoid using clocks as non-clock signals.

Example Code and/or Schematic

Consider the following schematic of the violation of the *Clock_check10* rule:

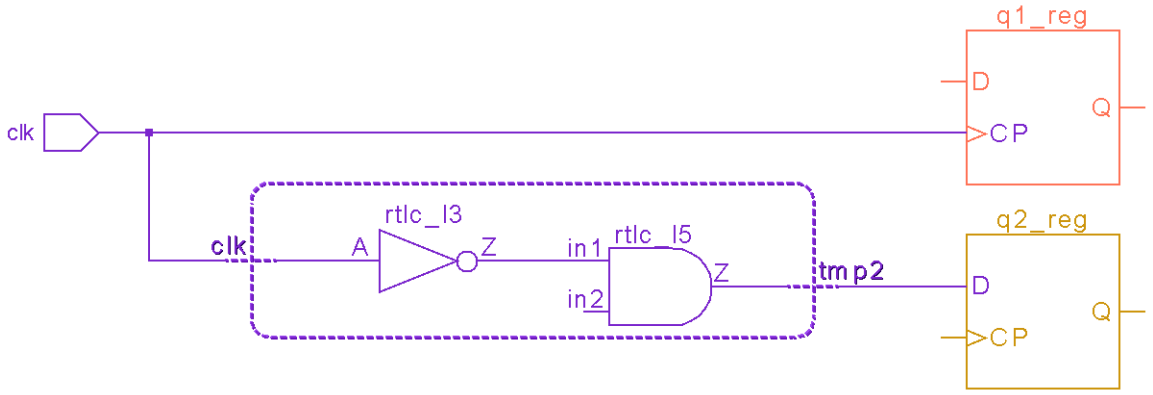


FIGURE 266. Schematic of the Clock_check10 Rule Violation

In the schematic above, the `clk` clock signal is being used as clock at `q1_reg` and data in the `q2_reg` flop. The rule highlights the clocks and the non-clock instances in different colors.

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

No report or related file

Clock_converge01

Reports the clock signal for which multiple fan-outs converge

When to Use

Use this rule to detect clocks with multiple converging fan-outs.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes`
- By using a combination of both the above methods

Description

The *Clock_converge01* rule reports clocks for which multiple fan-outs converge. See *Figure 270*.

It also reports a violation if convergence is at a mux select and its input.

Unate/Binate Analysis

You can configure the *Clock_converge01* rule to report violations related with same source reconvergence when the reconverging paths have different polarities or at least one path has an unknown polarity.

For details, see *no_unate_reconv*.

Rule Exceptions

The *Clock_converge01* rule has the following exceptions:

- This rule does not report a violation if the converged clock signal does not propagate to the clock pin of any of the sequential element in the design.

  For example, in the following scenario, the converged clock signal does not propagate to the `f1` flip-flop because the case analysis value 0 blocks the converged clock to propagate:

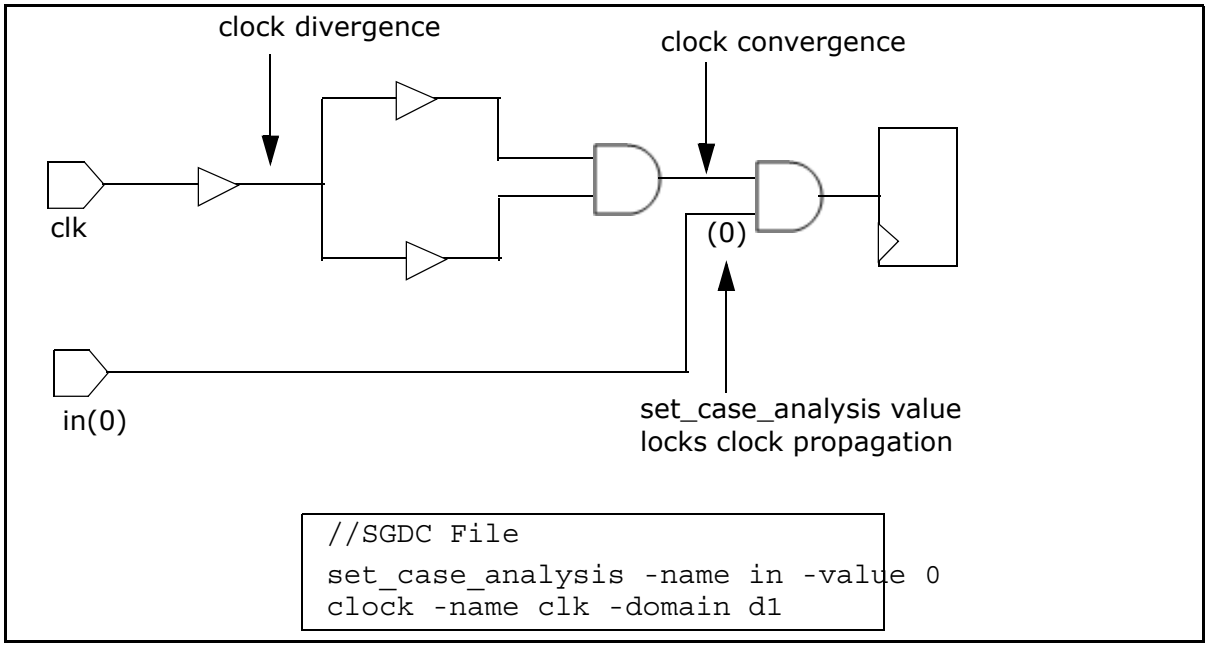


FIGURE 267. Clock_converge01 Rule Exception - 1

Similarly, in the following scenario also, the *Clock_converge01* rule does not report a violation because the converged clock signal is reaching an output port and not any sequential element:

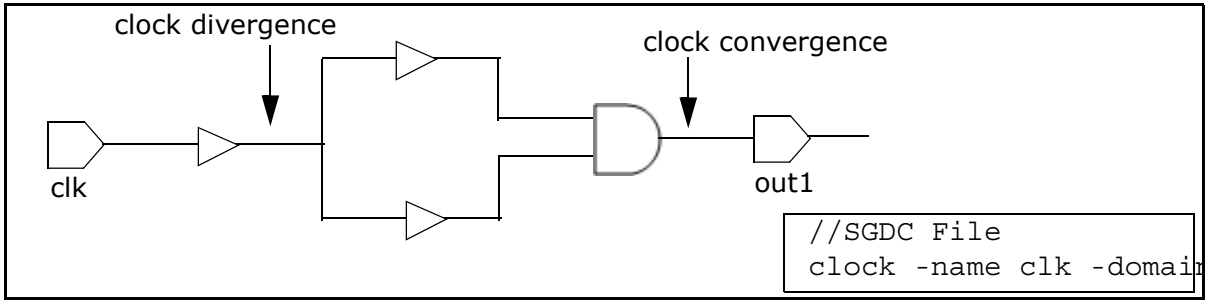


FIGURE 268. Clock_converge01 Rule Exception - 2

- The *Clock_converge01* rule does not report violations for the same source reconvergence on the input pin of a MUX. This is because, only

one input pin is active at a time. This scenario is shown in the following figure:

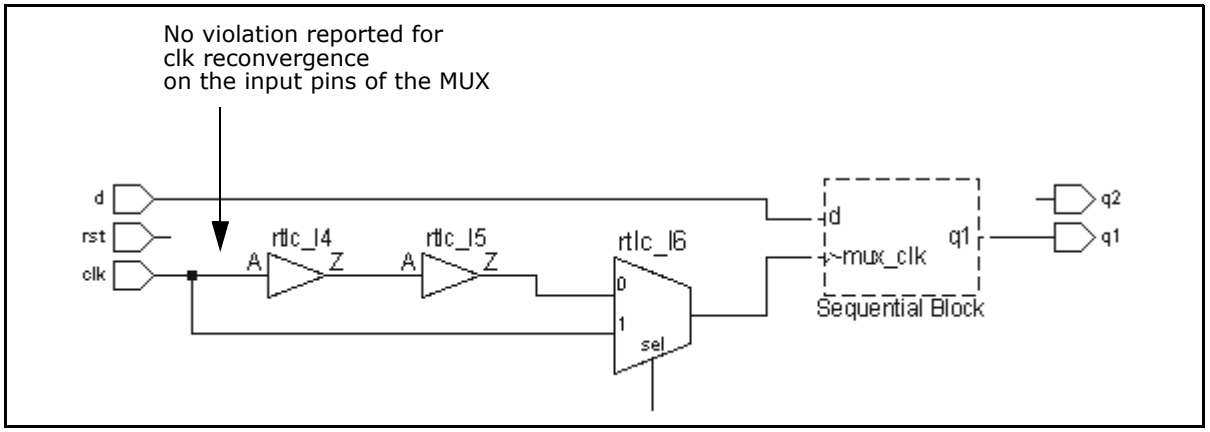


FIGURE 269. Example of the Clock_converge01 Rule Exception

Parameter(s)

- *cdc_express*: Default values is `no`. Set this parameter to `peakmem` to reduce peak memory. Other possible value is `yes`.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when multiple fan-outs of the clock *<clk-name>* converge at the net/instance *<obj-name>*:

[WARNING] Clock '<clk-name>' with multiple fanout converges on '<obj-name>'

Where, *<obj-name>* refers to *<inst-out-net-name>* in case of RTL designs, and *<inst-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs.

Potential Issues

This violation appears if your design contains clocks with multiple fan-outs converging deep in the design hierarchy.

Consequences of Not Fixing

If you do not fix this violation, the clock signal can have different delays along different paths before it re-converges. This may result in glitches or an incorrect clock waveform.

How to Debug and Fix

To fix this violation, avoid situations reported by this rule. Else, check the timing properly along different paths to ensure a correct clock waveform.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:

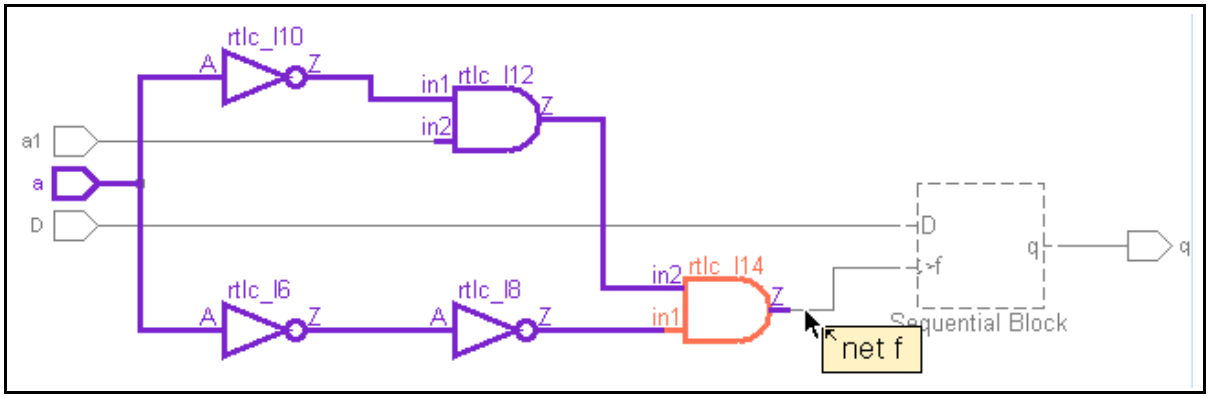


FIGURE 270. Clocks for which Multiple Fan-outs Converge

In the above schematic, the `a` clock with multiple fan-outs is converging on the `f` net.

Therefore, the *Clock_converge01* rule reports a violation.

Schematic Details

The *Clock_converge01* rule highlights the following in a different color in the schematic:

- Clock signal and its multiple fan-outs that are converging
- Converging instance

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

No report and related file

Clock_hier01

Reports clock-gating wrapper modules in clock-path

When to Use

Use this rule to check clock-gating wrapper modules in the clock-path hierarchy.

Description

The *Clock_hier01* rule reports all the modules that have a valid clock-gating structure. Presence of additional logic in this module will make it an invalid wrapper and get reported by the *Clock_hier03* rule.

This rule reports violation only for the propagated clock-path. Only one instantiation of each module is reported.

Clock Gating Structures Supported By the Clock_hier01 Rule

The following clock-gating structures are supported.

- **CGLP structure** (positive edge triggered clock-gating), as shown in the following figure:

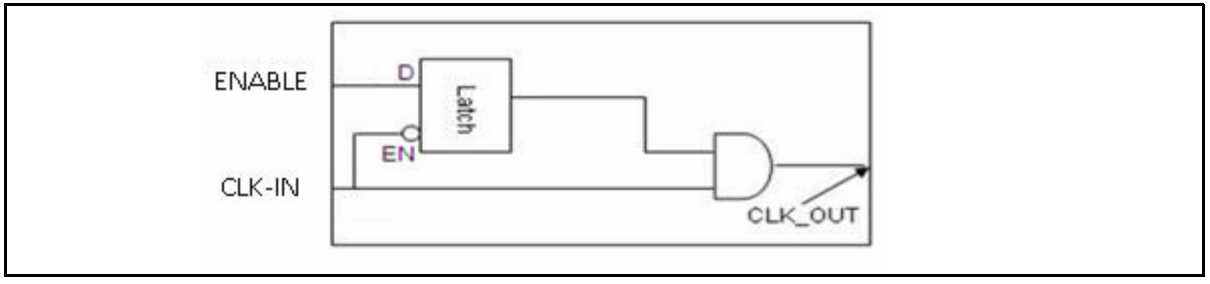


FIGURE 271. CGLP Structure

- **CGLN structure** (negative edge triggered clock-gating), as shown in the following figure:

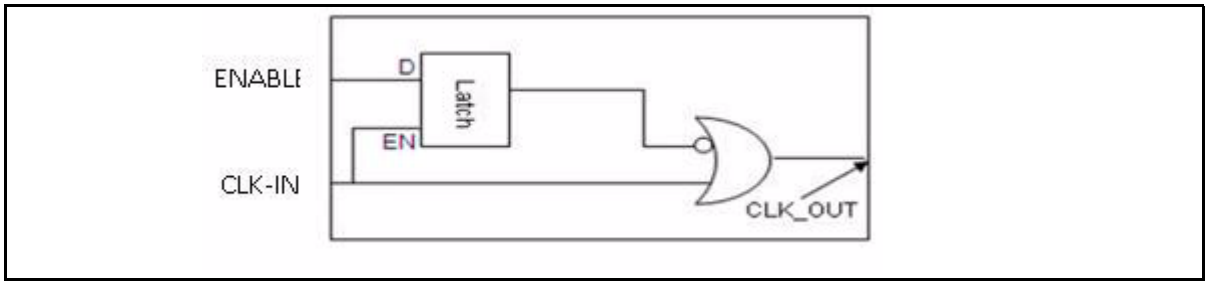


FIGURE 272. CGLN Structure

Input/output of the latch with an optional OR gate for scan-enable is supported. In addition to these auto-detected clock-gating structures, modules specified using the *clock_gate_cell* parameter get reported by this rule.

Rule Exceptions

This rule ignores instantiation of `clock_gating` structures from user-specified libraries.

Parameter(s)

- *enable_clock_gate_sync*: Default value is `yes`. Set this parameter to `no` to disable the *Clock-Gating Cell Synchronization Scheme.*
- *clock_gate_cell*: Default value is `NULL`. Set the value of this parameter to a comma or space-separated list of clock-gating cell names for the *Clock-Gating Cell Synchronization Scheme*.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.

- *clock_path_wrapper_modules* (Optional): Use this constraint to exclude modules from the checks performed by the *Clock_hier01*, *Clock_hier02*, and *Clock_hier03* rules.

Messages and Suggested Fix

The following message appears when the clock gating module *<mod-name>* is detected at the net *<net-name>*:

[INFO] Clock gating module '<mod-name>'(instance:'<hierarchy>') detected at net '<net-name>'

Potential Issues

This is an informational message. There are no potential issues.

Consequences of Not Fixing

None

How to Debug and Fix

View the **Incremental Schematic** of the violation message. The schematic highlights the clock-path to the clock-gating instance.

You can replace the clock-gating structure identified in a different hierarchical module by an appropriate library cell.

Example Code and/or Schematic

Consider the following example in which the clock-gating wrapper module is instantiated in the clock-path of `clk2`:

```
module gating_cell(en,clk,out);
….
..
always@(clk)
  begin
    if(!clk)
      temp<= en;
  end
assign gated_clk = temp & clk;
endmodule
```

For the above example, the Clock_hier01 rule reports the following

violation.

```
[INFO] Clock gating module 'gating_cell'(instance:'CG1')
detected at net 'test.t_clk2'
```

The following figure shows the schematic of this violation in which the clock-path to the clock-gating instance is highlighted:

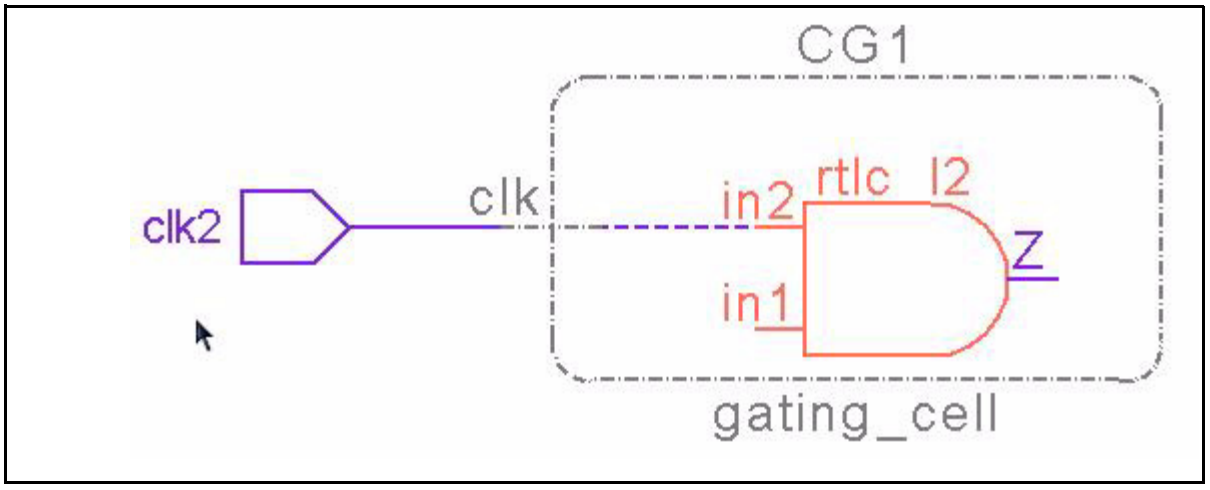


FIGURE 273. Schematic of the Clock_hier01 Rule Violation

Default Severity Label

Info

Rule Group

VERIFY

Reports and Related Files

No report and related file

Clock_hier02

Reports combinational wrapper modules in clock-path

When to Use

Use this rule to check combinational wrapper modules in the clock-path hierarchy.

Description

The *Clock_hier02* rule reports all combinational gates, in the clock-path, which are present inside a valid wrapper module. It reports only for the propagated clock-path. One instance of each wrapper module is reported.

The *Clock_hier02* rule does not consider an inverter with an output that is an internally generated net and is directly driving a clock pin because this inverter is created by SpyGlass during inversion of the negedge construct present.

```
wire clk2 = !clk & en;
```

Though these are two combinational elements, inverter and gate, this rule considers them together and, therefore, should be present inside one module.

Rule Exceptions

This rule ignores the instantiation of combinational logic from user-specified libraries.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *clock_path_wrapper_modules* (Optional): Use this constraint to exclude modules from the checks performed by the *Clock_hier01*, *Clock_hier02*, and *Clock_hier03* rules.

Messages and Suggested Fix

The following message appears when the clock gating module *<mod-name>* is detected at the net *<net-name>*:

```
[INFO] Clock gating module '<mod-name>'(instance:'<hierarchy>')
detected at net '<net-name>'
```

Potential Issues

This is an informational message. There are no potential issues.

Consequences of Not Fixing

Combinational logic is replaced by cells from libraries by synthesis tools. Therefore, if each combinational gate is wrapped in a different module, it is easier for synthesis tools to replace them with an equivalent library cell.

How to Debug and Fix

View the **Incremental Schematic** of the violation message. The schematic highlights the clock-path to the wrapper instance.

You can replace the valid wrapper-module with an equivalent cell from the library.

Example Code and/or Schematic

Example 1 - Understanding Wrapper Modules

A valid wrapper module is the parent module of the clock-path structure when it contains one clock-path structure. In the following figure:

- `Blk2` is not a valid wrapper module because it contains multiple combinational blocks.
- `Blk3` and `Blk4` are not considered as wrapper modules because of the multiple gates/blocks.
- `Blk6` and `Blk5` are considered as valid wrapper module and are reported by *Clock_hier02*.

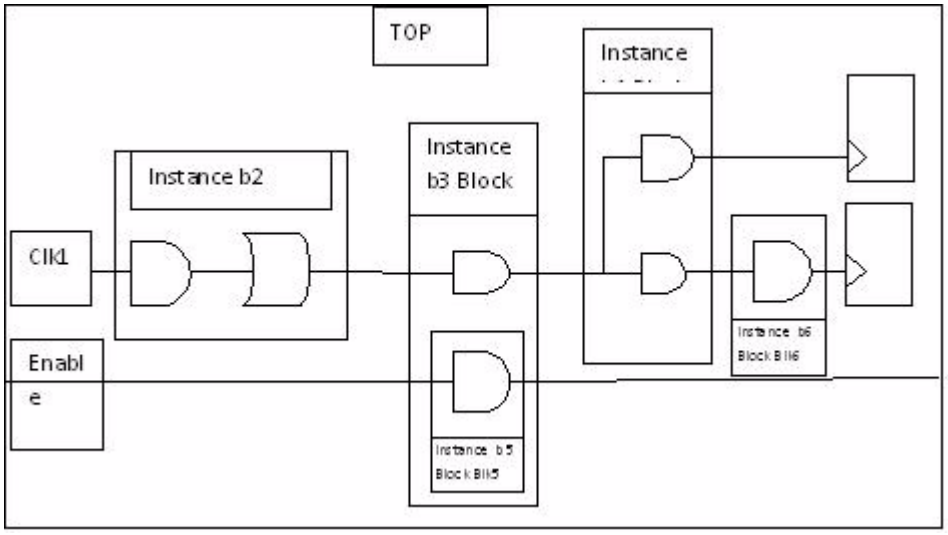


FIGURE 274. Wrapper Modules

Example 2 -Wrapper Modules with Generated Net

In this example, the `gated_cell` module is defined as:

```
module gated_cell(en1,en2,clk,out);
.
.
assign out = clk & en1 & en2;
endmodule
```

This gated cell is considered as a valid wrapper module and the following violation message is reported:

```
[INFO] Combinational logic wrapper module
'gated_cell'(instance:'CG3') detected at net 'test.g_clk2'
```

In addition, the following schematic is generated.

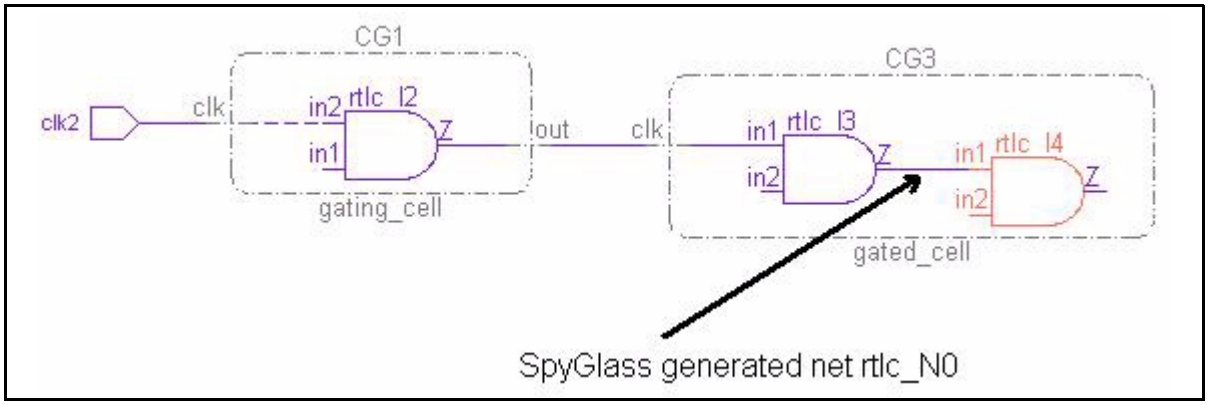


FIGURE 275. Wrapper Modules with Generated Net

Default Severity Label

Info

Rule Group

VERIFY

Reports and Related Files

None

Clock_hier03

Reports combinational gates that do have valid wrapper modules in the clock-path

When to Use

Use this rule to check the clock-path hierarchy.

Description

The *Clock_hier03* rule reports all combinational gates located in the clock-path, which are not present inside a valid wrapper module. It reports only for the propagated clock-path.

To understand wrapper modules, refer to *Example 1 - Understanding Wrapper Modules*.

Rule Exceptions

This rule ignores instantiation of combinational logic from user-specified libraries.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals.
- *clock_path_wrapper_modules* (Optional): Use this constraint to exclude modules from the checks performed by the *Clock_hier01*, *Clock_hier02*, and *Clock_hier03* rules.

Messages and Suggested Fix

The following message appears when a combinational gate, which is not present inside a valid wrapper module, is located in the clock-path at net *<net-name>*:

`[WARNING] Wrapper module not detected for clock path logic at net '<net-name>'`

Potential Issues

A combinational logic on the clock-path is not inside a valid wrapper module.

Consequences of Not Fixing

Typically, combinational logic is replaced by cells from libraries by synthesis tools. If the combinational logic is not wrapped inside a valid wrapper module, it is difficult for synthesis tools to replace them with an equivalent library cell.

How to Debug and Fix

View the **Incremental Schematic** of the violation message. The schematic highlights the clock-path to the violating logic.

To resolve this violation, either the combinational logic should be an instantiation of a library cell or an instantiation of a valid wrapper module.

Example Code and/or Schematic

In this example, the highlighted AND gate is not in a wrapper module. It is directly instantiated in the top module.

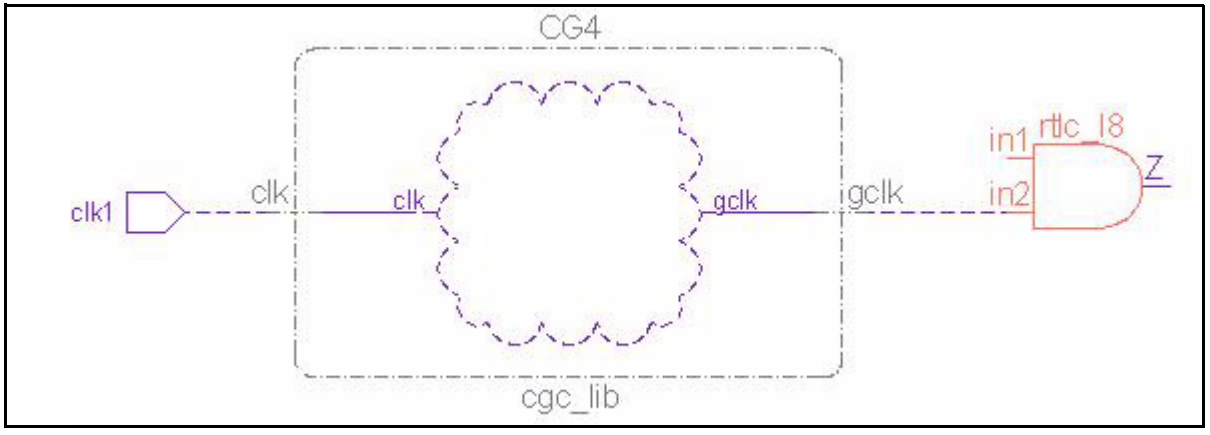


FIGURE 276. Schematic of the Clock_hier03 Rule Violation

Therefore, the *Clock_hier03* rule reports the following violation:

```
[WARNING] wrapper module not detected for clock path logic at
net 'test.gclk3'
```

To resolve this violation, either the AND gate should be an instantiation of a library cell or an instantiation of a valid wrapper module.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

None

Ac_xclock01

Reports non-deterministic clock-edges in the presence of clock-gates

When To Use

Use this rule to detect non-deterministic clock-edges occurring in the presence of clock-gates.

Prerequisites

- Use the *Advanced_CDC* and *adv_checker* license features.
- Specify ICG cells through .lib files.

Description

The *Ac_xclock01* rule reports the cases in which X values can be present on a clock path resulting in non-deterministic shifting during the scan operation.

This rule detects the logic value on the clock pin of positive and negative edge flip-flops when the clock driving the clock-gating cells is 0 and 1 and determines whether a non-deterministic edge can occur.

For details, see *Example 1*, *Example 2* and *Example 3*.

Assumptions About the Operation of the Circuit

The following is assumed about the operation of the circuit when this rule is run:

- Scan enable of the clock-gating cells is active high.
- Initially, SE=1, CLK=0.

Parameter(s)

- *show_all_xclock_flops*: Default value is `no`. Set this parameter to `yes` to generate a spreadsheet containing violation on a per net basis.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *fa_multicore*: Default value is `no`. Set this parameter to `yes` to invoke the multi core engine of SpyGlass for solving complex assertions.
- *fa_meta*: Default value is `no`. Set this parameter to `yes` to enable formal modeling of metastability.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock*: (Optional): Use this constraint to specify clock signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following violation message each for a positive edge and negative edge:

[AcXC1_1] [WARNING] Net '<name>' (Source clock: <source-clock>) incorrectly drives <positive | negative> edge of clock pin of (<element-type>) '<instance-output-name>' (Total Count: <count>)

The arguments of the above message are explained below:

Argument	Description
<net-name>	Hierarchical name of a net (in the clock path), which cannot drive a particular sequential element
<source-clock>	Name of the source clock driving the clock pin of a flip-clop
<element-type>	Can be `flop`, `latch`, or `library cell`

Argument	Description
<instance-output-name>	Name of the flip-flop
<count>	Total number of rule-violating sequential elements

Potential Issues

This violation may appear in the following cases:

- A negative ICG cell drives a positive edge flip-flop or a positive ICG cell drives a negative edge flip-flop.
- An inverter is present before an ICG cell reversing the polarity of the ICG cell.

Consequences of Not Fixing

If you do not fix this violation, the design circuit may produce incorrect output.

How to Debug and Fix

To fix this violation, perform any of the following actions:

- Ensure that a negative edge ICG cell triggers a negative edge flip-flop and a positive edge ICG cell triggers a positive edge flip-flop.
- Check the usage of an inverter before an ICG cell.

  Remove the inverter before the ICG cell or insert it at an appropriate location in the design.

Example Code and/or Schematic

Example 1

Consider the circuit shown in the following figure:

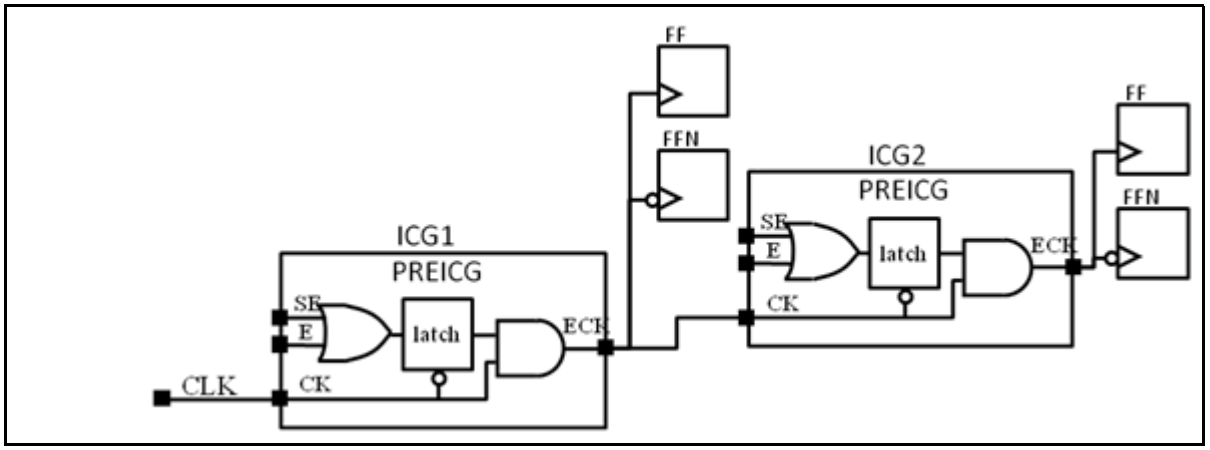


FIGURE 277. Scenario for No Ac_xclock01 Rule Violation

The above circuit contains two PREICG cells (`ICG1` and `ICG2`) in series. The following table shows the simulation values when `CLK` =1 and `CLK`=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	0	0
1	1	1	1

In this example, as the values of `ICG1.ECK` and `ICG2.ECK` are deterministic, both can drive flip-flops triggered on either the positive or the negative edge. Therefore, the *Ac_xclock01* rule does not report any violation in this case.

Example 2

Consider the circuit shown in the following figure:

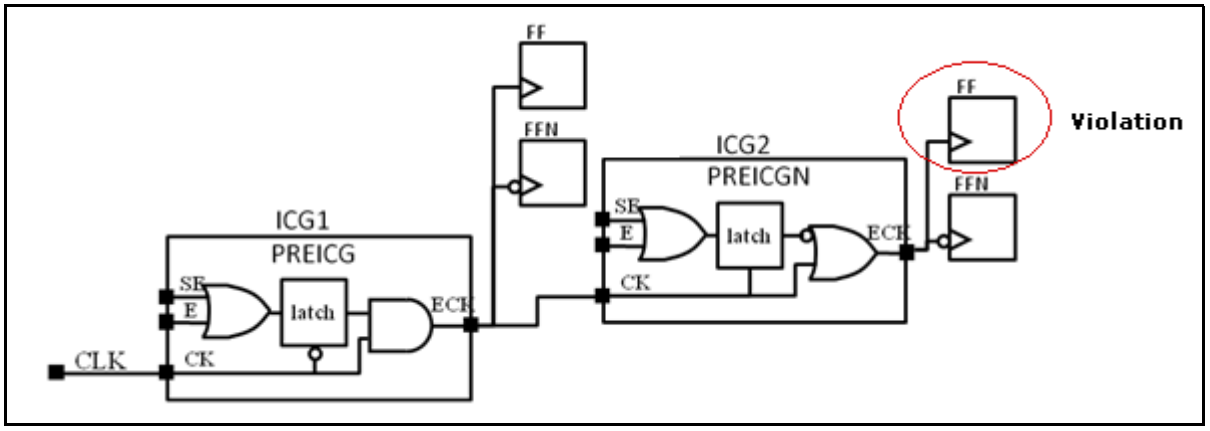


FIGURE 278. Scenario for Ac_xclock01 Rule Violation

In the above circuit, the PREICG cell drives the PREICGN cell.

This rule reports a violation in this case because of the following reasons:

- When CLK=0 and SE=1, ICG1.ECK=0, ICG1 latch is initialized to 1, and ICG2.ECK=X as the initial state of ICG2 latch is X.
- When CLK=1 and SE=1, ICG1.ECK and ICG2.ECK is 1. Therefore, ICG1.ECK can drive both positive and negative edge flip-flops. However, ICG2.ECK can drive only negative edge flip-flops.

The following table shows the simulation values when `CLK` =1 and `CLK`=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	0	X
1	1	1	1

Example 3

Consider the circuit shown in the following figure:

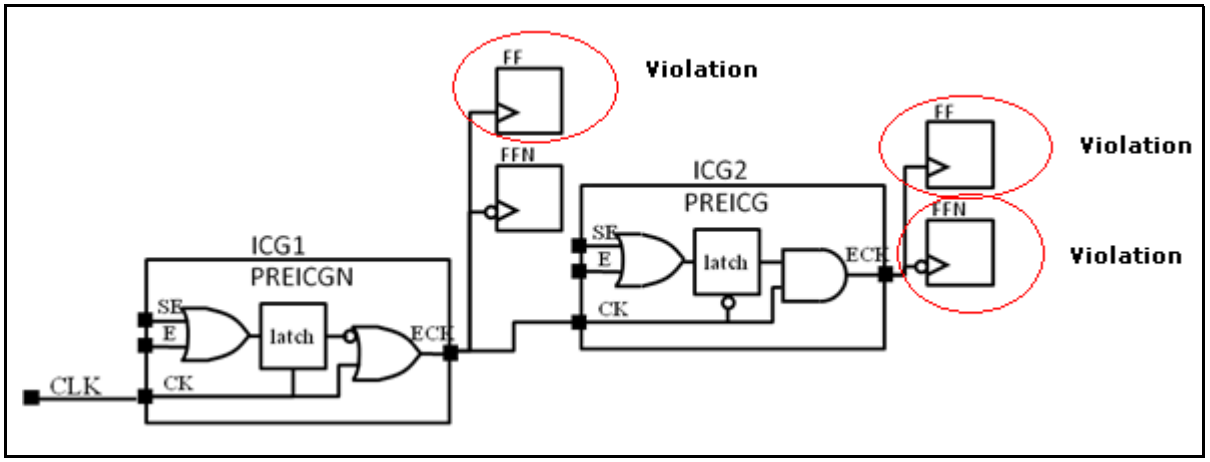


FIGURE 279. Scenario for Ac_xclock01 Rule Violation

For the above example, this rule reports violation because `ICG2.ECK` cannot drive any positive edge or negative edge flip-flop. The following table shows the simulation values when `CLK` =1 and `CLK`=0:

SE	CLK	ICG1.ECK	ICG2.ECK
1	0	X	X
1	1	1	X

Example 4

Consider the following spreadsheet, which is displayed when you click on a violation of this rule:

A ID	B Type	C Instance
6	Flop	testme.U1.f9.q

FIGURE 280. Spreadsheet of the Ac_xclock01 Rule

The above spreadsheet lists sequential elements that are triggered

incorrectly. To view the schematic of the violation, click *1* in the *ID* column and click .

The following figure shows the schematic for this case:

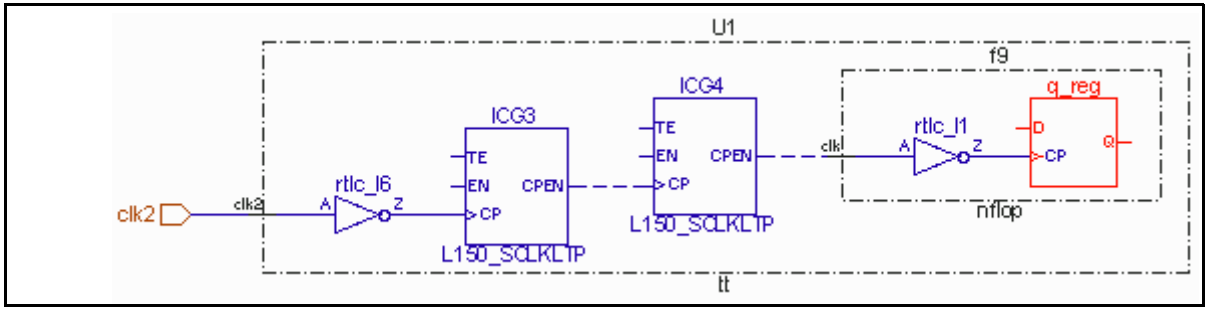


FIGURE 281. Schematic of the Ac_xclock01 Rule Violation

The above violation appears for the negative edge flip-flop, `U1.f9.q`, as the clock pin of this flip-flop is incorrectly driven by the `clk2` clock.

To fix this violation, you can perform any of the following actions:

- Check if the NOT gate before the `ICG3` cell is required in the circuit. If not, remove this NOT gate.
- Replace the `ICG3` cell with a negative edge ICG cell.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

This rule generates a spreadsheet that lists sequential elements that are driven incorrectly. For details, see *Example 4*.

Ac_converge01

Reports signals which are subjected to glitches in clock path

When to Use

Use this rule to catch glitches in a clock tree due to divergence and convergence of enable signals.

Prerequisites

Following are the prerequisites for running this rule:

- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using automatically-inferred resets by setting the *use_inferred_clocks* parameter to `yes`
  - By using both the above methods

Description

The details of the *Ac_converge01* rule reports enable signals that diverge and then reconverge before being used as a clock in the clock path. For example, this rule reports a violation for the 'd' signal in the following scenario:

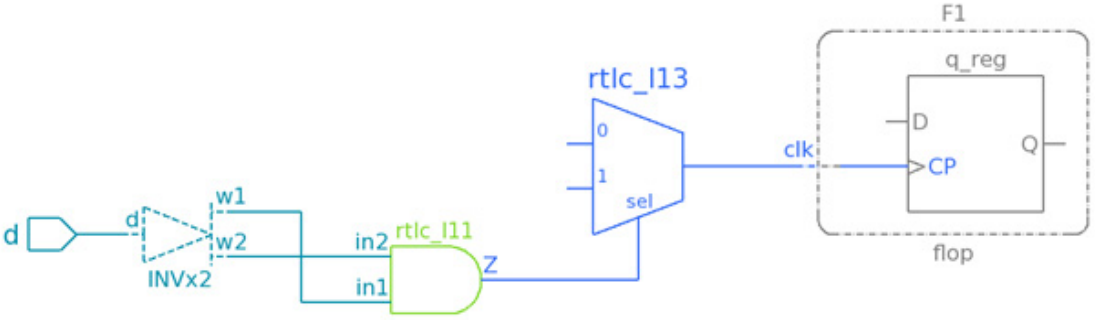


FIGURE 282. Clock Divergence and Reconvergence

The *Ac_converge01* rule only checks for combinational convergences and does not report a violation if an enable signal converges after sequential elements. For example, consider the following scenario:

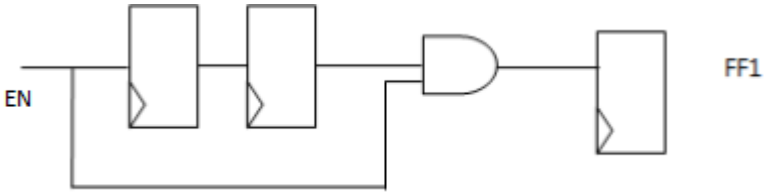


FIGURE 283. Case of Valid Synchronization

In case of multiple paths, this rule reports convergence on all the points.

For example, consider the scenario shown in the following figure:

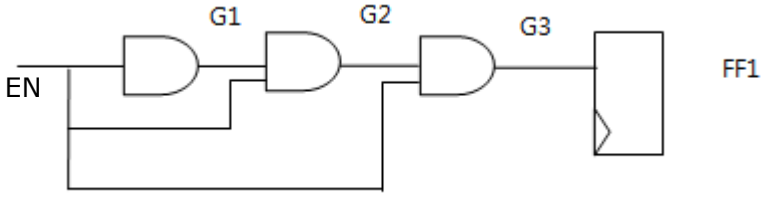


FIGURE 284. Presence of Multiple Paths

For the above scenario, the *Ar_converge01* rule reports on both G2 and G3.

The Ac_converge01 rule does not report violations for the same enable reconvergence on the input pin of a MUX. This is because, only one input pin is active at a time. For example, consider the following figure:

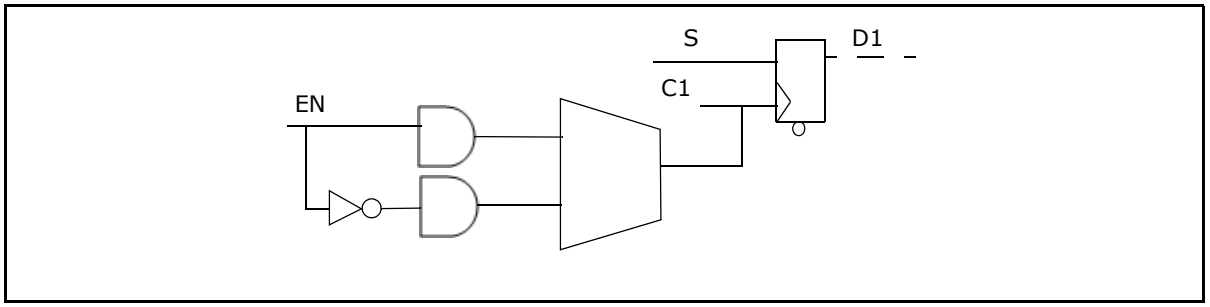


FIGURE 285. Example of the Ac_converge01 Rule Exception

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *clock*: (Optional): Use this constraint to specify clock signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for the enable signal that diverges and then reconverges back before being used as a clock:

[WARNING] Enable Signal '<sig-name>' converges at <type> <inst_name>

The arguments of the above message are explained below:

Argument	Description
<sig-name>	Name of the enable signal
<type>	Can be `combinational-gate` or `MUX`
<inst-name>	Name of the converging gate output net. Note the following points: • If the *report_inst_for_netlist* parameter is set to yes, this argument takes the value of the instance pin. • If the output net of the converging gate is an internal net, this argument takes the value of the closest net.

Potential Issues

This violation appears if your design contains enable signals that are diverging and then converging before being used in clock path.

Consequences of Not Fixing

If you do not fix this violation, glitches may get introduced in the design.

How to Debug and Fix

To debug and fix the violation of this rule, perform the following steps:

1. Check the violation of this rule from moresimple.rpt
2. Check for the corresponding violation schematic to view the divergence and convergence of enable signals in clock path.
3. If the converging instance is a MUX, provide the *set_case_analysis* constraint on the select pin of the MUX so that one of the input is selected.
4. Else, fix the RTL so that there are no multiple paths from the same enable to the destination to avoid glitches.

Example Code and/or Schematic

Consider the following design:

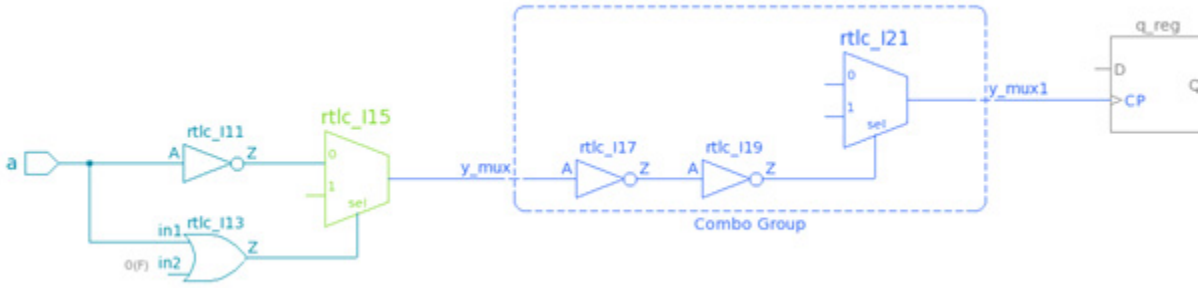


FIGURE 286. Ar_converge01 Spreadsheet

For the above design, the *Ac_converge01* rule reports the following violation:

```
Enable Signal 'test.a' converges at MUX 'test.y_mux'
```

Schematic Details

The *Ac_converge01* rule highlights the following details in the schematic:

- Two paths showing divergence and reconvergence of the reported enable signal
- Path from the convergence point till the clock pin of a sequential

Default Severity Label

Warning

Rule Group

VERIFY

Reset Checking Rules

The SpyGlass CDC solution has the following rules for checking the reset conditions:

Rule	Reports
Ar_converge01	Asynchronous reset signals having multiple converging fan-outs
Ar_converge02	Reset signals that diverge into multiple paths and reconverge on the data/enable and reset/set pin of the same flip-flop
Reset_check01	(Verilog Only) Reset signals that are not being used in the same mode as their respective pragma mode
Reset_check02	Latches, tristate signals, or XOR/XNOR gates found in a reset tree
Reset_check03	Reset signals that are being used at both levels to set or reset flip-flops synchronously
Reset_check04	Reset signals that are being used both as an asynchronous reset and synchronous reset
Reset_check05	Synchronous resets
Reset_check06	High fan-out reset signals not driven by specified placeholder cells
Reset_check07	Asynchronous set/reset pins that are driven by combinational logic
Reset_check09	Reports XOR/XNOR/AND/NAND gates found in a reset tree
Reset_check10	Asynchronous resets used as non-reset signals
Reset_check11	Asynchronous resets used as both active-high and active-low
Reset_check12	Flip-flops that do not get active reset during power on reset
Reset_overlap01	Resets that reach another reset domain.

Ar_converge01

Reports asynchronous reset signals that have multiple converging fan-outs

When to Use

Use this rule to catch glitches on an asynchronous reset tree due to divergence and convergence of reset signals.

Prerequisites

Following are the prerequisites for running this rule:

- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using automatically-inferred resets by setting the *use_inferred_resets* parameter to yes
  - By using both the above methods

Description

The details of the *Ar_converge01* rule is covered in the following topics:

- *Reason for the Ar_converge01 Rule Violation*
- *Unate/Binate Analysis*
- *Features of the Ar_converge01 Rule*
- *Rule Exceptions*
- *Ar_converge01.csv*

Reason for the Ar_converge01 Rule Violation

The *Ar_converge01* rule reports asynchronous reset signals that diverge and then reconverge before being used.

For example, this rule reports a violation for the `rst` reset in the following scenario:

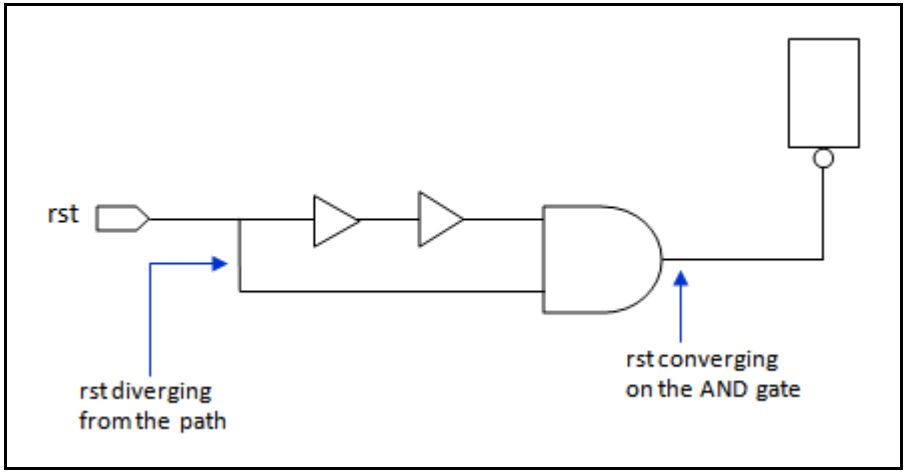


FIGURE 287. Reset Divergence and Reconvergence

Unate/Binate Analysis

You can configure the *Ar_converge01* rule to report violations related with same source reconvergence when the reconverging paths have different polarities or at least one path has an unknown polarity.

For details, see *no_unate_reconv*.

Features of the Ar_converge01 Rule

Following are features of this rule:

- It reports a violation for resets that traverse through multiple paths before reaching functional flip-flops. This scenario is shown in the following figure:

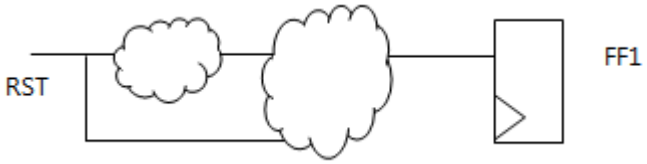


FIGURE 288. Reset Traversing Multiple Paths

- This rules only checks for combinational convergences and does not report a violation if a reset converges after sequential elements.

  For example, consider the following scenario:

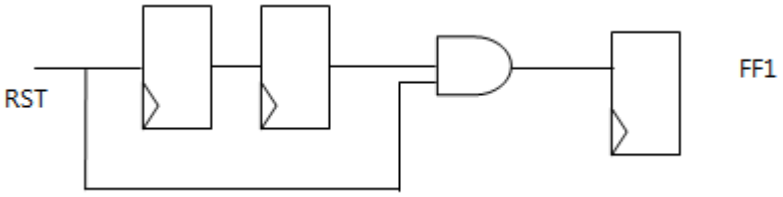


FIGURE 289. Case of Valid Synchronization

  For the above scenario, the *Ar_converge01* rule does not report a violation as this is the case of valid synchronization.

- This rule reports a violation for derived resets that diverge and then converge, as shown in the following figure:

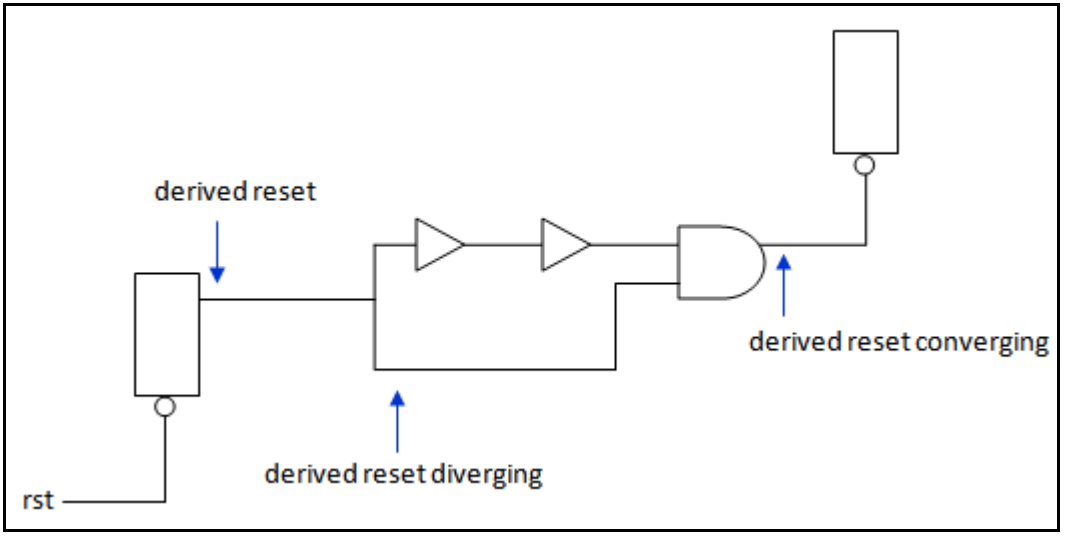


FIGURE 290. Divergence and Convergence of Derived Resets

- In case of multiple paths, this rule reports convergence on the last point with two paths (reaching to the last gate).

  For example, consider the scenario shown in the following figure:

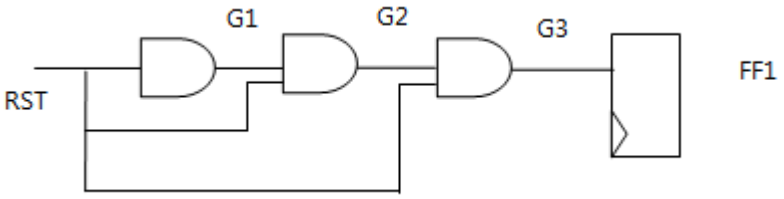


FIGURE 291. Presence of Multiple Paths

  For the above scenario, the *Ar_converge01* rule reports convergence on `G3`.

- If a converging gate output is an internal net, this rule reports the closest RTL net.

Rule Exceptions

The *Ar_converge01* rule has the following exceptions:

- The *Ar_converge01* rule does not report a violation if the path is sanitized and is equivalent to a buffer or an inverter.

  For example, consider the scenario shown in the following figure:

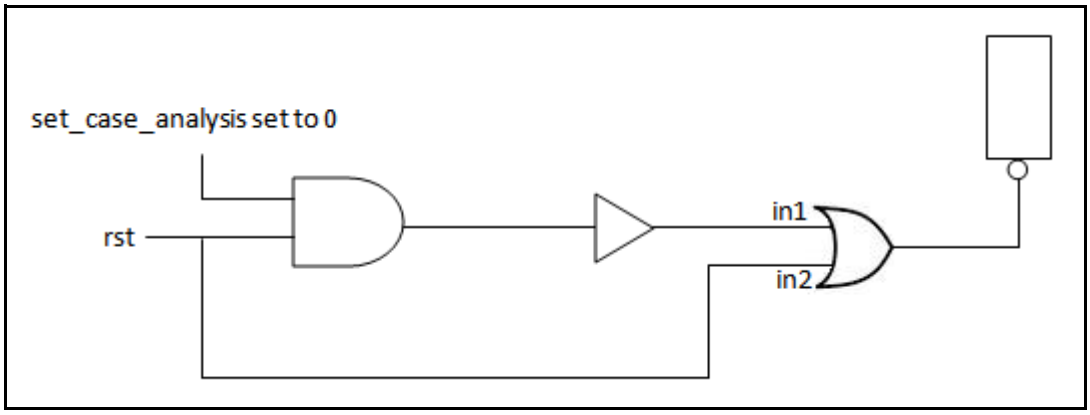


FIGURE 292. Path is Sanitized and is Equivalent to a Buffer/Inverter

  In the above scenario, when the *set_case_analysis* constraint is set to 0, the `rst` reset does not propagate to the `in1` input pin of the OR gate. This way, the OR gate acts as a buffer in this case. Therefore, the *Ar_converge01* rule does not report a violation.

- The *Ar_converge01* rule does not report violations for the same source reconvergence on the input pin of a MUX. This is because, only one input pin is active at a time. This scenario is shown in the following figure:

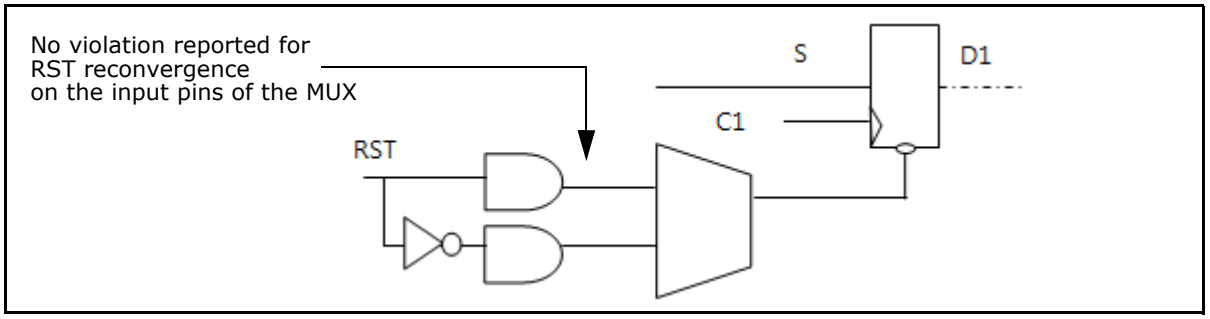


FIGURE 293. Example of the Ar_converge01 Rule Exception

- The *Ar_converge01* rule does not report convergence on black boxes.

Parameter(s)

use_inferred_resets: Default value is `no`. Set this parameter to `yes` to use the auto-generated reset information.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static*: Use this constraint to specify signals that have a value which is predominantly static.

Messages and Suggested Fix

The following message appears for the asynchronous reset signal that diverge and then reconverges back before being used:

```
[WARNING] Reset signal '<sig-name>' converges at <type>
```

<inst-name>

The arguments of the above message are explained below:

Argument	Description
<sig-name>	Name of the reset source
<type>	Can be `combinational-gate` or `MUX`
<inst-name>	Name of the converging gate output net. Note the following points: • If the *report_inst_for_netlist* parameter is set to yes, this argument takes the value of the instance pin. • If the output net of the converging gate is an internal net, this argument takes the value of the closest net.

Potential Issues

This violation appears if your design contains reset signals that are diverging and then converging before being used.

Consequences of Not Fixing

If you do not fix this violation, glitches may get introduced in the design.

How to Debug and Fix

To debug and fix the violation of this rule, perform the following steps:

1. Open the spreadsheet of this rule.

   To open the spreadsheet, right-click on the rule name in the *Results* pane and select the *Spreadsheet Viewer* option from the shortcut menu.
2. Check the list of reset signals in the spreadsheet.
3. Click on the violation in the spreadsheet and open the schematic to view the divergence and convergence of reset signals.
4. If the converging instance is a MUX, provide the *set_case_analysis* constraint on the select pin of the MUX so that one of the input is selected.

   Else, fix the RTL so that there are no multiple paths from the same source to the destination to avoid glitches.

Example Code and/or Schematic

Consider the following *Ar_converge01.csv* spreadsheet showing the details of a violation of this rule:

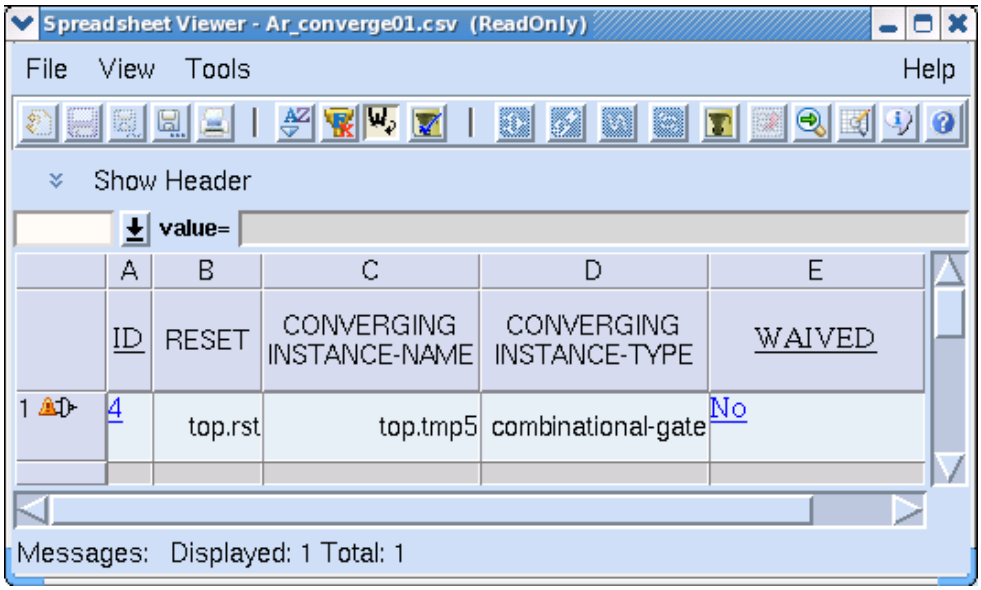


FIGURE 294. Ar_converge01 Spreadsheet

Information in the above spreadsheet indicates that the `top.rst` reset is the rule-violating reset that converges on the `top.tmp5` instance that is a combinational gate.

The schematic of this violation is shown below:

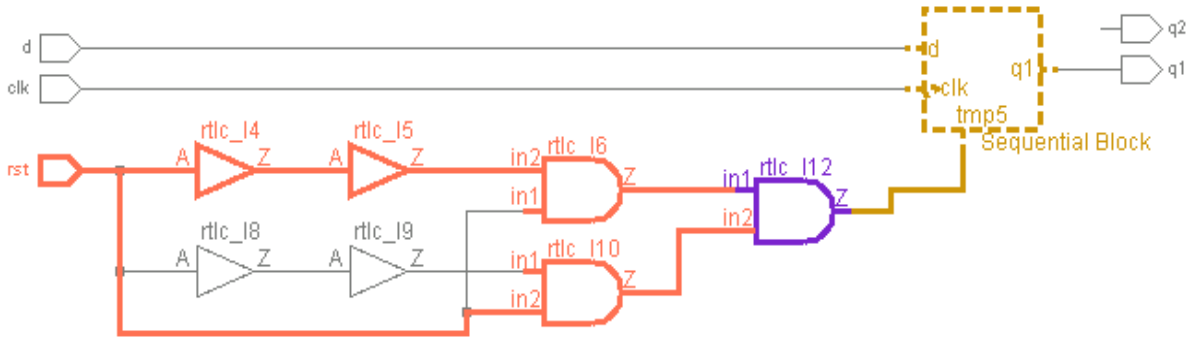


FIGURE 295. Schematic of the Ar_converge01 Rule Violation

The above schematic shows the `rst` reset is diverging and then converging on the AND gate.

Schematic Details

This rule highlights the following details in the schematic:

- Two paths showing divergence and reconvergence of the reported reset signal
- Path from the converging point till the reported RTL net
- Path from the convergence point till any one functional flip-flop

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

Ar_converge01.csv

This spreadsheet shows the details of violations reported by the *Ar_convergence01* rule.

Following is the example of this spreadsheet:

A	B	C	D	E
ID	RESET	CONVERGING INSTANCE-NAME	CONVERGING INSTANCE-TYPE	WAIVED
4	top.rst	top.tmp5	combinational-gate	No

FIGURE 296. Ar_converge01 Spreadsheet

The details of columns in the above spreadsheet are described below:

Column	Description
ID	Specifies a unique ID for a violation.
RESET	Specifies the name of the reset signal that is causing the violation.
CONVERGING INSTANCE NAME	Specifies the instance name on which the reported reset signal converges.
CONVERGING INSTANCE TYPE	Specifies the type of the instance on which the reported reset converges.
WAIVED	Specifies if the reported violation is waived.

Ar_converge02

Reports a reset signal which converges on data and reset pin of same flop

When To Use

Use this rule to catch convergence of reset signals on the data and reset pin of a flip-flop.

Prerequisites

Use the `Advanced_CDC` license for running this rule.

Description

The *Ar_converge02* rule reports a reset signal that diverges into multiple paths and reconverges on the data/enable and reset/set pin of the same flip-flop.

This rule automatically infers resets in a design. For example, consider the following figure:

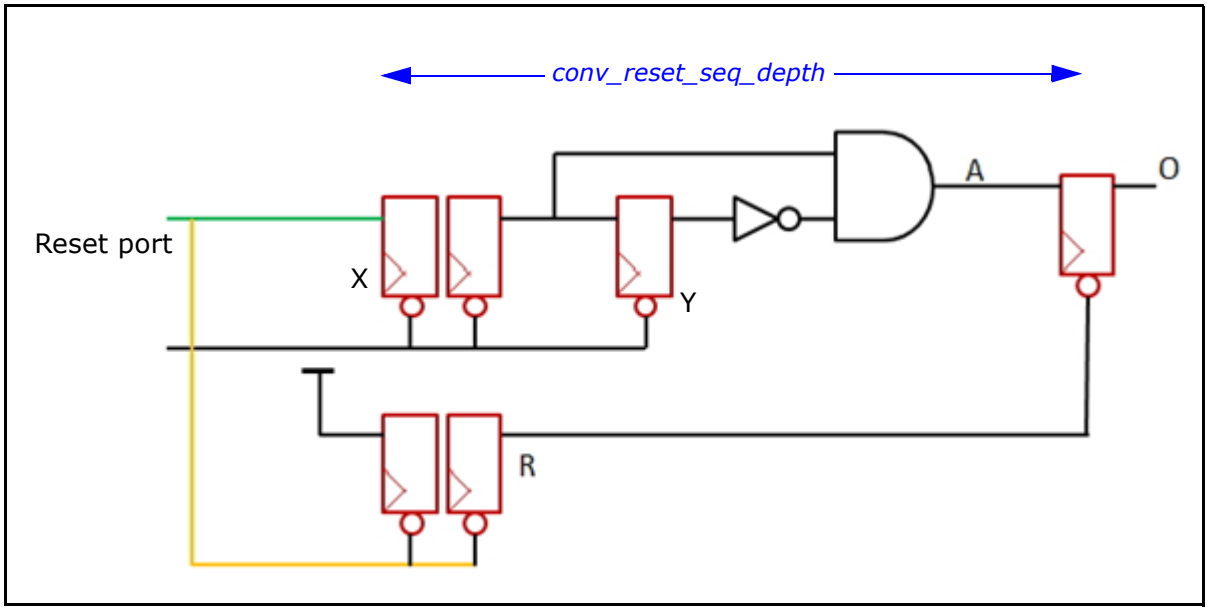


FIGURE 297. The Ar_converge02 violation

In the above example, based on the delay values of the yellow and green lines, a clock-cycle delay may occur on the `R` signal compared to the `A` signal, leading to functionality issue at the `O` flip-flop.

The following figure shows the expected and incorrect results in the above case:

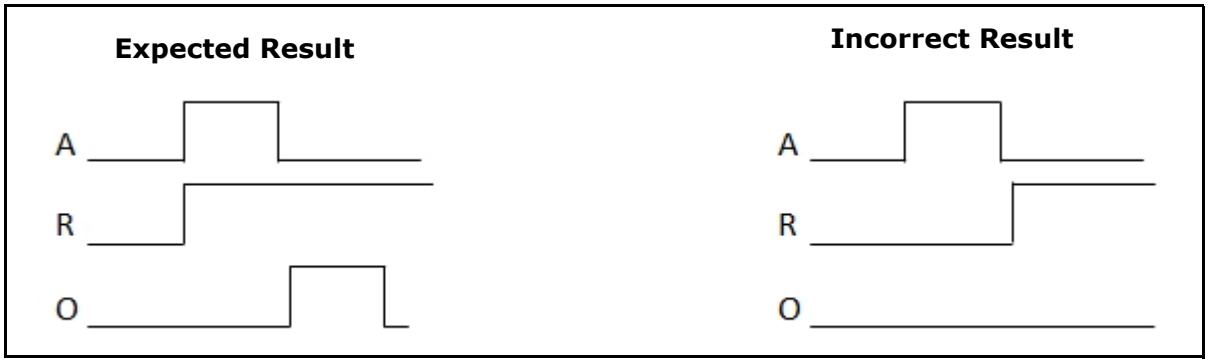


FIGURE 298.

Parameter(s)

- *conv_reset_seq_depth*: Default value is `2`. Set this parameter to a positive integer value to set the number of sequential elements beyond which a reset can propagate across a data terminal.
- *conv_reset_single_data_bit*: Default value is `no`. Set this parameter to `yes` to enable sequential traversal from only one bit of a data bus.
- *report_all_flops*: Default value is `no`. Set this parameter to `yes` to report all flip-flops whose resets are generated in asynchronous domains.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *ignore_bus_resets*: Default value is `yes`. Set this parameter to `no` to generate reset vector nets, which are not struct nets, in the *autoresets.sgdc* and the `generated_resets.sgdc` file.

Constraint(s)

- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when a reset diverges into multiple paths and reconverges on the set/reset and enable/data pin of the same flip-flop:

```
[WARNING] Reset signal '<sig-name>' converge on <set | reset>
and <enable | data> pins of flop <flip-flop-name>
```

Potential Issues

This violation appears if your design contains an asynchronous reset signal that reaches to the data input and reset pin of the same flip-flop.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

For example, in *Figure 297*, based on the delay values of the yellow and green lines, a clock cycle delay may occur on the `R` signal compared to the `A` signal leading to functionality issue at the `O` flip-flop.

How to Debug and Fix

To fix this violation, avoid using asynchronous resets as non-reset signals.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

```
//test.v

module top(input d,in1,in2,in3,clk,rst,rst11,output q1,q2, q3);
wire en;
wire t;
FFR a_use0(rst, clk, rst, t);
FFR a_use1(t, clk, rst & en, t1);
FFR a_use2(d, clk ,rst, t2);
FFR a_use11(d, clk, rst1, t11);
FFR a_use12(t11, clk, t11, t12);
reg q2;
wire rst12;
wire set = rst12 & en;
wire en_rst12 = rst12 & en;
always@(posedge clk or posedge rst12 or posedge set)
    if(rst12)
        q2 <= 1'b0;
    else if(set)
        q2 <= 1'b1;
    else if(en_rst12)
        q2 <= d;

endmodule
module FFR(input d, clk, rst, output reg q);
always@(posedge clk or posedge rst)
    if(rst)
        q<=1'b0;
    else
        q<=d;
endmodule
```

```
//test.sgdc
current_design top
clock -name clk
clock -name in1
clock -name in2
```

For the above example, the *Ar_converge02* rule reports violations for the `rst`, `rst12`, and `t11` signals.

The following schematic shows the violation for the `rst` signal that converge on the reset and data pins of the `top.a_use0.q` flip-flop:

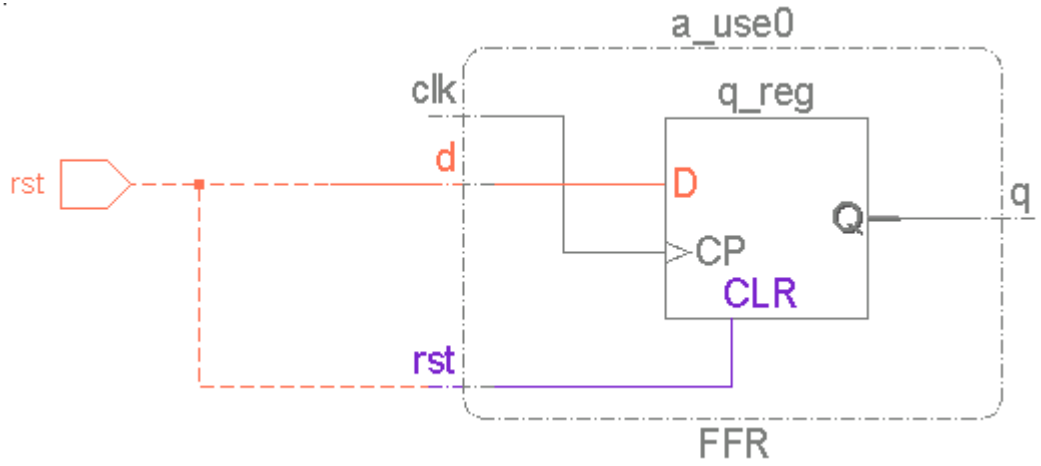


FIGURE 299. Schematic of the violation of the Ar_converge02 rule

Schematic Highlight

The *Ar_converge02* rule highlights the following in a different color:

- Reset path till the reset/set pin
- Reset path till the data/enable pin

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Reset_check01

Reports reset signals that are used in a different mode from their respective synthesis pragmas

When to Use

Use this rule to detect reset signals that are used in a different mode from their respective synthesis pragmas.

Description

The *Reset_check01* rule reports reset signals that are not used in the same mode as their respective synthesis pragma modes.

For example, this rule reports the reset signal that is used in:

- An asynchronous manner when the corresponding synthesis pragma, *sync_set_reset*, exists for that signal.
- A synchronous manner when the corresponding synthesis pragma, *async_set_reset*, exists for that signal.

Processing of Synthesis Pragmas

This rule checks the signals that are specified by using the *sync_set_reset* and *async_set_reset* pragma directives.

SpyGlass processes these synthesis pragmas based on the location where the pragma is specified, as described below:

- If the **pragma is specified inside a design unit description**, the scope of the pragma is from the point of specification up to the end of the design unit description including descriptions of master design units instantiation in the scope.
- If the **pragma is specified outside any design unit description**, the scope of the pragma is from the point of specification up to the end of the source file including design unit descriptions in the scope and descriptions of master design units instantiation in these design units.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

None

Messages and Suggested Fix

The following message appears at the location where a rule-violating reset signal *<sig-name>* is encountered:

[WARNING] Reset '<sig-name>' is not being used as specified through pragma

Potential Issues

This violation appears if your design contains the following types of reset signals:

- Synchronous reset signals that are used asynchronously
- Asynchronous reset signals are used synchronously

Consequences of Not Fixing

Using synchronous reset signals asynchronously and using asynchronous reset signals synchronously may result in failure of design functionality.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Click on the message in GUI.

   This step cross-references to the relevant RTL code.
2. Review the specified synthesis pragma and reset usage in this part of RTL.

Example Code and/or Schematic

This rule reports violations in the following example:

```
module test(data, clk, rst_sync, rst_async,
     out1, out2);
  input data, clk, rst_sync, rst_async;
  output out1, out2;

  reg out1, out2;

//synopsys sync_set_reset "rst_sync"
```

```
  always @(posedge clk or posedge rst_sync)
    if(rst_sync) out1 <= 0;
    else out1 <= data;

//synopsys async_set_reset "rst_async"
  always @(posedge clk)
    if(rst_async) out2 <= 0;
    else out2 <= data;
endmodule
```

In the above example, this rule reports violation because:

- The `rst_sync` signal has been specified in a *sync_set_reset* pragma but is being used in an asynchronous manner.
- The `rst_async` signal has been specified in an *async_set_reset* pragma but is being used in a synchronous manner.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and/or Related Files

No report and related file

Reset_check02

Reports latches, tristate signals, or XOR/XNOR gates in a reset tree

When to Use

Use this rule to detect latches, tristate signals, or XOR/XNOR gates in a reset tree.

Description

The *Reset_check02* rule reports latches, tristate signals, or XOR/XNOR gates in a reset tree.

This rule checks for such gates and reports them under informational message if:

- An enable pin of a latch or a tristate gate is tied to a constant logic value. In this case, the latch/tristate is permanently enabled or disabled.
- One of the inputs of a XOR/XNOR gate is tied to a constant logic value. In this case, the XOR/XNOR gate effectively works as buffer or inverter.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

report_inst_for_netlist: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.

Messages and Suggested Fix

Message 1

The following warning message appears at the location where a preset/clear signal of a flip-flop is being driven by latches, tristate signals, or XOR/XNOR gate:

```
[RstC2_1] [WARNING] Unexpected <gate-type> gate (at <name>) in
<sig-type> tree of flop (output <obj-type> <inst-name>)
```

Potential Issues

This violation appears if your design contains a reset tree that contains latches, tristate signals, or XOR/XNOR gates.

Consequences of Not Fixing

Presence of latches, tristate signals, or XOR/XNOR gate in a reset tree may block further propagation of a reset signal or may change the reset behavior.

As designs are very sensitive to resets, a reset tree should contain only permitted cells.

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

The following informational message appears at the location where a tristate/latch is permanently enabled/disabled or an input of XOR/XNOR is tied to active high/low:

`[RstC2_2] [INFO] Unexpected <gate-type> (<state>) gate (at <name>) in <sig-type> tree of flop (output <obj-type> <inst-name>)`

The arguments of the above message are explained below:

Argument	Description
<gate-type>	Can be `latch`, `tristate`, `XOR`, or `XNOR`
<name>	The name of the latch, tristate signal or the output of XOR/ XNOR gate
<sig-type>	Can be `preset` or `clear`
<obj-type>	`net` in case of RTL designs. `pin` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `net`

Argument	Description
<inst-name>	*<flop-output-net-name>* in case of RTL designs. *<flop-inst-name>.<pin-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs.
<state>	It can be `acting as buffer/inverter` in case of XOR/XNOR gates or `permanently enabled/disabled` in case of tristate gates or latches

Potential Issues

None

Consequences of Not Fixing

No fixing this violation may add to delays in a reset path.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Select an appropriate option from the *Edit->Show Case Analysis* menu option in the *Incremental Schematic* window.
3. Analyze the logic present in the reset-tree and provide the *set_case_analysis* constraint appropriately.

Example Code and/or Schematic

This section covers the following examples:

- *Example of a Latch Output Used as Clear*
- *Example of Tristate Signal used as Clear*
- *Example of XOR Gate Output used as Clear*

Example of a Latch Output Used as Clear

The *Reset_check02* rule reports a violation for the following example in which the `en_1` latch is found in the tree of the clear signal, `reset`, of the `q` flip-flop:

```
architecture logic of test is
```

```
  signal reset, en_1 : std_logic;
  begin
    process(clk, reset)
      begin
        if (reset = '1') then
          q <= '0';
        elseif (rising_edge(clk)) then
          q <= d;
        end if;
     end process;

    process(rst, en)
      begin
        if (rst = '0') then
          en_l <= en;
        end if;
    end process;

    reset <= en_l and rst;
end logic;
```

For this example, this rule reports the following violation message:

`Unexpected latch gate (at test.en_l) in clear tree of flop (output pin test.q).`

Example of Tristate Signal used as Clear

The *Reset_check02* rule reports a violation for the following example in which the `rst` tristate signal is used as the clear signal of the `q` flip-flop:

```
architecture logic of test is
  signal rst : std_logic;
  begin
    process(clk, rst)
      begin
        if (rst = '1') then
          q <= '0';
        elseif (rising_edge(clk)) then
          q <= d;
```

```
        end if;
    end process;

    process(en, rst1)
      begin
        if (en = '1') then
          rst <= rst1;
        else
          rst <= 'Z';
        end if;
    end process;

    process(en, rst2)
      begin
        if (en = '0') then
          rst <= rst2;
        else
          rst <= 'Z';
        end if;
    end process;
end logic;
```

For this example, this rule reports the following violation message:

`Unexpected tristate gate (at test.rst) in clear tree of flop (output pin test.q).`

Example of XOR Gate Output used as Clear

The *Reset_check02* rule reports a violation for the following example in which the `reset` clear signal of the `q` flip-flop is the output of an XOR signal:

```
architecture logic of test is
  signal reset : std_logic;
  begin
    process(clk, reset)
      begin
        if (reset = '1') then
          q <= '0';
```

```
        elseif (rising_edge(clk)) then
          q <= d;
        end if;
    end process;

    reset <= rst1 xor rst2;
end logic;
```

For this example, this rule reports the following violation message:

Unexpected XOR gate (at test.rst) in clear tree of flop (output pin test.q)

Schematic Details

The *Reset_check02* rule highlights an unexpected gate and path from this unexpected gate to a reset pin of a flip-flop.

Default Severity Label

Warning | Info

Rule Group

VERIFY

Reports and/or Related Files

No report or related file

Reset_check03

Reports synchronous reset signals that are used as active high as well as active low

When to Use

Use this rule to detect synchronous reset signals that are used as active high as well as active low.

Prerequisites

Specify synchronous preset/clear reset signals by using the *reset* `-sync` constraint.

Description

The *Reset_check03* rule reports reset signals that are used on both levels, that is, active high and active low, to set or reset flip-flops synchronously.

This rule reports any one active-high usage and one active-low usage for a synchronous reset in a design. You can expand the schematic to look at other occurrences of active-high or active-low usages.

Constraint(s)

- *reset* (Mandatory): Use this constraint to specify synchronous preset/clear reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

Messages and Suggested Fix

The following message appears at a location where the output of the flip-flop *<flop1-name>* that is being reset synchronously by using active high of the signal *<sig-name>* specified by using the `reset` constraint, and another flip-flop *<flop2-name>* that is being reset synchronously using the active low of the same signal *<sig-name>*:

[WARNING] Synchronous reset signal <sig-name> used as active high at <flop1-name> and as active low at <flop2-name>

NOTE: *For RTL designs,* `<flop1-name>` *and* `<flop2-name>` *are names of the output nets of the corresponding flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to yes,* `<flop1-name>` *and* `<flop2-name>` *are names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains a synchronous reset that resets a data path at both active levels, that is active high and active low.

Consequences of Not Fixing

Using synchronous resets as active high and active low simultaneously always keeps some logic of a design in a reset mode and blocks data propagation from that part.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Select appropriate option from the *Edit-> Show Case Analysis* menu option in the *Incremental Schematic* window.
3. Check if you need to apply *set_case_analysis* constraints to enable one polarity of the specified synchronous reset at a time.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

//test.v

```
module flop(d,clk,q);
   input d,clk;
   output q;
   reg q;
   always @(posedge clk)
       q<=d;
endmodule

module top(d1,d2,rst,clk1,clk2,q1,q2);
   input d1,d2,rst,clk1,clk2;
   output q1,q2;
   wire t1,t2,t3;
   not(t3,rst);
   or(t1,d1,rst);
   or(t2,t3,d2);
   flop F1(t1,clk1,q1);
   flop F2(t2,clk2,q2);
endmodule
```

// constr.sgdc

```
current_design top
clock -name clk1
clock -name clk2
reset -name rst -sync
```

In the above example, the `rst` signal name is specified by using the *reset* constraint and its `-sync` argument is also specified. Therefore, this rule reports the following violation in this case:

`Synchronous reset signal top.rst used as active high at top.F1.q and as active low at top.F2.q`

The following figure shows the schematic of this violation:

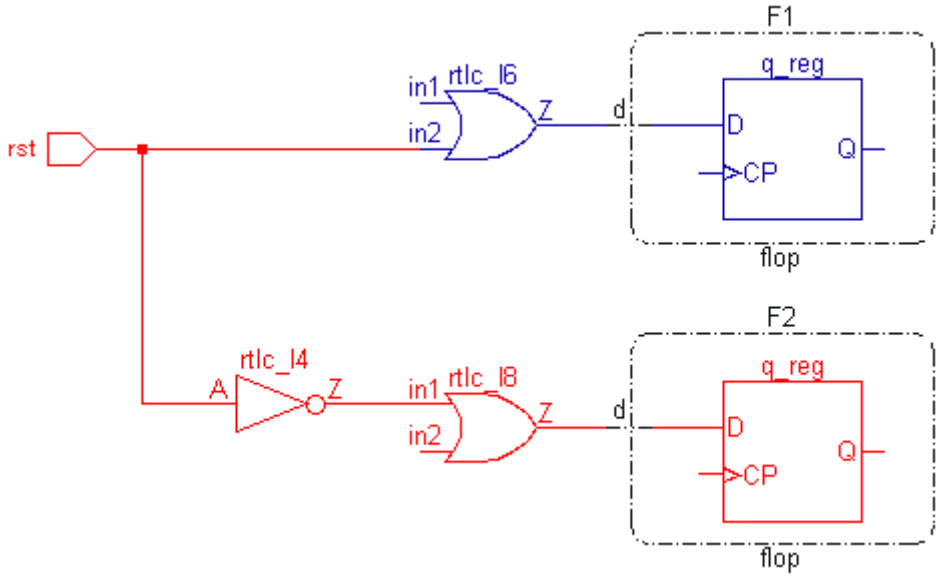


FIGURE 300. Schematic of the Reset_check03 Rule Violation

Schematic Details

The *Reset_check03* rule highlights the following path in different colors in the schematic:

- Path from the specified reset net to an input pin of one flip-flop where the reset net is being used as active high
- Path from the specified reset net to an input pin of another flip-flop where the reset net is being used as active low.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and/or Related Files

No report and related file

Reset_check04

Reports reset signals that are used asynchronously as well as synchronously for different flip-flops

When to Use

Use this rule to detect reset signals that reset a design asynchronously as well as synchronously.

Description

The *Reset_check04* rule reports reset signals that are used as both asynchronous and synchronous reset signals for different flip-flops.

This rule reports any one occurrence where an asynchronous reset is used synchronously or vice-versa. You can expand the schematic to look at other such occurrences.

Constraint(s)

- *reset* (Mandatory): Use this constraint to specify reset signals checked by this rule.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_derived_reset*: Default value is `none`. Set this parameter to `Reset_check04,Reset_check10` to enable the *Reset_check04* and *Reset_check10* rules to report asynchronous *Derived Resets*.

  Other possible values are `Reset_check04` and `Reset_check10`.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.

- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer, same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Messages and Suggested Fix

The following message appears if the output of a flip-flop that is reset synchronously or asynchronously by using the signal *<sig-name>* is first set when there is another flip-flop that is reset asynchronously or synchronously using the same signal:

[WARNING] <Asynchronous | Synchronous> reset "<sig-name>" (at "<flop2-name>") is used <synchronously | asynchronously> (at "<flop1-name>")

NOTE: *For RTL designs,* <flop1-name> *and* <flop2-name> *are names of the output nets of the corresponding flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to* yes, <flop1-name> *and* <flop2-name> *are names of the flip-flop instances. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains reset signals that are used as both asynchronous and synchronous reset signals for different flip-flops.

Consequences of Not Fixing

Using same reset signal asynchronously as well as synchronously may violate some of the design requirements.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. Select an appropriate option from the *Edit->Show Case Analysis* menu option in the *Incremental Schematic* window.
3. Check if you need to apply *set_case_analysis* constraints to enable one type of reset at a time. Else, waive this violation.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(clk,in1,in2,out,in3);
   input clk;
   input in1, in2,in3;
   output out;
   reg t1,t2,t3;
   reg srst;
   always@(posedge clk)
        srst <= in1;
    always@(posedge clk)
       if(!srst)
        t2 <= 1'b0;
       else
       t2 <= in2;
    always@(posedge clk or negedge srst)
       if(!srst)
        t3 <= 1'b0;
      else
      t3 <= in3;
     assign out = t3;
endmodule
```

// constr.sgdc

```
current_design test
clock -name clk
reset -name srst -sync
```

In the above example, the `srst` signal is used asynchronously as well as synchronously.

The following figure shows the schematic of this violation:

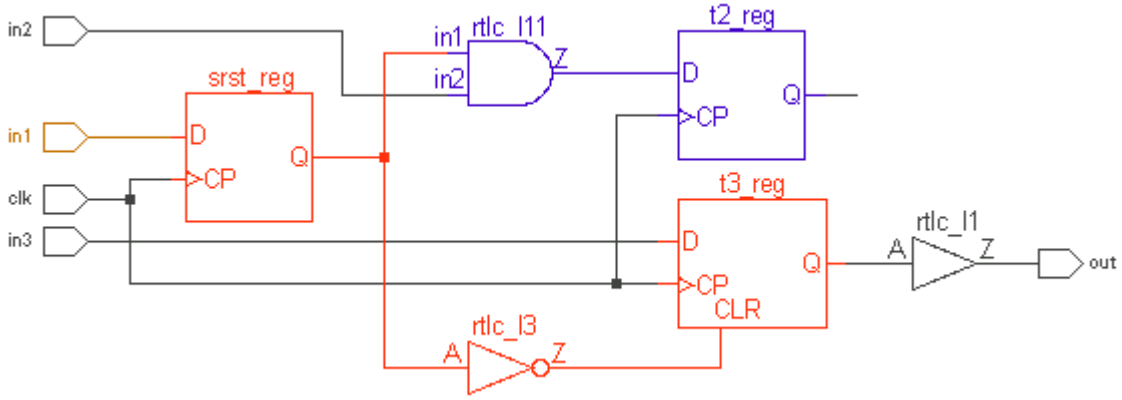


FIGURE 301. Schematic of the Reset_check04 Rule Violation

Schematic Details

The *Reset_check04* rule highlights the asynchronous and synchronous usages of a reset signal in different colors in the schematic.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

- *The Clock-Reset-Summary Report*
- *The SynchInfo Report*

Reset_check05

Reports synchronous resets in a design

NOTE: *The Reset_check05 rule will be deprecated in a future release. The rule is not included in CDC GuideWare goals now and do not perform checks until specifically included in the user-defined goal options. In this case, the rule performs the checks and SpyGlass includes a deprecation message in both the spyglass.out and spyglass.log files.*

When to Use

Use this rule to find synchronous reset signals in a design.

Description

The *Reset_check05* rule reports synchronous reset signals in a design.

If a signal is used as a reset signal at multiple places, this rule reports only the first usage of that signal. However, if a signal is used at a flip-flop as active high and the same signal is used at another flip-flop as active low, such signal is reported separately. Similarly, if a signal at a flip-flop is used as a set and at another flip-flop is used as a reset, it is reported separately.

Synthesis tools may optimize synchronous reset logic such that the corresponding sequential element may have ambiguous initial states.

Please note the following points:

- This rule works with simple synchronous reset constructs only.
- If a design unit containing a synchronous reset description is a parameterized design unit, this rule reports one message for each elaborated personality of the design unit.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

None

Messages and Suggested Fix

The following message appears at the location where a synchronous set/

reset description (using signal *<rst-name>*) is found in a design unit *<du-name>*:

[INFO] Candidate <active-type> synchronous <rst-type> '<rst-name>' found (design unit '<du-name>')

The arguments of the above message are explained below:

Argument	Description
<active-type>	Can be `active-high`, `active-low`, or `mixed-phase`
<rst-type>	Can be `Set` or `Reset`

Potential Issues

This violation appears if a design contains synchronous resets.

Consequences of Not Fixing

None

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Click on the message in GUI.
   This will cross-reference to the relevant RTL code.
2. Review the synchronous reset candidate reported by this rule message.
3. Either use all synchronous resets or all asynchronous resets.
4. If you prefer synchronous reset, disable or waive this rule.

Example Code and/or Schematic

Consider the following design file given as input for SpyGlass analysis:

```
module TOP( CLK, RST, SEL,
  DIN, BIN, QOUT, ZOUT);
  input CLK;
  input RST;
  input SEL;
  input [3:0] DIN;
  input [2:0] BIN;
```

```
output [3:0] QOUT;
output [4:0] ZOUT;
reg [3:0] QOUT;
reg [4:0] ZOUT;
reg RST3; // FF Gate
wire RST1; // AND Gate
wire RST6; // MUX Gate
wire RST9; // XOR Gate
assign RST1 = RST & BIN[0];
always@(posedge CLK)
  RST3 <= RST;
assign RST6 = (SEL)?RST:BIN[1];
assign RST9 = RST ^ BIN[2];
always @(posedge CLK) begin
  if (!RST1) //Active low synchronous reset
    QOUT[0] <= 1'b0;
  else
    QOUT[0] <= DIN[0];
end
always @(posedge CLK) begin
  if (RST1) //Active high synchronous reset
    ZOUT[0] <= 1'b0;
  else
    ZOUT[0] <= DIN[0];
end
always @(posedge CLK) begin
  if (RST1) //Active high synchronous reset
    QOUT[1] <= 1'b1;
  else
    QOUT[1] <= DIN[1];
end
always @(posedge CLK) begin
  if (!RST1) //Active low synchronous reset
    ZOUT[1] <= 1'b1;
  else
    ZOUT[1] <= DIN[1];
end
always @(posedge CLK) begin
```

```
    if (RST1)
      QOUT[2] <= 1'b0;
    else
      QOUT[2] <= DIN[2];
  end
endmodule
```

For the above example, this rule detects active high and active low synchronous reset signals, `RST1`.

Default Severity Label

Info

Rule Group

`VERIFY`

Reports and/or Related Files

No report or related file

Reset_check06

Reports high fan-out reset nets that are not driven by placeholder cells

When to Use

Use this rule to find fan-out reset nets that are not driven by certain types of instances, known as placeholder cells.

Prerequisites

Specify the following information before running this rule:

- Asynchronous reset signals by using the *reset* constraint or the *use_inferred_resets* parameter.
- Placeholder cells by using the *reset_placeholder_cells* parameter.

Description

The *Reset_check06* rule reports reset nets that have a high fan-out and are not driven by instances of any of the specified placeholder cells.

NOTE: *This rule is not run if you do not specify names of placeholder cells.*

Parameter(s)

- *reset_placeholder_cells*: Default value is `NULL`. Specify a list of placeholder cells that are intended to drive reset nets with a high fan-out.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *reset_fanout_max*: Default value is `24`. Set this parameter to a positive integer value to specify a maximum fan-out limit of resets.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to auto-infer asynchronous resets that do not match the specified strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

Constraint(s)

- *reset* (Optional): Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for a reset signal *<rst-name>* with a fan-out *<num>* that is more than the value specified by the *reset_fanout_max* parameter and is not driven by instances of any of the placeholder cells specified using the *reset_placeholder_cells* parameter:

```
[WARNING] Reset “<rst-name>” drives <num> flops that exceeds maximum allowed limit '$reset_fanout_max' and is not driven by any placeholder cells
```

Potential Issues

This violation is reported if your design contains reset nets of a high fan-out (specified by parameter *reset_fanout_max*) and the reset nets are not driven by any of the specified placeholder cells.

Consequences of Not Fixing

If a reset fan-out count exceeds the maximum permissible limit without using appropriate placeholder cells, the reset signal may not have sufficient strength to reset sequential elements.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Review the reset tree specific to the reset net reported by this rule message.
2. Check if you need to add the reset placeholder cells in the design for the reported reset.

   You may have missed to specify some of the reset placeholder cells in the *reset_placeholder_cells* parameter, which are already present in the fan-out of the reported reset net.
3. Check the value of the *reset_fanout_max* parameter, and if it is not set correctly, correct it.
4. You must ensure that violating cells that are intended are included in the *reset_placeholder_cells* parameter list. If the cells are not intended,

make proper changes in the design to reduce the fan-out of the cell below the value specified by the *reset_fanout_max* parameter.

Schematic Details

The *Reset_check06* rule highlights a reset net that is not driven by any of the specified placeholder cells, and exceeds the limit set by the *reset_fanout_max* parameter.

Example Code and/or Schematic

Consider the following example for which you have set the value of the *reset_placeholder_cells* parameter to `buffer` and the *reset_fanout_max* parameter to `2`:

```
module buffer (in,out);
  input in;
  output out;
  assign out = in;
endmodule

module not_buf(in,out);
  input in;
  output out;
  assign out = !in;
endmodule


module test(in, outa, outb, outc, outd, oute, outf,
rsta, rstc, clk, en);
  input in, en, rsta, rstc, clk;
  output reg outa, outb, outc, outd, oute, outf;
  buffer B(rsta, rstb);
  not_buf C(rstc, rstd); //cell does not belong to
                         // reset_placeholder_cells
                         //list
//Reset by rstb
  always @ (posedge clk or posedge rstb)
    if(rstb)    outa <= 0;
     else        outa <= in;
```

```
  always @ (posedge clk or posedge rstb)
    if(rstb)    outb <= 0;
    else        outb <= in;
  always @ (rstb or en)
    if(rstb)    outc <= 0;
    else if(en) outc <= in;
  //Reset by rstd
  always @ (rstd or en)
    if(rstd)    outd <= 0;
    else if(en) outd <= in;
  always @ (rstd or en)
    if(rstd)    oute <= 0;
    else if(en) oute <= in;
  always @ (posedge clk or posedge rstd)
    if(rstd)    outf <= 0;
    else        outf <= in;
endmodule
```

For the above example, the *Reset_check06* rule reports a violation for the `rstc` reset because:

- The `rstc` reset drives three flip-flops, `rstd`, when the maximum allowed limit set by the *reset_fanout_max* parameter is two.
- `rstc` is not driven by any placeholder cell.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Reset_check07

Reports asynchronous reset pins driven by a combinational logic or a mux

When to Use

Use this rule to detect flip-flops or latches for which clear/preset pins are driven by a combinational logic or a mux.

Description

The details of this rule are covered under the following topics:

- *Reason for the Reset_check07 Rule Violation*
- *One Violation Per Reset Cone*
- *Specifying Reset Signals*

Reason for the *Reset_check07* Rule Violation

The *Reset_check07* rule reports asynchronous clear/preset pins driven by a combinational logic or a mux (set *report_reset_path_mux* to `yes`).

For example, consider the following figure:

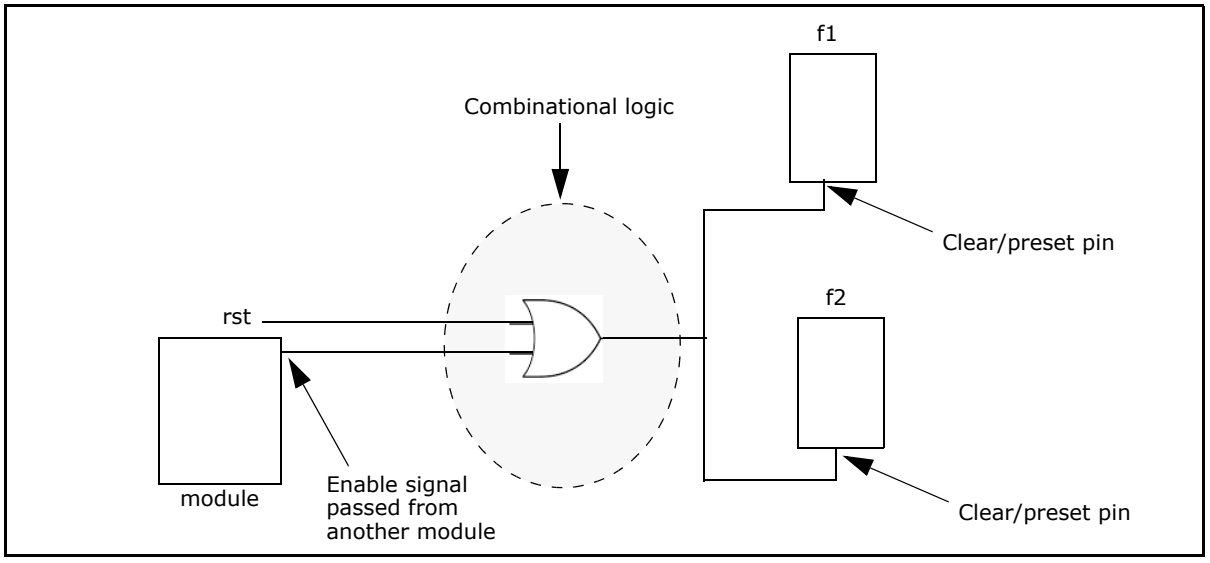


FIGURE 302. Example of the Reset_check07 Rule Violation

For the above example, this rule reports a violation as the clear/preset pins of the `f1` and `f2` flip-flops are driven by a combinational logic.

NOTE: *The Reset_check07 rule does not report violations when inputs/output of a combinational block in the reset path is quasi_static.*

One Violation Per Reset Cone

The *Reset_check07* rule reports one violation for each *Reset Cone*.

For example, in *Figure 302*, this rule reports violation for the `f1` flip-flop.

However, if the signal has multiple drivers, this rule reports a violation for each driver if all of them lie in different *Reset Cone*. For example, consider the following figure in which the input pins of the combinational logic are resets in a design:

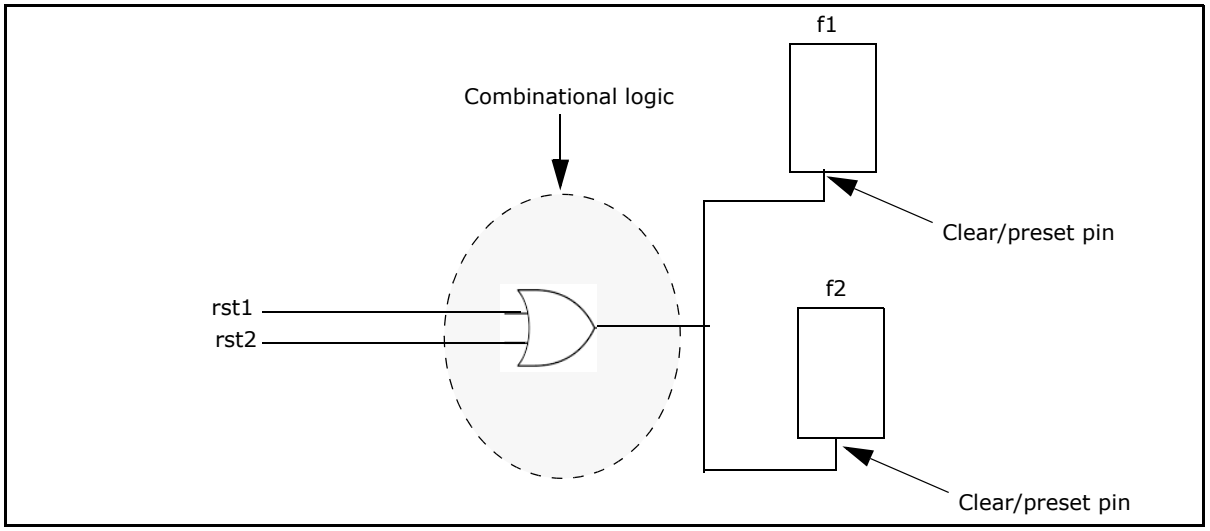


FIGURE 303. Example of the Reset_check07 Rule Violation

In the above scenario, the *Reset_check07* rule reports only one violation for flip-flops, as they lie in the same *Reset Cone*.

This rule reports a violation for one flip-flop per *Reset Cone*. The violations reported are not dependent on the number of clear/preset pins or the number of drivers. You can expand the *Reset Cone* in the schematic to view the other flip-flops.

Specifying Reset Signals

Combinational logic that is a part of reset synchronizers (for example, combinational logic present inside reset synchronize cells), is ignored only when you specify reset signals in any of the following ways to identify valid reset synchronization structures in a design:

- By using the *reset* constraint
- By setting the *use_inferred_resets* parameter to `yes`
- By a combination of the above methods

However, you do not need to specify reset signals for detection of combinational logic in a reset path.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *num_flops* with the `-reset` argument (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

NOTE: *The Reset_check07 rule does not report violations when inputs/output of a combinational block in the reset path is quasi_static.*

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *report_reset_path_mux*: Default value is `no`. Set this parameter to `yes` to report a violation when the asynchronous set/reset pins of a sequential element are driven by mux.
- *reset_synchronize_cells*: Default value is `NULL`. Specify a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *reset_num_flops*: Default value is `2`. Specify a positive integer value, greater than one, to specify different number of flip-flops.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to auto-detect clock signals.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

- *thru_reset_synchronizer*: Default value is `yes`. Set this parameter to `no` so that the *Reset_check07* rule does not report violations for any reset synchronizers that are driven by a combinational logic or mux.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Messages and Suggested Fix

The following message appears for a flip-flop or latch *<flop-name>* for which a *<pin-type>* pin is driven by a combinational logic or a mux:

[WARNING] <pin-type> pin of sequential element '<flop-name>' is driven by <combinational logic | mux>

Where, *<pin-type>* can be `Set` or `Clear`.

NOTE: *For RTL designs,* `<flop-name>` *is the name of the output net of the flip-flop. For netlist designs, if the report_inst_for_netlist parameter is set to yes, <flop-name> is the name of the flip-flop instance. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation is reported if your design contains flip-flops/latches for which set/reset pins are driven by a combinational logic or a mux.

Consequences of Not Fixing

If you do not fix this violation, there is a possibility of glitches on the set/reset signal.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, review the logic shown in the reset path.
3. Enable *Show Case Analysis* to check if the *set_case_analysis* constraints are missing in the SGDC file that makes the logic in the reset path glitch free.

4. If a combinational logic or a mux is expected to be the part of a reset synchronizer, run and analyze the *Ar_sync01* and *Ar_unsync01* rules to verify whether it is detected as a reset synchronizer.

To fix this issue, eliminate the combinational logic or mux found in the reset path. However, if your design style permits the logic and you wish to allow it in your design, disable this rule.

Example Code and/or Schematic

Consider the following example:

```
module core_top (rsta, in, out, en);
  input rsta;
  input in;
  input en;
  output reg out;
  wire rstb;
  wire comb;
  assign rstb = rsta & comb;
  always @ (en or rstb)
    if(rstb)      out = 0;
    else if (en) out = in;
endmodule
```

For the above example, the *Reset_check07* rule reports a violation because the clear pin of the latch is driven by an AND gate for which one of the input is a top-level reset.

The schematic of this violation is shown in the following figure:

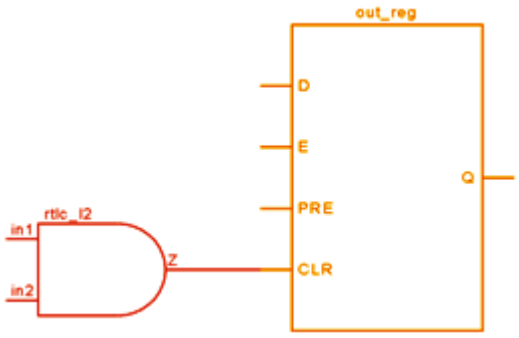


FIGURE 304. Schematic of the Reset_check07 Rule Violation

To fix this issue, remove the combinational logic present in the path of the reset.

However, if your design style permits the combinational logic and you wish to allow the logic in your design, disable this rule.

Schematic Details

The *Reset_check07* rule highlights the path from a combinational gate to the set/reset pin of a flip-flop/latch.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

The SynchInfo Report

Reset_check09

Reports XOR, XNOR, AND, or NAND gates found in a reset tree

When to Use

Use this rule during the RTL or pre-layout phase to detect any glitch generating logic in a design.

Description

The *Reset_check09* rule reports glitch generating logic, such as XOR, XNOR, AND, and NAND gate in the fan-in cone of preset or clear terminals of all flip-flops with asynchronous resets.

This rule traverses on paths until a primary port, sequential element, or black box is found. It reports a violation if any of the disallowed gates is detected and then it further continues traversal.

This rule reports a gate only once even if it appears in fan-in cone of multiple flip-flops.

Rule Exceptions

The *Reset_check09* rule does not report a violation if a logic is equivalent to a buffer after applying the *set_case_analysis* constraint.

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

- *unex_reset_gate_list*: Default value is `NULL`. Specify a comma or space-separated list of disallowed cell names.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report a violating instance name in case of netlist designs and a leaf-level net name for RTL designs.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Messages and Suggested Fix

The following message appears when the XOR, XNOR, AND, or NAND gate instance *<hier-inst-name>* is found in the fan-in cone of a reset pin of the flip-flop *<flop-name>* in the cell *<module-name>*:

[WARNING] Instance *<hier-inst-name>* (cell name *<module-name>*) found in the fanin cone of *<reset-type>* pin of sequential element *<flop-name>*

where, *<reset_type>* can be `preset` or `clear`

NOTE: *For RTL designs, <flop-name> is the name of the output net of the flip-flop. For netlist designs, if the report_inst_for_netlist parameter is set to* `yes`*, <flop-name> is the name of the flip-flop instance. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains any glitch-prone combinational cell in the *Reset Cone* of a sequential cell.

Consequences of Not Fixing

If you do not fix this violation, your design may contain glitches in the reset path.

How to Debug and Fix

To fix this violation, avoid using the XOR, XNOR, AND, or NAND gate instance in the fan-in cone of the preset or clear terminals of flip-flops with asynchronous resets.

Example Code and/or Schematic

Consider the following design file specified for SpyGlass analysis:

```
// test.v

module top(input d,clk, output reg q);
wire w3,w4,rst;
flop F1 (.d(d), .clk(clk), .rst(~(~w3&~w4)), .q(q));
assign rst = w3^w4;
always@(posedge clk or posedge rst)
   if(rst) q <= 1'b0;
   else q <=d;
endmodule
module flop(input d, clk, rst, output reg q);
   always@(posedge clk or posedge rst)
     if (rst) q <= 1'b0;
     else q <= d;
endmodule
```

FIGURE 305.

For the above example, the *Reset_check09* rule reports a violation for the gate instances `rtlc_N2` and `rst`.

The following figure shows the schematic highlighting the path containing the `rtlc_N2` instance:

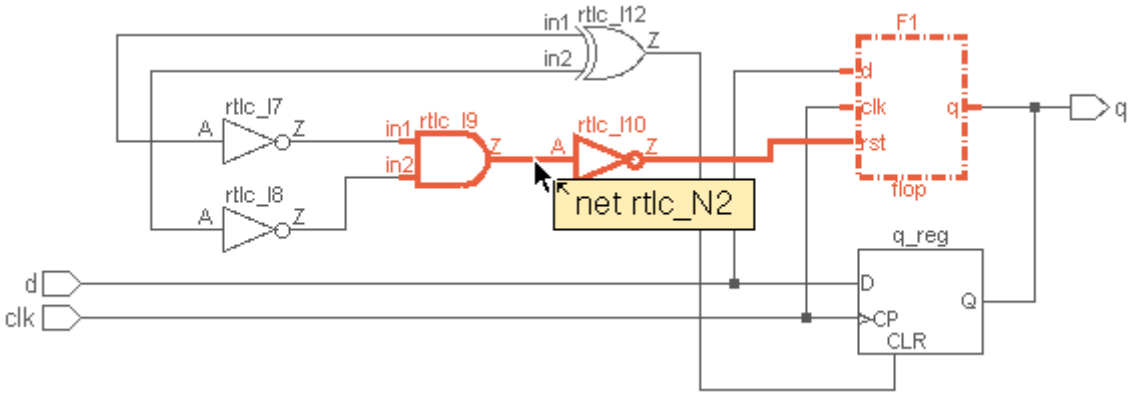


FIGURE 306. Schematic of the Reset_check09 Rule Violation

Schematic Details

The *Reset_check09* rule highlights the path from the reset terminal of flip-flop till the violating gate in its fan-in cone.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related files

No report or related file

Reset_check10

Reports asynchronous resets used as non-reset signals

When to Use

Use this rule to detect incorrect usage of asynchronous reset signals.

Prerequisites

Specify reset signals in any of the following ways:

- By using the *reset* constraint
- By using the automatically generated resets after setting the *use_inferred_resets* parameter to `yes`
- By using a combination of both the above methods

Description

The details of the *Reset_check10* rule are covered under the following topics:

- *Reason for the Reset_check10 Violation*
- *Cases of Reporting Asynchronous Derived Resets by the Reset_Check10 Rule*
- *Rule Exceptions*

Reason for the Reset_check10 Violation

The *Reset_check10* rule reports asynchronous resets that drive any of the following objects:

- Data terminal of sequential elements
- Control terminal of sequential elements
- Primary ports
- Black boxes
- Library cells

For example, this rule reports a violation in the following scenario in which the asynchronous reset `rst` is driving the data terminal `q2` of the `q2_reg` flip-flop:

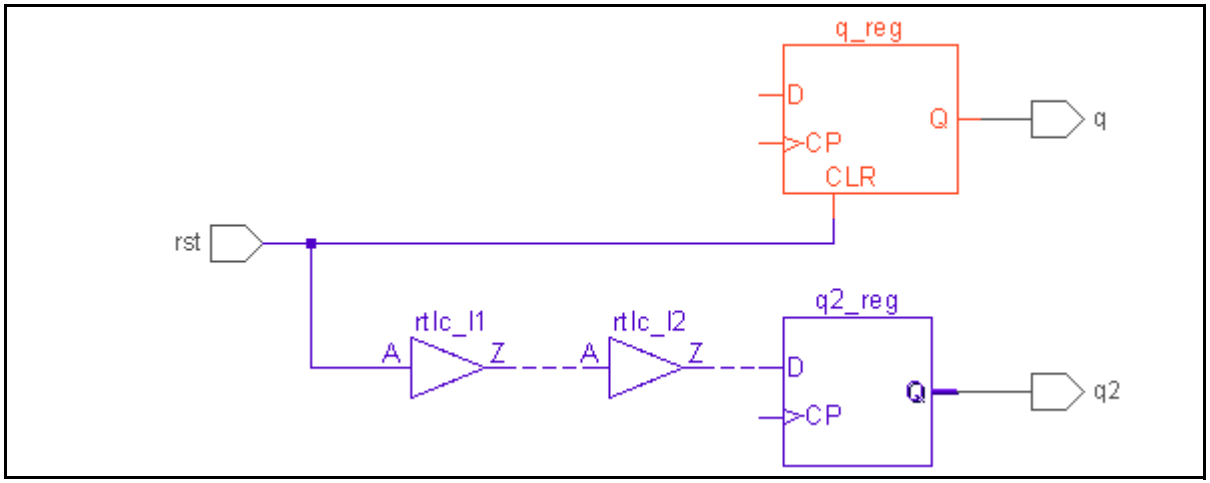


FIGURE 307. Schematic of the Reset_check10 Rule Violation

Cases of Reporting Asynchronous Derived Resets by the Reset_Check10 Rule

When the *report_derived_reset* parameter is set to *Reset_check10*, the following type of scenario is reported by the *Reset_check10* rule:

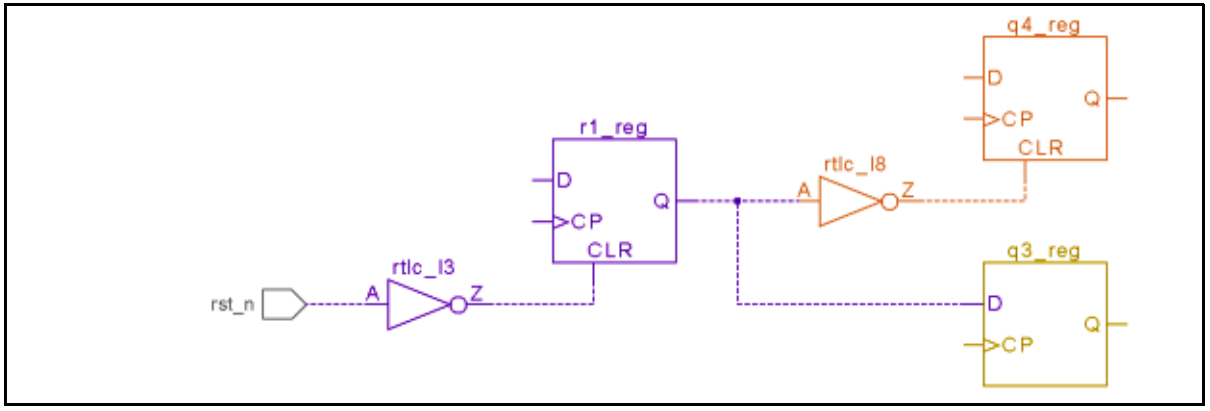


FIGURE 308.

In the above scenario, the *Reset_check10* violation appears as the output of `r1_reg` acts as the derived reset of `rst_n` and is used as a reset at

q4_reg and data at q3_reg.

However the following cases are not reported by the *Reset_check10* rule:

- When the same derived reset is used as a data and as a reset in the same flip-flop. This scenario is shown in the following figure:

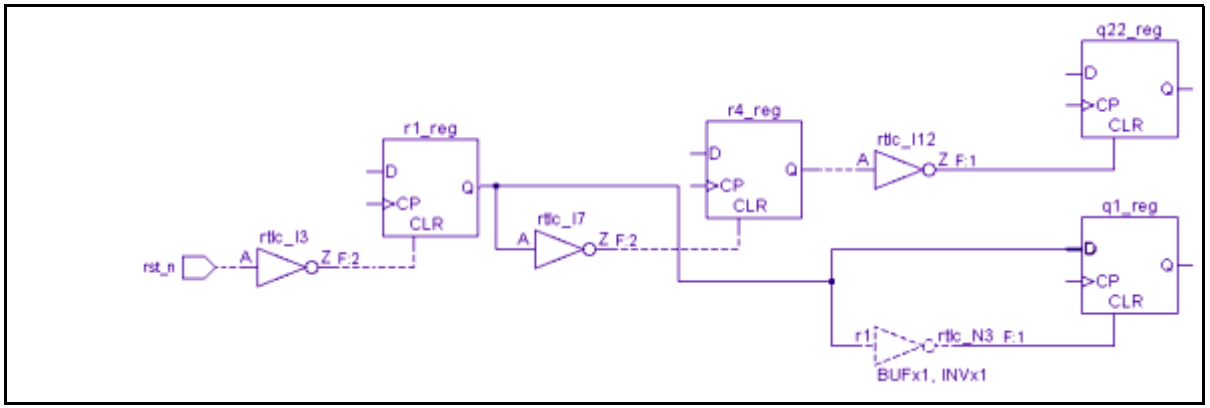


FIGURE 309.

In the above scenario, the *Reset_check10* rule does not report violation for the output of r1_reg that is used as a data signal at q1_reg. This is because q1_reg is using the same reset signal as data.

- If a reset passes through the enable of an isolation cell and the output of that cell is used as a data signal.

Rule Exceptions

This rule ignores data usage of an asynchronous reset if the D-flop is a part of a reset synchronizer structure reported by the *Ar_sync01* rule.

For example, this rule does not report a violation in the following scenario:

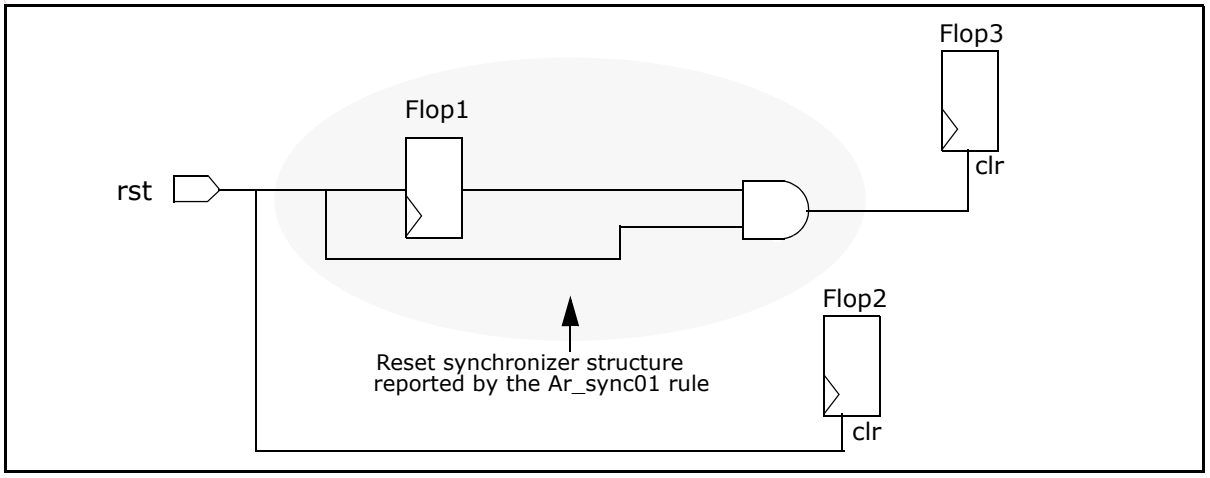


FIGURE 310. Reset_check10 Rule Exception

Constraint(s)

- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *reset_synchronizer* (Optional): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

- *async_reset_usage*: Default value is `data`. Specify a comma-separated list of values to specify asynchronous reset types to be reported for non-reset usage. Possible types are `data`, `control`, `ports`, `bbox`, `libcell` and `all`.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.

- *reset_synchronize_cells*: Default value is `NULL`. Specifies a comma-separated list of synchronizer cell names that are considered as valid synchronizers for asynchronous reset signals.
- *filter_named_resets*: Default value is clk, clock, scan. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- *report_derived_reset*: Default value is `none`. Set this parameter to `Reset_check04,Reset_check10` to enable the *Reset_check04* and *Reset_check10* rules to report asynchronous *Derived Resets*.

  Other possible values are `Reset_check04` and `Reset_check10`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer, same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.
- *cdc_reduce_pessimism*: Default value is `mbit_macro, no_convergence_at_syncreset, no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.

Messages and Suggested Fix

Message 1

The following message appears when the asynchronous signal *<reset-name>* reaches the data of a sequential element:

[WARNING] Asynchronous reset signal '*<reset-name>*' (at *<inst1-type>* '*<inst1-name>*') is reaching to data of *<inst2-name>*

The arguments of the above message are explained below:

Argument	Description
<reset-name>	Hierarchical name of the asynchronous reset signal
<inst1-type>	`flop` in case of flip-flops. `library-cell` in case of library cells
<inst1-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst1-name>* is the name of the instance where the reset signal is used asynchronously. Otherwise, it is the name of the output net of the instance where the reset signal is used asynchronously.
<inst2-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst2-name>* is the name of the instance whose data/control pin is driven by the reset signal. Otherwise, it is the name of the output net of the instance whose data/control pin is driven by the reset signal.

Potential Issues

This violation appears if your design contains an asynchronous reset signal that reaches to the data input of a sequential element.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To fix this violation, avoid using asynchronous resets as non-reset signals. In addition, check if the reset signal is a synchronous reset signal.

Message 2

The following message appears when the asynchronous signal *<reset-name>* reaches to the control of a sequential element:

[WARNING] Asynchronous reset signal '*<reset-name>*' (at *<inst1-type>* '*<inst1-name>*') is reaching to control of *<inst2-name>*

The arguments of the above message are explained below:

Argument	Description
<reset-name>	Hierarchical name of the asynchronous reset signal
<inst1-type>	`flop` in case of flip-flops. `library-cell` in case of library cells
<inst1-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst1-name>* is the name of the instance where the reset signal is used asynchronously. Otherwise, it is the name of the output net of the instance where the reset signal is used asynchronously.
<inst2-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst2-name>* is the name of the instance whose data/control pin is driven by the reset signal. Otherwise, it is the name of the output net of the instance whose data/control pin is driven by the reset signal.

Potential Issues

This violation appears if your design contains an asynchronous reset signal that reaches to the control input of a sequential element.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To debug and fix this violation, analyze the incremental schematic of the violation. Check if you can avoid using asynchronous resets as non-reset signals.

Message 3

The following message appears when the asynchronous signal *<reset-name>* reaches a primary port:

[WARNING] Asynchronous reset signal '*<reset-name>*' (at *<inst1-type>* '*<inst1-name>*') is reaching to port

<port-name>

The arguments of the above message are explained below:

Argument	Description
<reset-name>	Hierarchical name of the asynchronous reset signal
<inst1-type>	`flop` in case of flip-flops. `library-cell` in case of library cells
<inst1-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst1-name>* is the name of the instance where the reset signal is used asynchronously. Otherwise, it is the name of the output net of the instance where the reset signal is used asynchronously.
<port-name>	Refers to the port name where the reset signal is arriving.

Potential Issues

This violation appears if your design contains an asynchronous reset signal that reaches to an output port.

Consequences of Not Fixing

If you do not fix this violation, your design may contain functional issues.

How to Debug and Fix

To fix this violation, avoid using asynchronous resets as non-reset signals.

Message 4

The following message appears when the asynchronous signal *<reset-name>* reaches a black box instance at the pin *<pin-name>*:

[WARNING] Asynchronous reset signal '*<reset-name>*' (at *<inst1-type>* '*<inst1-name>*') is reaching to blackbox instance *<inst3-name> (blackbox/<pin-name>)*

The arguments of the above message are explained below:

Argument	Description
<reset-name>	Hierarchical name of the asynchronous reset signal
<inst1-type>	`flop` in case of flip-flops. `library-cell` in case of library cells
<inst1-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst1-name>* is the name of the instance where the reset signal is used asynchronously. Otherwise, it is the name of the output net of the instance where the reset signal is used asynchronously.
<inst3-name>	Refers to the name of the black box instance that is driven by the reset signal
<pin-name>	Refers to the pin name of the black box, where the reset signal is arriving.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

// test.v

```
module top(q,rst,clk,d,q2);
   input clk,rst,d;
   output reg q, q2;
   wire t;
   always@(posedge clk or posedge rst) begin
       if(rst)
        q<=1'b0;
       else
         q<=d;
       end
   assign t = rst;
   always@(posedge clk)
      q2<=t;
endmodule
```

// constr.sgdc

```
current_design top
reset -name rst
```

In the above example, the asynchronous reset `rst` hits the data input of the `q2` flip-flop. Therefore, the *Reset_check10* rule reports a violation.

The following figure shows the schematic of this violation:

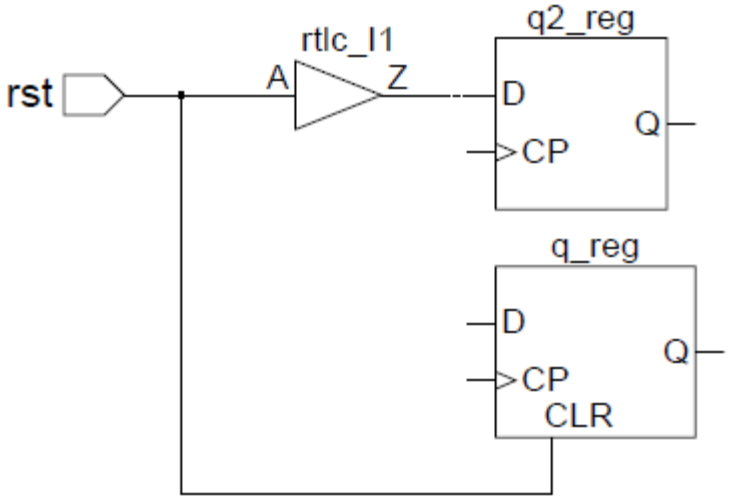


FIGURE 311. Schematic of the Reset_check10 Rule Violation

Schematic Details

The *Reset_check10* rule highlights the following information in the schematic:

- The path from an asynchronous reset source to one of the flip-flops/ sequential cell where it is used as an asynchronous reset.
- Path from an asynchronous reset source to the primary port or black box or the instance where the reset is used as data or control.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

The SynchInfo Report

Reset_check11

Reports asynchronous resets that are used as both active-high and active-low

When to Use

Use this rule to detect incorrect usage of asynchronous reset signals.

Prerequisites

Specify the following information before running this rule:

- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically generated resets after setting the *use_inferred_resets* parameter to `yes`
  - By using a combination of both the above methods
- Enable this rule by specifying the `set_goal_option addrules {Reset_check11}` command in the project file.

Description

The details of the rule are covered under the following topics:

- *Reason for the Reset_check11 Rule Violation*
- *Functioning of the Reset_check11 Rule*
- *Rule Exceptions*

Reason for the Reset_check11 Rule Violation

The *Reset_check11* rule reports asynchronous reset signals that are used at both levels, that is active high and active low, to set or reset flip-flops/ sequential-elements.

For example, in the following figure, the `rst` reset is used as active high at `q1` and active low at `q3`:

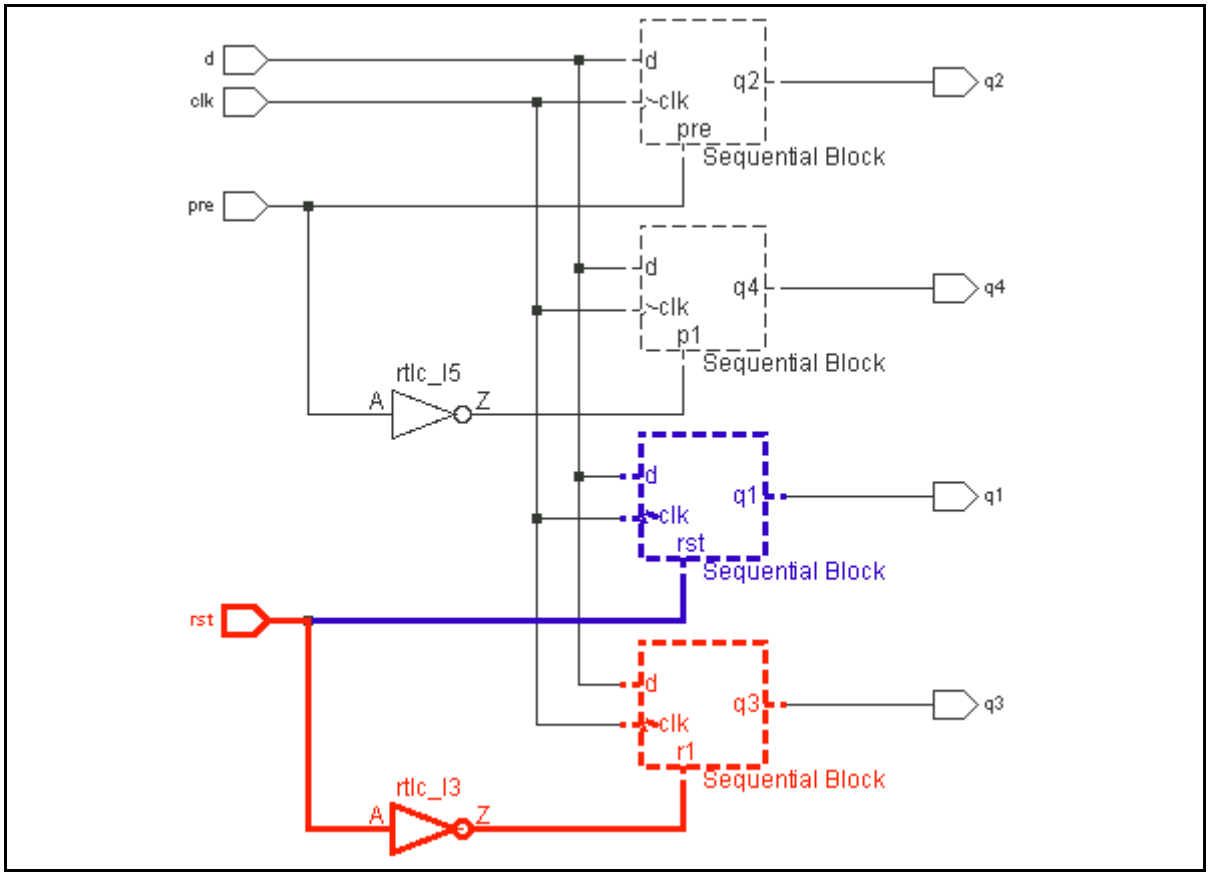


FIGURE 312. Schematic of the Reset_check11 Rule Violation

This rule reports any one active-high usage and one active-low usage for an asynchronous reset in a design. You can expand the schematic to view other occurrences of such active-high or active-low usages.

Functioning of the Reset_check11 Rule

The *Reset_check11* rule functions in the following manner:

1. It applies both active and inactive value on every reset signal and performs simulation every time for each reset twice, that is, one with the active value and another with the inactive value.

2. It then checks for the existence of two flip-flops such that one flip-flop can be asserted with the active value and the other flip-flop can be asserted with the inactive value.

   If such flip-flops exist, the *Reset_check11* rule reports a violation.

Rule Exceptions

The *Reset_check11* rule does not report a violation if a reset signal reaches a sequential element with an undetermined phase.

For example, this rule does not report a violation in the following scenario:

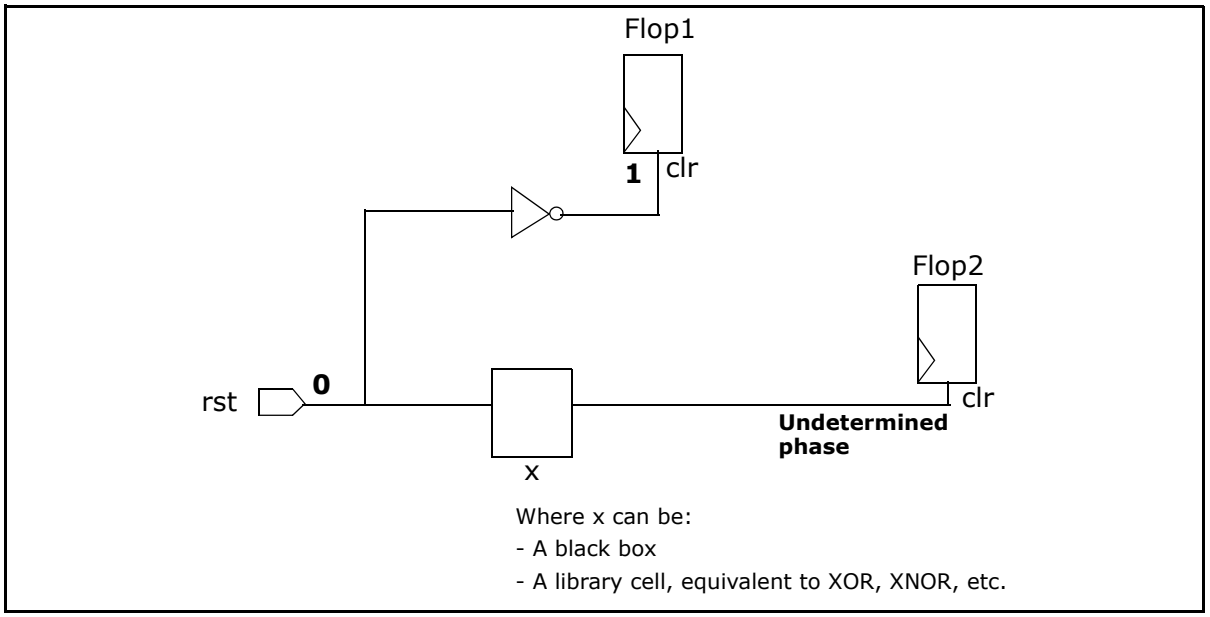


FIGURE 313. Reset_check11 Rule Exception

Constraint(s)

- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

Messages and Suggested Fix

The following message appears when the asynchronous signal *<reset-name>* is used at both active high and active low levels:

[WARNING] Asynchronous reset signal '*<reset-name>*' is used as active-high at '<inst1-name>' and active-low at '<inst2-name>'

The arguments of the above message are explained below:

Argument	Description
<reset-name>	Hierarchical name of the asynchronous reset signal
<inst1-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst1-name>* is the name of the instance where the reset signal is used as active-high. Otherwise, it is the name of the output net of the instance where the reset signal is used as active-high.
<inst2-name>	In case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`, *<inst2-name>* is the name of the instance where the reset signal is used as active-low. Otherwise, it is the name of the output net of the instance where the reset signal is used is used as active-low.

Potential Issues

This violation appears if your design contains an asynchronous reset that is used at both active-high and active-low levels.

Consequences of Not Fixing

If you do not fix this violation, the design does not initialize properly. As a result, functional analysis may not work properly in this case.

How to Debug and Fix

To fix this violation, use only one active level for a particular asynchronous reset in a design.

Example Code and/or Schematic

Consider the following file specified during SpyGlass analysis:

```
module test(rst,q1,q2,d1,d2,clk);
  input rst,d1,d2,clk;
  output q1,q2;
  reg q1,q2;
  always@(posedge clk or negedge rst) //active low clear
    if(!rst)
      q1 <= 0;
    else
      q1 <= d1;
  always@(posedge clk or posedge rst)
    // active high clear
    if(rst)
      q2 <= 0;
    else
      q2 <= d2;
endmodule
```

In the above example, the asynchronous reset signal `rst` is used as active-high at `q2` and as active-low at `q1`. Therefore, the *Reset_check11* rule reports a violation.

The following figure shows the schematic of this violation:

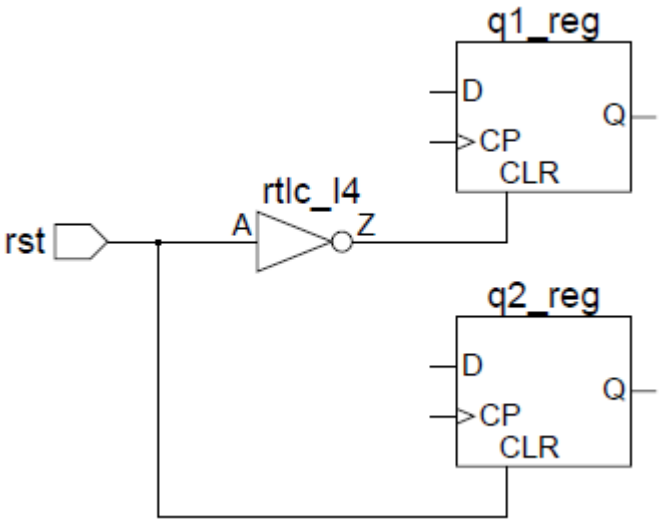


FIGURE 314. Schematic of the Reset_check11 Rule Violation

Schematic Details

The *Reset_check11* rule highlights the following information in the schematic:

- The path from an asynchronous reset net to the set/reset pin of one flip-flop/sequential-element where the reset is used as active-high.
- The path from the asynchronous reset net to the set/reset pin of another flip-flop/sequential-element where the reset is used as active low.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Reset_check12

Reports flip-flops, latches, and sequential elements that do not get an active reset during power on a reset

When to Use

Use this rule to detect sequential elements that are not asserted on the active value of asynchronous resets/sets.

Prerequisites

Specify the following information before running this rule:

- Specify resets by using the *reset* constraint.
- Enable this rule by specifying the `set_goal_option addrules {Reset_check12}` command in the project file.

Description

The details of the *Reset_check12* rule are covered under the following topics:

- *Reason for the Reset_check12 Rule Violation*
- *Features of the Reset_check12 Rule*
- *Rule Exceptions*

Reason for the Reset_check12 Rule Violation

The *Reset_check12* rule reports sequential elements that are not asserted on the active value of asynchronous resets/sets.

For example, this rule reports a violation in the following scenario as the `RST` reset is not asserted with the active value `0`:

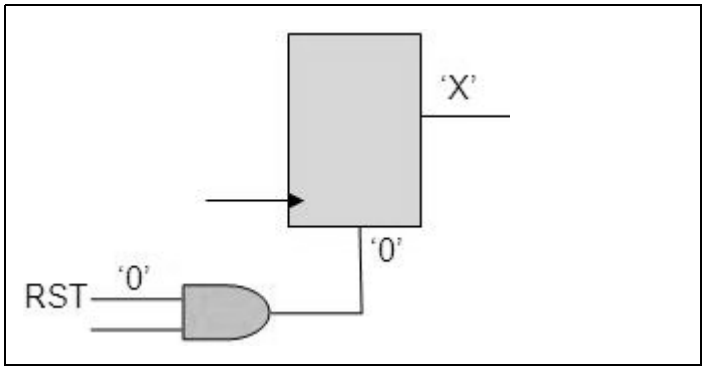


FIGURE 315. Reset_check12 Rule Violation

However, this rule does not report a violation in the following scenario because the `RST` reset is getting asserted as per the active value `0` specified in the SGDC file:

```
//constr.sgdc
current_design top
reset -name RST -value 0
```

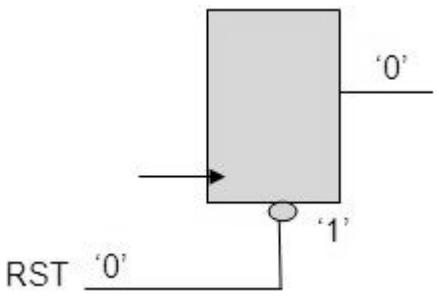


FIGURE 316. No Reset_check12 Rule Violation

Features of the Reset_check12 Rule

Following are the features of the *Reset_check12* rule:

- This rule applies active values on all resets and `x` on other nets in a design. It then reports sequential elements that are not receiving active the value on their asynchronous reset/set pins.

- If a sequential instance has both reset and set pins, then by default, this rule does not report a violation if the instance gets asserted from any of the reset or set pins.

  Set the *allow_any_async_pin* parameter to `no` to check for both the pins.
- If multiple resets/sets reach to a flip-flop, this rule checks if the reset/set pin is active after applying active values on all the resets/sets.

  For example, consider the following figure:

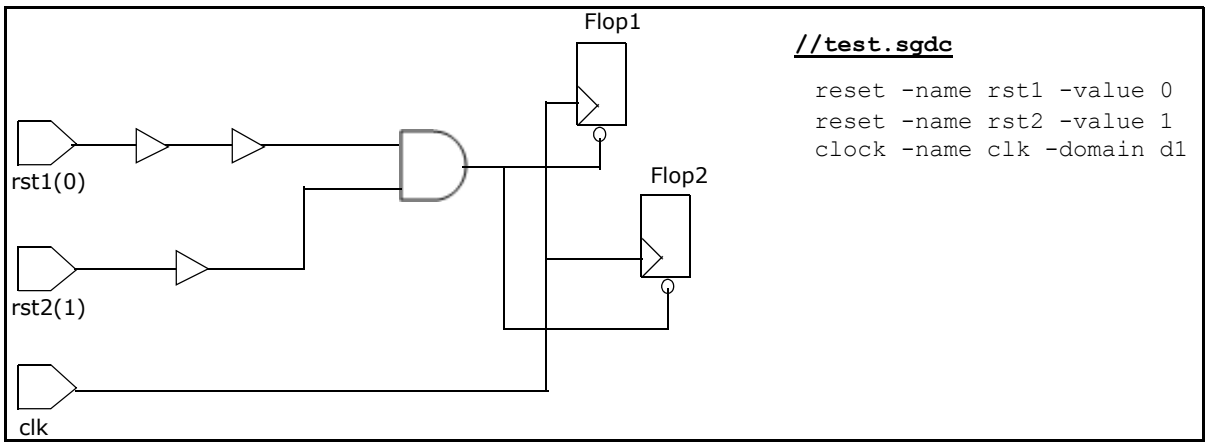


FIGURE 317. Reset_check12 Example

In the above figure, the combination of reset values reaching `Flop1` ensures that it is asserted. However, `Flop2` is not asserted.

Rule Exceptions

This rule has the following exceptions:

- It does not report a violation for automatically-inferred resets when the *use_inferred_resets* parameter is set to `yes`.
- It does not traverse over black boxes on which the *assume_path* is applied, because it is based on simulation.
- It does not report cases in which a reset is not specified for a sequential element. Such cases are reported by the *Reset_info09a* rule.
- It does not check resets for which an active value is not specified by the *reset* constraint.

- It does not report a violation for the blocked path present in the fan-in of the reset of a sequential element.

Constraint(s)

- *reset* (Mandatory): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.

Parameter(s)

- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report the violating instance name in case of netlist designs and the leaf-level net name for RTL designs.
- *allow_any_async_pin*: Default value is `yes`. Set this parameter to `no` to consider a flip-flop to be asserted if both set and reset pins assert that flip-flop.

Messages and Suggested Fix

Message 1

The following message appears if the sequential element *<name>* is not asserted on the active value of asynchronous sets/resets:

[RstC12_1] [WARNING] <cell-type> <name> is not asynchronously asserted

For instance pin, *Message 2* is reported.

Details of the arguments of the above message are described in the following table:

Argument	Description
<cell-type>	It can be `flop`, `latch`, or `sequential cell`.
<name>	In case of netlist designs, if the `report_inst_for_netlist` parameter is set to `yes`, *<name>* refers to the name of the output pin of the sequential instance that is not asserted asynchronously. Otherwise, *<name>* refers to the name of the output net of the sequential element that is not asserted asynchronously on active value of resets.

Potential Issues

This violation appears if the design contains a sequential element that is not asserted on the active value of asynchronous resets/sets.

Consequences of Not Fixing

If you do not fix this violation, basic functionality of an asynchronous reset/set is not served. In this case, the reported cell is not initialized and therefore, the chip does not have the expected functionality.

How to Debug and Fix

To debug this violation, perform the following steps:

1. View the incremental schematic of the violation message.
2. Review the logic shown in the reset/set path in the schematic.
3. Enable *Show Case Analysis* in the incremental schematic to check if the *set_case_analysis* constraints in the SGDC file or the logic in RTL design blocks the propagation of the value (`-value`) from the reset/set source to sequential elements.

   If required, check if the value reaching to the sequential element is the assertion value of the sequential element.

To fix this issue, eliminate the blocking logic of the reset value or correct the `-value` specification for reset in the SGDC file.

Message 2

The following message appears when the *report_inst_for_netlist* parameter is set to `yes`, and the sequential element of an instance pin is not asserted on the active value of asynchronous resets/sets:

```
[RstC12_1] [WARNING] <cell-type> instance pin <name> is not
asynchronously asserted
```

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 3

The following message appears if no *reset* constraint with an active value is defined for an asynchronous reset/set:

```
[RstC12_2] [WARNING] For design unit '<du-name>', no SGDC
command found for Asynchronous Reset with active value
```

Potential Issues

This violation appears if design contains an asynchronous reset/set but no *reset* constraint with an active value is defined for such reset/set.

Consequences of Not Fixing

If you do not fix this violation, such asynchronous resets/sets are not considered for rule checking.

How to Debug and Fix

To fix this violation, specify a *reset* constraint for the asynchronous reset/set with an active value.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

// test.v

```
module test(clk,rst,pre,din,out);
input clk,rst,pre,din;
output out;
reg out;

always@(posedge clk or negedge rst or posedge pre)
if(!rst)
  out <= 1'b0;
 else
 if(pre)
  out <=1'b1;
else
  out <= din;
endmodule
```

// test.sgdc

```
current_design test
clock -name clk
reset -name rst -value 1
reset -name pre -value 0
```

In the above example, `test.out` is neither asynchronously asserted by the reset nor by the preset. Therefore, the *Reset_check12* rule reports a violation.

The following figure shows the schematic of this violation:

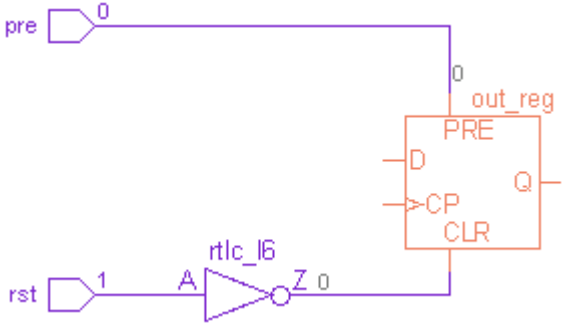


FIGURE 318. Schematic for the Reset_check12 Rule Violation

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

No report or related file

Reset_overlap01

Reset reaches another reset

When to Use

Use this rule to detect the reset reaching another reset.

Prerequisites

The prerequisites are as follows:

- Set the *stop_at_reset* parameter to yes to run this rule.
- Specify resets by using the *reset* constraint.

Description

The *Reset_overlap01* rule reports the cases when one reset reaches another reset.

Parameter(s)

- *stop_at_reset*: Default value is `yes`. Set this parameter to `no` to continue propagation of the reset reaching another reset in its path.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset* (Mandatory): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.

Messages and Suggested Fix

The following message appears when reset propagation of the reset <rst1-name> reaches the reset net <rst2-name>:

```
[WARNING] Reset propagation for reset '<rst1-name>' has reached
reset <rst2-name>'. Halting further propagation of reset
'<rst1-name>' on this path
```

Potential Issues

This violation appears when a design contains a reset that reaches to another reset from a different domain in the path.

Consequences of Not Fixing

If you do not fix this violation, reset propagation stops along the path where a reset reaches another clock of a different domain. This may result in an improper reset domain crossing analysis.

How to Debug and Fix

To debug this violation, perform the following steps:

- Check if the resets reported by the message are specified correctly in the SGDC file.
- Open the schematic of the violation and check if reset propagation occurs correctly.
- Make necessary changes to eliminate the conflict.

Example Code and/or Schematic

Consider the following figure:

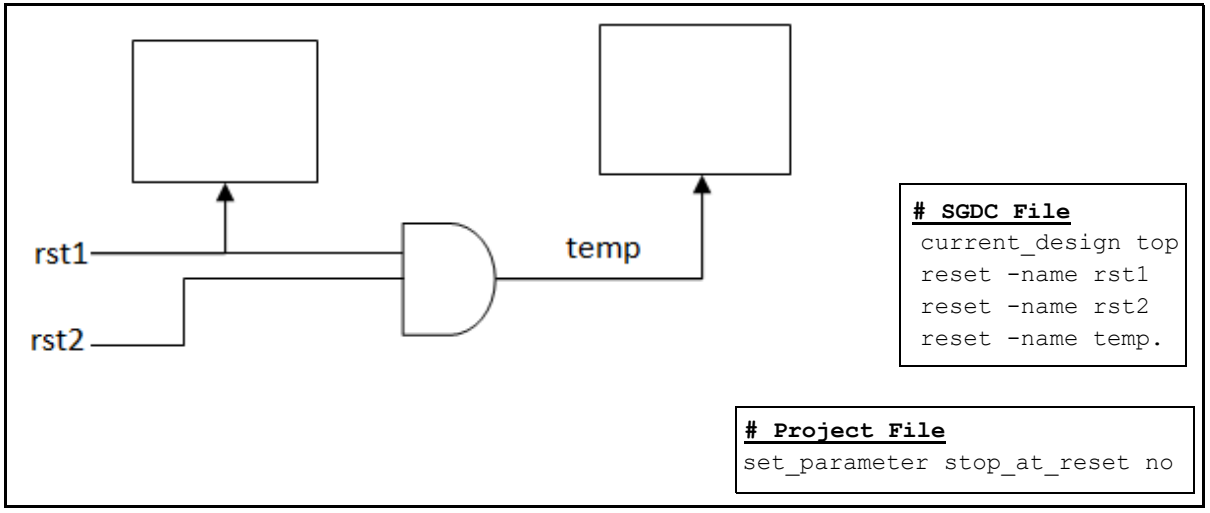


FIGURE 319.

In the above example, both the `rst1` and `rst2` resets reach the different reset domain `temp`. Therefore, the following *Reset_overlap01* rule violations appear:

- `Reset propagation for reset 'top.rst1' has reached reset 'top.temp'.Halting further propagation of reset 'test.rst1' on this path.`
- `Reset propagation for reset 'top.rst2' has reached reset 'top.temp'.Halting further propagation of reset 'test.rst2' on this path.`

Schematic Details

The schematic highlights the path from one reset to the point where another reset is reached.

Default Severity Label

Warning

Rule Group

Reports and Related Files

- *The Clock-Reset-Summary Report*
- *autoresets.sgdc*: This file contains all the primary resets and black box presets/clears specified in the SGDC format.

Clock and Reset Checking Rules

The SpyGlass CDC solution has the following rules for checking clock and reset conditions:

Rule	Reports
Clock_Reset_check01	Instances of specified cells found in a clock tree, reset tree, or fan-out of a net
Clock_Reset_check02	Potential race conditions between output of a flip-flop and its clock/preset/clear signal
Clock_Reset_check03	Potential race conditions between clock signal and reset/set signal of a flip-flop

Clock_Reset_check01

Reports unwanted cells found in clock or reset networks

When to Use

Use this rule to detect unwanted instances of cells appearing either in clock or reset networks or in the entire design.

Prerequisites

Specify names of allowed or disallowed cells by using the *network_allowed_cells* constraint.

Description

The *Clock_Reset_check01* rule reports a violation if instances of specified cells are found in clock, reset, or other networks.

This rule reports RTL inferred logic also if SpyGlass synthesis-generated primitive names (`RTL_*` or `M_RTL_*`) are given in disallowed list or not given in allowed list.

Parameter(s

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *network_allowed_cells* (Mandatory): Use this constraint to specify names of allowed and disallowed cells in clock trees.

If you specify clock or reset in the *-type* argument of this constraint, it is mandatory to specify the *clock* or *reset* constraint, respectively.

- *clock*: Use this constraint to specify clock signals.
- *reset*: Use this constraint to specify reset signals.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears for an instance *<inst-name>* of a cell *<cell-name>* that should not be present in a clock or reset tree based on the specified constraints:

```
[ClkRstC1_1] [WARNING] Instance '<inst-name>'(cell name '<cell-name>') should not be present
```

Potential Issues

This violation appears if your design contains instances of disallowed cells specified by using the *network_allowed_cells* constraint.

Consequences of Not Fixing

Instances of disallowed cells may add to delays on propagated networks thereby altering the expected output values.

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

The following message appears for an instance *<inst-name>* of a cell *<cell-name>* that should not be present in the fan-out of net *<net-name>* based on the specified constraints:

```
[ClkRstC1_2] [WARNING] Instance '<inst-name>'(cell name '<cell-name>') should not be present in the fanout of net '<net-name>'
```

NOTE: *In case of RTL inferred logic, the cell name '<cell-name>' will be SpyGlass Synthesis generated name (RTL_* or M_RTL_*).*

Potential Issues

This violation appears if your design contains instances of disallowed cells in clock/reset networks.

Consequences of Not Fixing

Disallowed cells may result in timing issues on clock paths. It may also change polarity or add to delays.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, check how the invalid cell is connected in the specified network.
3. Open *The CKSGDCInfo Report* to see the list of inferred cells, when wildcard is specified.
4. You can also view case analysis settings along with the violation of this rule.

Example Code and/or Schematic

Consider the following example:

```
module test(clk1,clk2,q,d);
input clk1;
input clk2;
input [1:0] d;
output reg [1:0] q;
reg clock;
wire clockby2;
wire t1;
B1I B1(.A(clk1),.Z(tclk));
assign tclock = clock ^ clk1;
assign clockby2 = clock;
always@(posedge clk1)
  clock <= !clockby2;
always@(posedge tclk)
  q[0] <= d[0];
always@(negedge tclock)
  q[1] <= d[1];
endmodule
```

The *Clock_Reset_check01* rule reports a violation for the above example as an invalid cell `B1I` is present in the clock path, as shown in the following schematic:

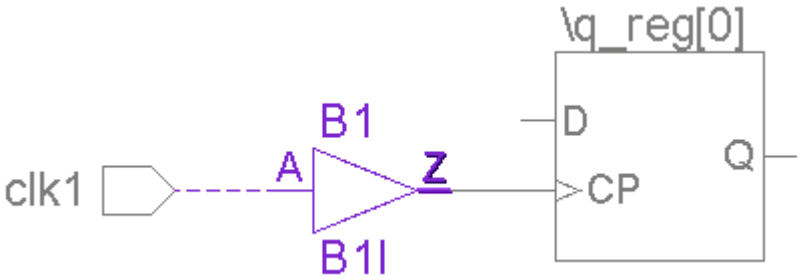


FIGURE 320. Schematic for the Clock_Reset_check01 Rule Violation

To fix the above violation, remove the `B1I` cell from the clock path.

Schematic Details

The *Clock_Reset_check01* rule highlights the cell that should not appear in the clock or reset network.

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

The CKSGDCInfo Report

Clock_Reset_check02

Reports potential race conditions between flip-flop output and its clock/reset pin

When to Use

Use this rule to detect race-condition between output of a flip-flop and clock/reset pins.

Description

The *Clock_Reset_check02* rule reports a violation if a feedback path exists between the output of a flip-flop to its clock/reset pins.

Parameter(s)

- *ignore_race_thru_latch*: Default value is `no`. Set this parameter to `yes` to ignore cases in which a latch exists in a feedback path between the output of a flip-flop pin and its clock pin.
- *report_user_defined_clock*: Default value is `no`. Set this parameter to `yes` to enable the enable the Clock_Reset_check02 rule to report the clock that is highlighted in the RTL viewer and schematic.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the first line of a design unit where a potential race condition is detected between the output of a flip-flop and its clock/preset/clear signal *<name>*:

```
[WARNING] Potential race between flop (output <obj-type>
<inst-name>) and its <sig-type> (<name>) detected
```

NOTE: *If there is a latch in the feedback path, this rule reports Message 2.*

The arguments of the above message are explained below:

Argument	Description
<sig-type>	Can be `clock`, `preset` or `clear`.
<obj-type>	`net` in case of RTL designs. `pin` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `net`.
<inst-name>	*<flop-output-net-name>* in case of RTL designs. *<flop-inst-name>.<out-pin-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs.

Potential Issues

This violation appears if a feedback loop exists in your design possibly through a combinational logic from the output pin of a flip-flop to its clock pin or asynchronous set/reset pin.

Consequences of Not Fixing

If you do not fix this violation, timing and behavior in such cases can be difficult to predict.

How to Debug and Fix

For information on debugging, click *How to Debug and Fix*.

Message 2

The following message appears at the first line of a design unit where a potential race condition is detected between the output of a flip-flop *<flop-name>* and its clock/preset/clear signal *<name>* and a latch is detected in the feedback path:

`[WARNING] Potential race between flop (output <obj-type> <inst-name>) and its <sig-type> (<name>) detected through a latch`

The arguments of the above message are explained below:

Argument	Description
<sig-type>	Can be `clock`, `preset` or `clear`
<obj-type>	`net` in case of RTL designs. `pin` in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is `net`.
<inst-name>	*<flop-output-net-name>* in case of RTL designs. *<flop-inst-name>.<out-pin-name>* in case of netlist designs, if the *report_inst_for_netlist* parameter is set to `yes`. Otherwise, it is same as in case of RTL designs.

Potential Issues

This violation appears if your design contains one or more latches in a feedback path.

Consequences of Not Fixing

If you do not fix this violation, timing and behavior in such cases can be difficult to predict.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, check the race in clock/preset/clear path.
3. Run the *Info_Case_Analysis* rule to check if mux-select/enable/control signals have been applied properly in the race path. Else, constraint them properly.

Example Code and/or Schematic

Example 1

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(q,d,clk,rst,sel,en);
input d;
input clk;
input rst;
input en;
input sel;
output q;
reg q,q1;
wire clear;
assign clear = (sel)?q1 :rst;
always@(en)
  if(en)
    q1 <= q;
always@(posedge clk or negedge clear)
begin
  if(!clear)
    q <= 0;
  else
    q <= d;
end
endmodule
```

// constraints.sgdc

```
current_design test
clock -name clk -domain d1
reset -name rst -value 0
set_case_analysis -name en -value 1
```

The *Clock_Reset_check02* rule reports a violation for the above example as a race condition exists between the output pin Q of a flip-flop and the clear pin CLR. This is shown in the following schematic:

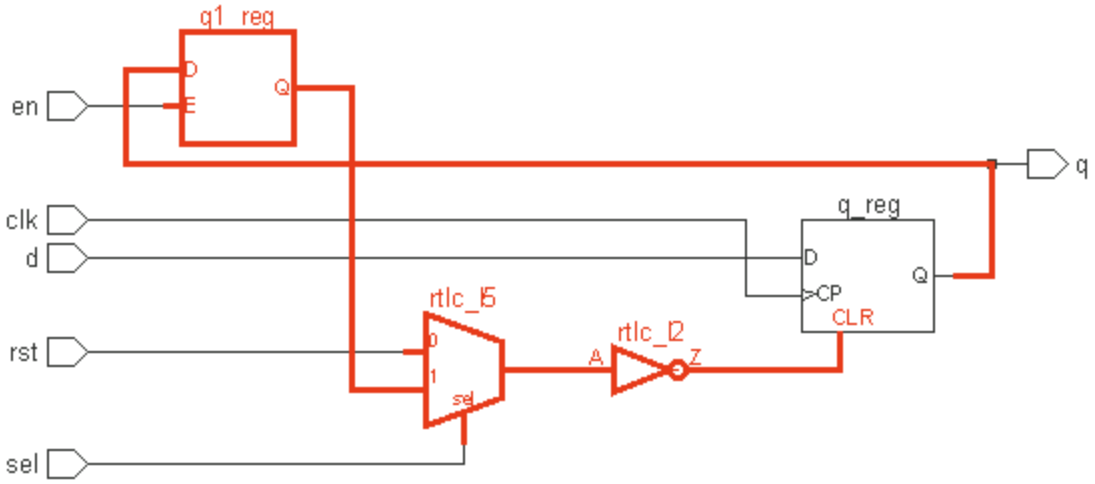


FIGURE 321. Schematic for the Clock_Reset_check02 Rule Violation

To resolve the race condition in the above example:

- Constraint the mux-select pin so that the output of the flip-flop is not feeding the clear path.
- If the latch present in the feedback path can resolve the race condition, set the *ignore_race_thru_latch* parameter.

Example 2

Consider the following example where a potential race condition exists between the output of flip-flop `q_temp` and its clock `clk_temp`:

```
architecture logic of test is
  signal q_temp : std_logic;
  signal clk_temp : std_logic;
    begin
      process(clk_temp)
        begin
          if (clk_temp'event and clk_temp='1') then
            q_temp <= d;
          end if;
      end process;
```

```
      q <= q_temp;
      clk_temp <= q_temp and clk;
end logic;
```

For this example, SpyGlass generates the following message:

`Potential race between flop (output pin q_temp) and its clock (clk_temp) detected`

Example 3

Consider the following example where a potential race condition exists between the output of flip-flop `q_temp` and its preset `rst_temp`:

```
architecture logic of test is
  signal rst_temp, q_temp, tmp : std_logic;
    begin
      process(clk, rst_temp)
        begin
          if (rst_temp='1') then
            q_temp <= '1';
          elseif (clk'event and clk='1') then
            q_temp <= d;
          end if;
       end process;

      tmp <= rst_temp and rst;
      rst_temp <= tmp and q_temp;
      q <= q_temp;
end logic;
```

For this example, SpyGlass generates the following message:

`Potential race between flop (output pin q_temp) and its preset (rst_temp) detected`

Example 4

Consider the following example where a potential race condition exists between the output of flip-flop `q_temp` and its clear `rst_temp`:

```
architecture logic of test is
  signal rst_temp, q_temp, tmp : std_logic;
    begin
      process(clk, rst_temp)
```

```
        begin
          if (rst_temp='1') then
            q_temp <= '0';
          elseif (clk'event and clk='1') then
            q_temp <= d;
          end if;
        end process;

      tmp <= rst_temp and rst;
      rst_temp <= tmp and q_temp;
      q <= q_temp;
end logic;
```

For this example, SpyGlass generates the following message:

`Potential race between flop (output pin q_temp) and its clear (rst_temp) detected`

Example 5

Consider the following example where a potential race condition exists between the output of flip-flop `q_temp` and its clock `clk_temp` and the feedback path contains a latch:

```
architecture logic of test is
  signal tmp, clk_temp, q_temp : std_logic;
  begin
    process(clk_temp)
      begin
        if (clk_temp'event and clk_temp='1') then
          q_temp <= d;
        end if;
    end process;
    process(en, q_temp)
      begin
        if (en='1') then
          tmp <= q_temp;
        end if;
    end process;

    clk_temp <= tmp and clk;
```

```
    q <= q_temp;
end logic;
```

For this example, SpyGlass generates the following message:

```
Potential race between flop (output pin q_temp) and its clock
(clk_temp) detected through a latch
```

Schematic Details

The *Clock_Reset_check02* rule highlights the feedback loop from flip-flop output to its clock/preset/clear pin.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and/or Related Files

No report or related file

Clock_Reset_check03

Reports potential race condition between flip-flop clock and reset pins

When to Use

Use this rule to detect race-conditions between clock and reset pins of a flip-flop.

Description

The *Clock_Reset_check03* rule reports potential race conditions between clock signals and reset/set signals of a flip-flop.

Race conditions occasionally occur when clock and reset signals of a flip-flop come into conflict. In such cases, gate output takes a finite, non-zero amount of time to react to any change in inputs. As a result, simultaneous propagation of clock and reset signals produce undesired outputs.

Parameter(s)

None

Constraint(s)

set_case_analysis (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears at the location where flip-flop *<flop-name>* is first set when there is a potential race condition between its clock signal and reset/preset signal:

```
[WARNING] Potential race between clock and reset pins of flop
<flop-name>
```

NOTE: *For RTL designs, <flop-name> is the name of the output net of the flip-flop. For netlist designs, if the report_inst_for_netlist parameter is set to yes, <flop-name> is the name of the flip-flop instance. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if your design contains a net that drives a clock pin as well as a reset/set pin of a flip-flop, possibly through some combinational logic.

Consequences of Not Fixing

If you do not fix this violation, timing and therefore behavior may be difficult to predict.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. View the *Incremental Schematic* of the violation message.
2. In the schematic, check the race between clock and reset/preset pins.
3. Run the *Info_Case_Analysis* rule to check if control signals have been applied properly in the race path. Else, constraint them properly.

Example Code and/or Schematic

Consider the following example:

```
module test(q,d,clk,in1,pre,enable,sel);
input d;
input clk;
input in1;
input enable,sel,pre;
output q;
reg q;
wire reset;
wire temp;
wire tclk;
assign temp = in1 & (!sel);
assign tclk = clk & enable;
assign reset = (sel)?temp : pre & enable;
always@(posedge tclk or posedge reset)
  begin
    if(reset)
      q <= 1'b1;
    else
      q <= d;
  end
```

```
endmodule
```

For the above example, the *Clock_Reset_check03* rule reports a violation as race condition exists between clock and reset pins of the `q` flip-flop.

The schematic of this violation is shown below:

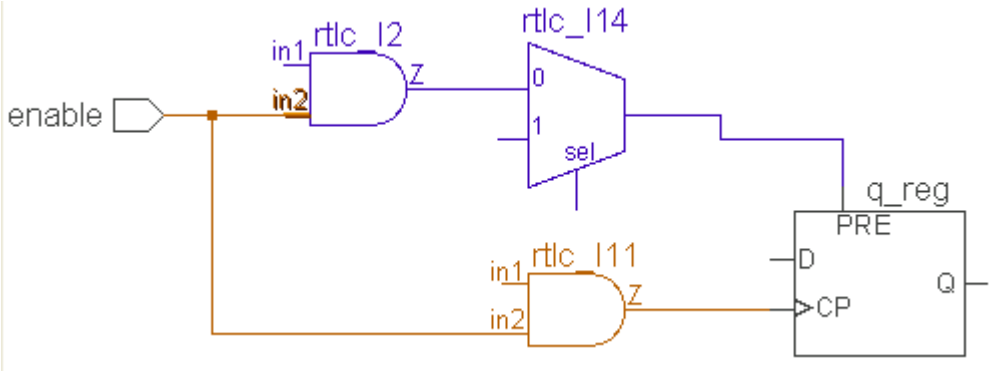


FIGURE 322. Schematic for the Clock_Reset_check03 Rule Violation

In the above schematic, the same `enable` net is reaching to the clock as well as the preset path.

To fix this violation, constraint the MUX select signal to activate proper paths by using the *set_case_analysis* constraint, as shown below:

```
set_case_analysis -name sel -value 1
```

Schematic Details

The *Clock_Reset_check03* rule highlights the following details in a schematic:

- Net driving a clock pin as well as a reset/set pin of a flip-flop.
- Path from a net to a clock pin.
- Path from a net to a reset/set pin.

Default Severity Label

Warning

Rule Group

VERIFY

Report and Related File

No related reports or files

Delta Delay Rules

The SpyGlass Delta Delay rules are as follows:

Rule	Reports
Clock_delay01	Flip-flop pairs triggered by the same clock where the number of cells in the data path to the destination flip-flop is more than the net difference in number of cells in the clock paths of the two flip-flops
Clock_delay02	Clocks where the delay (number of cells) is not same for all paths from clock source to each flip-flop triggered by the clock
DeltaDelay01	Flip-flops/latches that have a different delta clock delay value
DeltaDelay02	Flip-flops that can cause simulation problems due to delta delay issues
NoClockCell	Logic found in clock trees
PortTimeDelay	Ports with missing or unexpected time delay settings in assignments

In addition, a number of *Must Rules* are automatically run to check the validity of user-specified SpyGlass Design Constraints.

Clock_delay01

Reports flip-flop pairs whose data path delta delay is less than the difference in their clock path delta delays

When to Use

Use this rule to detect a simulation mismatch in clock and data paths for the given flip-flop pair.

Prerequisites

Specify clock names to be checked by using the *delay_check_clk_list* parameter.

Description

The *Clock_delay01* rule reports flip-flop pairs triggered by the same clock based on the following equation:

$$N_{ddp} < N_{dcp} - N_{scp}$$

Where:

- N_{ddp} is the number of cell instances in the data path from the source flip-flop to the destination flip-flop.
- N_{dcp} is the number of cell instances in the path from the clock source to the destination flip-flop's clock pin.
- N_{scp} is the number of cell instances in the path from the clock source to the source flip-flop's clock pin.

NOTE: *The Clock_delay01 rule is switched off by default.*

Parameter(s)

- *delay_check_clk_list*: Default value is `all`. Set this parameter to a comma or space-separated list of clock source names.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears at the location where the output of the destination flip-flop (triggered by clock *<clk-name>*) is first assigned:

```
[ClkD1_1] [WARNING] Simulation mismatch for destination
register '<flop2-name>' and source register '<flop1-name>',
clocked by '<clk-name>'
```

NOTE: *For RTL designs, <flop1-name> and <flop2-name> are names of the output nets of the source and destination flip-flops. For netlist designs, if the report_inst_for_netlist parameter is set to yes, <flop1-name> and <flop2-name> are names of the source and destination flip-flop instances. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears if delay in a data path is less than the effective delay in the clock path.

Consequences of Not Fixing

If you do not fix this violation, it may result in a possible hold violation in the design.

How to Debug and Fix

To fix this violation, perform the following actions:

1. View the incremental schematic of the violation to analyze the flip-flop pair delay in the clock and data path.
2. Based on your analysis, perform any of the following actions:
   - ❒ Reduce the clock skew.
   - ❒ Add some delay in the data path.

Message 2

The following message appears when an invalid (non existent or non clock) object *<name>* is specified with the `delay_check_clk_list` parameter:

```
[ClkD1_2] [WARNING] Invalid clock '<name>' specified in
parameter 'delay_check_clk_list'
```

Potential Issues

This violation appears if an invalid object is specified with the *delay_check_clk_list* parameter.

An object is considered as invalid in any of the following cases:

- If the object does not exist in the current design
- If the object is not inferred as a clock

Consequences of Not Fixing

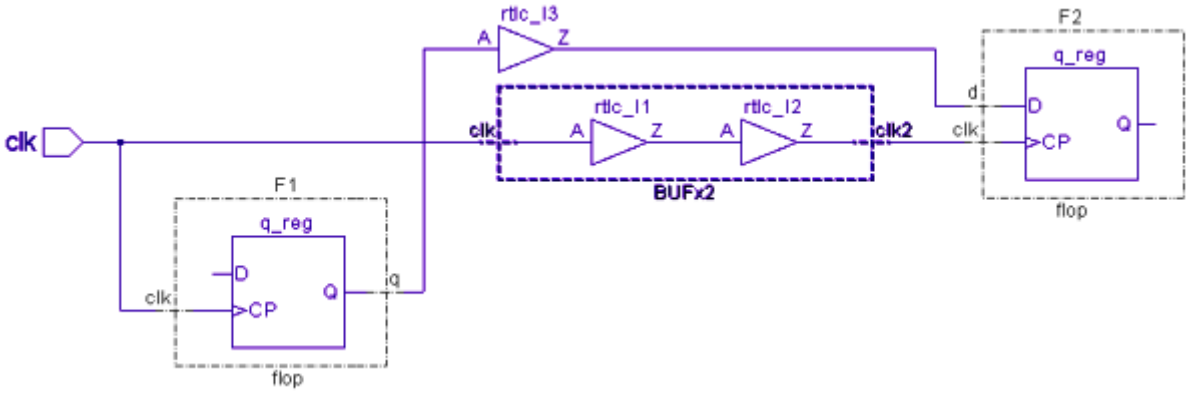
If you do not fix this violation, the *delay_check_clk_list* parameter is not considered during SpyGlass analysis, which may not be as per your expectation.

How to Debug and Fix

To fix this violation, specify a valid clock name with the *delay_check_clk_list* parameter.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
set_parameter report_inst_for_netlist set yes
set_parameter delay_check_clk_list clk
```

FIGURE 323. Schematic of the Clock_delay01 Rule Violation

In the above example, the delay between the data path of the F1 and F2 flip-flops is lesser than the clock path delay.

Therefore, the *Clock_delay01* rule reports a violation.

Schematic Highlight

The *Clock_delay01* rule highlights the following information in the schematic:

- The path from the source flip-flop to the destination flip-flop
- The path from the clock source to the clock pin of each flip-flop

Default Severity Label

Warning

Rule Group

DELTADELAY

Reports and Related Files

No report or related file

Clock_delay02

Reports unbalanced clock trees

When to Use

Use this rule to check for unbalanced clock trees.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to generate clock signals automatically
- Combination of both the above methods

Description

The *Clock_delay02* rule checks clocks propagating with a different number of cell instances. That is, it checks for delays in different flip-flop paths from the same clock source.

This rule reports the flip-flop with a maximum delay.

Parameter(s)

- *cdc_express*: Default values is `no`. Set this parameter to `peakmem` to reduce peak memory. Other possible value is `yes`.
- *report_inst_for_netlist*: Default value is no. Set this parameter to yes to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears for the clock *<clk-name>* that has an imbalanced clock tree:

```
[WARNING] Clock Tree '<clk-name>' is unbalanced - Maximum
delta-delay imbalance is '<num>' at '<flop-name>'
```

NOTE: *For RTL designs, <flop-name> is the name of the output net of the flip-flop with maximum delay <num>. For netlist designs, if the report_inst_for_netlist parameter is set to yes, <flop-name> is the name of the flip-flop instance with maximum delay <num>. Otherwise, the message details are same as for the RTL designs.*

Potential Issues

This violation appears when a clock signal propagates through different number of instances in the clock path of different flip-flops.

Consequences of Not Fixing

If you do not fix this violation, the design may contain unbalanced clock trees. This may generate incorrect simulation results.

How to Debug and Fix

To debug and fix this violation, perform the following actions:

- Check the maximum clock delay path in the incremental schematic.
- Ensure that the propagation delay on the clock path of each flip-flop is within the acceptable minimum and maximum delay range.

Example code and/or Schematic

Consider the following schematic of a violation of this rule:

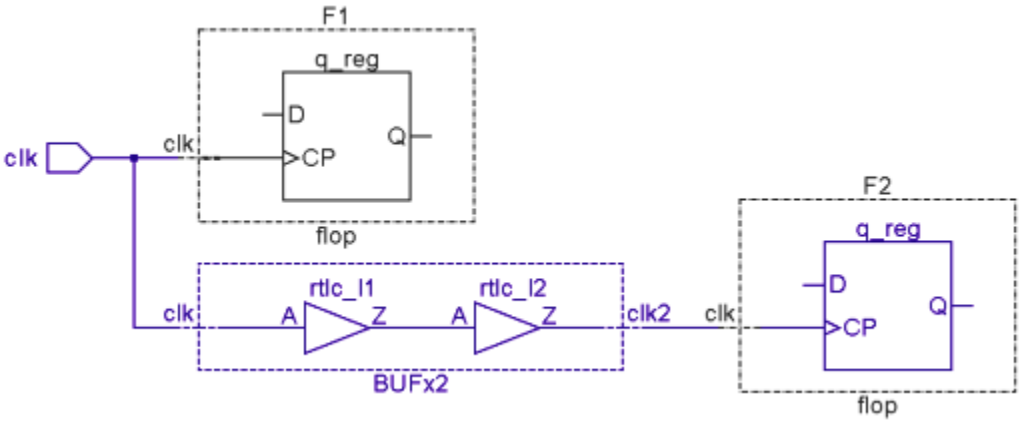


FIGURE 324. Schematic of the Clock_delay02 Rule Violation

In the above schematic, the clock signal `clk` propagates through different number of buffers in the clock path of the `F1` and `F2` flip-flops.

Therefore, the *Clock_delay02* rule reports a violation for the `F2.q` flip-flop.

Schematic Details

This rule highlights the path from the clock source to the flip-flop with maximum delay.

Default Severity Label

Warning

Rule Group

`DELTADELAY`

Reports and Related Files

No report or related file

DeltaDelay01

Flags flip-flops/latches, which may have different delta clock delay values

When to Use

Use this rule to check clock skew issues in a clock network.

Prerequisites

Specify the following details before running this rule:

- A *Simulator File* containing simulator-specific delta delay information for RTL constructs by using the *simulator_file_name* parameter.

  You can use the *Sample Simulator File* present in the installation directory.
- Clock signals in any of the following ways:
  - By using the *clock* constraint
  - By setting the *use_inferred_clocks* parameter to `yes` to generate clock signals automatically
  - Combination of both the above methods
- The `Advanced_CDC` and `adv_checker` licenses.

Description

The *DeltaDelay01* rule reports flip-flops/latches that have different delta clock-delay values.

You can specify a *Simulator File* containing delta delay information.

NOTE: *The DeltaDelay01 rule is switched off by default.*

NOTE: *Clock traversal automatically stops at black box instances.*

Handling Constructs Other Than Those Specified in a Simulator File

For the constructs other than those specified in the *Simulator File*, the delta delay value is assumed zero. In this case, the *DeltaDelay01* rule reports all the flip-flops/latches in the clock tree if one or more flip-flops/latches have a different delta clock delay value.

NOTE: *When the design save/restore feature is enabled, the DeltaDelay01 rule does not consider changes in the simulator file. If delay values are changed, the values should be saved first and then restored*

Rule Exceptions

The *DeltaDelay01* rule does not report a violation if the delta clock delay value is zero for a flip-flop/latch and the expected value is not specified for the *deltacheck_start* constraint.

Parameter(s)

- *simulator_file_name*: Default value is `NULL`. Specify a *Simulator File* that contains simulator-specific delta delay information for RTL constructs.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report the violating instance name in case of netlist designs and a leaf-level net name for RTL designs.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *deltacheck_start* (Optional): Use this constraint to specify start points, such as clock ports, clock pins, or clock nets for *DeltaDelay01* rule checking.
- *deltacheck_stop_signal* (Optional): Use this constraint to specify design points, such as ports, pins, or nets where the *DeltaDelay01* rule should stop further traversal.
- *deltacheck_stop_module* (Optional): Use this constraint to specify design units where the *DeltaDelay01* rule should stop further traversal along the clock tree.
- *deltacheck_stop_instance* (Optional): Use this constraint to specify instances where the *DeltaDelay01* rule should stop further traversal along the clock tree.
- *deltacheck_ignore_module* (Optional): Use this constraint to specify design units to be ignored for delta delay value checking.
- *deltacheck_ignore_instance* (Optional): Use this constraint to specify instances to be ignored for delta delay value checking.

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears when a cone of the clock *<clk-name>* triggers *<num>* flip-flops/latches that have a different delta clock delay value *<value>*:

[DD1_1] [WARNING] For clock '<clk-name>', <num> flops/latches have a delay value <value>

Potential Issues

This violation appears if your design contains a clock that has different delay values in different paths.

Consequences of Not Fixing

If you do not fix this violation, data is passed from a shorter delay path to a larger delayed path. This happens when a clock has different delay values in different paths.

This generates simulation errors, both at the top-level and module-level, and it is difficult to identify the cause of such problems in simulation.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

- View the incremental schematic of the violation message and review the delta delay information displayed.

  Ensure that propagation delay along each clock path meets the minimum and maximum delay requirements as per the specifications.
- If the delay is not specified correctly for an RTL construct, make the required modifications in the simulator file specified by the *simulator_file_name* parameter.
- Check if the expected delay value for a clock specified in the *deltacheck_start* constraint is correct.

- View *The DeltaDelay-Summary Report*, *The DeltaDelay-Concise Report*, and *The DeltaDelay-Detailed Report* for details.

Message 2

The following message appears when a cone of the clock *<clk-name>* triggers one flip-flop/latch that has a different delta clock delay value *<value>*:

```
[DD1_2] [WARNING] For clock '<clk-name>', 1 flop/latch has a
delay value <value>
```

Potential Issues

See *Potential Issues*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Example Code and/or Schematic

Consider a design for which the following *deltacheck_start* constraint is set:

```
deltacheck_start -name top.clk1 -value +3
```

In addition, consider that the file specified by the *simulator_file_name* parameter contains the following data:

```
VHDL:signal_assignment_01:1
VHDL:signal_assignment_02:2
Verilog:port_connection_04:3
```

In this case, the *DeltaDelay01* rule reports flip-flops in the design, as shown in the following figure:

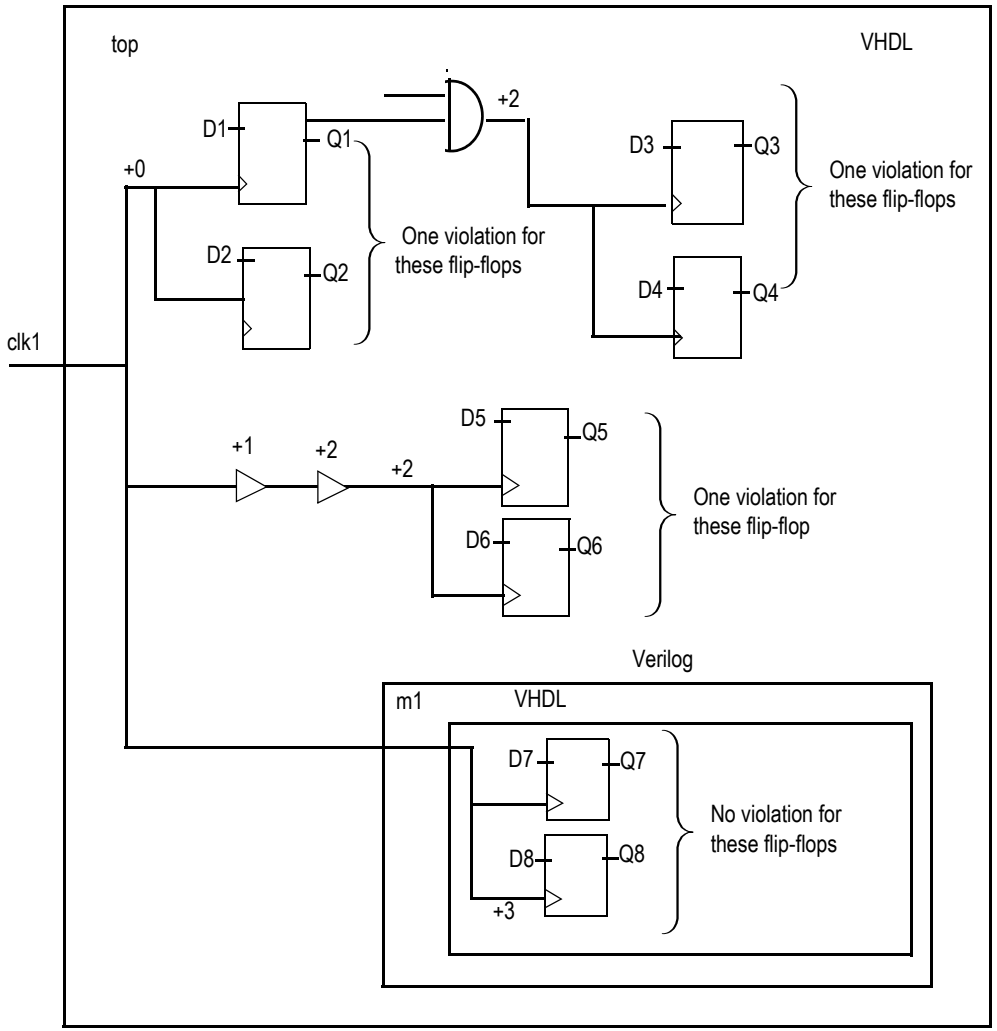


FIGURE 325. DeltaDelay01 Rule Violation

Schematic Highlight

The *DeltaDelay01* rule highlights a single flip-flop/latch at each clock cone in the path of the clock in the schematic.

The schematic uses the following naming conventions:

Notation Symbol	Represents
F	Flop
L	Latch

NOTE: *Only a single flip-flop/latch is highlighted at each clock cone of the clock in the schematic. In addition, the schematic displays the total number of flip-flops/latches violating at the same clock cone with the same delta clock delay value.*

Default Severity Label

Warning

Rule Group

DELTADELAY

Reports and/or Related Files

- *The DeltaDelay-Summary Report*
- *The DeltaDelay-Concise Report*
- *The DeltaDelay-Detailed Report*

DeltaDelay02

Reports flip-flops that can cause simulation problems due to delta delay issues

When to Use

Use this rule to check flip-flops that can cause simulation problems due to delta delay issues.

Prerequisites

Specify the following details before running this rule:

- A simulator mode file containing simulator-specific delta delay information for RTL constructs by using the *simulator_file_name* parameter.

NOTE: *A sample simulator file,* simulator_file.txt*, is present in the* SPYGLASS_HOME/policies/clock *directory. You can directly pass this file to the* `simulator_file_name` *parameter or modify it to specify delay values for different simulators.*

NOTE: *When the design save/restore feature is enabled, the DeltaDelay02 rule does not consider changes in the simulator mode file. If delay values are changed, the values should be saved first and then restored.*

- Clock signals in any of the following ways:
  - By using the *clock* constraint
  - By setting the *use_inferred_clocks* parameter to `yes` to generate clock signals automatically
  - Combination of both the above methods
- The `Advanced_CDC` and `adv_checker` licenses

Description

The *DeltaDelay02* rule reports the following:

- A synchronous data path when both the following conditions are true:
  - If the delay in clock path of source flip-flop is less than the delay in destination clock path
  - If an explicit physical delay statement (after `clause` in VHDL or # in Verilog) is not present in source flip-flop assignment

The following figure shows the scenario in which this rule reports a violation:

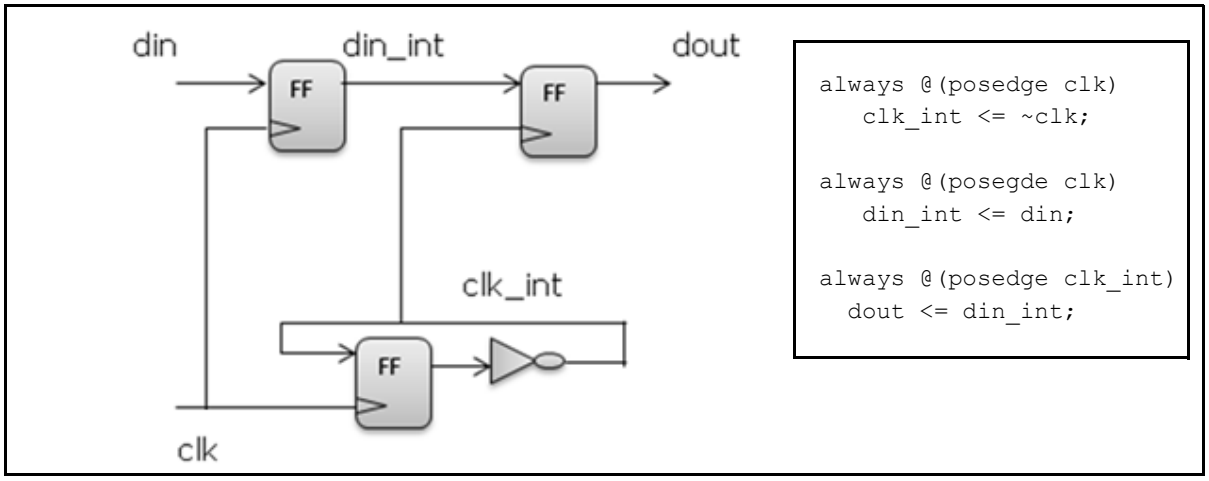


FIGURE 326. DeltaDelay02 Violation - Synchronous Data Path

In the above scenario, delay in the clock path of the source flip-flop is less than the delay in the clock path of the destination flip-flop. In addition, an explicit delay statement is not provided at the destination flip-flop. Therefore, this rule reports *Message 1* in this case.

- A derived clock if a flip-flop used in the derived clock has an explicit delay assignment.

  The following figure shows the scenario in which this rule reports a violation:

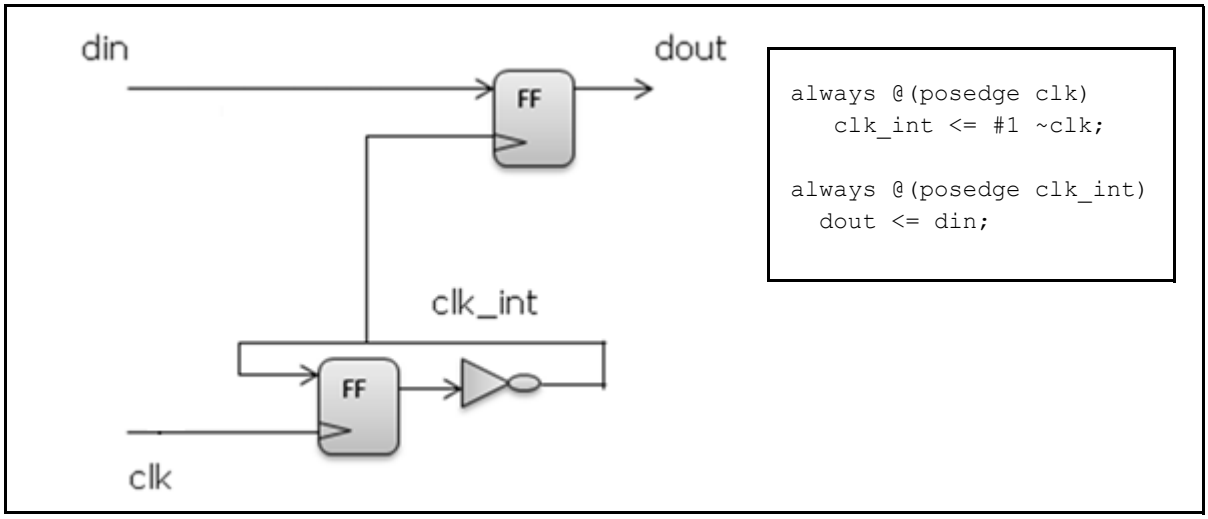


FIGURE 327. DeltaDelay02 Violation - Explicit Delay Assignment for Derived Clock

In the above scenario, an explicit physical delay assignment is provided in the derived clock `clk_int`. Therefore, this rule reports *Message 2* in this case.

This rule stops clock traversal at a black box.

NOTE: *The DeltaDelay02 rule is switched off by default.*

Messages in Schematic

After computing delay, this rule shows any of the following messages in the schematic:

- *Higher Delay*

  When the delay at destination is greater than the delay at source.

- *Different Cells*

  When different library cells appear in the paths towards source and destination, as shown in the following figure:

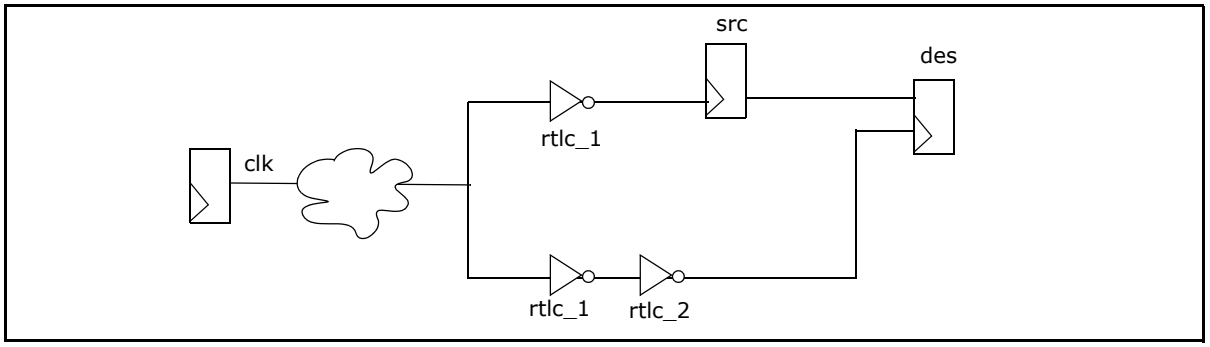


FIGURE 328. Different library cells in the paths of source and destination

- *Higher Cell Count*

  When the same library cells appear in the paths towards source and destination, but the number of cells is greater on the destination path, as shown in the following figure:

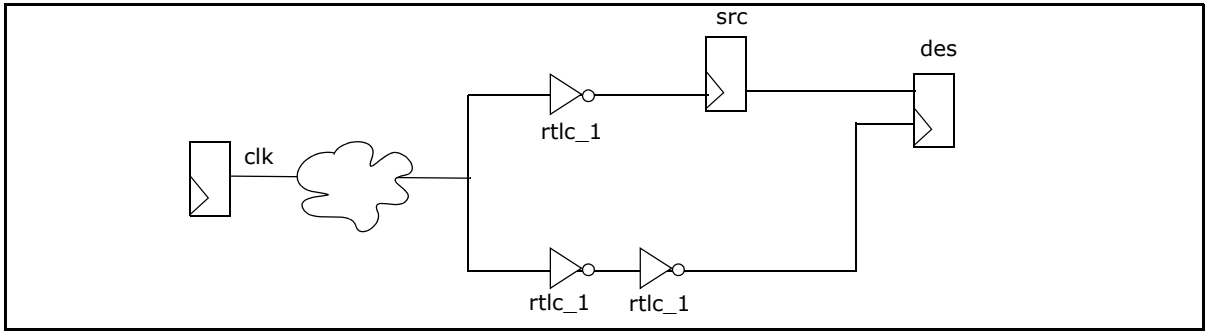


FIGURE 329. Different number of same library cells in the paths of source and destination

Parameter(s)

- *cdc_bus_compress*: Default value is `Ac_glitch03`. Set this parameter to `DeltaDelay02` to check all the bits of the source bus by the *DeltaDelay02* rule. For information on the other possible values, see *Possible values of the cdc_bus_compress parameter*.

- *simulator_file_name*: Default value is `NULL`. Specify a simulator mode file that contains simulator-specific delta delay information for RTL constructs.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report the violating instance name in case of netlist designs and a leaf-level net name for RTL designs.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *deltacheck_start* (Optional): Use this constraint to specify start points, such as clock ports, clock pins, or clock nets for *DeltaDelay02* rule checking.
- *deltacheck_stop_signal* (Optional): Use this constraint to specify design points, such as ports, pins, or nets where the *DeltaDelay02* rule should stop further traversal.
- *deltacheck_stop_module* (Optional): Use this constraint to specify design units where the *DeltaDelay02* rule should stop further traversal along the clock tree.
- *deltacheck_stop_instance* (Optional): Use this constraint to specify instances where the *DeltaDelay02* rule should stop further traversal along the clock tree.
- *deltacheck_ignore_module* (Optional): Use this constraint to specify design units to be ignored for delta delay value checking.
- *deltacheck_ignore_instance* (Optional): Use this constraint to specify instances to be ignored for delta delay value checking.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears if the *<num-flops>* source flip-flops do

not have explicit delay to avoid simulation problems:

[DD2_1] [ERROR] <num-flops> flop(s) do not have explicit delay to avoid simulation problem (delta delay issue relative to clock '<clk-name>')

Where:

- *<Num-flops>* is the count of the source flip-flops that do not have explicit physical delay to avoid simulation problem.
- *<clk-name>* is the clock that is causing the delta delay issue.

Potential Issues

This violation appears if your design contains flip-flops that do not have explicit delay to avoid simulation problems.

Consequences of Not Fixing

If you do not fix this violation, the specified flip-flops can cause simulation problems due to delta delay issues.

How to Debug and Fix

The rule reports one violation per source clock and a spreadsheet is associated with each message. The spreadsheet contains all the source flip-flops that satisfy the conditions mentioned above. Each row in spreadsheet highlights data path and source and destination clock paths with delay values annotated. See *DeltaDelay02 Spreadsheet - Synchronous Data Path*.

To fix this violation, ensure that the source flip-flop in the data path has explicit physical delay to avoid simulation problems.

Message 2

The following message appears if the *<der-clk-name>* derived clock has explicit delay:

[DD2_3] [ERROR] Explicit delay on derived clock '<der-clk-name>' must be removed to avoid simulation problems (delta-delay relative to clock '<clk-name>'

Where *<clk-name>* is the clock that is causing the delta delay issue.

Potential Issues

This violation appears if there is an explicit delay on the specified derived clock.

Consequences of Not Fixing

If you do not fix this violation, the specified flip-flops can cause simulation problems due to delta delay issues.

How to Debug and Fix

To debug this violation, open the incremental schematic. It highlights the path from derived clock to the clock pin of a flip-flop.

To fix this violation, ensure that the flip-flops used in the derived clocks do not have explicit physical delay.

Example Code and/or Schematic

Consider the following files (in addition to a simulation file) specified for

SpyGlass analysis:

```
// test.v

module test (in, clk, out);
  input in, clk;
  output out;

  reg der_clk_1;
  always @(posedge clk)
      der_clk_1 <= ~der_clk_1;

  reg der_clk_2;
  always @(posedge clk)
      der_clk_2 <= #1 ~der_clk_2;

  reg src1;
  always @(posedge clk)
      src1 <= in;

  reg src2;
  always @(posedge der_clk_1)
      src2 <= in;

  reg des1;
  always @(posedge der_clk_1)
      des1 <= src1;

  reg des2;
  always @(posedge der_clk_2)
      des2 <= src2;
endmodule
```

```
// test.sgdc

current_design test
clock -name clk
```

For the above example, the *DeltaDelay02* rule reports the following two violations:

- Explicit delay on derived clock 'test.der_clk_2' must be removed to avoid simulation problems (delta-delay relative to clock 'test.clk')

  The highlighted portion in the following schematic shows this violation:

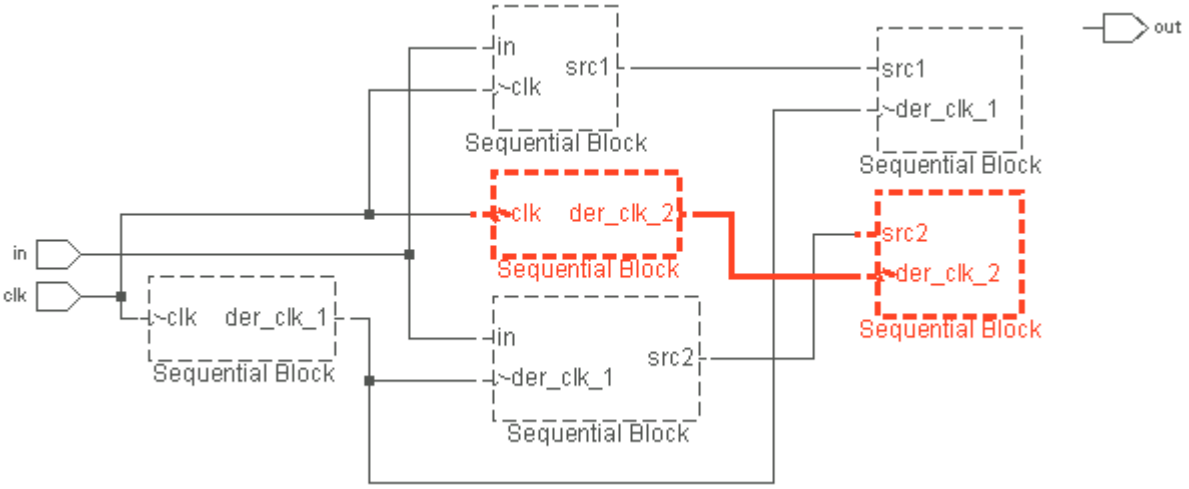


FIGURE 330. DeltaDelay02 Example - Explicit Delay Assignment for Derived Clock

- 1 flop(s) do not have explicit delay to avoid simulation problem (delta delay issue relative to clock 'test.clk')

  When you double-click on this violation, the following spreadsheet appears:

A	B	C
ID	Source Flop	Number of Occurrences
1	test.src1	1

FIGURE 331. DeltaDelay02 Spreadsheet - Synchronous Data Path

The following figure shows the schematic of this violation:

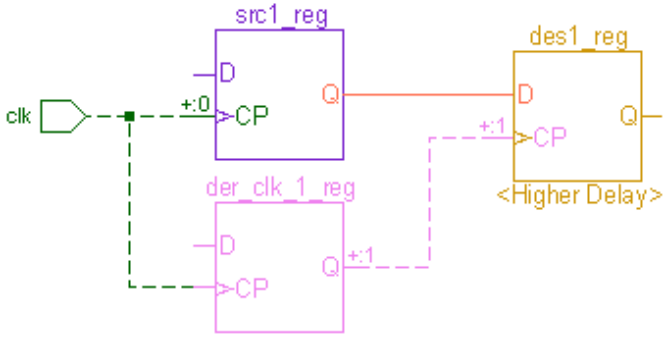


FIGURE 332. DeltaDelay02 Example - Synchronous Data Path

Default Severity Label

Error

Rule Group

DELTADELAY

Reports and/or Related Files

The DeltaDelay02-Detailed Report

NoClockCell

Reports any logic found in clock trees

When to Use

Use this rule to detect any logic present in clock trees.

Prerequisites

Specify the following information before running this rule:

- Use the `Advanced_CDC` and `adv_checker` license features.
- Specify the *noclockcell_start* constraint.

Description

The *NoClockCell* rule reports a violation if any logic is present in the path of clock ports, nets, or pins specified by using the *noclockcell_start* constraint.

This rule stops clock traversal automatically at black box instances except for instances of single input and single output black box design units.

NOTE: *The NoClockCell rule is switched off by default.*

Rule Exceptions

If the names specified by the *noclockcell_start* constraint are not valid clock signals, this rule does not check for such signals.

Parameter(s)

- *allow_vhdl_on_clock_path*: Default value is `no`. Set this parameter to `yes` to report the use of Verilog constructs in clock trees.
- *report_inst_for_netlist*: Default value is `no`. Set this parameter to `yes` to report violating instance name in case of netlist designs and leaf-level net name for RTL designs.

Constraint(s)

- *noclockcell_start* (Mandatory): Use this constraint to specify start points, such as ports or nets for rule-checking.

- *noclockcell_stop_signal* (Optional): Use this constraint to specify design points, such as ports, pins, or nets where the *NoClockCell* rule should stop further traversal along the clock tree.
- *noclockcell_stop_module* (Optional): Use this constraint to specify a 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.
- *noclockcell_stop_instance* (Optional): Use this constraint to specify an instance where the *NoClockCell* rule should stop further traversal along the clock tree when the clock pin of the specified instance is hit.

Messages and Suggested Fix

Message 1

The following message appears when the cell instance *<inst-name>* of the type *<gate-type>* is found in the path of the clock signal *<clk-name>*:

```
[WARNING] Gate '<inst-name> (gate-type)' found in path of clock
'<clk-name>'
```

NOTE: *If an instance is internally generated, Message 2 is reported.*

Potential Issues

This violation appears if a cell is present in the path of a clock.

Consequences of Not Fixing

It is design methodology choice. If you do not want to check for such cases, disable or waive this rule.

How to Debug and Fix

To fix this violation, check the clock path from the schematic of the violation message, and remove the cells present in the clock path.

Message 2

The following message appears when an instance is internally generated

(RTL designs). This message is reported on the output net of the connected instance <*net-name*>:

[WARNING] Gate output'<net-name>' found in path of clock '<clk-name>'

Potential Issues

This violation appears if some logic is present in the path of a clock.

Consequences of Not Fixing

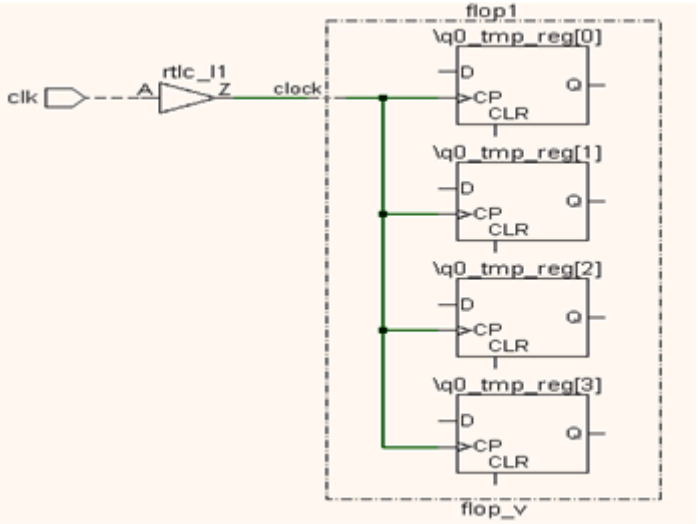
It is design methodology choice. If you do not want to check for such cases, disable or waive this rule.

How to Debug and Fix

Check the clock path from the schematic of the violation message, and remove the logic present in the clock path.

Example Code and/or Schematic

Consider the following schematic:



//top.sgdc

```
current_design top
clock -name clk -domain d1
noclockcell_start -name clk
```

FIGURE 333. Schematic of the NoClockCell Rule Violation

For the above example, the *NoClockCell* rule reports a violation for the `clk` clock because a buffer exists between this clock and flip-flops.

Schematic Highlight

The *NoClockCell* rule highlights the path from the clock source to the instance found in its path.

Default Severity Label

Warning

Rule Group

DELTADELAY

Reports and Related Files

The NoClockCell-Summary Report

PortTimeDelay

Reports ports with missing or unexpected time delay settings

When to Use

Use this rule to check unexpected time delays for entity ports.

Prerequisites

Specify the following details before running this rule:

- Specify the design units to be checked by using the *port_time_delay* constraint.
- Specify the `Advanced_CDC` and `adv_checker` license.

Description

The *PortTimeDelay* rule reports ports that have missing or unexpected time delay settings in assignments.

NOTE: *Please note the following points about the PortTimeDelay rule:*

- *This rule is switched off by default.*
- *This rule is applicable for only VHDL and mixed mode. It is not applicable to Verilog modules.*

Understanding Internal and External Mode Delay Assignments

The following points describe the internal and external mode delay assignments:

- In an internal mode delay assignment, a port of a design unit is assigned a time delay value within that design unit.
- In an external mode delay assignment, the signal connected to the port of a design unit is assigned a time delay value.

Considering Cases in which the PortTimeDelay Rule Reports a Violation

The *PortTimeDelay* rule reports a violation for the following types of ports:

- Ports that are specified by the `-notimedelay_ports` argument, and such ports are assigned a delay value in the internal and/or external mode.

- Ports that are not specified by the `-notimedelay_ports` argument of the *port_time_delay* constraint, and any of the following conditions hold true:
  - Ports are assigned a delay value in both internal and external modes
  - Ports are neither assigned a delay value in the internal mode nor in the external mode

To make an IP/module insensitive to the variation of delta-time on a clock path, a time delay should be added on all IO pads to the IP/modules with respect to the clock.

Messages and Suggested Fix

Message 1

The following message appears when the port *<port-name>* of the instance *<inst-name>* is not specified by the `-notimedelay_ports` argument of the *port_time_delay* constraint, but the port is assigned a delay value in both internal and external mode:

```
[WARNING] <Input | Output> port '<port-name>' of instance
<inst-name> has both internal and external mode delay
assignments
```

Potential Issues

This violation appears when a design instance contains a port that is assigned a delay value in both internal and external mode.

Consequences of Not Fixing

If you do not fix this violation, the design module becomes sensitive to variation of delay values on a clock path.

As a result, during simulation, value assignment to the port may happen in a different simulation time as compared to the change in the clock.

How to Debug and Fix

To fix this issue, assign a time delay value to the reported port either in an internal mode or an external mode.

Message 2

The following message appears when the port *<port-name>* is not specified by the `-notimedelay_ports` argument of the *port_time_delay* constraint, and the port is not assigned a delay value in an internal mode or an external mode with respect to any instance of the master design unit:

```
[WARNING] <Input | Output> port '<port-name>' of instance
<inst-name> has no delay assignment
```

Potential Issues

This violation appears when a design instance contains a port that is not assigned a delay value in either an internal or an external mode.

Consequences of Not Fixing

If you do not fix this violation, the design module becomes sensitive to variation of delay values on a clock path.

As a result, during simulation, value assignment to the port may happen in a different simulation time as compared to the change in the clock.

How to Debug and Fix

To fix this issue, specify appropriate time delay settings for the reported port. That is, assign a time delay value to the reported port either in an internal mode or an external mode.

Message 3

The following message appears when the port *<port-name>* is not specified by the `-notimedelay_ports` argument of the *port_time_delay* constraint, and the port is not assigned in an internal mode inside the master design unit *<du-name>* and it is also not assigned in an external mode with respect to all instances of the master design unit:

```
[WARNING] <Input | Output> port '<port-name>' of module
'<du-name>' has no delay assignment
```

Potential Issues

This violation appears if a design module contains a port for which no delay value is assigned in internal or external mode.

Consequences of Not Fixing

If you do not fix this violation, the design module becomes sensitive to variation of delay values on a clock path.

As a result, during simulation, value assignment to the port may happen in a different simulation time as compared to the change in the clock.

How to Debug and Fix

To fix this issue, specify appropriate time delay settings for the reported port. That is, assign a time delay value to the reported port either in an internal mode or an external mode.

Message 4

The following message appears when the port *<port-name>* specified by the `-notimedelay_ports` argument of the *port_time_delay* constraint is assigned a delay value in an external mode with respect to the instance *<inst-name>*:

```
[WARNING] <Input | Output> port '<port-name>' of instance
<inst-name> has a delay assignment
```

Potential Issues

This violation appears when a design instance contains a port that is assigned a delay value in an external mode.

Consequences of Not Fixing

If you do not fix this violation, the design module becomes sensitive to variation of delay values on a clock path.

As a result, during simulation, value assignment to the port may happen in

a different simulation time as compared to the change in the clock.

How to Debug and Fix

To fix this issue, remove the delay value assigned in an external mode from the reported port.

Message 5

The following message appears when the port *<port-name>* specified by the `-notimedelay_ports` argument of the *port_time_delay* constraint is assigned a delay value in an internal mode with respect to all instances of the master design unit *<du-name>*:

```
[WARNING] <Input | Output> port '<port-name>' of module
'<du-name>' has a delay assignment
```

Potential Issues

This violation appears if a design module contains a port that is assigned a delay value in an internal mode.

Consequences of Not Fixing

If you do not fix this violation, the design module becomes sensitive to variation of delay values on a clock path.

As a result, during simulation, value assignment to the port may happen in a different simulation time as compared to the change in the clock.

How to Debug and Fix

To fix this issue, remove the delay value assigned in an internal mode from the reported port.

Example Code and/or Schematic

Consider the following example:

```
ENTITY top IS
PORT(in_NOdelay, in2, clk1, clk2 : IN bit;
```

```
     out_delay, out_NOdelay: OUT bit);
END top;

ARCHITECTURE rtl OF top IS
COMPONENT ent IS
   PORT(in_NOdelay, in_delay, clk1, clk2 : IN bit;
        out_delay, out_NOdelay : OUT bit);
END COMPONENT;

SIGNAL w_delay: BIT;
BEGIN
w_delay <= in2 AFTER 1 ns; -- External mode
U1 : ent PORT MAP(in_NOdelay => in_NOdelay,
                  in_delay => w_delay,
                  clk1 => clk1,
                  clk2 => clk2,
                  out_NOdelay => out_NOdelay,
                  out_delay => out_delay);

END rtl;
ENTITY ent IS
    PORT(in_NOdelay, -- Violation
         in_delay, -- delay specified in assignment to
                   -- connected net, w_delay
         clk1, -- Ignored in port_time_delay constraint
         clk2 : IN BIT; -- Ignored in port_time_delay
                        -- constraint
         out_delay,  -- Violation
         out_NOdelay: OUT BIT -- No violation, delay
                    -- specified in entity definition
);
END ent;

ARCHITECTURE arc OF ent IS
SIGNAL d_w : BIT;
BEGIN
    out_delay <= d_w AFTER 1 ns; -- Internal mode
    PROCESS(clk1)
```

```
    BEGIN
        IF (clk1'event and clk1='1') THEN
            out_NOdelay <= in_delay;
        END IF;
    END PROCESS;
END arc;
```

In the above example, the `in_NOdelay` and `out_NOdelay` ports are reported because no time delay is specified for these ports and they are not specified with the `-notimedelay_ports` argument.

However, the `clk1` and `clk2` ports are not reported because they are specified by the `-notimedelay_ports` argument and are not assigned with a time delay value. The port `in_delay` is not reported as delay for this port is specified in assignment to connected net `w_delay` (external mode). Similarly, for port `out_delay`, the time delay value is specified within entity definition (internal mode). Therefore, no violation is reported for it.

Default Severity Label

Warning

Rule Group

`DELTADELAY`

Reports and Related Files

No report or related file

Block Constraint Generation Rules

Rules under this category are used during constraint generation.

Following are the rules under this category:

Rule	Description
Ac_blksgdc01	Generates relevant SpyGlass CDC solution constraints at block boundary
Clock_info15	Generates the PortClockMatrix report and abstracted model for input ports
Setup_port01	Generates abstract_port constraints on the input ports of a block

NOTE: *For details on abstraction-based bottom up SpyGlass CDC solution verification flow, refer to the SpyGlass CDC solution Hierarchical Methodology Guide.*

Ac_blksgdc01

Migrates top-level constraints of SpyGlass CDC solution to block-level boundaries

When to Use

Use this rule to generate block-level constraints from top-level design for use in block-level CDC verification.

Prerequisites

Specify the name of a block module by using the `sgdc -export <block-name>` command in the top-level SGDC file. Do not specify a block instance in this command.

Description

The *Ac_blksgdc01* rule generates block-level constraints in the following SGDC file by migrating top-level constraints to the block-level boundary:

spyglass_reports/clock-reset/<block-name>_<inst-name>_genblock.sgdc

The above file contains SGDC constraints for block-input ports. Review this file and use it for block-level verification.

This file contains specification of the following constraints generated for a block:

clock	*define_reset_order*	*set_case_analysis*	*abstract_port*
assume_path	*quasi_static*	*signal_in_domain*	*num_flops*
reset	*sg_clock_group*		

Constraints Generation for Parameterized Modules

For a parameterized module, the `-param <parameter-value>` option is generated with the `current_design` command. If a default parameter value is associated with the module, the `-def_param` option is generated with the `current_design` command. See *Example 2 - Constraints generation for parameterized modules*.

Parameter(s)

The following are the common parameters used by the *Ac_blksgdc01* rule:

- `sta_based_clock_relationship`: Default value is `no`. Set this parameter to `yes` to compute domains based on the specification of the *sg_clock_group* constraint.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *define_reset_order* (Optional): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.
- *sg_clock_group* (Mandatory): Use this constraint to define asynchronous relationship between clocks.

- *reset_filter_path* (Optional): Use this constraint to specify reset paths so that the reset crossings on these paths are ignored from SpyGlass analysis.

Messages and Suggested Fix

Message 1

This rule reports the following message to indicate that an SGDC file is generated for the instance *<inst-name>*:

```
[AcBS1_1] [INFO] SGDC file generated for instance '<inst-name>'
(block: '<block-name>') using top level constraints migration
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 2

This rule reports the following message to indicate that no SGDC file is generated for the instance *<inst-name>*:

```
[AcBS1_2] [INFO] No SGDC file generated for instance '<inst-
name>' (block: '<block-name>') using top level constraints
migration
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

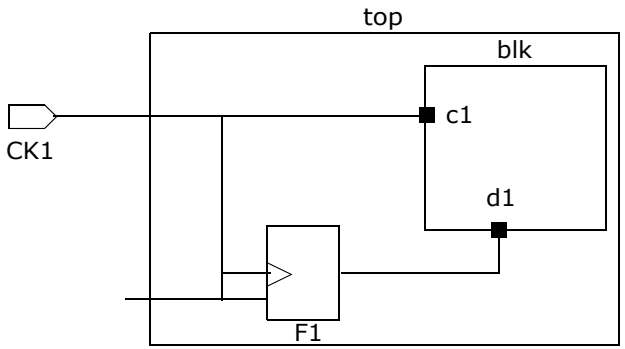
Example Code and/or Schematic

This section covers the following examples:

- *Example 1 - Top-level clock reaching block-level port*
- *Example 2 - Constraints generation for parameterized modules*
- *Example 3 - Generated sg_clock_group constraint*
- *Example 4 - Migration of reset_filter_path constraint*

Example 1 - Top-level clock reaching block-level port

Consider the following figure:




```
// Top-level SGDC file

current_design top
clock -name top.CK1 -domain d1
sgdc -export blk
```

FIGURE 334. Design for which *Ac_blksgdc01* Rule Generates Constraints

In the above figure, the top-level clock `CK1` is going into the block-level port `c1`, and the flip-flop `F1` clocked by `CK1` is going to the block-level port `d1`.

In this case, the *Ac_blksgdc01* rule generates the following SGDC constraints for the `blk` block:

```
current_design blk -def_param
```

```
clock -name c1 -domain domain1
abstract_port -scope cdc -module blk -ports d1 -clock c1
```

In the above figure, if `CK1` does not reach any block-level port, the *Ac_blksgdc01* rule generates the following SGDC constraints for the `blk` block:

```
current_design blk -def_param
abstract_port -scope cdc -module blk -ports d1 -clock VCLK0
clock -tag VCLK0 -domain d1
```

Where, `VCLK0` is a SpyGlass-generated virtual clock.

Example 2 - Constraints generation for parameterized modules

Consider the following files specified for SpyGlass analysis:

```
// test.v
module block1(a, b, c, c1, c2, c3, out);
    input [4:0] a, b, c;
    input c1, c2, c3;
    output [4:0] out;
    reg [4:0] temp1,temp2, temp3;
    always @(posedge c1)
        begin
            temp1 <= a;
        end
    always @(posedge c1)
        begin
            temp2 <= b;
        end
    always @(posedge c1)
        begin
            temp3 <= c;
        end
    wire [4:0] s;
    assign s = !temp3 ? temp1 : temp2;
    block #4 b1(.s(s), .clk1(c1), .clk2(c2), .clk3(c3), .out(out));
    block b2(.s(s), .clk1(c1), .clk2(c2), .clk3(c3), .out(out));
endmodule
module block(s,clk1,clk2,clk3,out);
parameter N = 2;
input clk1, clk2, clk3;
input [N-1:0] s;
output [N-1:0] out;
reg [N-1:0] src, des;
always @(posedge clk1)
    src <= s;
always @(posedge clk2)
    des <= src;
assign out = des;
endmodule
```

```
// clock.sgdc
current_design block1
clock -name c1
clock -name c2
clock -name c3
sgdc -export block
```

In the above example:

- The parameter value 4 is specified for the `b1` block. Therefore, the `-param 4` option is generated with the `current_design` command in the block_b1_genblock.sgdc file, as shown below:

```
current_design "block" -param { N=4 }
clock -name clk1 -domain c1
clock -name clk2 -domain c2
clock -name clk3 -domain c3
abstract_port -ports s[0:3] -scope cdc -clock clk1 -combo
```

```
yes
```

- A default parameter is associated with the `b2` block. Therefore, the `-def_param` option is generated with the `current_design` command in the block_b2_genblock.sgdc file, as shown below:

```
current_design "block" -def_param
clock -name clk1 -domain c1
clock -name clk2 -domain c2
clock -name clk3 -domain c3
abstract_port -ports s[0:1] -scope cdc -clock clk1
-combo yes
```

Example 3 - Generated sg_clock_group constraint

Consider the following files specified for SpyGlass analysis:

Verilog File

```
module block1(a, b, c, c1, c2, c3, out);
  input a, b, c, c1, c2, c3;
  output out;
  reg temp1,temp2, temp3;
  always @(posedge c1)
    begin
       temp1 <= a;
    end
  always @(posedge c1)
    begin
       temp2 <= b;
    end
  always @(posedge c1)
    begin
       temp3 <= c;
    end
  assign s = !temp3 ? temp1 : temp2;
  assign c12 = c1 && c2;
  assign c13 = c1 && c3;
  assign c32 = c3 && c2;
  block b1(.s(s), .clk1(c12), .clk2(c13), .clk3(c32), .out(out));
endmodule
```

Sgdc File

```
current_design block1
clock -name c1 -tag T1
clock -name c2 -tag T2
clock -name c3 -tag T3
sg_clock_group -group1 { T1 } -group2 { T2 }
sg_clock_group -group1 { T1 } -group2 { T3 }
sg_clock_group -group1 { T2 } -group2 { T3 }
sgdc -create_constraints block
```

Generated Block File

```
####Hierarchical instance name of the block: block1.b1
 current_design "block" -def_param
#####start::clock constraints#####
## top_down_new_domain_4 :: d0,d1
clock -name clk1 -domain top_down_new_domain_4 -tag sg_tag_1
## top_down_new_domain_5 :: d0,d2
clock -name clk2 -domain top_down_new_domain_5 -tag sg_tag_2
## top_down_new_domain_3 :: d1,d2
clock -name clk3 -domain top_down_new_domain_3 -tag sg_tag_3
#####end::clock constraints######
#####start::sg_clock_group constraints#####
sg_clock_group -group1 sg_tag_1 -group2  sg_tag_2
sg_clock_group -group1 sg_tag_1 -group2  sg_tag_3
sg_clock_group -group1 sg_tag_2 -group2  sg_tag_3
#####end::sg_clock_group constraints######


#####start::abstract_port constraints#####
abstract_port -ports s -scope cdc -clock VCLK0 -combo yes
#####end::abstract_port constraints######
#####start::virtual clock constraints#####
clock -tag VCLK0 -domain d0
#####end::virtual clock constraints######
```

In the above example, the sg_clock_group constraint is generated as shown above.

Example 4 - Migration of reset_filter_path constraint

Consider the following figure.

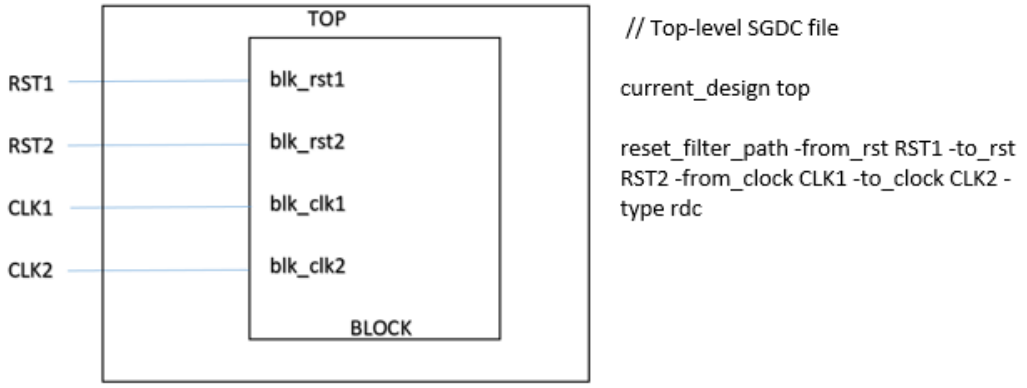


FIGURE 335.

In the design shown in *Figure 335*, the `Ac_blksgdc01` rule generates the following SGDC constraints for the `BLOCK` block:

```
reset_filter_path -from_rst blk_rst1 -to_rst blk_rst2 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

In addition, if the `RST1` reset is driving one more block pin `blk_rst3` as shown below, the above constraint in the top-level sgdc would be migrated to block as multiple constraints:

```
reset_filter_path -from_rst blk_rst1 -to_rst blk_rst2 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

```
reset_filter_path -from_rst blk_rst3 -to_rst blk_rst2 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

Similar would be the behavior for other fields of the constraint.

Consider the design shown in *Figure 336*.

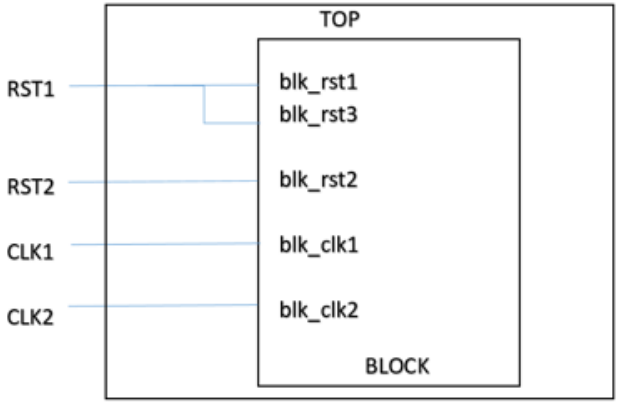


FIGURE 336.

In addition, consider that the following constraint is specified in the top-level SGDC file:

```
reset_filter_path -from_rst RST1 RST2 -to_rst RST3 -from_clk
CLK1 -to_clk CLK2 -type rdc
```

Note that there are two objects specified in the `-from_rst` argument.

The above constraint is equivalent to:

```
reset_filter_path -from_rst RST1 -to_rst RST3 -from_clk CLK1
-to_clk CLK2 -type rdc
```

```
reset_filter_path -from_rst RST2 -to_rst RST3 -from_clk CLK1
-to_clk CLK2 -type rdc
```

Therefore, these constraints are migrated as:

```
reset_filter_path -from_rst blk_rst1 -to_rst blk_rst3 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

```
reset_filter_path -from_rst blk_rst2 -to_rst blk_rst3 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

Similar would be the behavior for other arguments of the constraint.

NOTE: *If the -from_obj or -to_obj arguments are provided in the constraint, such constraints are not migrated to the block level.*

Default Severity Label

Info

Rule Group

BLOCK_CONSTR_GENERATION

Reports and Related Files

The Ac_blksgdc01 rule generates the file <block-name>_<inst-name>_genblock.sgdc that is located in the spyglass_reports/clock-reset/ directory.

This file contains SGDC constraints for block input ports. See *Example 1 - Top-level clock reaching block-level port* and *Example 2 - Constraints generation for parameterized modules*.

Block Abstraction Rules

Rules under this category are used during block abstraction.

Following are the rules under this category:

Rule	Description
Ac_abstract01	Generates relevant SpyGlass CDC solution constraint for block abstraction

NOTE: *For details on abstraction-based bottom up SpyGlass CDC solution verification flow, refer to the SpyGlass CDC Solution Hierarchical Methodology Guide.*

Ac_abstract01

Generates SpyGlass CDC constraints for block abstraction

When to Use

Use this rule to generate an abstract view for a block so that this abstract view is used during SpyGlass CDC verification for an SoC.

Prerequisites

Specify an SGDC file containing block constraints on input ports. See *Example 2 - Specifying parameters through set_option param {<param>}*.

Description

The *Ac_abstract01* rule generates *The <block-name>_cdc_abstract.sgdc File* that represents the abstract view for a block.

This file is saved in the *$projectdir/<block-name>/cdc_abstract/cdc_abstract/spyglass_reports/abstract_view/cdc/* directory.

This abstract view:

- Is a set of SpyGlass design constraints describing the behavior of block ports. See *Example 1*.

NOTE: *The abstract_port constraint is generated with respect to the generated clock instead of the master clock when the generated clock reach the output port of the block.*

- Is used during SoC-level validation and verification.

The `Ac_abstract01` rule generates the following *abstract_port* constraint if a qualified signal (source merges with a valid qualifier at a valid gate) reaches the output port.

```
abstract_port -ports out -sync inactive -clock clk2 -from
clk1 -to clk2
```

Note that in the above constraint, `clk2` is the destination domain of the qualifier.

For example, consider the schematic shown in *Figure 337*.

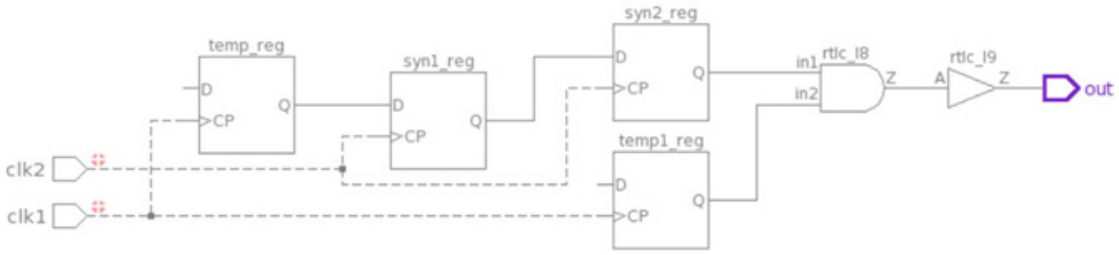


FIGURE 337.

In the above design, if the *check_qualified_signal_at_soc* parameter is set to `yes`, the `Ac_abstract01` rule generates the following *abstract_port* constraint because a qualified signal reaches the output port:

```
abstract_port -ports out -clock "clk2" -from "clk1" -to
"clk2" -sync inactive -sync_names "top.syn2"
```

However, if a non-qualified signal (sources merged with invalid gate, such as XOR) reaches the output port, the `Ac_abstract01` rule generates the following *abstract_port* constraints if the *check_qualified_signal_at_soc* parameter is set to `yes`:

```
abstract_port -port out -sync inactive -clock clk2 -from clk1
-to clk2 (clk2 is the domain of qualifier)

abstract_port -clock clk1 (clk1 is the domain of source)
```

Rule Exceptions

In the abstract view, the *abstract_port* constraint is not generated on the output of quasi-static flip-flops.

Parameter(s)

- *check_qualified_signal_at_soc*: Default value is `no`. Set this parameter to `yes` to not report data-mismatch violations if a qualified signal reaches to abstract block input having same domain as the destination domain.

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *assume_path* (Optional): Use this constraint to specify paths through black box instances.
- *cdc_false_path* (Optional): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *define_reset_order* (Optional): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *signal_in_domain* (Optional): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

Message 1

The following message appears to indicate that an abstracted SGDC file is generated for the module *<module-name>*:

[ACAbs1] [INFO] Abstracted sgdc file for module '<module-name>' is generated

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 2

The following message appears to indicate that an abstracted SGDC file is generated for the module *<module-name>*:

[AcAbs2] [INFO] `Abstracted sgdc file for module '<module-name>' is not generated`

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

This section covers the following examples:

- *Example 1*
- *Example 2 - Specifying parameters through set_option param {<param>}*
- *Example 3 - Generation of abstract_port with -combo no*

- *Example 4 - Generation of abstract_port with --combo_ifn*

Example 1

Consider the following figure of a design:

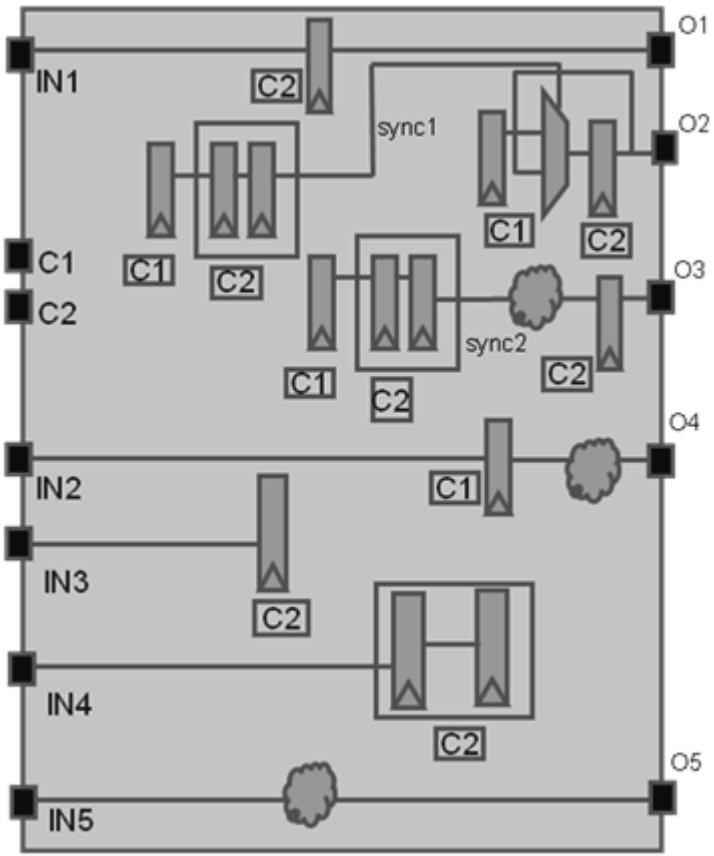


FIGURE 338. Design on which the Ac_abstract01 Rule is Run

When your run the *Ac_abstract01* rule on the above design, various constraints are generated at the output ports of the above design during abstraction.

The following points describe the generated constraints for these output

ports:

- The O1 port is driven by a flip-flop in the C2 domain, which in turn is coming from the IN1 input. In this case, the following constraint is generated:

  ```
  abstract_port -scope cdc -module BLOCK -ports O1 -clock C2
  -related_ports IN1
  ```

- The O2 port is the output of a synchronized data crossing in the C2 domain with the C1 source. In this case, the following constraint is generated:

  ```
  abstract_port -scope cdc -module BLOCK -ports O2 -clock C2
  -sync inactive -from C1 -to C2 -sync_names "BLOCK.sync1"
  ```

- The O3 port is the output of a conventional multi-flop synchronizer from the C1 to C2 domain. It also has a flip-flop after the synchronizer. In this case, the following constraint is generated:

  ```
  abstract_port -scope cdc -module BLOCK -ports O3 -clock C2
  -sync active -from C1 -to C2 -seq yes -sync_names
  "BLOCK.sync2"
  ```

- The O4 port is driven by a flip-flop in the C2 domain through a combinational logic that is connected to the IN2 input. In this case, the following constraint is generated:

  ```
  abstract_port -scope cdc -module BLOCK -ports O4 -clock C1
  -related_ports IN2 -combo yes
  ```

- The O5 port is connected to the IN5 port through a combinational logic. In this case, the following constraint is generated:

  ```
  assume_path -name BLOCK -input IN5 -output O5
  ```

In addition, the *abstract_file* constraint is appended to the above constraints to refer to the input side constraints that were passed in the abstraction run.

Example 2 - Specifying parameters through set_option param {<param>}

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1, in2, clk1, clk2, out1, out2);
parameter W = 4;
    input [W-1 : 0] in1, in2;
    input clk2, clk1;
    output [W-1 : 0] out1, out2;
    assign out2 = in1;
    reg [W-1 : 0] temp1, temp2, temp3,temp4, temp5;
    wire [W-1 : 0] w;
    always @(posedge clk1)
        temp1 <= in2;
    always @(posedge clk2)
        temp2 <= in1;
    always @(posedge clk2)
        temp3 <= temp1;
    always @(posedge clk2)
        temp4 <= temp3;
    assign w = temp4 & in2;
    always @(posedge clk2)
        temp5 <= w;
    assign out1 = temp2 & temp5;
  endmodule
```

// clocks.sgdc

```
current_design top
abstract_port -ports in1
      -clock clk2
abstract_port -ports in2
      -clock clk1
clock -name clk1
clock -name clk2
```

For the above example, there can be the following cases:

- Case 1

  If you pass a parameter value for the `top` module by using the `set_option param {top.W=2}` project file command, the following constraints are generated in the top_cdc_abstract.sgdc file:

```
current_design "top" -param { W=2 }
assume_path -name top -input in1[0] -output out2[0]
assume_path -name top -input in1[1] -output out2[1]

abstract_port -ports out1[0] -scope cdc -clock clk2
-combo yes -related_ports in1[0]
abstract_port -ports out1[1] -scope cdc -clock clk2
-combo yes -related_ports in1[1]
abstract_port -ports out1[0:1] -scope cdc -clock clk2
-combo yes -from clk1 -to clk2 -sync inactive
-sync_names "top.temp4[0:1]"
```

- Case 2

  If you do not pass any parameter value to the `top` module, the following constraints are generated in the top_cdc_abstract.sgdc file:

```
current_design "top" -def_param
assume_path -name top -input in1[0] -output out2[0]
assume_path -name top -input in1[1] -output out2[1]
assume_path -name top -input in1[2] -output out2[2]
assume_path -name top -input in1[3] -output out2[3]

abstract_port -ports out1[0] -scope cdc -clock clk2 -combo
yes -related_ports in1[0]
abstract_port -ports out1[1] -scope cdc -clock clk2
-combo yes -related_ports in1[1]
abstract_port -ports out1[2] -scope cdc -clock clk2
-combo yes -related_ports in1[2]
abstract_port -ports out1[3] -scope cdc -clock clk2
-combo yes -related_ports in1[3]
abstract_port -ports out1[0:3] -scope cdc -clock clk2
-combo yes -from clk1 -to clk2 -sync inactive
-sync_names "top.temp4[0:3]"
```

Example 3 - Generation of abstract_port with -combo no

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1, clk1, clk2, clk3, out1,out2);
 input clk1, clk2,clk3,in1;
 output reg out1,out2;
 reg src, dest,dest1;
 always @(posedge clk2)
     begin
     dest<=in1;
     out1<=dest;
     end
 always @(posedge clk3)
     begin
     dest1<=in1;
     out2<=dest1;
     end
 endmodule
```

// constr.sgdc

```
current_design top
clock -name clk1
clock -name clk2
clock -name clk3
input -name in1 -clock vck
```

virtual clock

The following figure shows the schematic of the above example:

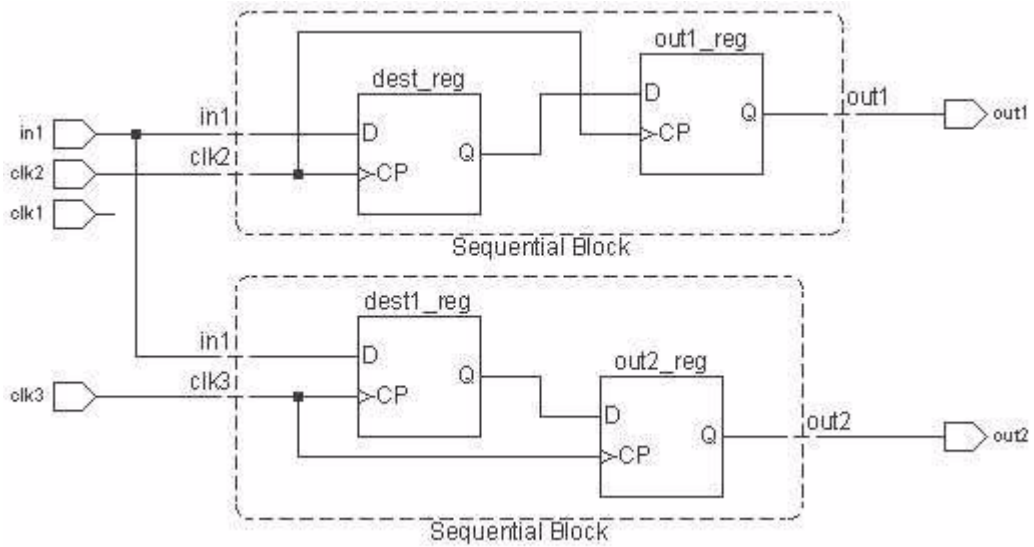


FIGURE 339. Example in which -combo no is generated

In the above example, the `in1` input port is involved in a crossing synchronized by *Conventional Multi-Flop Synchronization Scheme* and this port is clocked by a virtual clock.

Therefore, the *Ac_abstract01* rule generates `-combo no` with the *abstract_port* constraint generated for `in1` in the top_cdc_abstract.sgdc file:

```
abstract_port -ports in1 -scope cdc -clock vck -combo no
```

Example 4 - Generation of abstract_port with --combo_ifn

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1, clk1, clk2, out);
input clk1, clk2, in1;
output reg out;
reg src, dest;
always @(posedge clk1)
    src<=in1;
always @(posedge clk2)
    begin
    dest<=in1;
    out<=dest;
    end
endmodule
```

// constr.sgdc

```
current_design top
clock -name clk1
clock -name clk2
input -name in1 -clock vck
```

The following figure shows the schematic of the above example:

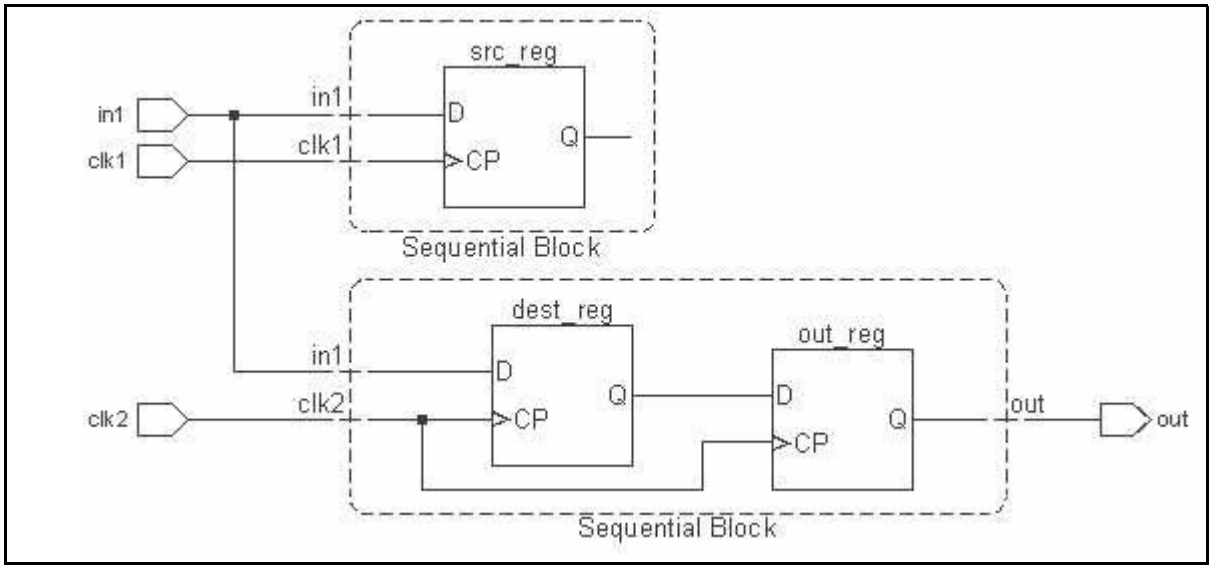


FIGURE 340. Example in which -combo_ifn is generated

In the above example, the `in1` input port clocked by the `vck` virtual clock is synchronized in the same clock domain (`clk2`) as that of the synchronizer.

Therefore, the *Ac_abstract01* rule generates `-combo_ifn` with the *abstract_port* constraint generated for `in1` in the top_cdc_abstract.sgdc file:

```
abstract_port -ports in1 -scope cdc -clock vck -combo no
-combo_ifn clk2
```

In addition, the rule generates `-combo_ifn` with the *abstract_port* constraint for real clocks as well. In the above example, if the `in1` input port clocked by the `clk1` clock is synchronized in the same clock domain (`clk2`) as that of the synchronizer, the *Ac_abstract01* rule generates `-combo_ifn` with the *abstract_port* constraint generated for `in1` in the top_cdc_abstract.sgdc file:

```
abstract_port -ports in1 -scope cdc -clock clk1 -combo no
-combo_ifn clk2
```

However, the above constraint is not generated if `in1` is synchronized in different clock domains of different synchronizers. This scenario is shown in

the following figure where `in1` is synchronized in the `clk2` and `clk3` domains of the `S1` and `S2` synchronizers, respectively:

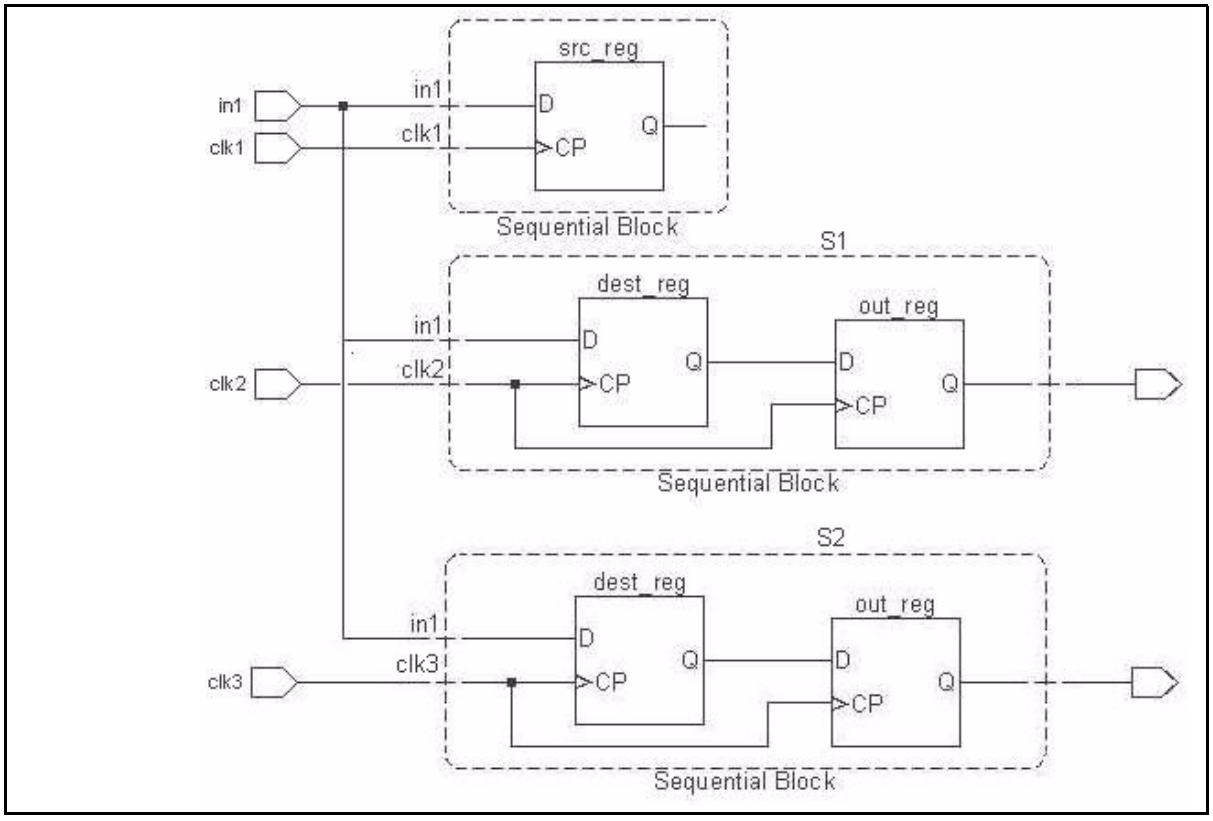


FIGURE 341. Example in which -combo_ifn is not generated

Default Severity Label

Info

Rule Group

ADV_CLOCKS

Reports and/or Related Files

The *Ac_abstract01* rule generates *The <block-name>_cdc_abstract.sgdc File*.

The <block-name>_cdc_abstract.sgdc File

This file represents the abstract view of a block. It contains block information in the form of constraints, such as *abstract_port*, *assume_path*, *clock*, *reset*, and *set_case_analysis*.

This file is saved in the *spyglass_reports/abstract_view/* directory. To change the directory, use the following project-file command:

```
set_option block_abstract_directory <directory>
```

Options Generated in the <block-name>_cdc_abstract.sgdc File

The following options appear in this file based on the specified conditions:

Option	Condition
-param <parameter-value> option with the current_design command	This option appears for parameterized modules. See *Case 1* of *Example 2 - Specifying parameters through set_option param {<param>}*.
-def_param option with the current_design command	This option appears if a default parameter value is associated with a module. See *Case 2* of *Example 2 - Specifying parameters through set_option param {<param>}*.
-combo no option with the *abstract_port* constraint	This option appears if all the following conditions hold true for an input port: • The port is specified by the *input* or *abstract_port* constraint. • The port is involved in the crossings synchronized by *Synchronizing Cell Synchronization Scheme* or *Conventional Multi-Flop Synchronization Scheme*. • The port is clocked by a virtual clock. See *Example 3 - Generation of abstract_port with -combo no*.

Option	Condition
-combo_ifn option with the *abstract_port* constraint	This option appears when all the following conditions hold true for an input port: • The port is specified by the *input* or *abstract_port* constraint. • The port is involved in the crossings synchronized by *Synchronizing Cell Synchronization Scheme* or *Conventional Multi-Flop Synchronization Scheme*. • The port is synchronized in the same clock domain as that of the synchronizer. • The port is clocked by a virtual clock. See *Example 4 - Generation of abstract_port with --combo_ifn*.
-sync inactive -seq no with the *abstract_port* constraint	This option appears (instead of the *-sync active* option) for a control crossings if *qualifier -ignore* is specified for that crossing.
-tag option with the *clock* constraint	This option appears for virtual clocks that are specified to the *input*/*abstract_port* constraints. Note that: • The `sg_virtual` string is appended to the domain name of the virtual clock if the domain of this clock is not mapped to any user-specified domain. • The `sg_merged` string is appended to the domain name of the virtual clock if this domain is mapped to multiple user-specified domains.
-ignore option with the *abstract_port* constraint	This option appears if all the fan-in of an output port are hanging or blocking.

Note that if a synchronous reset propagates through a synchronized crossing and that synchronized signal drives an output port, *abstract_port* is generated without `-sync active` for that output.

Crossings Involving Output Ports

If a crossing involving an output port is synchronized, the following *abstract_port* constraint is generated for that synchronized crossing:

```
abstract_port -ports <output-port> -clock <output-clock-
name> -sync inactive -from <src-clock-name> -to <dest-clock-
```

name>

However, if that crossing is not synchronized, the *abstract_port* constraint is generated for the corresponding *output* constraint.

Block Constraint Validation Rules

Rules under this category validate consistency of an abstract model in context of a higher-level block.

Please note the following points for the rules of this category:

- To enable these rules, specify the `set_option sgdc_validate yes` command in the project file.

  Specifying this command enables validation of block-level constraints in context of a top-level domain.

- Before running rules of this category, use the following command in the migration file to specify a block and its SGDC file:

  `sgdc -import <block-name> <block-abstracted-sgdc>.sgdc`

NOTE: *For details on abstraction-based bottom up SpyGlass CDC solution verification flow, refer to the SpyGlass CDC Solution Hierarchical Methodology Guide.*

Following are the rules under this category:

Rule	Description
Ac_abstract_validation01	Reports block abstraction mismatch with top level design
SGDC_abstract_mapping01	Reports clock mapping of an abstracted instance
SGDC_clock_validation01	Checks whether block level clock is missing
SGDC_clock_validation02	Checks whether top-level clock reaches block port
SGDC_clock_domain_validation01	Checks whether top-level clocks of different domains are connected to same domain block port
SGDC_clock_domain_validation02	Checks whether top-level clocks of same domain are connected to different domain block ports
SGDC_set_case_analysis_validation01	Reports a violation if values between block level and top-level set_case_analysis are conflicting

Rule	Description
SGDC_set_case_analysis_validation02	Checks whether set_case_analysis is missing
SGDC_reset_validation01	Checks missing reset constraint at block level
SGDC_reset_validation02	Checks missing reset constraint at top-level
SGDC_reset_validation03	Reports conflicting top and block level asynchronous and synchronous reset types
SGDC_reset_validation04	Checks conflict between top and block level reset value
SGDC_virtualclock_validation01	Reports a virtual clock specified in combination with other real or virtual clock in the abstract_port constraint
SGDC_virtualclock_validation01	Checks validity of virtual clock
SGDC_input_validation01	Checks for incorrect input constraint
SGDC_input_validation02	Checks for missing input constraint
SGDC_num_flops_validation01	Reports a violation if a clock domain of clocks specified in the -from_clk and -to_clk argument of the num_flops constraints are same.
SGDC_num_flops_validation02	Checks the conflict between the number of flip-flops specified in num_flops constraint
SGDC_output_validation01	Checks for incorrect output constraint
SGDC_output_validation02	Checks for missing output constraint
SGDC_abstract_port_validation01	Checks whether abstract_port is constrained with proper clocks
SGDC_abstract_port_validation02	Checks whether -sync is correctly specified
SGDC_abstract_port_validation03	Checks the validity of clocks specified in -from/-to field
SGDC_abstract_port_validation04	Checks for combinational logic
SGDC_qualifier_validation01	Clocks specified in -from_clk and -to_clk field of qualifier constraint have same domain

Rule	Description
SGDC_qualifier_validation02	Checks for missing qualifier constraint
SGDC_cdc_false_path_validation01	Clocks specified in -from and -to field of cdc_false_path constraint have same domain.
SGDC_define_reset_order_validation01	Checks whether top-level reset reaches the reset pin specified in define_reset_order constraint
SGDC_define_reset_order_validation02	Resets specified in -from and -to field of define_reset_order constraint are same
SGDC_quasi_static_validation01	Reports unconstrained quasi_static ports of an abstract view
SGDC_quasi_static_validation02	Reports quasi-static ports, which are not driven from top-level quasi-static signals, of an abstract view

Ac_abstract_validation01

Reports block abstraction mismatch with top level design

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract view in the context of a higher-level hierarchy.

Prerequisites

Specify the following details before running this rule:

- The *sgdc* `-import` constraint.
- The *clock* constraint to check for *Clocks Mismatch*, *Data Path Domain Mismatch*, and *Virtual Clocks Mismatch*.
- The *set_case_analysis* constraint to check for *Case Analysis Mismatch*.
- The *quasi_static* constraint to check for *Quasi Static Mismatch*.
- The *reset* constraint to check for *Reset Mismatch*.
- The *qualifier* constraint to check for *Qualifier Mismatch*.
- The *num_flops* constraint to check for *num_flops Mismatch*.
- The *abstract_port* constraint to check for *Combo Check Mismatch*.

NOTE: *For information on parameters and constraints used by this rule, see Parameter(s) and Constraint(s).*

Description

The *Ac_abstract_validation01* rule reports a violation for the following types of mismatches:

Clocks Mismatch	*Clock Domain Mismatch*	*Virtual Clocks Mismatch*
Case Analysis Mismatch	*Quasi Static Mismatch*	*Data Path Domain Mismatch*
Reset Mismatch	*Qualifier Mismatch*	*Combo Check Mismatch*
num_flops Mismatch		

Reporting violations for all the above mismatches may result in noise and multiple iterations to fix them. To avoid this, you can restrict this rule to validate only block-level assumptions with respect to the top level by setting *abstract_validate_express* to `yes`.

Clocks Mismatch

This mismatch occurs in the following cases:

- If a top-level clock reaches to a clock port of a block, but that clock port is not constrained by the *clock* constraint.
- If a block-level clock port is not driven from a top-level clock port.

  This can occur when the *clock* constraint is defined on a block port, but a top-level clock does not reach that block port.

NOTE: *For clocks mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 1. Also see Example - Clock Mismatch.*
- *Generates Clock Mismatch Spreadsheet.*

The rule also generates the `derived_reset_info.csv` file. The file includes information on the derived reset flops and the end flop where the derived reset flop is used as a reset.

The file is generated in the `spyglass_reports/clock-reset/` directory.

Clock Domain Mismatch

This mismatch occurs in the following cases:

- If multiple clock ports in the same domain of an abstract view are triggered from top-level clocks of different domains.

  See *Example 1 - Clock Domain Mismatch*.
- If virtual clock specified at a block port and the clock port are in the same domain of an abstract view and they are triggered from top-level clocks of different domains

  See *Example 2 - Clock Domain Mismatch*.
- If virtual clocks *<virtual-clock1>* and *<virtual-clock2>* specified at the ports *<block-port1>* and *<block-port2>*, respectively, are in the same domain of an abstract view and are triggered from top-level clocks of different domains

See *Example 3 - Clock Domain Mismatch*.

- If multiple clock ports in different domains of an abstract view are triggered from the top-level clocks of the same domain.

  See *Example 4 - Clock Domain Mismatch*.

NOTE: *For clock domain mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 2 and Message 3.*
- *Generates Clock Domain Mismatch Spreadsheet.*

If the `sta_based_clock_relationship` parameter is set to true, SpyGlass CDC reports clock domain mismatch violations based on the *sg_clock_group* constraint specified on the block and the top level rather than the domain specified for the clock. See *Example 5 - Clock Domain Mismatch*.

Virtual Clocks Mismatch

This mismatch occurs in the following cases:

- If multiple ports of the same block specified with the same virtual clock are driven by different domains from top level
- If no top-level clock is reaching the block port specified with a virtual clock

NOTE: *For virtual clocks mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 4 and Message 5. See Example 1 - Virtual Clocks Mismatch and Example 2 - Virtual Clocks Mismatch.*
- *Generates Virtual Clocks Mismatch Spreadsheet.*

Case Analysis Mismatch

This mismatch occurs in the following cases:

- If there is a mismatch between the following values:
  - Constant value specified by the *set_case_analysis* constraint for a block-level port
  - Constant value propagated from the top-level
- If a simulated value reaches a top-level net connected to a block-level port, but no *set_case_analysis* constraint is specified on the block-level port

- If a simulated value does not reach to a top-level net connected to a block-level port, but the *set_case_analysis* constraint is specified on the block-level port

NOTE: *For case analysis mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 6. Also see Example - Case Analysis Mismatch.*
- *Generates Case Analysis Mismatch Spreadsheet.*

Quasi Static Mismatch

This mismatch occurs in the following cases:

- If a top-level quasi-static signal reaches a block port on which a *quasi_static* constraint has not been specified.
- If a *quasi_static* constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

NOTE: *For quasi static mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 7. Also see Example - Quasi Static Mismatch.*
- *Generates Quasi static Mismatch Spreadsheet.*

Data Path Domain Mismatch

This mismatch occurs if an abstract-block port is driven from a sequential instance, and there is a mismatch between the clock pin driving this sequential instance and the clock specified in the `-clock` argument of the *abstract_port* or the input constraint.

NOTE: *The Ac_abstract_validation01 rule does not report data path domain mismatch violations if a qualified signal reaches to abstract block input having same domain as the destination domain. Set the check_qualified_signal_at_soc parameter to* `yes` *to not report such data-mismatch violations.*

NOTE: *For data path domain mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 8. Also see Example - Data Path Domain Mismatch.*
- *Generates Data Path Domain Mismatch Spreadsheet.*

Reset Mismatch

This mismatch occurs in the following cases:

- If a top-level reset reaches a block port for which no *reset* constraint is specified.
- If the *reset* constraint is specified for a block-level port, but no top-level reset drives that block-level port.
- If an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.
- If the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

NOTE: *For reset mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 9. Also see Example - Reset Mismatch.*
- *Generates Reset Mismatch Spreadsheet.*

The rule also generates the `derived_reset_info.csv` file. The file includes information on the derived reset flops and the end flop where the derived reset flop is used as a reset.

The file is generated in the `spyglass_reports/clock-reset/` directory.

Qualifier Mismatch

This mismatch occurs in the following cases:

- If a synchronized signal reaches to an input port of a block for which:
  - The qualifier or *abstract_port* constraint is not defined with the -sync argument, or
  - The *abstract_port* constraint is defined with the `-sync` argument, but clocks specified by the `-from` or `-to` argument of that *abstract_port* constraint do not match with the source or destination clocks of the synchronizer reaching to that input port.
- If a synchronizer does not reach to an input port of a block for which the `-sync` argument of the *abstract_port* constraint is specified

Automatically Fixing the abstract_port Constraint of the Reported Port

The **CDC SoC abstract auto update flow** works if a synchronized signal reaches to a block port and any of the following cases is true:

- *abstract_port* is defined at the block port without `-sync` and the domain of the synchronized signal matches with the clock specified in *abstract_port*
- *assume_path* is defined at the block port

Set the *autofix_abstract_port* parameter to `yes` to modify the *abstract_port* constraint in the context of SoC for the reported port, and save the modified constraints in the *<module_name>_<instance_name>_cdc_soc_abstract_autoupdate.sgdc* file.

By default, in addition to the modified constraints, this file also contains a copy of all the unmodified input side *abstract_port* constraints present in block-level SGDC file (abstract block). Set the *autofix_dump_allinputs* to `no` to generate only the modified constraints.

The following figure shows the example of using the *autofix_abstract_port* parameter:

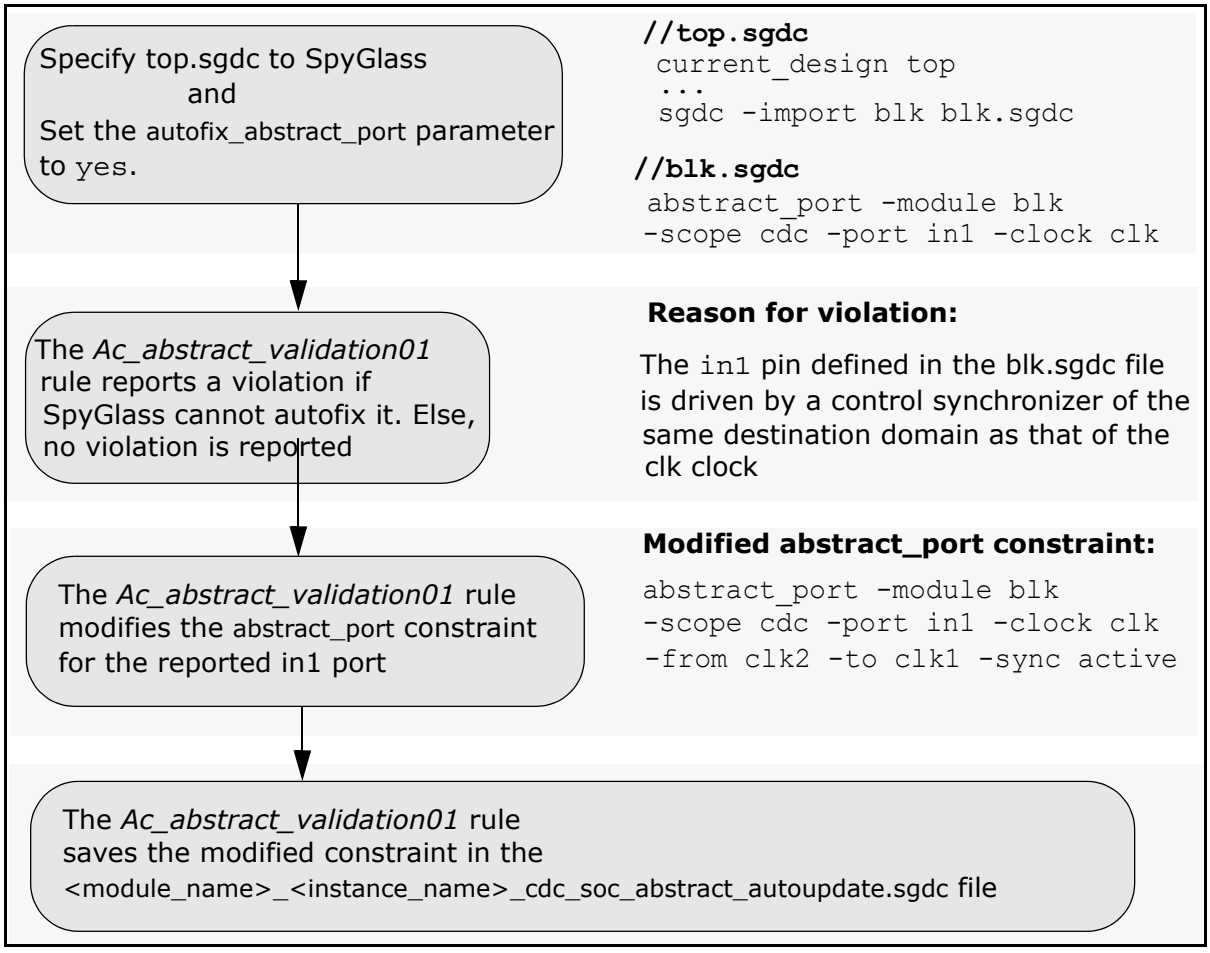


FIGURE 342. Using the autofix_abstract_port Parameter

NOTE: *For qualifier mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 10. Also see Example - Qualifier Mismatch.*
- *Generates Qualifier Mismatch Spreadsheet.*

Combo Check Mismatch

This mismatch occurs if a combinational logic exists between a block-level input port and the output of a sequential element at the top-level when the `-combo no` argument of the *abstract_port* constraint is specified for that block in the following cases:

- If at the block level, the *abstract_port* constraint is defined along with the `-combo_ifn` argument as shown below:

  ```
  abstract_port  -ports a -clock VCK1 -combo_ifn ck2 -combo no
  ```

  In this case, this rule reports a violation if a sequential element reaches the block port after a combinational logic and if the sequential cell has a clock domain that is different from the clock domain of the clock specified in the `-combo_ifn` argument.

  Note that the rule reports a violation if a real clock is used in place of a virtual clock.

- If at the block level, the *abstract_port* constraint is defined with real clocks as shown below:

  ```
  abstract_port  -ports a -clock clk1 -combo no
  ```

  In this case, the *Ac_abstract_validation01* rule will report a violation if a sequential element reaches the block port after combinational logic.

- If at the block level, the *abstract_port* constraint is defined with only virtual clock as shown below:

  ```
  abstract_port  -ports a -clock VCK1 -combo no
  ```

  In this case, this rule reports a violation if the sequential element reaches the block port after combinational logic.

Automatically Fixing the abstract_port Constraint of the Reported Port

Set the *autofix_abstract_port* parameter to `yes` to modify the *abstract_port* constraint in the context of SoC for the reported port, and save the modified constraints in the *<module_name>_<instance_name>_cdc_soc_abstract_autoupdate.sgdc* file.

By default, in addition to the modified constraints, this file also contains a

copy of all the unmodified input side *abstract_port* constraints present in block-level SGDC file (abstract block). Set the *autofix_dump_allinputs* to `no` to generate only the modified constraints.

The following figure shows the example of using the *autofix_abstract_port* parameter:

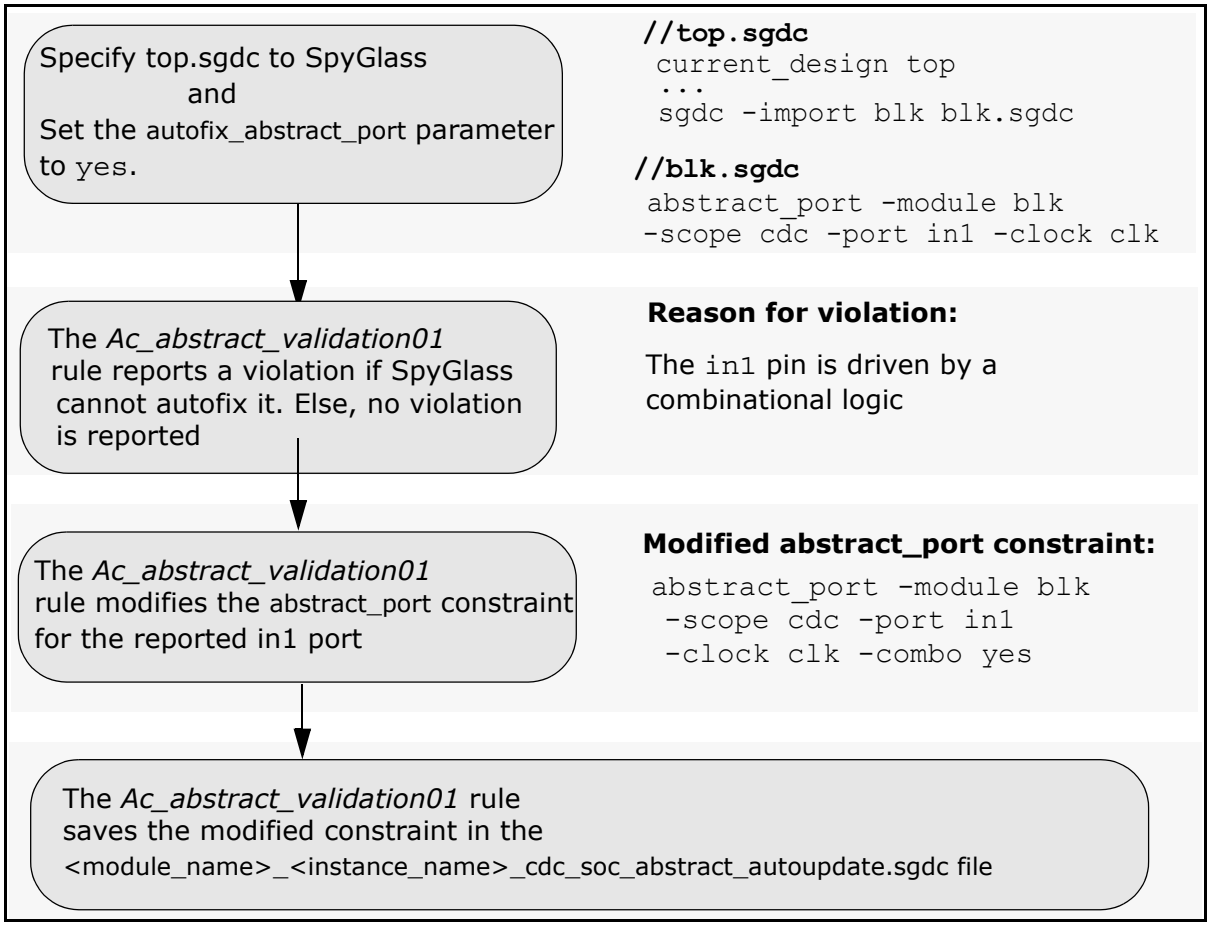


FIGURE 343. Using the autofix_abstract_port Parameter

NOTE: *For combo check mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 11. Also see Example - Combo Check Mismatch.*

- *Generates Combo Check Mismatch Spreadsheet.*

num_flops Mismatch

This mismatch occurs if the value of the *num_flops* constraint defined in the top-level SGDC file for a clock-pair is different from the value of the *num_flops* constraint defined in the block-level SGDC file for the same clock pair.

NOTE: *For the num_flops mismatch, the Ac_abstract_validation01 rule:*

- *Reports Message 12. Also see Example - num_flops Mismatch.*
- *Generates num_flops Mismatch Spreadsheet.*

The Ac_abstract_validation01 rule does not report data mismatch violations if a qualified signal reaches to abstract block input that has the same domain as the destination domain. Set the *check_qualified_signal_at_soc* parameter to `yes` to not report such data-mismatch violations.

For example, consider the schematic shown in *Figure 344*.

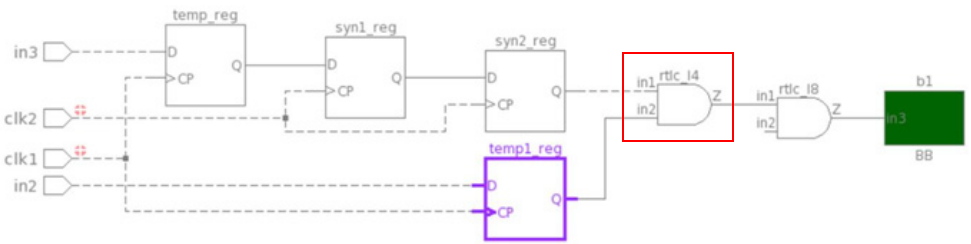


FIGURE 344.

In addition, consider that the following *abstract_port* constraint is specified on the `in3` port at the block level:

```
abstract_port -module BB -ports in3 -clock clk2
```

In this case, if the *check_qualified_signal_at_soc* parameter is set to `yes`, the `Ac_abstract_validation01` rule does not report a violation because a

qualified signal (highlighted in red in *Figure 344*) reaches to the `in3` pin of the abstracted module (`BB`).

The *check_qualified_signal_at_soc* parameter does not work as described above in the following scenarios:

- If the destination block input has multiple *abstract_port* constraints specified in different clock domains
- If the destination block has *abstract_port* constraint specified with multiple clocks
- If the source block has multiple *abstract_port* constraints with `-sync active` specified in different source domains
- If the source blocks, which have multiple *abstract_port* constraints specified, belong to domains that are different from the destination domain
- If the source block has the *abstract_port* and *qualifier* constraint specified on the same pin

In addition, if a qualified signal reaches to the *abstract_port* constraint specified with a virtual clock, the `Ac_abstract_validation01` rule reports violations for the virtual clock mismatch even if the *check_qualified_signal_at_soc* parameter is specified.

Parameter(s)

- *autofix_abstract_port*: Default value is `yes`. Set this parameter to `no` to disable this rule from modifying the reported *abstract_port* constraints in the context of SoC.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *abstract_validate_express*: Default value is `no`. Set this parameter to `yes` to enable validation of only user-specified block assumptions with respect to the top-level block. Missing block assumptions are not checked in this case.
- *validate_reduce_pessimism*: Default value is `none`. Set this parameter to `hanging_nets` to ignore reporting on the hanging block ports. Other possible values are `constant`, `quasi_static`, `ignore_domain_overconstraint`, and `all`.

- `sta_based_clock_relationship`: Default value is `no`. Set this parameter to `yes` to compute domains based on the specification of the *sg_clock_group* constraint.
- *check_qualified_signal_at_soc*: Default value is `no`. Set this parameter to `yes` to not report data-mismatch violations if a qualified signal reaches to abstract block input having same domain as the destination domain.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *input* (Optional): Use this constraint to specify a clock domain at input ports.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *sgdc* `-import` (Mandatory): Use this constraint to specify a block-level SGDC file to be imported.
- *validation_filter_path* (Optional): Use this constraint to filter data domain violations reported during block validation.
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

Message 1

The following message appears for the *<inst-name>* instance that has

Clocks Mismatch:

```
[AcAbsV1_1] [WARNING] Instance '<inst-name>' of block '<module-name>' has <num> clock mismatches
```

NOTE: *For this message, see:*

- *Clock Mismatch Spreadsheet*
- *Example - Clock Mismatch*

Potential Issues

See *Clocks Mismatch*.

Consequences of Not Fixing

The consequences vary based on the following situations:

- If a top-level clock reaches to an unconstrained clock port of a block

  Consequences: Some valid clock ports may get missed in the SGDC file of an abstract view. As a result, SpyGlass may not perform synchronization checks for such potential clock signals.

- If a block-level clock port is not driven from a top-level clock port.

  Consequences:

  - If the path of top-level clock is blocked before reaching to a clock port of a block, it may result in incorrect violations at the top-level.
  - If the block port is not a clock but it is defined as a clock in the block-level SGDC file by mistake, the block-level CDC verification may be inaccurate.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Clock Mismatch Spreadsheet*.

To fix this violation, perform appropriate actions based on the following cases:

- If a top-level clock reaches to an unconstrained clock port of a block

  Action: Specify the clock constraint on the clock port of the reported block instance and analyze the specification or propagation of the top-

level clock.

- If a block-level clock port is not driven from a top-level clock port.

  Action: Open the schematic and perform the following actions:

  - Analyze the top-level design for propagation of a clock to the block port.

    Check if the path of the top-level clock is blocked before reaching to the clock port of the block. In this case, fix the logic accordingly.
  - Check if the top-level net driving the clock port of a block is a clock, but it is not defined in top-level SGDC file. In this case, define the clock in the SGDC file.
  - If the block port is not a clock but it is defined as a clock in block-level SGDC file by mistake, perform the following actions:
    - Remove the clock specification from block-level SGDC file.
    - Re verify the block-level CDC verification.

Message 2

The following message appears for the *<inst-name>* instance that has *Clock Domain Mismatch Spreadsheet*:

```
[AcAbsV1_3] [WARNING] Instance '<inst-name>' of block
'<module-name>' has <num> clock domain mismatches. Reason: Same
domain block clocks connected to different domain top level
clocks
```

Potential Issues

This violation appears due to *Clock Domain Mismatch* when any of the following cases hold true:

- If multiple clock ports in the same domain of an abstract view are triggered from top-level clocks of different domains

  See *Example 1 - Clock Domain Mismatch*.
- If virtual clock specified at a block port and the clock port are in the same domain of an abstract view and they are triggered from top-level clocks of different domains

  See *Example 2 - Clock Domain Mismatch*.

- If virtual clocks *<virtual-clock1>* and *<virtual-clock2>* specified at the ports *<block-port1>* and *<block-port2>*, respectively, are in the same domain of an abstract view and are triggered from top-level clocks of different domains

  See *Example 3 - Clock Domain Mismatch*.

Consequences of Not Fixing

The consequences vary based on the following situations:

- If multiple clock ports in the same domain of an abstract view are triggered from the top-level clocks of different domains.

  Consequence: SpyGlass may map the reported virtual clock to an incorrect top-level domain. This may result in spurious synchronization violations during the block verification stage.

- If top-level clocks of the same domain trigger block ports of a different domain.

  Consequence: It may result in spurious synchronization results during verification phase of higher-level blocks.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Clock Domain Mismatch Spreadsheet*.

To fix this violation, perform appropriate actions based on the following cases:

- If multiple clock ports in the same domain of an abstract view are triggered from the top-level clocks of different domains.

  Action:

  a. Analyze the specification or propagation of top-level clocks.

  b. Ensure that the specification of the same domain virtual clocks is consistent with the specification of top-level clocks identified in the first step.

- If top-level clocks of the same domain trigger block ports of a different domain.

  Action:

a. Verify the specification of different domains on multiple clock ports.
b. Analyze the specification or propagation of the top-level clock.

Message 3

The following message appears for the *<inst-name>* instance that has *Clock Domain Mismatch*:

```
[AcAbsV1_2] [WARNING] Instance '<inst-name>' of block '<module-name>' has <num> clock domain mismatches. Reason: Different domain block clocks connected to same domain top level clocks
```

Potential Issues

This violation appears due to *Clock Domain Mismatch* when multiple clock ports in different domains of an abstract view are triggered from the top-level clocks of same domain. See *Example 4 - Clock Domain Mismatch*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 4

The following message appears for the *<inst-name>* instance that has *Virtual Clocks Mismatch*:

```
[AcAbsV1_9] [WARNING] Instance '<inst-name>' of block '<module-name>' has <num> virtual clock mismatches. Reason: No top-level domain reaches to block ports
```

Potential Issues

This violation appears in case of *Virtual Clocks Mismatch* when no top-level domain reaches to block ports.

See *Example 1 - Virtual Clocks Mismatch*.

Consequences of Not Fixing

If you do not fix these violations, SpyGlass analysis may produce inaccurate synchronization results during block verification, thereby generating incorrect abstract view model.

This may further generate incorrect synchronization violations in the SoC verification stage.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Virtual Clocks Mismatch Spreadsheet*.

To fix this violation, perform the following actions:

- Analyze design connectivity between the top-level sequential element and a block input port.
- Verify the virtual clock specified by the *abstract_port* or input constraint.

Message 5

The following message appears for the *<inst-name>* instance that has *Virtual Clocks Mismatch*:

[AcAbsV1_8] [WARNING] Instance '<inst-name>' of block '<module-name>' has <num> virtual clock mismatches. Reason: Conflicting domains reach to block ports

Potential Issues

This violation appears in case of *Virtual Clocks Mismatch* when conflicting domains reach block ports.

See *Example 2 - Virtual Clocks Mismatch*.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 6

The following message appears for the *<inst-name>* instance that has *Case Analysis Mismatch*:

[AcAbsV1_4] [WARNING] Instance '<inst-name>' of block '<module-name>' has <num> case analysis mismatches

NOTE: *For this message, see:*

- *Case Analysis Mismatch Spreadsheet*
- *Example - Case Analysis Mismatch*

Potential Issues

See *Case Analysis Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If the specified value at the block-level port is incorrect, block-level CDC verification is inaccurate.
- If the specified value at the block-level port is correct but constant propagation at the top-level is incorrect, it indicates a logical issue at the top-level because of which incorrect value is propagated at the block-level.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Case Analysis Mismatch Spreadsheet*.

To fix this violation, perform appropriate actions based on the following cases:

- If block-level ports are constrained to values that do not match with constant values propagated from the top-level

Action:

a. Check the value specification of the *set_case_analysis* constraint on a block port.

b. Analyze the top-level design for propagation of a constant value to the block port.

- If a constant value propagates from the top-level, but the port of the abstract view is not constrained with the *set_case_analysis* constraint.

  Action:

  a. Analyze the top-level design for propagation of a constant value to the block port.

  b. Specify the *set_case_analysis* constraint on the block port.

- If a block port is constrained with the *set_case_analysis* constraint, but no constant value propagates from the top-level

  Action:

  a. Analyze the top-level design for propagation of a constant value to the block port.

  b. Remove the *set_case_analysis* constraint if a valid constant value does not reach the block port.

Message 7

The following message appears for the *<inst-name>* instance that has *Quasi Static Mismatch*:

```
[AcAbsV1_11] [WARNING] Instance '<inst-name>' of block
'<module-name>' has <num> quasi-static mismatches
```

NOTE: *For this message, see:*

- *Quasi static Mismatch Spreadsheet*
- *Example - Quasi Static Mismatch*

Potential Issues

See *Quasi Static Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If a top-level quasi-static signal reaches a block port on which a quasi_static constraint has not been specified.

  Consequence: The design may not operate in the desired mode. In addition, the *quasi_static* constraints would not be propagated to block outputs.

- If a *quasi_static* constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

  Consequence: Some crossings in the design may not be detected.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Quasi static Mismatch Spreadsheet*.

To fix this violation, perform appropriate actions based on the following cases:

- If a top-level quasi-static signal reaches a block port on which a *quasi_static* constraint has not been specified.

  Action:

  a. Specify *quasi_static* constraint on a block port, or

  b. Analyze the specification or propagation of the top-level quasi-static signal to the block port.

- If a *quasi_static* constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

  Action:

  a. If the block quasi-static specification is correct, add the missing *quasi_static* at the top-level, else

  b. Remove the *quasi_static* constraint from the block port.

Message 8

The following message appears for the *<inst-name>* instance that has *Data Path Domain Mismatch*:

```
[AcAbsV1_6] [WARNING] Instance '<inst-name>' of block
```

'<module-name>' has <num> data path domain mismatches

NOTE: *For this message, see:*

- *Data Path Domain Mismatch Spreadsheet*
- *Example - Data Path Domain Mismatch*

Potential Issues

See *Data Path Domain Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during verification phase of the hierarchical verification flow.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Data Path Domain Mismatch Spreadsheet*.

To fix this violation, perform the following steps:

1. Analyze the design connectivity between the top-level sequential element and the block input port.
2. Check if the clock domain specified by the abstract_port or the input constraint is consistent with the clock domains driving sequential elements identified in the first step.
3. Check SGDC (*abstract_port*) back-annotation for block *abstract_port* for which the violation was reported and back-annotation of top level sequential element to indicate the differing clock.

Message 9

The following message appears for the *<inst-name>* instance that has *Reset Mismatch*:

[AcAbsV1_5] [WARNING] Instance '<inst-name>' of block '<module-name>' has <num> reset mismatches

NOTE: *For this message, see:*

- *Reset Mismatch Spreadsheet*
- *Example - Reset Mismatch*

Potential Issues

See *Reset Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If a top-level reset reaches a block port for which no *reset* constraint is specified.

  Consequence: Some potential resets may not propagate during the verification of the abstract view. This may result in the following:

  - The block may not achieve its initial state.
  - In the absence of synchronous resets, SpyGlass may report violations related with unsynchronized clock domains.

- If the reset constraint is specified for a block-level port, but no top-level reset drives that block-level port.

  Consequence: The reported port of an abstract view may not be considered as a valid reset signal. This may alter the initial state of the block during verification.

- If an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.

  Consequence:

  - Incorrect reset analysis may happen at the block-level. That is, the initial state of the block may get altered during its verification.
  - SpyGlass may generate incorrect clock domain violations during block verification if synchronous resets are not properly specified.

- If the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

  Consequence:

  - It may result in an incorrect initial state of an abstract view during verification.

- It may generate spurious reset simulation results for the abstract view.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Reset Mismatch Spreadsheet*.

To fix this violation, perform appropriate actions based on the following cases:

- If a top-level reset reaches a block port for which no *reset* constraint is specified.

  Action: Perform the following steps:

  a. Specify the *reset* constraint on the reported block port.

  b. Analyze the specification or propagation of the top-level reset to the block.

- If the *reset* constraint is specified for a block-level port, but no top-level reset drives that block-level port.

  Action: Perform the following actions:

  a. Remove the *reset* constraint from the reported block port.

  b. Analyze the specification or propagation of a top-level reset to the block port.

- If an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.

  Action:

  a. Specify an appropriate *reset* constraint on the block-level port.

  b. Analyze the specification or propagation of the top-level reset to a block.

- If the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

  Action:

  a. Check the value specified by the *reset* constraint on the block port.

  b. Assign a proper value in the *reset* constraint.

  c. Verify that the top-level reset of the block port has the same active value identified in step 2.

Message 10

The following message appears for the *<inst-name>* instance that has *Qualifier Mismatch*:

[AcAbsV1_7] [WARNING] `Instance '<inst-name>' of block '<module-name>' has <num> qualifier mismatches`

NOTE: *For this message, see:*

- *Qualifier Mismatch Spreadsheet*
- *Example - Qualifier Mismatch*

Potential Issues

See *Qualifier Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during the block verification stage.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Qualifier Mismatch Spreadsheet*.

To fix this violation, perform appropriate actions based on the following cases:

- If clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint for an abstract view exist in the same top-level domain.

  Action: Perform the following actions:

  - Analyze the clocks specified in the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint.
  - Analyze specification or propagation of top-level clocks.

- If a synchronizer does not reach to an input port of a block for which the `-sync` argument of the *abstract_port* constraint is specified.

  Action: Perform the following actions:

- Analyze the fan-in cone of an input port for the presence of a synchronizer.
- Remove the `-sync` argument from the *abstract_port* constraint.

Message 11

The following message appears for the *<inst-name>* instance that has *Combo Check Mismatch*:

```
[AcAbsV1_10] [WARNING] Instance '<inst-name>' of block
'<module-name>' has <num> combo check mismatches
```

NOTE: *For this message, see:*

- *Combo Check Mismatch Spreadsheet*
- *Example - Combo Check Mismatch*

Potential Issues

See *Combo Check Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, it results in an incorrect setup at the block level.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *Combo Check Mismatch Spreadsheet*.

To fix this violation, perform the following actions:

- Remove the `-combo no` argument from the *abstract_port* constraint.
- Analyze the combinational logic between the sequential element and block-level input port.

Message 12

The following message appears for the *<inst-name>* instance that has *num_flops Mismatch*:

```
[AcAbsV1_12] [WARNING] Instance '<inst-name>' of block
'<module-name>' has <num> combo num_flops mismatches
```

NOTE: *For this message, see:*

- *num_flops Mismatch Spreadsheet*
- *Example - num_flops Mismatch*

Potential Issues

See *num_flops Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following may occur depending upon different situations:

- If the num_flops constraint is incorrectly specified in the block-level SGDC file, it may result in inaccurate block-level CDC verification.
- If domains of top-level clocks are not specified correctly, it may result in inaccurate top-level CDC verification.

How to Debug and Fix

Double-click on the violation to open the message-based spreadsheet. For details on this spreadsheet, see *num_flops Mismatch Spreadsheet*.

To fix this violation, perform the following actions:

- Analyze the clock specification in the `-from_clk` and `-to_clk` arguments of the *num_flops* constraint.
- Analyze the specification or propagation of top-level clocks.

Example Code and/or Schematic

This section covers the following examples:

- *Example - Clock Mismatch*
- *Example 1 - Clock Domain Mismatch*
- *Example 2 - Clock Domain Mismatch*

- *Example 3 - Clock Domain Mismatch*
- *Example 4 - Clock Domain Mismatch*
- *Example 5 - Clock Domain Mismatch*
- *Example 1 - Virtual Clocks Mismatch*
- *Example 2 - Virtual Clocks Mismatch*
- *Example - Case Analysis Mismatch*
- *Example - Quasi Static Mismatch*
- *Example - Data Path Domain Mismatch*
- *Example - Combo Check Mismatch*
- *Example - Qualifier Mismatch*
- *Example - Reset Mismatch*
- *Example - num_flops Mismatch*

Example - Clock Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1,clk1,clk2,clk3,
  clk4,clk5,in2,in3,out,sel);
input in1,in2,clk2,clk1,clk4,
      clk5,sel,in3,clk3;
output out;
wire clk_2, seq_out;
reg clk_div2, clk_latch;
reg temp, syn1, syn2;
PLL p1(.clk2(clk2), .clk3(clk3),
      .clk_out(clk_2));
SYNCP2QV15 seg_cell (.D(in2),
    .CK(clk4), .Q(seq_out));
wire bb_out;
always @(posedge clk1)
 clk_div2= in3;
 always @(sel)
    if(sel)
    clk_latch = clk5;
 always @(posedge clk_2)
    begin
    syn1 <=temp;
    syn2<=syn1;
    end
assign sync2_tmp = syn2 && in2;
assign clk_absport = clk_2 && seq_out
     && clk_div2 && clk_latch && in1;
 BB b1(.clk1(clk1),.clk2(clk_absport),
 .clk3(clk3),.in3(sync2_tmp),.out(out),
 .out1(out));
 BB b2(.clk1(clk1),.clk2(clk_absport),
 .clk3(clk3),.in3(sync2_tmp),.out(out),
 .out1(out));
endmodule
module BB(clk1,clk2,clk3,in1,out,out1, in3, in2);
input clk1,clk2,in1,clk3;
input in3, in2;
output out;
output out1;
assign out1 = clk1;
endmodule
module PLL(clk1, clk2, clk3 , clk_out);
input clk1, clk2 , clk3;
output clk_out;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1
clock -name clk2
clock -name clk3
clock -name clk4
clock -name clk5
set_case_analysis -name sel -value 1
assume_path -name PLL -input clk2
           -output clk_out
sgdc -import BB  block.sgdc
```

// block.sgdc

```
current_design BB
clock -name clk1
```

For the above example, the *Ac_abstract_validation01* rule reports two *Clocks Mismatch* violations for the `b1` and `b2` instances.

The following figure shows the *Clock Mismatch Spreadsheet* for this violation for the `b2` instance:

A ID	B Block Port Name	C Block Clock Defined	D Top Clock
12	clk2	no	top.clk2,top.clk5
13	clk3	no	top.clk3

FIGURE 345. The Clock Mismatch Spreadsheet

The information in the above spreadsheet indicates that top-level clocks reach to the clock ports `clk2` and `clk3` of the abstract block, but no *clock* constraint is defined on these clock ports.

Click on the link *12* in the above spreadsheet and open the schematic, as shown below:

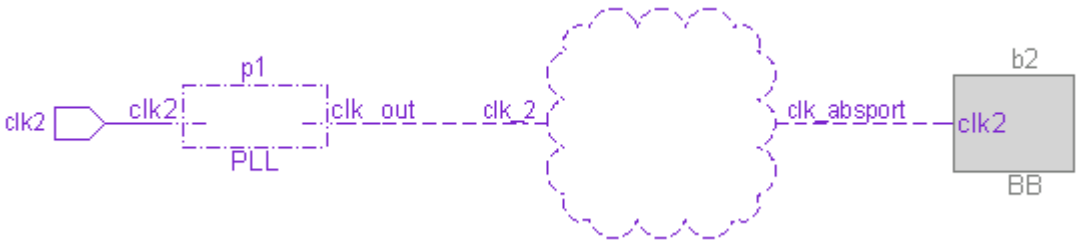


FIGURE 346. Schematic Showing Clock Mismatch

Example 1 - Clock Domain Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1,clk1, clk2, clk3,
   clk4, clk5, in2,in3, out,sel);
input in1,in2,clk1,clk2,sel,in3,
   clk3,clk4, clk5;
output out;
assign clk1_2_3 = clk3 & clk1 && clk2;
assign clk2_3_4 = clk2 && clk4 && clk3;
BB b1(.clk1(clk1_2_3),.clk2(clk1_2_3),
  .clk3(clk2_3_4),.in3(sync2_tmp),
  .out(out), .out1(out));
BB b2(.clk1(clk2_3_4),.clk2(clk2_3_4),
  .clk3(clk1_2_3),.in3(sync2_tmp),
  .out(out), .out1(out));
endmodule
module BB(clk1,clk2,clk3,in1,
       out,out1,in3,in2);
input clk1,clk2,in1,clk3;
input in3, in2;
output out;
output out1;
assign out1 = clk1;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
clock -name clk3 -domain domain3
clock -name clk4 -domain domain4
sgdc -import BB  block.sgdc
```

// block.sgdc

```
current_design BB
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
clock -name clk3 -domain domain2
```

For the above example, the *Ac_abstract_validation01* rule reports multiple violations for *Clock Domain Mismatch*. The following *Clock Domain Mismatch Spreadsheet* shows one of the violation:

A	B	C	D
ID	Block Domain	Block Clocks	Top Clocks
2	domain2	clk3 clk2	top.clk4 top.clk1

FIGURE 347. The Clock Domain Mismatch Spreadsheet

The information in the above spreadsheet indicates that multiple clocks, `clk2` and `clk3`, of the same domain, `domain2`, of the abstract view are

triggered from the top-level clocks, `clk1` and `clk4`, of different domains.

Click on the link *4* in the above spreadsheet and open the schematic, as shown below:

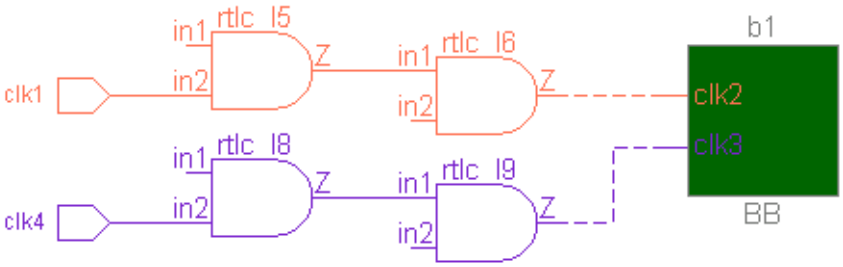


FIGURE 348. Schematic Showing Clock Domain Mismatch

Example 2 - Clock Domain Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(clk1,clk2,clk3,
    in3,in1,in2,out1);
input clk1,clk2,clk3,in3,in1,in2;
output out1;
reg r1,r2,r3;
bbox B1 (.i1(r1),.i2(r2 & r3),
  .ck1(clk1),.ck2(clk2),
  .ck3(clk3),.o1(out1),
  .o2(wr1));
always@(posedge clk2)
    r1 <= in1;
always@(posedge clk3)
    r3 <= in3;
always@(posedge clk1)
    r2 <= in2;
endmodule

module bbox(input i1,i2,ck1,
 ck2,ck3, output o1,o2);
endmodule
```

// constr.sgdc

```
current_design test
clock -name clk1 -domain d1
clock -name clk2
clock -name clk3 -domain d1
sgdc -import bbox block.sgdc
```

// block.sgdc

```
current_design bbox
clock -name ck1 -domain d1
clock -name ck2 -domain d2
clock -name ck3 -domain d1
clock -tag vck1  -domain d1
abstract_port -module bbox
  -ports i1 -clock vck1
```

For the above example, the *Ac_abstract_validation01* rule reports two *Clock Domain Mismatch*.

The following figure shows the *Clock Domain Mismatch Spreadsheet* for this violation:

A ID	B Block Domain	C Block Clocks	D Top Clocks
7	d1	vck1 ck1	test.clk1 test.clk2
8	d1	vck1 ck3	test.clk3 test.clk2

FIGURE 349. The Clock Domain Mismatch Spreadsheet

The information in the above spreadsheet indicates that the `vck1` virtual clock in the `d1` domain of the abstract block is triggered by top-level clocks of different domains.

Click on link *7* in the above spreadsheet and open the schematic, as shown below:

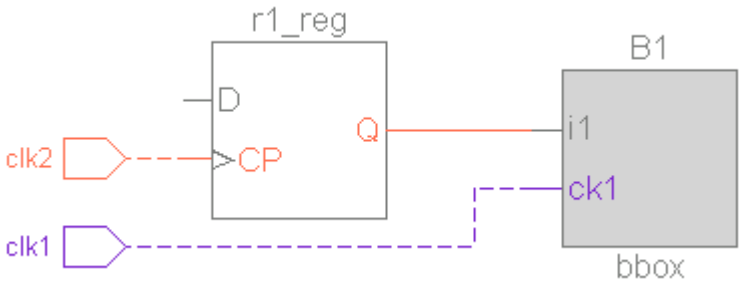


FIGURE 350. Schematic Showing Clock Domain Mismatch

Example 3 - Clock Domain Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(input in1,in2,in3,
        clk1,clk2,output out);
wire vclk1,vclk2;
reg temp;
assign gclk = vclk1  & vclk2;
always@(posedge gclk)
   temp <= in1;
block inst(temp,in2,in3,
         clk1,clk2,out);
endmodule
module block(input in1,in2,
   in3,clk1,clk2,output out);
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1
clock -name clk2
abstract_port -module top -ports in2
              -clock clk1
abstract_port -module top -ports in3
              -clock clk2
sgdc -import block block.sgdc
```

// block.sgdc

```
current_design block
clock -name clk1
clock -name clk2
clock -tag vclk1 -domain d1
clock -tag vclk2 -domain d1
abstract_port -module block
     -ports in2 -clock vclk1
input -name in3 -clock vclk2
```

For the above example, the *Ac_abstract_validation01* rule reports one *Clock Domain Mismatch*, as shown in the following *Clock Domain Mismatch Spreadsheet*:

A	B	C	D
ID	Block Domain	Block Clocks	Top Clocks
3	d1	vclk2 vclk1	top.clk2 top.clk1

FIGURE 351. The Clock Domain Mismatch Spreadsheet

The information in the above spreadsheet shows that the `vclk1` virtual clock of the `in2` port and `vclk2` virtual clock of the `in3` port are in the same domain `d1`, and these virtual clocks are triggered by different top-level clocks `top.clk1` and `top.clk2`.

Click on the link *7* and open the schematic:

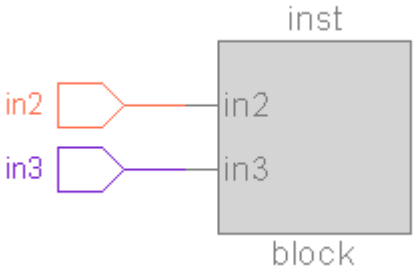


FIGURE 352. Schematic Showing Clock Domain Mismatch

Example 4 - Clock Domain Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module block(clk1,clk2,d1,en,q1);
input clk1,clk2;
input [0:3] d1;
input en;
output [0:3] q1;
reg [0:3] src,dest;
reg enable,sync1,sync2;
always@(posedge clk1)
   begin
    src <= d1;
    enable<= en;
   end
always@(posedge clk2)
    begin
     sync1 <= enable;
     sync2 <= sync1;
    end
always@(posedge clk2)
    if(sync2)
     dest <= src;
assign q1 = dest;
endmodule
module top(clk1,clk2,clk3,testclk,testen,in1,in2,sel1,sel2,out1,out2);
input clk1,clk2,clk3;
input [0:3] in1,in2;
output [0:3] out1,out2;
input sel1,sel2;
input testclk;
input testen;
reg [0:3] t1;
wire [0:3] wr1,wr2;
assign mclk1 = (testen)? clk2 : testclk;
assign mclk2 = (testen)? clk3 : testclk;
block B1(.clk1(clk1),.clk2(clk1),.d1(in1),.en(sel1),.q1(wr1));
block B2(.clk1(clk2),.clk2(clk2),.d1(in2),.en(sel2),.q1(wr2));
always@(posedge clk2)
    t1 <= wr1;
assign out1 = t1;
assign out2 = wr2;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d2
clock -name clk3 -domain d3
clock -name testclk -domain d4
set_case_analysis -name testen
   -value 0
sgdc -import block block.sgdc
```

// block.sgdc

```
current_design block
clock -name clk1 -domain d1
clock -name clk2
output -name q1 -clock clk2
```

For the above example, the *Ac_abstract_validation01* rule reports one *Clock Domain Mismatch*. The details are shown in the following *Clock Domain Mismatch Spreadsheet*:

A	B	C	D
ID	Block Clocks	Top Clocks	Top internal domain Tag
3	clk1 clk2	top.clk2 top.clk2	1

FIGURE 353. The Clock Domain Mismatch Spreadsheet

The information in the above spreadsheet indicates that the `clk1` and `clk2` clocks of the abstract view are of different domains, and they are triggered by the top-level clocks of the same domain.

In the above spreadsheet, click on link *3* and open the schematic. The following figure shows the schematic:

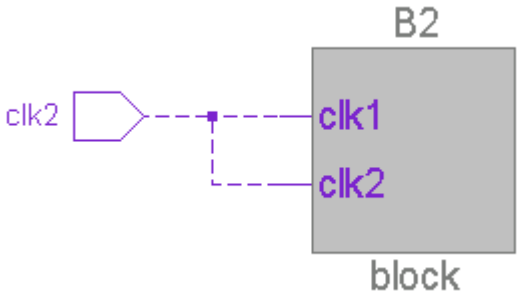


FIGURE 354. Schematic Showing Clock Domain Mismatch

Example 5 - Clock Domain Mismatch

Consider the following files specified for SpyGlass analysis:

Verilog File

```
module top(in, clk1, clk2, out);
input in, clk1, clk2;
output out;
wire w1;
assign w1 = in;
block U1 (w1, clk1, clk2, out);
endmodule

module block (in,ck1, ck2, out);
input in , ck1, ck2;
output reg out;
always @(posedge ck1)
   out <= in;
endmodule
```

Block sgdc File

```
current_design block
clock -name ck1  -tag T1 -domain d1
clock -name ck2  -tag T2 -domain d2
clock -tag T3 -domain d3
sg_clock_group -group1 { T1 } -group2 { T2 }
sg_clock_group -group1 { T1 } -group2 { T3 }
abstract_port -ports out -clock ck1
```

Top Sgdc File

```
sgdc -import block block.sgdc
clock -name clk1 -tag T1 -domain d1
clock -name clk2 -tag T2 -domain d2
```

For the above example, the *Ac_abstract_validation01* rule reports clock domain mismatch as shown in the graphic below:

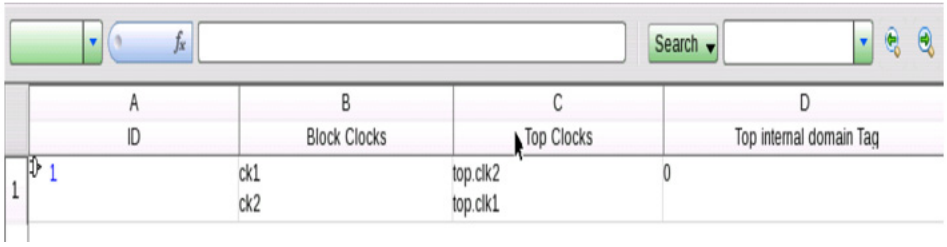


FIGURE 355. The Clock Domain Mismatch Spreadsheet

Example 1 - Virtual Clocks Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(clk1,clk2,in1,in2, out1,out2);
input clk1,clk2;
input in1,in2;
output out1,out2;
reg r1,r2;
wire wr1,wr2;
BBOX B2 (.d1(in1), .clk1(in1), .clk2(in1),
    .out1(w1), .out2(w2), .out_clk(w3));
block B1(.d1(w1 && w3),.d2(w2),.ck1(clk1),
    .ck2(clk2),.out1(wr1),.out2(wr2));
assign out1 = wr1;
assign out2 = wr2;
endmodule
module BBOX (d1, clk1, clk2, out1, out2,
    out_clk);
input d1, clk1, clk2;
output out1, out2, out_clk;
endmodule
module block(d1,d2,ck1,ck2,out1,out2);
input d1,d2;
input ck1,ck2;
output out1,out2;
reg out1,out2;
always@(posedge ck1)
    out1 <= d1;
 always@(posedge ck1)
    out2 <= d2;
endmodule
```

// top.sgdc

```
current_design test
clock -name clk1
clock -name clk2
sgdc -import block block.sgdc
sgdc -import BBOX bbox.sgdc
```

// block.sgdc

```
current_design block
clock -name ck1
clock -name ck2

abstract_port -module block
   -ports d1 -clock V1
abstract_port -module block
   -ports d2 -clock V2
```

// bbox.sgdc

```
current_design BBOX
abstract_port -module BBOX
     -ports out1 -clock clk1
abstract_port -module BBOX
     -ports out2 -clock clk2
clock -name out_clk
```

For the above example, the *Ac_abstract_validation01* rule reports two *Virtual Clocks Mismatch* violations for the `test.B1` instance of the `block` block because no top-level domain reaches to the block ports.

The following figure shows the *Virtual Clocks Mismatch Spreadsheet* for this violation:

A ID	B Virtual Clock Name	C Port Name	D Top Clocks	E Top Internal Domain Tag
1	V1	d1	<unconstrained>	-
2	V2	d2	<unconstrained>	-

FIGURE 356. The Virtual Clocks Mismatch Spreadsheet

The information in the above spreadsheet indicates that no top-level clock reaches to the `V1` and `V2` virtual clocks of the abstract block `B1`. The following schematic shows this scenario:

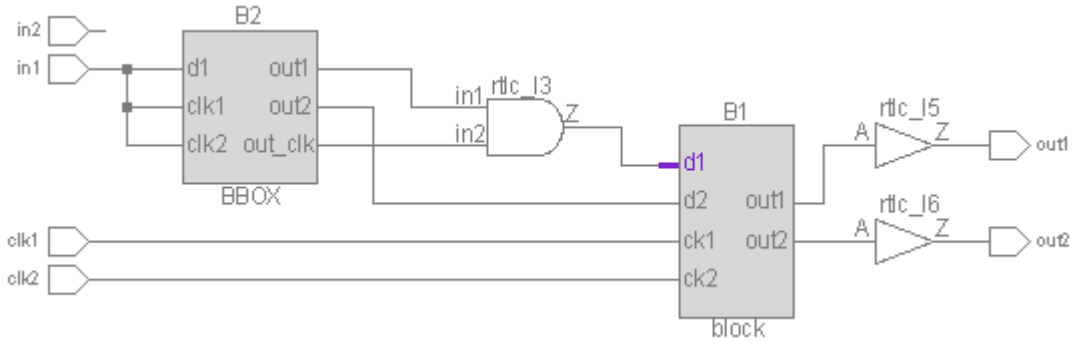


FIGURE 357. Schematic Showing Virtual Clocks Mismatch

Example 2 - Virtual Clocks Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(clk1,clk2,in1,in2, out1,out2);
  input clk1,clk2;
  input in1,in2;
  output out1,out2;
  reg r1,r2;
  wire wr1,wr2;
  BBOX B2 (.d1(in1), .clk1(clk1),
     .clk2(clk2), .out1(w1), .out2(w2));
  always @(posedge clk1)
      r1 <= in1;
  block B1(.d1(w1),.d2(r1),.ck1(clk1),
     .ck2(clk2),.out1(wr1),.out2(wr2));
  assign out1 = wr1;
  assign out2 = wr2;
endmodule
module block(d1,d2,ck1,ck2,out1,out2);
  input d1,d2;
  input ck1,ck2;
  output out1,out2;
  reg out1,out2;
  always@(posedge ck1)
      out1 <= d1;
   always@(posedge ck1)
      out2 <= d2;
endmodule
```

// top.sgdc

```
current_design test
clock -name clk1
clock -name clk2
sgdc -import block block.sgdc
abstract_port -module BBOX
   -ports out1 -clock clk1 clk2
abstract_port -module BBOX
   -ports out2 -clock clk2
```

// block.sgdc

```
current_design block
clock -name ck1
clock -name ck2

abstract_port -module block
    -ports d1 -clock V1
abstract_port -module block
    -ports d2 -clock V1
```

For the above example, the *Ac_abstract_validation01* rule reports two virtual clocks mismatches for the `test.B1` instance of the `block` block because conflicting domains reach to the block ports.

The following figure shows the *Virtual Clocks Mismatch Spreadsheet* for this violation:

A ID	B Virtual Clock Name	C Port Name	D Top Clocks	E Top Internal Domain Tag
1	V1	d2	test.clk1	0
2	V1	d1	test.clk1,test.clk2	2

FIGURE 358. The Virtual Clocks Mismatch Spreadsheet

The information in the above spreadsheet indicates that the `d1` and `d2` ports of the abstract block `B1` are specified with the same virtual clock `V1`, but both these ports are driven with different top-level clocks.

The following schematic shows this scenario:

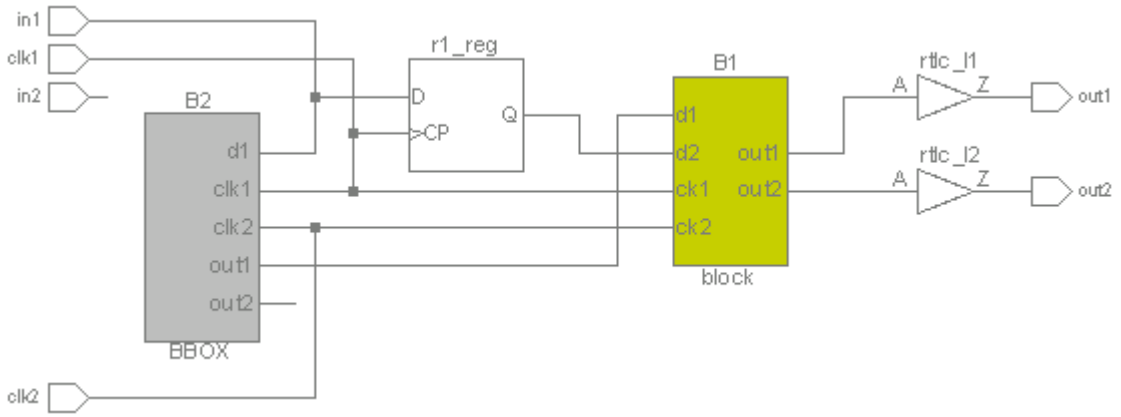


FIGURE 359. Schematic Showing Virtual Clocks Mismatch

Example - Case Analysis Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1,clk1, clk2, clk3, in2,
       in3, out,sel);
input in1,in2,clk1,clk2,sel,in3,clk3;
output out;
reg temp,syn1,syn2, temp1;
assign clk_1 = clk1 & clk2;
wire bb_out;
always @(posedge clk_1)
 temp = in3;
 always @(posedge clk1)
    temp1 = in2;
 always @(posedge clk2)
    begin
    syn1 <=temp;
    syn2<=syn1;
    end
    assign sync2_tmp = syn2 && temp1;
BB b1(.clk1(clk1),.clk2(clk2),
   .clk3(clk3),.in3(sync2_tmp),
   .out(out), .out1(out));

BB b2(.clk1(clk1),.clk2(clk2),
   .clk3(clk3),.in3(sync2_tmp),
    .out(out), .out1(out));
endmodule
module BB(clk1,clk2,clk3,in1,out,out1, in3, in2);
input clk1,clk2,in1,clk3;
input in3, in2;
output out;
output out1;
assign out1 = clk1;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
sgdc -import BB  block.sgdc
```

// block.sgdc

```
current_design BB
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
set_case_analysis -name in3 -value 0
set_case_analysis -name in2 -value 0
abstract_port -module BB -ports in3
   -clock clk2 -from in3 -to in1
   -sync active

abstract_port -module BB -ports in3
    -clock clk1
```

For the above example, the *Ac_abstract_validation01* rule reports two *Case Analysis Mismatch* violations for the `b1` and `b2` blocks. The details for the `b1` block are shown in the following *Case Analysis Mismatch Spreadsheet*:

A ID	B Port Name	C Block Value	D Top Value
1	in3	0	-
2	in2	0	-

FIGURE 360. The Case Analysis Mismatch Spreadsheet

The above spreadsheet shows that the value of the *set_case_analysis* constraint is set to 0 on the `in3` and `in2` ports of the `b1` block, but no simulated value reaches to the top-level net that is connected to these ports. Therefore, this rule reports a violation.

In the above spreadsheet, click on link 1 and open the schematic. The following figure shows the schematic:

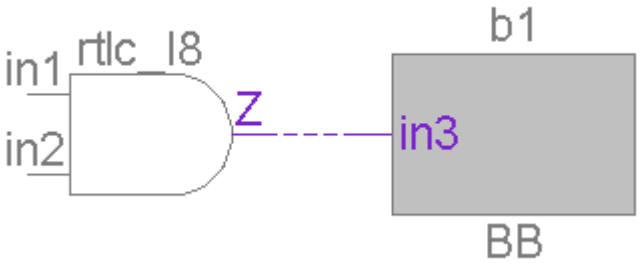


FIGURE 361. Schematic Showing Case Analysis Mismatch

Example - Quasi Static Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(in1,in2,in3,clk1,clk2,out1,out2);
input [0:1] in1,in3;
input in2;
input clk1,clk2;
output out1,out2;
reg out1;
wire wr2;
BBOX bb1(.a(in2),.b(in1[0]),.clk(clk2),.out(wr2));
block b1(in1,in2,clk1,w1);
block b2(in3,wr2,clk1,w2);
always@(clk2)
    out1 <= w1 & w2;
endmodule
module BBOX(input a,b,clk,output out);
endmodule
module block(a,b,clk,out);
input [0:1] a;
input b,clk;
output out;
endmodule
```

// test.sgdc

```
current_design test
clock -name clk1
clock -name clk2
quasi_static -name in1
quasi_static -name wr2
sgdc -import block block.sgdc
```

// block.sgdc

```
current_design block
clock -name clk
```

For the above example, the *Ac_abstract_validation01* rule reports a violation for *Quasi Static Mismatch*. The violation details are shown in the following *Quasi static Mismatch Spreadsheet*:

A	B	C	D
ID	Port Name	Block port quasi_static	Top quasi_static
1	a[1:0]	no	yes

FIGURE 362. The Quasi Static Mismatch Spreadsheet

The information in the above spreadsheet indicates that a top-level quasi static signal reaches the `a[1:0]` port of the abstract block, but no *quasi_static* constraint is defined on this port.

Click on link 1 in the above spreadsheet and open the schematic, as shown below:

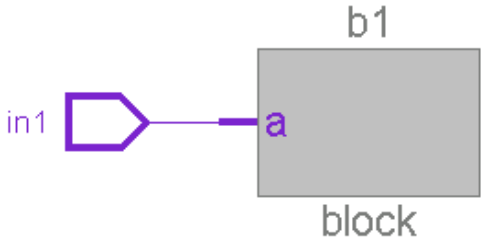


FIGURE 363. Schematic Showing Quasi Static Mismatch

Example - Data Path Domain Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1,clk1, clk2, clk3,
           in2,in3, out,sel);
input in1,in2,clk1,clk2,sel,in3,clk3;
output out;
reg temp,syn1,syn2, temp1;
assign clk_1 = clk1 & clk2;
wire bb_out;
always @(posedge clk_1)
 temp = in3;
 always @(posedge clk1)
    temp1 = in2;
 always @(posedge clk2)
    begin
    syn2<=temp;
    end
    assign sync2_tmp = syn2 && temp1;
BB b1(.clk1(clk_1),.clk2(clk2),
   .clk3(clk3),.in3(sync2_tmp),
   .out(out), .out1(out));

BB b2(.clk1(clk_1),.clk2(clk2),
   .clk3(clk3),.in3(sync2_tmp),
   .out(out), .out1(out));
endmodule

module BB(clk1,clk2,clk3,in1,out,out1, in3, in2);
input clk1,clk2,in1,clk3;
input in3, in2;
output out;
output out1;
assign out1 = clk1;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
sgdc -import BB  block.sgdc
```

// block.sgdc

```
current_design BB
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
set_case_analysis -name in3 -value 0
set_case_analysis -name in2 -value 0
abstract_port -module BB -ports in3
   -clock clk1
```

For the above example, the *Ac_abstract_validation01* rule reports three *Data Path Domain Mismatch* violations each for the `b1` and `b2` instances.

The following figure shows the *Data Path Domain Mismatch Spreadsheet* for the `b1` instance:

A ID	B Port Name	C Block Domain(s)	D Top Domain(s)
6	in3	-	top.clk1
7	in3	-	top.clk2
8	in3	top.clk1,top.clk2(clk1)	-

FIGURE 364. The Data Path Domain Mismatch Spreadsheet

In the above spreadsheet:

- The first row indicates that the `in3` pin of the abstract block is driven from a sequential instance driven by the `top.clk1` clock, but no *abstract_port* constraint is defined for that pin.
- The second row indicates that the `in3` pin of the abstract block is driven from a sequential instance driven by `top.clk2` clock, but no *abstract_port* constraint is defined for that pin.
- The third row indicates that the `in3` pin of the abstract block is constrained with the `top.clk1` and `top.clk2` clocks (`abstract_port -module BB -ports in3 -clock clk1`), but no sequential cell drives this pin.

In the above spreadsheet, click on link 6 to open the schematic. The following figure shows the schematic:

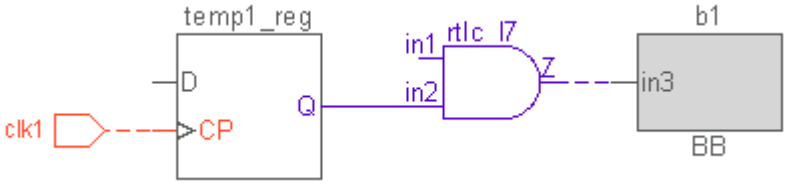


FIGURE 365. Schematic Showing Data Path Domain Mismatch

Example - Combo Check Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1,clk1, clk2, clk3,
   in2,in3, out,sel);
  input in1,in2,clk1,clk2,sel,in3,clk3;
  output out;
  reg temp,syn1,syn2;
  always @(posedge clk1)
   syn1 <= in1;
  always @(posedge clk2)
   syn2 <= in2;
  assign sync2_tmp = syn1 && syn2;
  BB b1(.clk1(clk1),.clk2(clk2),
       .in3(sync2_tmp),.out(out),
       .out1(out));

  BB b2(.clk1(clk1),.clk2(clk2),
      .in3(sync2_tmp),.out(out),
      .out1(out));
endmodule
  module BB(clk1,clk2,in1,out,out1,
          in3, in2);
  input clk1,clk2,in1;
  input in3, in2;
  output out;
  output out1;
  assign out1 = clk1;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
sgdc -import BB  block.sgdc
```

// block.sgdc

```
current_design BB
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
set_case_analysis -name in2 -value 0
set_case_analysis -name in3 -value 0
 abstract_port -module BB -ports in3
      -clock clk2 -combo no
```

For the above example, the *Ac_abstract_validation01* rule reports one *Combo Check Mismatch* violation each for the `b1` and `b2` instance. The violation details for the `b1` instance are shown in the following *Combo Check Mismatch Spreadsheet*:

A	B	C	D
ID	Port Name	Block Combo	Top Combo
7	in3	no	yes

FIGURE 366. The Combo Check Mismatch Spreadsheet

The information in the above spreadsheet indicates that for the `in3` port, the abstract_port constraint has `-combo no` defined. However, the top-level data reaching to this port through a combinational logic.

In the above spreadsheet, click on link *7* and open the schematic. The following figure shows the schematic:

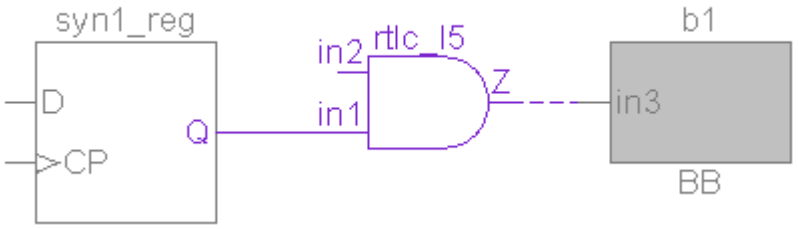


FIGURE 367. Schematic Showing Combo Check Mismatch

Example - Qualifier Mismatch

The following figure shows the *Qualifier Mismatch Spreadsheet* showing information related to *Qualifier Mismatch*:

A	B	C	D	E	F	G	H
ID	Port Nam	lock Source Clo	Block Destination Clocl	op Source Clo	op Destination Cloc	Block Sync Type	op Sync Typ
14	d1	-	-	test.clk1	test.clk4	-	active
15	d1	-	-	test.clk4	test.clk3	-	active
16	d1	test.B1_V1	test.clk2	-	-	active	-
17	d1	test.clk1	test.clk3	-	-	active	-

FIGURE 368. The Qualifier Mismatch Spreadsheet

Information in the above spreadsheet indicates the following:

- (For row ID 14): The top-level qualifier having the source domain as `test.clk1` and the destination domain as `test.clk4` reaches the pin of the abstract view, but no matching qualifier is specified at this pin.

- (For row ID 15): The top-level qualifier having the source domain as `test.clk4` and the destination domain as `test.clk3` reaches the pin of the abstract view, but no matching qualifier is specified at this pin.
- (For row ID 16): No top-level qualifier reaches the abstract block's port for which the abstract_port constraint is defined with the `-sync` argument, the source clock as `test.B1_V` and the destination clock as `test.clk2`.
- (For row ID 17): No top-level qualifier reaches the abstract block's port for which the abstract_port constraint is defined with the `-sync` argument, the source clock as `test.clk1` and the destination clock as `test.clk3`.

Click on the link 14 in the above spreadsheet and open the schematic, as shown in the following figure:

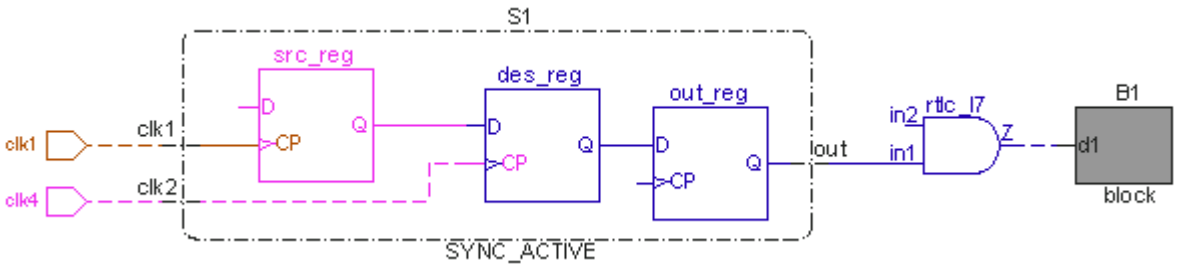


FIGURE 369. Schematic Showing Qualifier Mismatch

Example - Reset Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(input clk1,clk2,rst1,
   rst2,rst3,rst4,rst5,rst6,
   rst7,rst8,in1,in2,output out);

block bbox(.clk1(clk1),.clk2(clk2),
    .rst1(rst1),.rst2(rst2),
    .rst3(rst3),.rst4(rst4),
    .rst5(rst5),.rst6(rst6),.rst7(rst7),
    .rst8(rst8),.in1(in1),.in2(in2),
    .out(out));
endmodule
module block (input clk1,clk2,rst1,
    rst2,rst3,rst4,rst5,rst6,rst7,
    rst8,in1,in2,output out) ;
endmodule
```

// top.sgdc

```
current_design top
clock -name clk1
clock -name clk2
reset -name rst1 -value Z -sync
reset -name rst2 -value z -sync
reset -name rst3 -value X -sync
reset -name rst4 -value x -sync
reset -name rst5 -value 1 -sync
reset -name rst6 -value 1 -sync
reset -name rst7 -value 0 -sync
reset -name rst8 -value 0 -sync
sgdc -import block block.sgdc
```

// block.sgdc

```
current_design block
clock -name clk1
clock -name clk2
reset -name rst1 -value 1 -sync
reset -name rst2 -value 0 -sync
reset -name rst3 -value 0 -sync
reset -name rst4 -value 1 -sync
reset -name rst5 -value x -sync
reset -name rst6 -value X -sync
reset -name rst7 -value z -sync
reset -name rst8 -value Z -sync
```

For the above example, the *Ac_abstract_validation01* rule reports eight *Reset Mismatch* violations. The violation details are shown in the following *Reset Mismatch Spreadsheet*:

A	B	C	D	E	F	G
ID	Port Name	Top Reset	Block Reset Type	Top Reset Type	Block Reset Value	Top Reset Value
1	rst1	top.rst1	Sync	Sync	1	-
2	rst2	top.rst2	Sync	Sync	0	-
3	rst3	top.rst3	Sync	Sync	0	-
4	rst4	top.rst4	Sync	Sync	1	-
5	rst5	top.rst5	Sync	Sync	-	1
6	rst6	top.rst6	Sync	Sync	-	1
7	rst7	top.rst7	Sync	Sync	-	0
8	rst8	top.rst8	Sync	Sync	-	0

FIGURE 370. The Reset Mismatch Spreadsheet

The information in the above spreadsheet indicates that the active value of the top-level resets is different from the active value of the block-level reset ports driven by the top-level resets.

Click on link 1 and open the schematic:

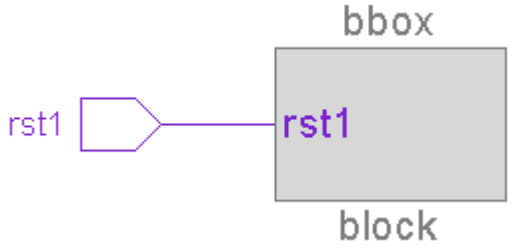


FIGURE 371. Schematic Showing Reset Mismatch

Example - num_flops Mismatch

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(in1,clk1, clk2, clk3,
           in2,in3, out,sel);
input in1,in2,clk1,clk2,sel,in3,clk3;
output out;
reg temp,syn1,syn2, temp1;
assign clk_1 = clk1 & clk2;
wire bb_out;
always @(posedge clk_1)
 temp = in3;
 always @(posedge clk1)
    temp1 = in2;
 always @(posedge clk2)
    begin
    syn2<=temp;
    end
   assign sync2_tmp = syn2 && temp1;
 BB b1(.clk1(clk1),.clk2(clk2),
   .in1(temp1),.in3(sync2_tmp),
    .out(out));

 BB b2(.clk1(clk1),.clk2(clk2),
    .in3(sync2_tmp),.out(out));
endmodule

module BB(clk1,clk2,in1,out, in3, in2);
input clk1,clk2,in1;
input in3, in2;
output out;
assign out = in1;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1 -domain domain1
clock -name clk2 -domain domain2
num_flops -from_clk clk1
          -to_clk clk2 -value 4
sgdc -import BB  block.sgdc
```

// block.sgdc

```
current_design BB
input -name in1 -clock  clk1
num_flops -from_clk clk1
          -to_clk clk2 -value 2
```

For the above example, the *Ac_abstract_validation01* rule reports one *num_flops Mismatch* violation each for the `b1` and `b2` instance.

The following figure shows the *num_flops Mismatch Spreadsheet* for the num_flops mismatch for the `b1` instance:

A	B	C	D	E
ID	num_flops Source	num_flops Destination	Block Value	Top Value
1	clk1	clk2	2	4

FIGURE 372. The num_flops Mismatch Spreadsheet

The information in the above spreadsheet indicates that the value (4) of the *num_flops* constraint defined in the top-level SGDC file for a clock-pair is different from the value (2) of the *num_flops* constraint defined in the block-level SGDC file for the same clock pair.

Therefore, this rule reports a violation due to this mismatch.

Default Severity Label

Warning

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

- Rule-based spreadsheet - Ac_abstract_validation01.csv

  This spreadsheet shows information on all the *Ac_abstract_validation01* rule violations (one violation per row).

  The following figure shows the rule-based spreadsheet of this rule:

A	B	C	D	E	F
ID	Block-Name	Block Instance Name	Problem Type	Count	WAIVED
E	BB	top.b1	Clock Mismatch	2	No
17	BB	top.b2	Clock Mismatch	2	No
B	BB	top.b1	Data Path Domain Mismatch	2	No
14	BB	top.b2	Data Path Domain Mismatch	3	No
8	BB	top.b1	Num_flops Mismatch	1	No
10	BB	top.b2	Num_flops Mismatch	1	No

FIGURE 373. Rule-Based Spreadsheet of the Ac_abstract_validation01 Rule

In the above spreadsheet, double-click on a violation to open the message-based spreadsheet.

NOTE: *If you run the Ac_abstract_validation01 rule in the batch mode, the rule-based spreadsheet contains an additional column,* ***CSV File****. This column shows the path of the corresponding message-based spreadsheet. Refer to this column to correlate the row of the rule-based spreadsheet with the corresponding message-based spreadsheet.*

- Message-based spreadsheet

  This spreadsheet shows the details of the selected violation. For information on different message-based spreadsheets generated by this rule, see *The Spreadsheets of the Ac_abstract_validation01 Rule*.

Ac_abstract_validation02

Mismatch between the abstract block and top-level design

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract view in the context of a higher-level hierarchy.

Prerequisites

Use the `Advanced_CDC` license.

In addition, specify the following details before running this rule:

- The *sgdc* `-import` constraint.
- The *clock* constraint to check for *Clocks Mismatch*, *Data Path Domain Mismatch*, and *Virtual Clocks Mismatch*.
- The *set_case_analysis* constraint to check for *Case Analysis Mismatch*.
- The *quasi_static* constraint to check for *Quasi Static Mismatch*.
- The *reset* constraint to check for *Reset Mismatch*.
- The *qualifier* constraint to check for *Qualifier Mismatch*.
- The *num_flops* constraint to check for *num_flops Mismatch*.
- The *abstract_port* constraint to check for *Combo Check Mismatch*.

Description

The *Ac_abstract_validation02* rule reports a violation for the following types of mismatches between the abstract block and the top-level design:

Clocks Mismatch	*Clock Domain Mismatch*	*Virtual Clocks Mismatch*
Case Analysis Mismatch	*Quasi Static Mismatch*	*Data Path Domain Mismatch*
Reset Mismatch	*Qualifier Mismatch*	*Combo Check Mismatch*
num_flops Mismatch		

The *Ac_abstract_validation02* rule reports all the above mismatches under one spreadsheet, *The Ac_abstract_validation02 Spreadsheet*.

Reporting violations for all the above mismatches may lead to noise and

multiple iterations to fix them. To avoid this, you can restrict this rule to validate only block-level assumptions with respect to the top level by setting *abstract_validate_express* to `yes`.

Clocks Mismatch

This mismatch occurs in the following cases:

- If a top-level clock reaches to a clock port of a block, but that clock port is not constrained by the `clock` constraint.
- If a block-level clock port is not driven from a top-level clock port.

  This can occur when the `clock` constraint is defined on a block port, but a top-level clock does not reach that block port.

Clock Domain Mismatch

This mismatch occurs in the following cases:

- If multiple clock ports in the same domain of an abstract view are triggered from top-level clocks of different domains.
- If virtual clock specified at a block port and the clock port are in the same domain of an abstract view and they are triggered from top-level clocks of different domains
- If virtual clocks *<virtual-clock1>* and *<virtual-clock2>* specified at the ports *<block-port1>* and *<block-port2>*, respectively, are in the same domain of an abstract view and are triggered from top-level clocks of different domains
- If multiple clock ports in different domains of an abstract view are triggered from the top-level clocks of the same domain.

If the `sta_based_clock_relationship` parameter is set to true, SpyGlass CDC reports clock domain mismatch violations based on the `sg_clock_group` constraint specified on the block and the top level rather than the domain specified for the clock.

Virtual Clocks Mismatch

This mismatch occurs in the following cases:

- If multiple ports of the same block specified with the same virtual clock are driven by different domains from top level
- If no top-level clock is reaching the block port specified with a virtual clock

Case Analysis Mismatch

This mismatch occurs in the following cases:

- If there is a mismatch between the following values:
  - Constant value specified by the `set_case_analysis` constraint for a block-level port
  - Constant value propagated from the top-level
- If a simulated value reaches a top-level net connected to a block-level port, but no `set_case_analysis` constraint is specified on the block-level port
- If a simulated value does not reach to a top-level net connected to a block-level port, but the `set_case_analysis` constraint is specified on the block-level port

Quasi Static Mismatch

This mismatch occurs in the following cases:

- If a top-level quasi-static signal reaches a block port on which a `quasi_static` constraint has not been specified.
- If a `quasi_static` constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

Data Path Domain Mismatch

This mismatch occurs if an abstract-block port is driven from a sequential instance, and there is a mismatch between the clock pin driving this sequential instance and the clock specified in the `-clock` argument of the `abstract_port` or the input constraint.

Reset Mismatch

This mismatch occurs in the following cases:

- If a top-level reset reaches a block port for which no `reset` constraint is specified.

- If the reset constraint is specified for a block-level port, but no top-level reset drives that block-level port.
- If an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.
- If the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

Qualifier Mismatch

This mismatch occurs in the following cases:

- If a synchronized signal reaches to an input port of a block for which:
  - The qualifier or the abstract_port constraint is not defined with the -sync argument, or
  - The abstract_port constraint is defined with the -sync argument but clocks specified by the -from or -to argument of that abstract_port constraint do not match with the source or destination clocks of the synchronizer reaching to that input port.
- If a synchronizer does not reach to an input port of a block for which the -sync argument of the abstract_port constraint is specified

Automatically Fixing the abstract_port Constraint of the Reported Port

The **CDC SoC abstract auto update flow** works if a synchronized signal reaches to a block port and any of the following cases is true:

- abstract_port is defined at the block port without -sync and the domain of the synchronized signal matches with the clock specified in abstract_port
- assume_path is defined at the block port

Set the autofix_abstract_port parameter to yes to modify the abstract_port constraint in the context of SoC for the reported port, and save the modified constraints in the *<module_name>_<instance_name>_cdc_soc_abstract_autoupdate.sgdc* file.

By default, in addition to the modified constraints, this file also contains a copy of all the unmodified input side abstract_port constraints present in block-level SGDC file (abstract block). Set autofix_dump_allinputs

to no to generate only the modified constraints.

Figure 374 shows the example of using the autofix_abstract_port parameter:

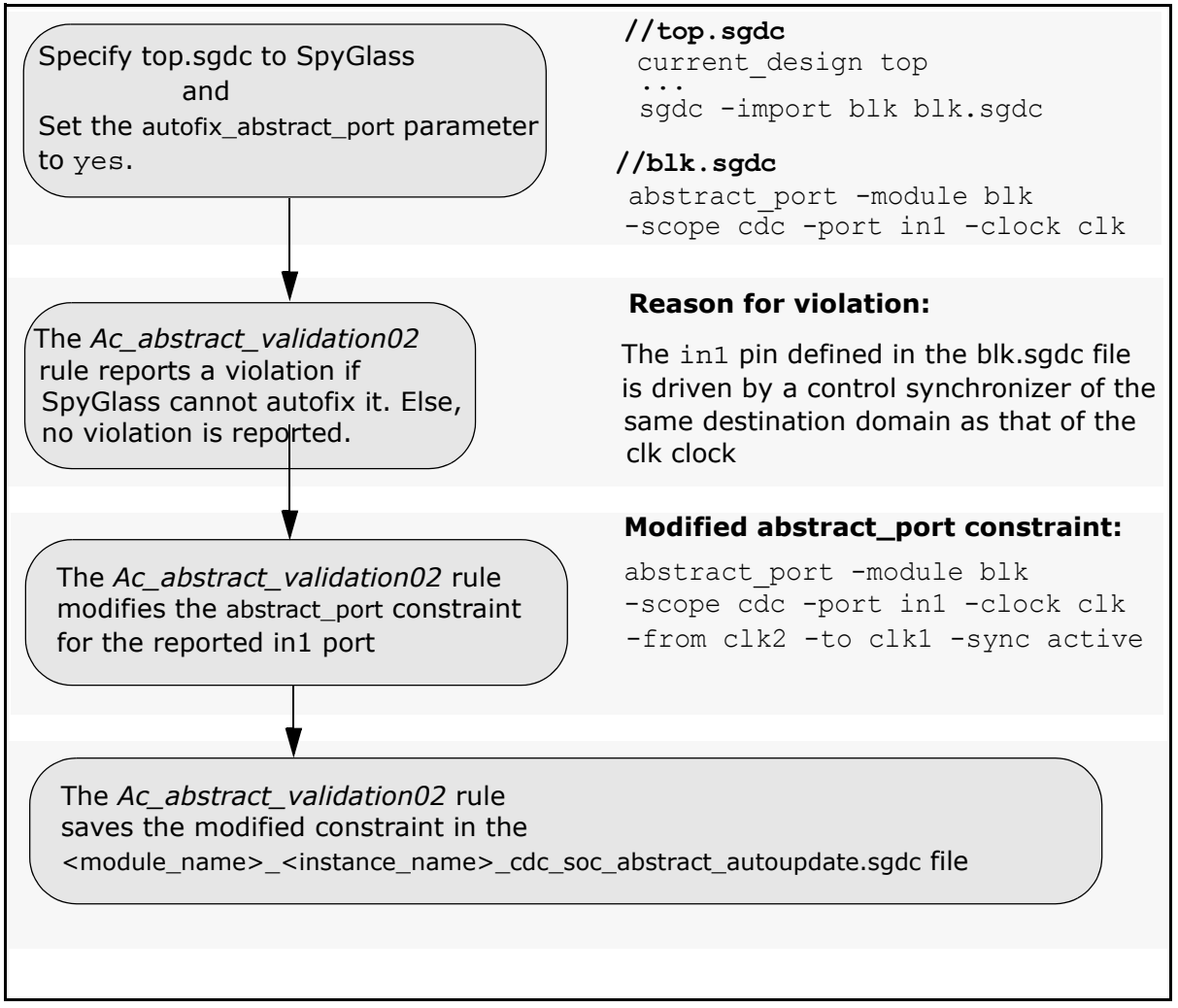


FIGURE 374. Using the autofix_abstract_port Parameter

Combo Check Mismatch

This mismatch occurs if a combinational logic exists between a block-level input port and the output of a sequential element at the top-level when the `-combo no` argument of the `abstract_port` constraint is specified for that block in the following cases:

- If at the block level, the `abstract_port` constraint is defined along with the `-combo_ifn` argument as shown below:

  ```
  abstract_port -ports a -clock VCK1 -combo_ifn ck2 -combo
  no
  ```

  In this case, this rule reports a violation if a sequential element reaches the block port after a combinational logic and if the sequential cell has a clock domain that is different from the clock domain of the clock specified in the `-combo_ifn` argument.

- If at the block level, the `abstract_port` constraint is defined with real clocks as shown below:

  ```
  abstract_port -ports a -clock clk1 -combo no
  ```

  In this case, the *Ac_abstract_validation01* rule will report a violation if a sequential element reaches the block port after combinational logic and if the sequential cell has a clock domain that is different from the clock domain of the clock specified in the `-clock` argument.

- If at the block level, the `abstract_port` constraint is defined with only virtual clock as shown below:

  ```
  abstract_port -ports a -clock VCK1 -combo no
  ```

  In this case, this rule reports a violation if the sequential element reaches the block port after combinational logic.

Automatically Fixing the abstract_port Constraint of the Reported Port

Set the `autofix_abstract_port` parameter to `yes` to modify the `abstract_port` constraint in the context of SoC for the reported port, and save the modified constraints in the *<module_name>_<instance_name>_cdc_soc_abstract_autoupdate.sgdc* file.

By default, in addition to the modified constraints, this file also contains a

copy of all the unmodified input side `abstract_port` constraints present in block-level SGDC file (abstract block). Set `autofix_dump_allinputs` to `no` to generate only the modified constraints.

Figure 375 shows the example of using the `autofix_abstract_port` parameter:

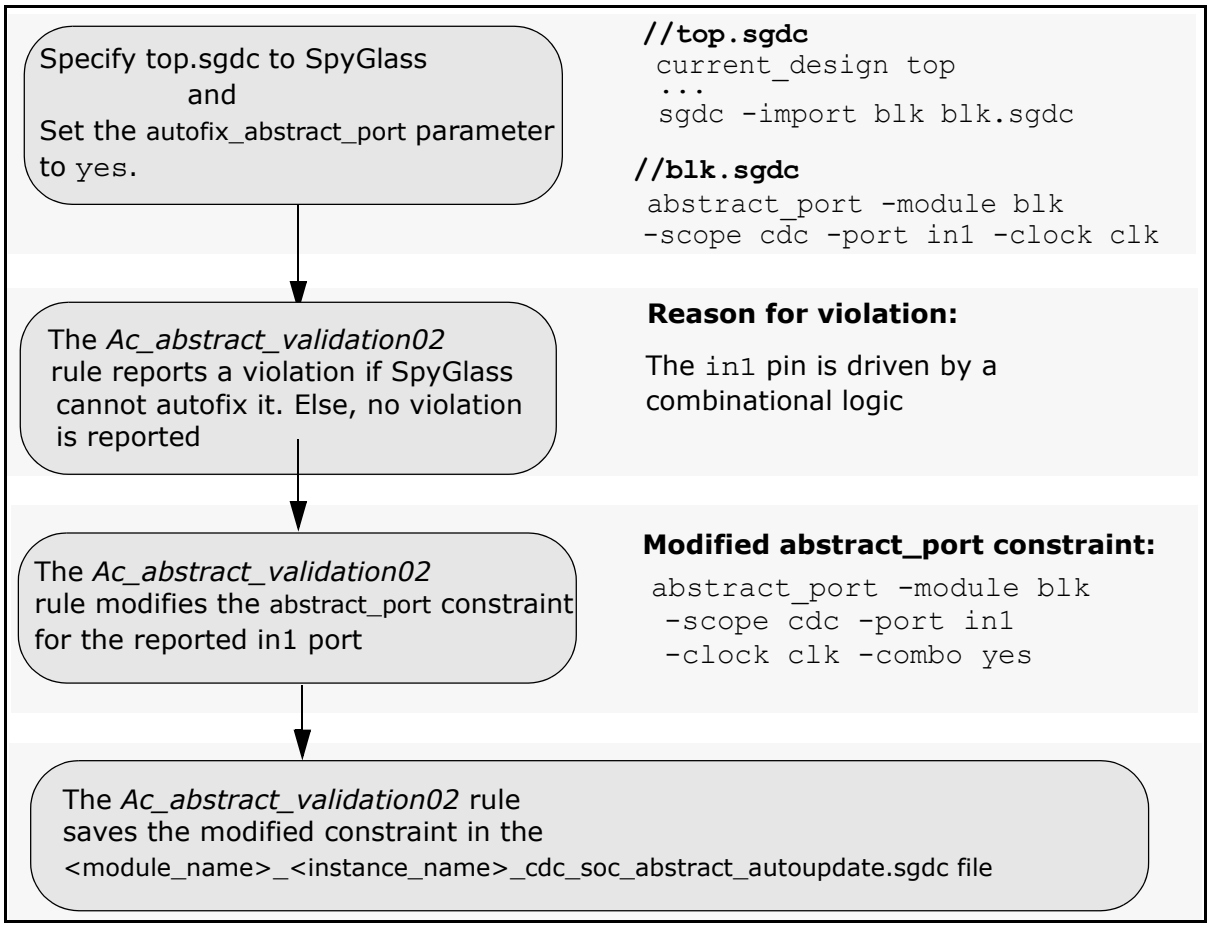


FIGURE 375. Using the autofix_abstract_port Parameter

num_flops Mismatch

This mismatch occurs if the value of the `num_flops` constraint defined in the top-level SGDC file for a clock-pair is different from the value of the `num_flops` constraint defined in the block-level SGDC file for the same clock pair.

Parameter(s)

- `validate_group_type`: Default value is `default`. Set this parameter to `port` to group the rows of *The Ac_abstract_validation02 Spreadsheet* based on the ports of an abstracted block.
- *autofix_abstract_port*: Default value is `yes`. Set this parameter to `no` to disable this rule from modifying the reported `abstract_port` constraints in the context of SoC.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *abstract_validate_express*: Default value is `no`. Set this parameter to `yes` to enable validation of only user-specified block assumptions with respect to the top-level block. Missing block assumptions are not checked in this case.
- *validate_reduce_pessimism*: Default value is `none`. Set this parameter to `hanging_nets` to ignore reporting on the hanging block ports. Other possible values are `constant`, `quasi_static`, `ignore_domain_overconstraint`, and `all`.
- `sta_based_clock_relationship`: Default value is `no`. Set this parameter to `yes` to compute domains based on the specification of the `sg_clock_group` constraint.
- *dump_detailed_info*: Default value is `none`. Set this parameter to a supported value to enable the rule to include detailed information in the generated rule/message-based spreadsheet.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.

- *input* (Optional): Use this constraint to specify a clock domain at input ports.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *quasi_static* (Optional): Use this constraint to specify signals whose value is predominantly static.
- *reset* (Optional): Use this constraint to specify reset signals in your design.
- *qualifier* (Optional): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *num_flops* (Optional): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *sgdc* `-import` (Mandatory): Use this constraint to specify a block-level SGDC file to be imported.
- *validation_filter_path* (Optional): Use this constraint to filter data domain violations reported during block validation.
- *sg_clock_group* (Optional): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

Message 1

The following message appears for *Clocks Mismatch*:

`Clock Mismatch:` `Top-level clocks` *`<clock-name>`*`, block-level clock` *`<block-level-clock-name>`*`, block instance` *`<inst-name>`* `(block:` *`<block-name>`*`)`

Potential Issues

See *Clocks Mismatch*.

Consequences of Not Fixing

The consequences vary based on the following situations:

- If a top-level clock reaches to an unconstrained clock port of a block

  Consequences: Some valid clock ports may get missed in the SGDC

file of an abstract view. As a result, SpyGlass may not perform synchronization checks for such potential clock signals.

- If a block-level clock port is not driven from a top-level clock port.

  Consequences:

  - If the path of top-level clock is blocked before reaching to a clock port of a block, it may result in incorrect violations at the top-level.
  - If the block port is not a clock but it is defined as a clock in the block-level SGDC file by mistake, the block-level CDC verification may be inaccurate.

How to Debug and Fix

To fix this violation, perform appropriate actions based on the following cases:

- If a top-level clock reaches to an unconstrained clock port of a block

  Action: Specify the clock constraint on the clock port of the reported block instance and analyze the specification or propagation of the top-level clock.

- If a block-level clock port is not driven from a top-level clock port.

  Action: Open the schematic and perform the following actions:

  - Analyze the top-level design for propagation of a clock to the block port.

    Check if the path of the top-level clock is blocked before reaching to the clock port of the block. In this case, fix the logic accordingly.
  - Check if the top-level net driving the clock port of a block is a clock, but it is not defined in top-level SGDC file. In this case, define the clock in the SGDC file.
  - If the block port is not a clock but it is defined as a clock in block-level SGDC file by mistake, perform the following actions:
    - Remove the clock specification from block-level SGDC file.
    - Re verify the block-level CDC verification.

Message 2

The following message appears for *Clock Domain Mismatch*:

`Clock Domain Mismatch (Same block clock domain connected to different top clock domains): Top-level clocks <top-level-`

clock-name>, block-level clocks <*block-level-clock-name*>, block instance <*block-instance*> (block: <*block-name*>)

Potential Issues

This violation appears due to *Clock Domain Mismatch* when any of the following cases hold true:

- If multiple clock ports in the same domain of an abstract view are triggered from top-level clocks of different domains
- If virtual clock specified at a block port and the clock port are in the same domain of an abstract view and they are triggered from top-level clocks of different domains
- If virtual clocks <*virtual-clock1*> and <*virtual-clock2*> specified at the ports <*block-port1*> and <*block-port2*>, respectively, are in the same domain of an abstract view and are triggered from top-level clocks of different domains

Consequences of Not Fixing

The consequences vary based on the following situations:

- If multiple clock ports in the same domain of an abstract view are triggered from the top-level clocks of different domains.

  Consequence: SpyGlass may map the reported virtual clock to an incorrect top-level domain. This may result in spurious synchronization violations during the block verification stage.

- If top-level clocks of the same domain trigger block ports of a different domain.

  Consequence: It may result in spurious synchronization results during verification phase of higher-level blocks.

How to Debug and Fix

To fix this violation, perform appropriate actions based on the following cases:

- If multiple clock ports in the same domain of an abstract view are triggered from the top-level clocks of different domains.

  Action:

  a. Analyze the specification or propagation of top-level clocks.

b. Ensure that the specification of the same domain virtual clocks is consistent with the specification of top-level clocks identified in the first step.

- If top-level clocks of the same domain trigger block ports of a different domain.

  Action:

  a. Verify the specification of different domains on multiple clock ports.

  b. Analyze the specification or propagation of the top-level clock.

Message 3

The following message appears for *Clock Domain Mismatch*:

`Clock Domain Mismatch (Different domain block clocks connected to same domain top level clocks): Top-level clocks <top-level-clock-name>, block-level clocks <block-level-clock-name>, block instance <block-instance> (block: <block-name>)`

Potential Issues

This violation appears due to *Clock Domain Mismatch* when multiple clock ports in different domains of an abstract view are triggered from the top-level clocks of same domain.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 4

The following message appears for *Virtual Clocks Mismatch*:

`Virtual Clock Mismatch: Top-level attributes (Clock: '<top-level-clock-name>'), block-level attributes (Clock: <virtual-clock-name>), block ports <block-port-name>, block instance <inst-name> (block: <block-name>)`

Potential Issues

This violation appears in case of *Virtual Clocks Mismatch* when no top-level domain reaches to block ports.

Consequences of Not Fixing

If you do not fix these violations, SpyGlass analysis may produce inaccurate synchronization results during block verification, thereby generating incorrect abstract view model.

This may further generate incorrect synchronization violations in the SoC verification stage.

How to Debug and Fix

To fix this violation, perform the following actions:

- Analyze design connectivity between the top-level sequential element and a block input port.
- Verify the virtual clock specified by the `abstract_port` or `input` constraint.

Message 5

The following message appears for *Case Analysis Mismatch*:

`Case Analysis Mismatch: Top-level value <value>, block-level value <value>, block port <block-port-name>, block instance <instance-name> (block: <block-name>)`

Potential Issues

See *Case Analysis Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If the specified value at the block-level port is incorrect, block-level CDC verification is inaccurate.
- If the specified value at the block-level port is correct but constant propagation at the top-level is incorrect, it indicates a logical issue at the top-level because of which incorrect value is propagated at the block-level.

How to Debug and Fix

To fix this violation, perform appropriate actions based on the following cases:

- If block-level ports are constrained to values that do not match with constant values propagated from the top-level

  Action:

  a. Check the value specification of the `set_case_analysis` constraint on a block port.

  b. Analyze the top-level design for propagation of a constant value to the block port.

- If a constant value propagates from the top-level, but the port of the abstract view is not constrained with the `set_case_analysis` constraint.

  Action:

  a. Analyze the top-level design for propagation of a constant value to the block port.

  b. Specify the `set_case_analysis` constraint on the block port.

- If a block port is constrained with the `set_case_analysis` constraint, but no constant value propagates from the top-level

  Action:

  a. Analyze the top-level design for propagation of a constant value to the block port.

  b. Remove the `set_case_analysis` constraint if a valid constant value does not reach the block port.

Message 6

The following message appears for *Quasi Static Mismatch*:

`Quasi-static Mismatch: Top-level quasi-static <value>, block-level quasi-static <value>, block port <block-port-name>, block instance <instance-name> (block: <block-name>)`

Potential Issues

See *Quasi Static Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If a top-level quasi-static signal reaches a block port on which a quasi_static constraint has not been specified.

  Consequence: The design may not operate in the desired mode. In addition, the `quasi_static` constraints would not be propagated to block outputs.

- If a `quasi_static` constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

  Consequence: Some crossings in the design may not be detected.

How to Debug and Fix

To fix this violation, perform appropriate actions based on the following cases:

- If a top-level quasi-static signal reaches a block port on which a `quasi_static` constraint has not been specified.

  Action:

  a. Specify `quasi_static` constraint on a block port, or

  b. Analyze the specification or propagation of the top-level quasi-static signal to the block port.

- If a `quasi_static` constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

  Action:

  a. If the block quasi-static specification is correct, add the missing `quasi_static` at the top-level, else

  b. Remove the `quasi_static` constraint from the block port.

Message 7

The following message appears for *Data Path Domain Mismatch*:

`Data Path Domain Mismatch: Top-level clocks <clock-name>,`

block-level clocks *<clock-name>*, block port *<block-port-name>*, block instance *<instance-name>* (block: *<block-name>*)

Potential Issues

See *Data Path Domain Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during verification phase of the hierarchical verification flow.

How to Debug and Fix

To fix this violation, perform the following steps:

1. Analyze the design connectivity between the top-level sequential element and the block input port.
2. Check if the clock domain specified by the abstract_port or the input constraint is consistent with the clock domains driving sequential elements identified in the first step.
3. Check SGDC (`abstract_port`) back-annotation for block `abstract_port` for which the violation was reported and back-annotation of top level sequential element to indicate the differing clock.

Message 8

The following message appears for *Reset Mismatch*:

Reset Mismatch: Top-level resets *<reset-name>* (Active low asynchronous reset), block-level reset *<block-reset-name>* (Active low synchronous reset), block instance *<instance-name>* (block: *<block-name>*)

Potential Issues

See *Reset Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If a top-level reset reaches a block port for which no `reset` constraint is specified.

  Consequence: Some potential resets may not propagate during the verification of the abstract view. This may result in the following:

  - The block may not achieve its initial state.
  - In the absence of synchronous resets, SpyGlass may report violations related with unsynchronized clock domains.

- If the reset constraint is specified for a block-level port, but no top-level reset drives that block-level port.

  Consequence: The reported port of an abstract view may not be considered as a valid reset signal. This may alter the initial state of the block during verification.

- If an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.

  Consequence:

  - Incorrect reset analysis may happen at the block-level. That is, the initial state of the block may get altered during its verification.
  - SpyGlass may generate incorrect clock domain violations during block verification if synchronous resets are not properly specified.

- If the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

  Consequence:

  - It may result in an incorrect initial state of an abstract view during verification.
  - It may generate spurious reset simulation results for the abstract view.

How to Debug and Fix

To fix this violation, perform appropriate actions based on the following

cases:

- If a top-level reset reaches a block port for which no `reset` constraint is specified.

  Action: Perform the following steps:

  a. Specify the `reset` constraint on the reported block port.

  b. Analyze the specification or propagation of the top-level reset to the block.

- If the `reset` constraint is specified for a block-level port, but no top-level reset drives that block-level port.

  Action: Perform the following actions:

  a. Remove the `reset` constraint from the reported block port.

  b. Analyze the specification or propagation of a top-level reset to the block port.

- If an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.

  Action:

  a. Specify an appropriate `reset` constraint on the block-level port.

  b. Analyze the specification or propagation of the top-level reset to a block.

- If the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

  Action:

  a. Check the value specified by the `reset` constraint on the block port.

  b. Assign a proper value in the `reset` constraint.

  c. Verify that the top-level reset of the block port has the same active value identified in step 2.

Message 9

The following message appears for *Qualifier Mismatch*:

Qualifier Mismatch: Top-level qualifier (sync type: *<sync-type>* **from:** *<clock-name>* **to:** *<clock-name>*), **block-level qualifier (sync type:** *<sync-type>* **from:** *<clock-name>* **to:** *<clock-name>*), **block port** *<block-port-name>*, **block**

instance *<instance-name>* **(block:** *<block-name>***)**

Potential Issues

See *Qualifier Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during the block verification stage.

How to Debug and Fix

To fix this violation, perform appropriate actions based on the following cases:

- If clocks specified by the `-from_clk` and `-to_clk` arguments of the `qualifier` constraint for an abstract view exist in the same top-level domain.

  Action: Perform the following actions:

  - Analyze the clocks specified in the `-from_clk` and `-to_clk` arguments of the `qualifier` constraint.
  - Analyze specification or propagation of top-level clocks.

- If a synchronizer does not reach to an input port of a block for which the `-sync` argument of the `abstract_port` constraint is specified.

  Action: Perform the following actions:

  - Analyze the fan-in cone of an input port for the presence of a synchronizer.
  - Remove the `-sync` argument from the `abstract_port` constraint.

Message 10

The following message appears for *num_flops Mismatch*:

`Num-flops Mismatch: Top-level num_flop (value: <value>, from_clk: <clock-name>, to_clk: <clock-name>), block-level num_flops (value: <value>, from_clk: <clock-name>, to_clk:`

<clock-name>), block instance *<value>*, from_clk: *<clock-name>*, to_clk: *<instance-name>* (block: *<value>*, from_clk: *<clock-name>*, to_clk: *<block-name>*)

Potential Issues

See *num_flops Mismatch*.

Consequences of Not Fixing

If you do not fix this violation, the following may occur depending upon different situations:

- If the num_flops constraint is incorrectly specified in the block-level SGDC file, it may result in inaccurate block-level CDC verification.
- If domains of top-level clocks are not specified correctly, it may result in inaccurate top-level CDC verification.

How to Debug and Fix

To fix this violation, perform the following actions:

- Analyze the clock specification in the `-from_clk` and `-to_clk` arguments of the `num_flops` constraint.
- Analyze the specification or propagation of top-level clocks.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

```
// test.v
  module test(in1,clk1, clk2, clk3, in2,in3, out,sel);
  input clk1,clk2,sel,clk3;
  input [3:0] in1,in2,in3;
  output [3:0] out;
  reg [3:0] temp,syn1,syn2, temp1;
  wire [3:0] sync2_tmp;
  assign clk_1 = clk1 & clk2;
  wire bb_out;
  always @(posedge clk_1)
   temp = in3;
   always @(posedge clk1)
      temp1 = in2;
   always @(posedge clk2)
      begin
      syn2<=temp;
      end
      assign sync2_tmp = syn2 & temp1 & temp;
  BB b1(.clk1(clk1),.clk2(clk2),.clk3(clk_1),.in3(sync2_tmp),.in2(sync2_tmp),
         .out(out), .out1(out));
  BB b2(.clk1(clk1),.clk2(clk2),.clk3(clk_1),.in3(sync2_tmp),.out(out),
  .out1(out));
  endmodule
  module BB(clk1,clk2,clk3,in1,out,out1, in3, in2);
   input clk1,clk2,in1,clk3;
   input [3:0] in3, in2;
   output [3:0] out;
   output out1;
   assign out1 = in3;
  endmodule
// test.sgdc
  current_design test
  clock -name clk1 -domain domain1
  clock -name clk2 -domain domain2
  sgdc -import BB  block.sgdc

// block.sgdc
  current_design BB
  clock -name clk1 -domain domain1
  clock -name clk2 -domain domain2
  set_case_analysis -name in3 -value 0
  set_case_analysis -name in2 -value 0
  abstract_port -module BB -ports in3 -clock clk1
  abstract_port -module BB -ports in3 -clock clk2
  abstract_port -module BB -ports in3 -clock clk3
  abstract_port -module BB -ports in2 -clock clk1
  abstract_port -module BB -ports in2 -clock clk3
  abstract_port -module BB -ports in2 -clock clk2
```

For the above example, the *Ac_abstract_validation02* rule reports various types of mismatches between the abstract block and the top-level design.

To view the details of all the mismatches, double-click on the violation of this rule to open *The Ac_abstract_validation02 Spreadsheet*.

Reports and Related Files

The Ac_abstract_validation02 Spreadsheet

SGDC_abstract_mapping01

Reports clock mapping of an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow.

Description

The *SGDC_abstract_mapping01* rule reports the mapping between block-level and top-level clocks.

The clocks can be real or virtual clocks.

Parameter(s)

report_top_block_info: Default value is `yes`. Set this parameter to `no` to stop reporting of information about the clock domains and tags in the clock mapping spreadsheet.

Constraint(s)

None

Messages and Suggested Fix

The following message appears:

`[INFO] Clock mapping of instance '<inst-name>' of block '<block-name>'`

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Check the *<block-name>_<instance-name>_ClockMapping.csv* file to see the

mapping between the block-level and top-level clocks.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test(clk1,clk2,in1,in2, out1,out2);

input clk1,clk2;
input in1,in2;
output out1,out2;
reg r1,r2;
wire wr1,wr2;

always@(posedge clk1)
   r1 <= in1;
always@(posedge clk2)
   r2 <= in2;

wire temp_clk = clk1 & clk2;

block B1(.d1(r1),.d2(r2),.ck1(temp_clk),.ck2(clk2),.out1(wr1),.out2(wr2));

assign out1 = wr1;
assign out2 = wr2;
endmodule

module block(d1,d2,ck1,ck2,out1,out2);

input d1,d2;
input ck1,ck2;
output out1,out2;
reg out1,out2;

always@(posedge ck1)
   out1 <= d1;
always@(posedge ck1)
   out2 <= d2;
endmodule
```

```
//top.sgdc
current_design test
clock -name clk1 -tag T1
clock -name clk1 -tag T2 -add
clock -name clk2 -domain d2


sgdc -import block block.sgdc
```

```
//block.sgdc
current_design block
clock -name ck1
clock -name ck2

abstract_port -module block -ports d1 -clock V1
abstract_port -module block -ports d2 -clock V2
```

For the above example, the following spreadsheet is generated showing mapping between the clocks of the B1 instance with the top-level clocks:

A Block Clock	B Block Clock Domain	C Block Clock Tag	D Top Clock(s)	E Top Clock Domain(s)	F Top Clock Tag(s)
ck1	ck1	SG_AUTO_TAG_0	test.clk1 test.clk2	clk1 d2	T1,T2 SG_AUTO_TAG_1
ck2	ck2	SG_AUTO_TAG_1	test.clk2	d2	SG_AUTO_TAG_1
V1	sg_virtual_V1	SG_AUTO_VIRT_TAG_V1	test.clk1	clk1	T1,T2
V2	sg_virtual_V2	SG_AUTO_VIRT_TAG_V2	test.clk2	d2	SG_AUTO_TAG_1

FIGURE 376. Spreadsheet Generated by the SGDC_abstract_mapping01 Rule

Default Severity Label

Info

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

- <block-name>_<instance-name>_ClockMapping.csv

  This spreadsheet shows the mapping between the block-level and top-level clocks. *Figure 376* shows this spreadsheet.

  The following table describes the columns of this spreadsheet:

Column Name	Description
Block Clock	Specifies the name of the clock on the block
Block Clock Domain	Specifies the clock domain name on the block
Block Clock Tag	Specifies the clock tag name on the block
Top Clock(s)	Specifies the top-level clocks mapped to the block-level clock
Top Clock Domain(s)	Specifies the top-level clock domain name(s)
Top Clock Tag(s)	Specifies the top-level clock tag name(s)
Top-level internal domain tag	Specifies a unique tag number generated for the top-level clock. For details, see *Using the Clock Domain Tag*.

SGDC_clock_validation01

Reports unconstrained clock ports of an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_clock_validation01* rule reports a violation if a top-level clock reaches to a clock port of a block, but that clock port is not constrained by the *clock* constraint.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears when the top-level clock *<top-level-clock>* reaches to the unconstrained clock port *<clock-port>* of the block instance *<inst-name>*:

```
[ERROR] Top-level clock '<top-level-clock>' reaches to
unconstrained port '<clock-port>' of block instance
'<inst-name>' (block: '<abstract-view-name>')
```

Potential Issues

This violation appears if the design contains an unconstrained clock port present on a block instance, and a top-level clock reaches to that unconstrained clock port.

Consequences of Not Fixing

If you do not fix this violation, some valid clock ports may get missed in the SGDC file of an abstract view. As a result, SpyGlass may not perform synchronization checks for such potential clock signals.

How to Debug and Fix

To fix this violation, specify the *clock* constraint on the clock port of the reported block instance and analyze the specification or propagation of the top-level clock.

Example Code and/or Schematic

Consider the following schematic of a design:

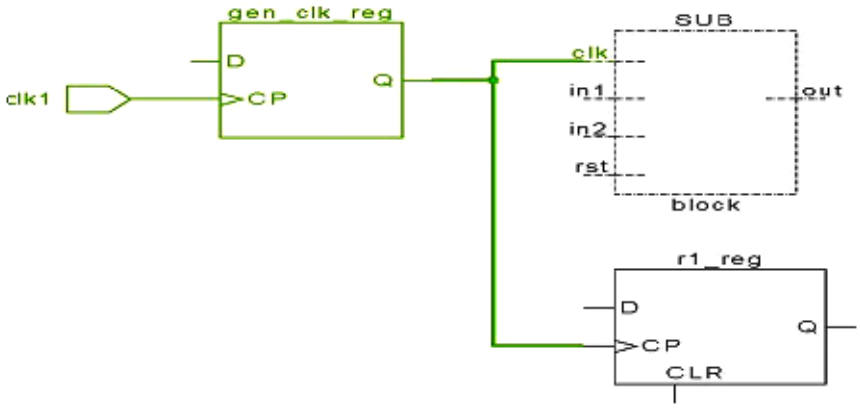


FIGURE 377. Schematic of the SGDC_clock_validation01 Rule Violation

In the above example, the `SUB` block is the abstract view that has the clock port `clk`. However, this clock port is not constrained by the *clock* constraint.

In this case, the *SGDC_clock_validation01* rule reports a violation because the top-level clock `clk1` reaches the clock port `clk`, which is unconstrained.

To fix this violation, specify the *clock* constraint for the `clk` clock port.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No related files and reports

SGDC_clock_domain_tag

Reports clock constraints whose -tag and -domain fields have the same name

When to Use

Use this rule to validate that the same name is not used for the `-tag` and `-domain` fields of the *clock* constraint.

Description

The *SGDC_clock_domain_tag* rule reports a violation if the same name is used for the `-tag` and `-domain` fields of the *clock* constraint.

Parameter(s)

None

Constraint(s)

clock (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears when the *clock* constraint has the same name in the `-tag` and `-domain` fields:

`[WARNING] Constraint 'clock': Same name '<string>' used in '-tag' and '-domain' fields`

Potential Issues

This violation appears if the *clock* constraint has the same name for the `-tag` and `-domain` fields of the constraint.

Consequences of Not Fixing

If you do not fix this violation, CDC analysis might be inaccurate.

How to Debug and Fix

To fix this violation, ensure that the `-tag` and the `-domain` fields of the *clock* constraint have different names.

Example Code and/or Schematic

Consider the following code:

```
current_design duname

clock -name top.clk1
-tag CK1
-domain CK1
```

In the above scenario, the *SGDC_clock_domain_tag* rule reports a violation.

Default Severity Label

WARNING

Rule Group

SETUP

Reports and Related Files

No related file

SGDC_clock_validation02

Reports clock ports of an abstract view, which are not driven from top-level clocks

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_clock_validation02* rule reports a violation when a block-level clock port is not driven from a top-level clock port.

This can occur when the *clock* constraint is defined on a block port, but a top-level clock does not reach that block port.

Parameter(s)

None

Constraint(s)

clock (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears when a clock specified for a block instance is not propagated from the top-level clock:

```
[ERROR] Clock '<clock-name>' specified for block instance
'<inst-name>' (block: '<block-name>') is not propagated from
top-level clock
```

Details of the arguments of the above violation message are described in the following table:

Argument	Description
<clock-name>	Specifies the abstract-view port that is constrained as a clock
<inst-name>	Specifies the instance name of an abstract view
<block-name>	Specifies the abstract-view name

Potential Issues

This violation appears if your design contains a block-level clock port on which the *clock* constraint is applied, but the top-level clock does not reach that block port.

Consequences of Not Fixing

If you do not fix this violation, the following may occur depending upon different cases:

- If the path of top-level clock is blocked before reaching to a clock port of a block, it may result in incorrect violations at the top-level.
- If the block port is not a clock but it is defined as a clock in the block-level SGDC file by mistake, the block-level CDC verification may be inaccurate.

How to Debug and Fix

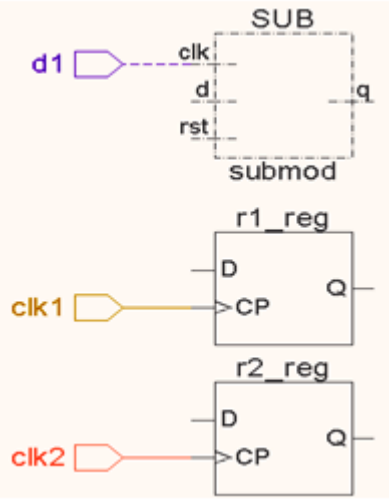
To fix this violation, open the schematic of the violation of the *SGDC_clock_validation02* rule, and perform the following actions:

- Analyze the top-level design for propagation of a clock to the block port.

  Check if the path of the top-level clock is blocked before reaching to the clock port of the block. In this case, fix the logic accordingly.
- Check if the top-level net driving the clock port of a block is a clock, but it is not defined in top-level SGDC file. In this case, define the clock in the SGDC file.
- If the block port is not a clock but it is defined as a clock in block-level SGDC file by mistake, perform the following actions:
  - ❒ Remove the clock specification from block-level SGDC file.
  - ❒ Re-verify the block-level CDC verification.

Example Code and/or Schematic

Consider the following schematic of a design:




```
// SGDC file for the block module,submod
   current_design submod
   clock -name clk -domain d1

// Top-level SGDC file defining the top-level
// clocks clk1 and clk2
   current_design top
   clock -name clk1 -domain d1
   clock -name clk2 -domain d2
```

FIGURE 378. Schematic of the SGDC_clock_validation02 Rule Violation

In the above example, the clock port `clk` of the block is driven by the top-level port `d1`, which is not defined as a clock in the top-level SGDC file.

As a result, the *SGDC_clock_validation02* rule reports a violation.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_clock_domain_validation01

Reports same domain clock ports of an abstract view driven from different top-level clock domains

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_clock_domain_validation01* rule reports a violation if multiple clock ports in the same domain of an abstract view are triggered from the top-level clocks of different domains.

Message Details

Message 1

The following message appears if multiple clock ports in the same domain of an abstract view are triggered from top-level clocks of different domains:

`[SClkDV1_1] [ERROR] Top-level clocks '<top-clock1>' and '<top-clock2>' of different domains are connected to same domain clock ports '<clock-port1>' and '<clock-port2>'(block level domain: '<domain-name>') of block instance '<inst-name>' (block: '<block-name>')`

Potential Issues

This violation appears if the top-level clocks from different domains are connected with the same domain clock ports of an abstract view.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report spurious

synchronization violations during the block verification stage.

How to Debug and Fix

To fix this violation, perform following steps:

1. Analyze the specification or propagation of top-level clocks.
2. Ensure that the specification of the same domain on multiple clock ports is consistent with the specification of top-level clocks identified in the first step.

Message 2

The following message appears if the virtual clock *<virtual-clock>* specified at the port *<block-port>* and the clock port that are in the same domain of an abstract view are triggered from top-level clocks of different domains:

```
[SClkDV1_2] [ERROR] Top level clocks '<top-clock1>' and '<top-clock2>' of different domains are connected to same domain clocks '<virtual-clock>'(at '<block-port>') and '<clock-port>'(block level domain: '<domain-name>') of block instance '<inst-name>' (block: '<block-name>')
```

Potential Issues

This violation appears if the top-level clocks from different domains are connected with the same domain clock ports of an abstract view.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may map the reported virtual clock to an incorrect top-level domain. This may result in spurious synchronization violations during the block verification stage.

How to Debug and Fix

To fix this violation, perform following steps:

1. Analyze the specification or propagation of top-level clocks.

2. Ensure that the specification of the same domain virtual clocks is consistent with the specification of top-level clocks identified in the first step.

Message 3

The following message appears if the virtual clocks *<virtual-clock1>* and *<virtual-clock2>* specified at the ports *<block-port1>* and *<block-port2>* that are in the same domain of an abstract view are triggered from top-level clocks of different domains:

```
[SClkDV1_3] [ERROR] Top level clocks '<top-clock1>' and '<top-clock2>' of different domains are connected to same domain clocks '<virtual-clock1>'(at '<block-port1>') and '<virtual-clock2>'(at '<block-port2>') (block level domain: '<domain-name>') of block instance <inst-name>' (block: '<block-name>')
```

Potential Issues

This violation appears if the top-level clocks from different domains are connected with the same domain virtual clocks of an abstract view.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may map the reported virtual clocks to incorrect top-level domains. This may result in spurious synchronization violations during the block verification stage.

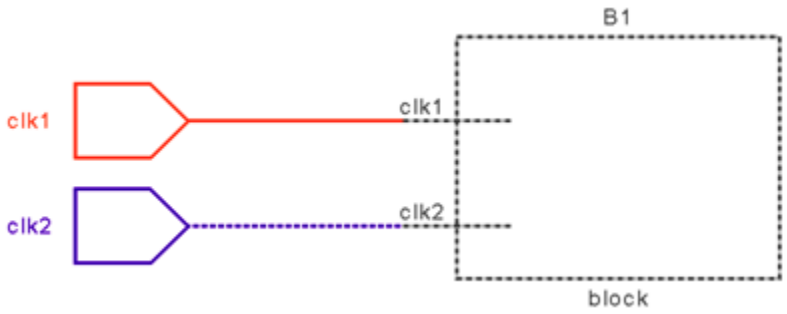
How to Debug and Fix

To fix this violation, perform following steps:

1. Analyze the specification or propagation of top-level clocks.
2. Ensure that the specification of the same domain on multiple virtual clocks is consistent with the specification of top-level clocks identified in the first step.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// block.sgdc
clock -name clk1 -domain d1
clock -name clk2 -domain d1
```

```
// top.sgdc
clock -name clk1 -domain d1
clock -name clk2 -domain d2
```

FIGURE 379. Schematic of the SGDC_clock_domain_validation01 Rule Violation

In the above schematic, the `clk1` and `clk2` clocks from different domains are connected with the same domain clock ports of the abstract view `B1`.

As a result, the *SGDC_clock_domain_validation01* rule reports a violation.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_clock_domain_validation02

Reports different domain clock ports of an abstract view being driven from the same top-level clock domain

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *clock* constraint.

Description

The *SGDC_clock_domain_validation02* rule reports a violation if top-level clocks of the same domain trigger block ports of a different domain.

Rule Exception

The *SGDC_clock_domain_validation02* rule does not report a violation for virtual clocks.

Parameter(s)

None

Constraint(s)

clock (Mandatory): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if top-level clocks of the same domain trigger different domain block ports:

```
[SClkDV2_1] [WARNING] Top-level clock '<clock-name>' of same domain is connected to different domain clock ports '<clock-port1>' and '<clock-port2>' (block level domain: '<domain1>'
```

'<domain2>') of block instance '<inst-name>' (block: '<block-name>')

Potential Issues

This violation appears if your design contains top-level clocks of the same domain, and these clocks are connected to different domain clock ports.

Consequences of Not Fixing

If you do not fix this violation, it may result in spurious synchronization results during verification phase of higher-level blocks.

How to Debug and Fix

To debug this violation, perform the following steps:

- Verify the specification of different domains on multiple clock ports.
- Analyze the specification or propagation of the top-level clock.

Example Code and/or Schematic

Consider the following figure:

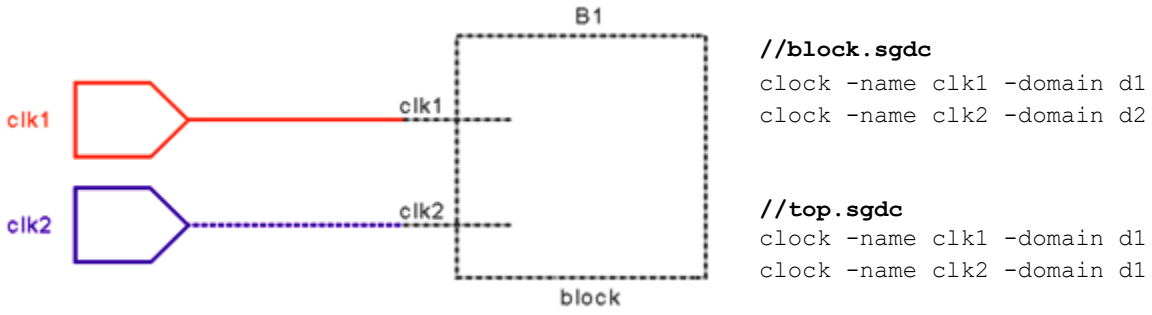


FIGURE 380. Schematic of the SGDC_clock_domain_validation02 Rule Violation

In the above example, the top-level clocks of the same domain trigger different domain block ports.

Therefore, the *SGDC_clock_domain_validation02* rule reports a violation.

Default Severity Label

Warning

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_set_case_analysis_validation01

Reports a violation if the constant value simulated from the top-level does not match with the constant value specified in a block-level constraint file

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following details before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *set_case_analysis* constraint.

Description

The *SGDC_set_case_analysis_validation01* rule reports a violation if there is a mismatch between the following values:

- Constant value specified by the *set_case_analysis* constraint for a block-level port
- Constant value propagated from the top-level

The constant is propagated across flip-flops or other sequential cells only if you specify the `set_option enable_const_prop_thru_seq yes` command in the project file.

Parameter(s)

None

Constraint(s)

set_case_analysis (Mandatory): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when a constant value simulated from the

top-level does not match with the constant value specified in a block-level constraint file:

```
[ERROR] Simulated value at port '<port-name>' of instance
'<inst-name>' (block: '<block-name>') is '<value1>' but
specified value in block level constraint file is '<value2>'
```

Potential Issues

This violation appears if block-level ports are constrained to values that do not match with constant values propagated from the top-level.

Consequences of Not Fixing

If you do not fix this violation, the following issues may arise depending upon different situations:

- If the specified value at the block-level port is incorrect, block-level CDC verification is inaccurate.
- If the specified value at the block-level port is correct but constant propagation at the top-level is incorrect, it indicates a logical issue at the top-level because of which incorrect value is propagated at the block-level.

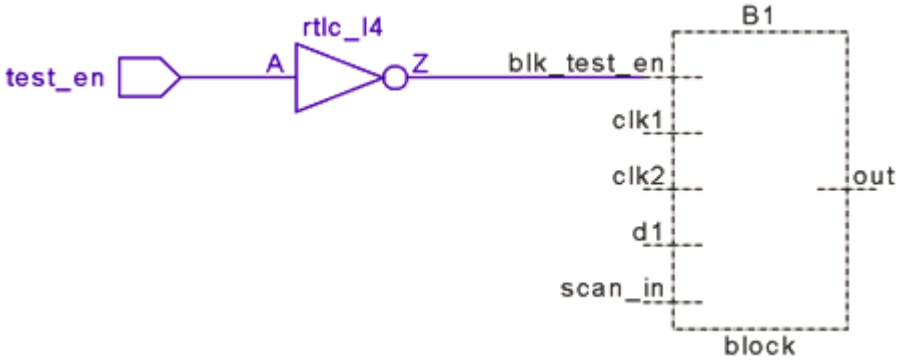
How to Debug and Fix

To fix this violation, perform the following steps:

1. Check the value specification of the *set_case_analysis* constraint on a block port.
2. Analyze the top-level design for propagation of a constant value to the block port.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// Top-level SGDC file
current_design top
set_case_analysis -name top.test_en -value 0

// Block-level SGDC file
current_design block
set_case_analysis -name block.blk_test_en -value 0
```

FIGURE 381. Schematic of the SGDC_set_case_analysis_validation01 Rule Violation

In the above example, the *SGDC_set_case_analysis_validation01* rule reports a violation because at the top-level, the constant value `1` is propagated at the net connected to the `blk_test_en` block pin, whereas in the block-level SGDC file, the value specified is `0`.

To fix this violation, modify the *set_case_analysis* constraint specification of the block-level SGDC file to the following:

```
set_case_analysis -name block.blk_test_en -value 1
```

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_set_case_analysis_validation02

Reports missing constants between top and block

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following details before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *set_case_analysis* constraint.

Description

The *SGDC_set_case_analysis_validation02* rule reports a violation in the following cases:

- If a simulated value reaches a top-level net connected to a block-level port, but no *set_case_analysis* constraint is specified on the block-level port
- If a simulated value does not reach to a top-level net connected to a block-level port, but the *set_case_analysis* constraint is specified on the block-level port

Parameter(s)

None

Constraint(s)

set_case_analysis (Mandatory): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears if a simulated value reaches a top-level net connected to a block-level port, but no *set_case_analysis* constraint is

specified on the block-level port:

```
[SScaV2_1] [ERROR] Simulated value '<value>' reaches to port
'<port-name>' of block instance '<inst-name>' (block: '<block-
name>') however no set_case_analysis is specified in block
level constraint file
```

Potential Issues

This violation appears if a constant value propagates from the top-level, but the abstract view port is not constrained with the *set_case_analysis* constraint.

Consequences of Not Fixing

If you do not fix this violation, the design may not operate in the desired mode.

How to Debug and Fix

To fix this violation, perform the following steps:

- Analyze the top-level design for propagation of a constant value to the block port.
- Specify the *set_case_analysis* constraint on the block port.

Message 2

The following message appears if a simulated value does not reach a top-level net connected to a block-level port, but the *set_case_analysis* constraint is specified on the block-level port:

```
[SScaV2_2] [ERROR] Simulated value does not reach to port
'<port-name>' of block instance '<inst-name>' (block: '<block-
name>') where as set_case_analysis defined in block-level
constraint file
```

Potential Issues

This violation appears if a block port is constrained with the *set_case_analysis* constraint, but no constant value propagates from the

top-level.

Consequences of Not Fixing

If you do not fix this violation, the reported ports can block or enable propagation of unexpected signals across the abstract view.

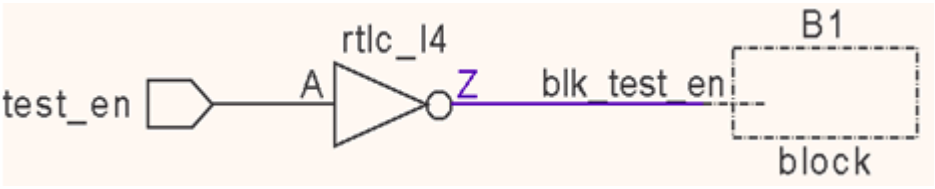
How to Debug and Fix

To fix this violation, perform the following steps:

- Analyze the top-level design for propagation of a constant value to the block port.
- Remove the *set_case_analysis* constraint if a valid constant value does not reach the block port.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// SGDC file for the block module

current_design block
set_case_analysis -name block.blk_test_en -value 0
```

FIGURE 382. Schematic of the SGDC_set_case_analysis_validation02 Rule Violation

In the top-level SGDC file, the *set_case_analysis* constraint is not defined for the `test_en` signal.

In this case, the *SGDC_set_case_analysis_validation02* rule reports a violation because at the top-level, no constant value propagates at the net connected to the `blk_test_en` block pin, whereas the constant value is specified in the block-level SGDC file.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_set_case_analysis_validation03

Reports top module output ports on which user defined set_case_analysis value differs from value obtain from propagation

When to Use

If you have specified the *set_case_analysis* constraint on some output ports of the top design/module, use this rule to check if any or different constant/set_case_analysis value is propagated to those output ports.

Prerequisites

Specify the *set_case_analysis* constraint on any output port.

Description

The *SGDC_set_case_analysis_validation03* rule reports a violation if:

- The constant value that is propagated (through simulation) to any output port of the top module differs from the value which is defined on that output port.
- Any constant value is not propagated to the output port but the *set_case_analysis* constraint is defined on that port.

Parameter(s)

None

Constraint(s)

set_case_analysis (Mandatory): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears if the constant value propagated to any output port of the top module *<module_name>* differs from the value defined on that output port:

```
[Warning] Simulated value at output ports of top module
<module-name> differs with specified values in constraint file.
```

Potential Issues

This violation appears if the output port is constrained with the *set_case_analysis* constraint but the constant signal received from the fanin cone is different or no constant value is received on that port.

Consequences of Not Fixing

If you do not fix this violation, the constrained port will pass incorrect information to other interacting modules when these modules are used in the SoC level.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

- Identify the problematic output ports by referring the rule based spreadsheet for the top module.
- If any set_case_analysis value is not propagated to an output port on which *set_case_analysis* constraint is defined, remove the set_case_analysis constraint from that port. Or, set the value of the specified set_case_analysis constraint so that it equals to the simulated value.
- If the specified constraint is correct, verify the fanin cone for incorrect propagation of the constraint.

Example Code and/or Schematic

Consider the following schematic:

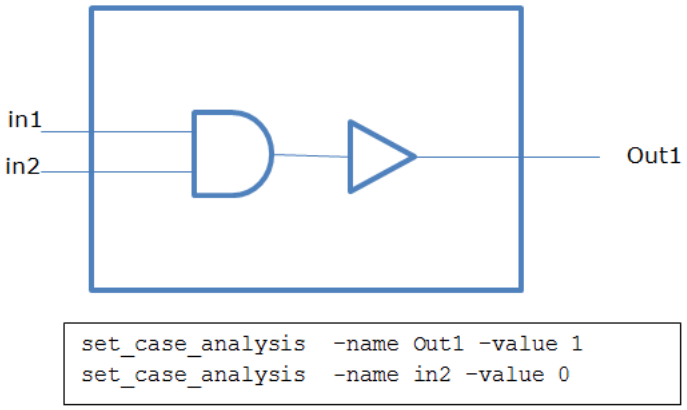


FIGURE 383. Schematic of the SGDC_set_case_analysis_validation03 Rule Violation

In the above example, the defined value of the Out1 output port is 1. However, the simulated/propagated value is zero. Therefore for this top module, the *SGDC_set_case_analysis_validation03* rule reports a violation.

Default Severity Label

Warning

Rule Group

None

Reports and Related Files

SGDC_set_case_analysis_validation03.csv

SGDC_reset_filter_path_validation01

Reports block-level reset_filter_path constraints which do not have a matching top-level reset_filter_path constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_filter_path* constraint
- Specify the `clock` constraint for objects given in `-from_clock`/`-to_clock`/`-clock` arguments of the `reset_filter_path` constraint
- Specify the `reset` constraint for objects given in `-from_rst`/`-to_rst` arguments of the *reset_filter_path* constraint

Description

The `SGDC_cdc_false_path_validation01` rule reports a violation if the *reset_filter_path* constraint specified on an abstracted block does not have an equivalent *reset_filter_path* constraint at the top level. the rule matches the `-from_rst`, `-to_rst`, `-from_clock`, `-to_clock`/`-clock`, and `-type` arguments.

Parameter(s)

None

Constraint(s)

- *reset_filter_path* (Mandatory): Use this constraint to specify reset paths so that the reset domain crossings across these paths are ignored from SpyGlass analysis.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The rule reports the following message when the `reset_filter_path` constraint given in an SGDC file of an abstract view does not have an equivalent *reset_filter_path* constraint at the top level.

```
[ERROR] For block instance '<block-inst>' (block: <block
name>), Constraint reset_filter_path specified at the block
level with fields -from_rst '<from-reset>', -to_rst '<to-
reset>', -from_clock '<from-clock>', -to_clock/-clock '<to-
clock/clock>' and -type '<type>' has no equivalent constraint
at the top level
```

Potential Issues

This violation appears if the *reset_filter_path* constraint specified in an SGDC file of an abstract view does not have an equivalent *reset_filter_path* constraint at the top level.

Consequences of Not Fixing

If you do not fix this violation, ignored paths by reset_filter_path are inconsistent between top and abstracted block. This may result in an inconsistency in the violations reported for the top and the abstracted block by the Ar_resetcross01 and Ar_sync_group rules.

How to Debug and Fix

To fix this violation, first analyze the following:

- The specification of *reset_filter_path* constraint in block level as well as top level.
- The specification or propagation of top-level clocks.
- The specification or propagation of top-level resets.

Next, if the block-level constraints are incorrect, specify the correct block-level constraints and run block-level verification again. If the top-level constraints are incorrect with respect to the block, specify the correct top-level constraints and run top-level verification again.

Example Code and/or Schematic

Consider the following schematic:

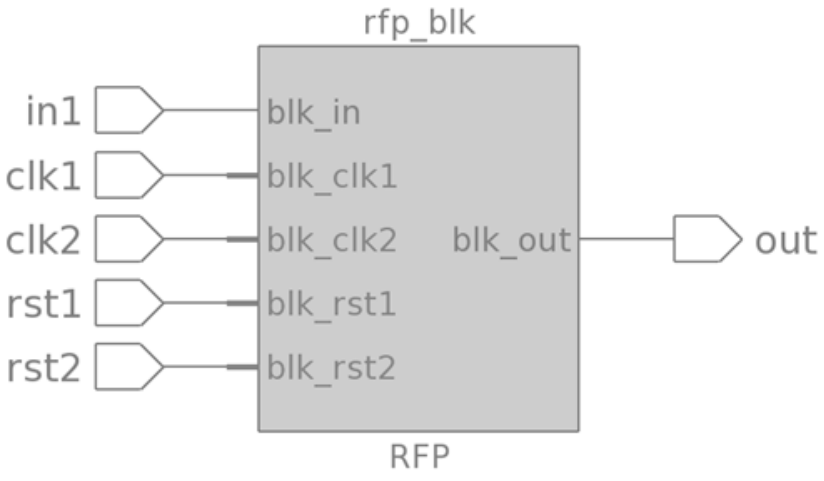


FIGURE 384.

In addition, consider the following SGDC files:

//Block-level SGDC file

```
current_design "RFP"
clock -name blk_clk1 -domain clk1
clock -name blk_clk2 -domain clk2
reset -name blk_rst1 -value 0
reset -name blk_rst2 -value 0
reset_filter_path -from_rst blk_rst1 -to_rst blk_rst2 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

//Top-level SGDC file

```
current_design top
clock -name clk1
clock -name clk2
reset -name rst1 -value 0
reset -name rst2 -value 0
sgdc -import RFP RFP.sgdc
```

In the above example, only one *reset_filter_path* constraint is specified in the block-level SGDC file. However, no matching *reset_filter_path* constraint is specified in the top-level SGDC file.

Therefore, the `SGDC_reset_filter_path_validation01` rule reports a violation in this case.

Default Severity Label

Error

Rule Group

None

Reports and Related Files

No report or related file

SGDC_reset_validation01

Reports unconstrained reset ports of an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_reset_validation01* rule reports a violation if a top-level reset reaches a block port for which no *reset* constraint is specified.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears if the top-level reset *<reset-name>* propagates to the block reset port *<port-name>* that is not constrained by the *reset* constraint:

[ERROR] Top-level reset '<reset-name>' reaches to unconstrained port '<port-name>' of block instance '<inst-name>' (block: '<block-name>')

Potential Issues

This violation appears if a top-level reset net is connected to an abstract-view port on which no *reset* constraint is specified.

Consequences of Not Fixing

If you do not fix this violation, some potential resets may not propagate during abstract-view verification.

This may result in the following:

- The block may not achieve its initial state.
- In the absence of synchronous resets, SpyGlass may report violations related with unsynchronized clock domains.

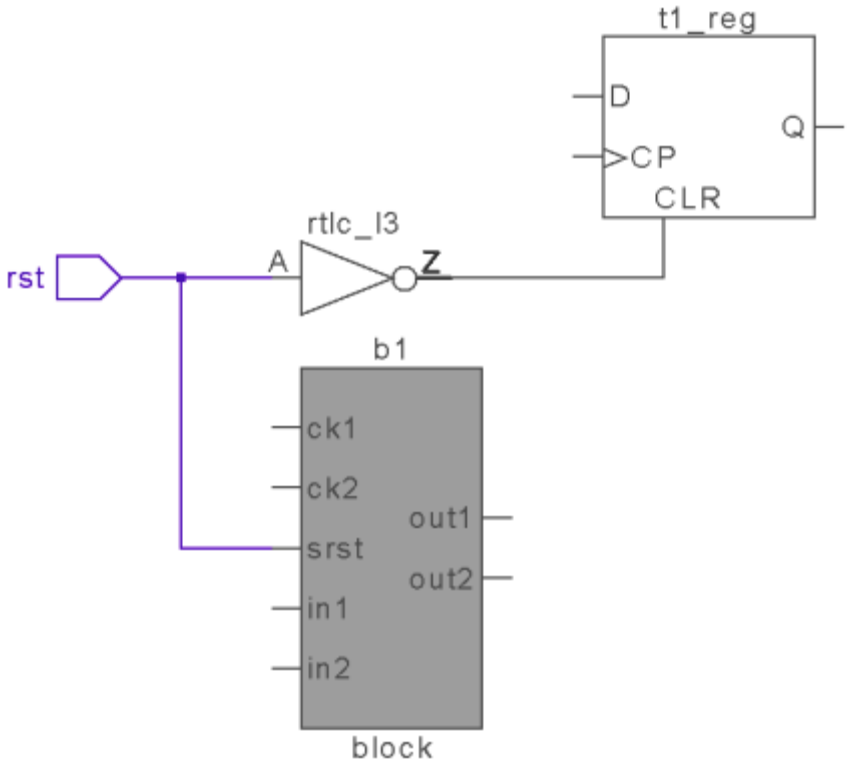
How to Debug and Fix

To debug and fix this violation, perform the following steps:

- Specify the *reset* constraint on the reported block port.
- Analyze the specification or propagation of the top-level reset to the block.

Example Code and/or Schematic

Consider the abstract view `B1` shown in the following schematic:




```
//top.sgdc
Reset -name rst -async
```

FIGURE 385. Schematic of the SGDC_reset_validation01 Rule Violation

In the above example, the top-level reset `rst` reaches the `srst` block port on which no *reset* constraint is defined.

As a result, the *SGDC_reset_validation01* rule reports a violation.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_reset_validation02

Reports abstract-view reset ports that are not driven by top-level resets

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *reset* constraint.

Description

The *SGDC_reset_validation02* rule reports a violation if the *reset* constraint is specified for a block-level port, but no top-level reset drives that block-level port.

Parameter(s)

None

Constraint(s)

reset (Mandatory): Use this constraint to specify reset signals in your design.

Messages and Suggested Fix

The following message appears if the block-level port on which the *reset* constraint is specified is not driven by a top-level reset port:

```
[ERROR] Reset '<reset-name>' specified for block instance
'<block-instance name> (block: <block-name>) is not propagated
from top-level reset
```

Potential Issues

This violation appears if the port of an abstract view is constrained by the *reset* constraint, but no top-level reset drives that port.

Consequences of Not Fixing

If you do not fix this violation, the reported port of an abstract view may not be considered as a valid reset signal. This may alter the initial state of the block during verification.

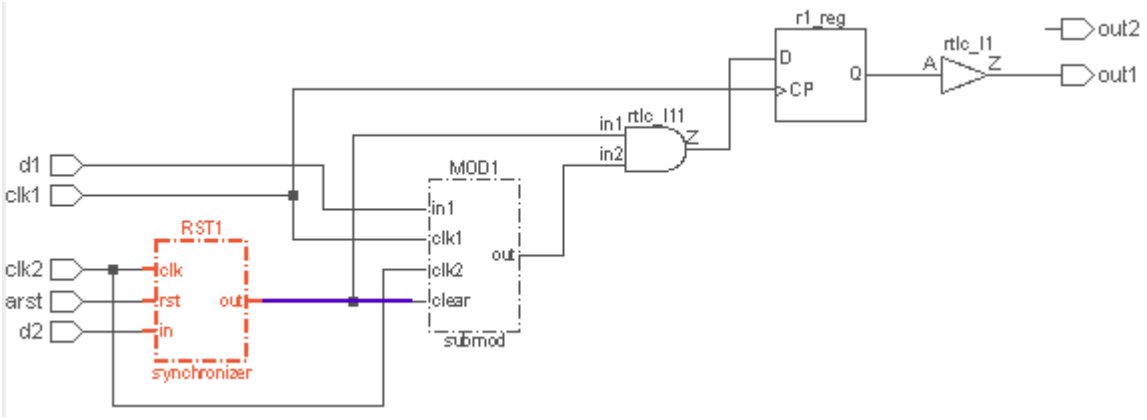
How to Debug and Fix

To debug and fix this violation, perform the following actions:

- Remove the *reset* constraint from the reported block port.
- Analyze the specification or propagation of a top-level reset to the block port.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// submod.sgdc

current_design submod
clock -name clk1
clock -name clk2
reset -name clear
```

FIGURE 386. Schematic of the SGDC_reset_validation02 Rule Violation

In the above example, the reset `clear` of the `MOD1` block is not driven by any top-level reset.

Therefore, the *SGDC_reset_validation02* rule reports a violation.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_reset_validation03

Reports conflicting top and block level asynchronous and synchronous reset types

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_reset_validation03* rule reports a violation if an asynchronous reset specified at a top-level reaches to a synchronous reset port of an abstract view or vice-versa.

Messages and Suggested Fix

Message 1

The following message appears if the top-level asynchronous reset *<reset-name>* propagates to a synchronous block-level port:

```
[SRstV3_1] [ERROR] Top-level asynchronous reset '<reset-name>'
reaches to the net '<net-name>' connected to synchronous reset
pin '<port-name>' of block instance '<block-instance name>
(block: <block-name>)'
```

Potential Issues

This violation appears if your design contains a top-level asynchronous reset that reaches to a synchronous reset port of an abstract view, or vice-versa.

Consequences of Not Fixing

If you do not fix this violation, the following may happen:

- Incorrect reset analysis may happen at the block-level. That is, the initial state of the block may get altered during its verification.
- SpyGlass may generate incorrect clock domain violations during block verification if synchronous resets are not properly specified.

How to Debug and Fix

Perform the following actions to fix this violation:

- Specify an appropriate *reset* constraint on the block-level port.
- Analyze the specification or propagation of the top-level reset to a block.

Message 2

The following message appears if the top-level synchronous reset *<reset-name>* propagates to an asynchronous block-level port:

```
[SRstV3_2] [ERROR] Top-level synchronous reset '<reset-name>'
reaches to the net '<net-name>' connected to asynchronous reset
pin '<port-name>' of block instance '<block-instance name>
(block: <block-name>)'
```

Potential Issues

This violation appears if your design contains a top-level synchronous reset that reaches to an asynchronous reset port of an abstract view, or vice-versa.

Consequences of Not Fixing

If you do not fix this violation, the following may happen:

- Incorrect reset analysis may happen at the block-level. That is, the initial state of the block may get altered during its verification.
- SpyGlass may generate incorrect clock domain violations during block verification if synchronous resets are not properly specified.

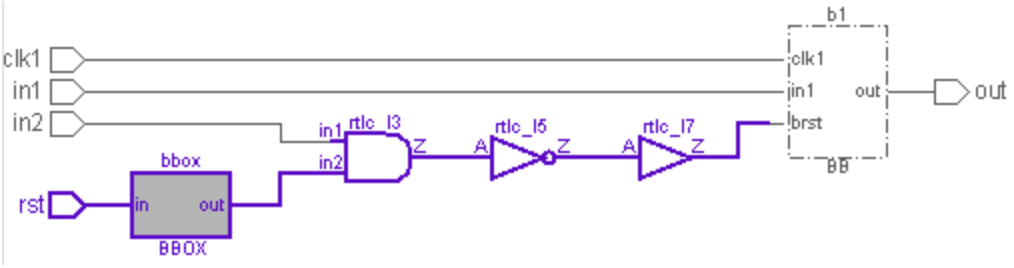
How to Debug and Fix

Perform the following actions to fix this violation:

- Specify an appropriate *reset* constraint on the block-level port.
- Analyze the specification or propagation of the top-level reset to a block.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// top.sgdc
current_design top
clock -name clk1 -domain domain1
reset -name rst
assume_path -name BBOX -input in -output out
sgdc -import BB block.sgdc

// block.sgdc
current_design BB
reset -name brst -sync
```

FIGURE 387. Schematic of the SGDC_reset_validation03 Rule Violation

In the above example, the top-level asynchronous reset `rst` reaches the synchronous reset port `brst` of the abstract view `BB`.

Therefore, the *SGDC_reset_validation03* rule reports a violation.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_reset_validation04

Reports the conflicting active value specified on a reset port of an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *reset* constraint for the top-level design and the port of an abstract view.

Description

The *SGDC_reset_validation04* rule reports a violation if the active value of the top-level reset is different from the active value of the block-level reset port driven by that top-level reset.

Parameter(s)

None

Constraint(s)

reset (Mandatory): Use this constraint to specify reset signals in your design.

Messages and Suggested Fix

The following message appears if the active value of the top-level reset does not match with the active value of the block-level reset port:

```
[ERROR] Reset '<reset-name>' specified for instance
'<inst-name>' (block: '<block-name>') has different
active-value compared to top level reset '<top-level-reset>'
```

Potential Issues

This violation appears when the reset value propagated from the top-level reset does not match with the reset value of the block-level reset port driven by that top-level reset.

Consequences of Not Fixing

If you do not fix this violation, the following may occur:

- It may result in an incorrect initial state of an abstract view during verification.
- It may generate spurious reset simulation results for the abstract view.

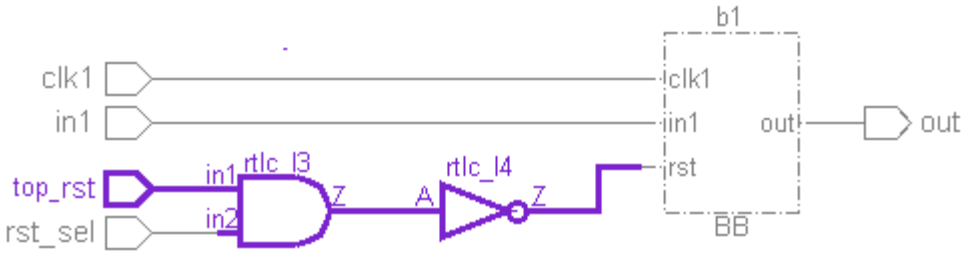
How to Debug and Fix

To debug and fix this violation, perform the following steps:

1. Check the value specified by the *reset* constraint on the block port.
2. Assign a proper value in the *reset* constraint.
3. Verify that the top-level reset of the block port has the same active value identified in step 2.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// top.sgdc
current_design top
clock -name clk1 -domain domain1
reset -name top_rst -value 0
sgdc -import BB block.sgdc
```

```
// block.sgdc
current_design BB
reset -name rst -value 0
abstract_port -module BB -ports in1
              -clock clk1
```

FIGURE 388. Schematic of the SGDC_reset_validation04 Rule Violation

In the above example, the active value of the top-level reset `top_rst` reaching the reset port `rst` does not match with the reset value of `rst`.

Therefore, the *SGDC_reset_validation04* rule reports a violation.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_virtualclock_validation01

Reports mapping of block level virtual clocks with top-level clocks

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_virtualclock_validation01* rule reports a message if the block-level virtual clocks map to the top-level clocks.

The rule reports a message in the following cases:

- Multiple ports of the same block specified with the same virtual clock are driven by different domains from top-level
- If no top-level clock is reaching the block port specified with virtual clock

Parameter(s)

validate_reduce_pessimism: Default value is `none`. Set this parameter to `hanging_nets` to ignore reporting on the hanging block ports. Other possible values are `constant`, `quasi_static`, `ignore_domain_overconstraint`, and `all`.

Constraint(s)

- *input* (Optional): Use this constraint to specify a clock domain at input ports.
- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.

Message Details

Message 1

The following message appears if a virtual clock is mapped with a top-level

clock:

```
[SCLKV01] [INFO] Virtual clock '<clock-name>' of instance
'<inst-name>' (block: '<block-name>') mapped with top-level
clock '<top level-clock>'
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 2

The following message appears if a virtual clock is not mapped.

```
[SCLKV4] [ERROR] Virtual clock '<clock-name>' of instance
'<inst-name>' (block: '<block-name>') is unmapped. Reason:
Conflicting domains reach to block ports
```

Potential Issues

This violation appears if the design contains multiple ports of the same block specified with the same virtual clock and these ports are driven by different domains from top-level.

Consequences of Not Fixing

See *Consequences of Not Fixing*.

How to Debug and Fix

See *How to Debug and Fix*.

Message 3

The following message appears if a virtual clock is not mapped.

```
[SCLKV3] [WARNING] Virtual clock '<clock-name>' of instance
'<inst-name>' (block: '<block-name>') is unmapped. Reason: No
top-level domain reaches to block port(s)
```

Potential Issues

This violation appears if the design does not contain any top-level clock reaching to the block port specified with virtual clock

Consequences of Not Fixing

If you do not fix these violations, SpyGlass analysis may produce inaccurate synchronization results during block verification, thereby generating incorrect abstract view.

This may further generate incorrect synchronization violations in the SoC verification stage.

How to Debug and Fix

To fix these violations, perform the following actions:

- Analyze design connectivity between the top-level sequential element and a block input port.
- Verify the virtual clock specified by the *abstract_port* or *input* constraint.
- Use the *validate_reduce_pessimism* parameter to ignore reporting on block ports that are hanging or have a constant value reaching them.

A spreadsheet is generated for each violation reported for unmapped virtual clock. It contains all the unmapped top-level sequential logic reaching the port specified with the reported virtual clock. The sample spreadsheet is shown below:

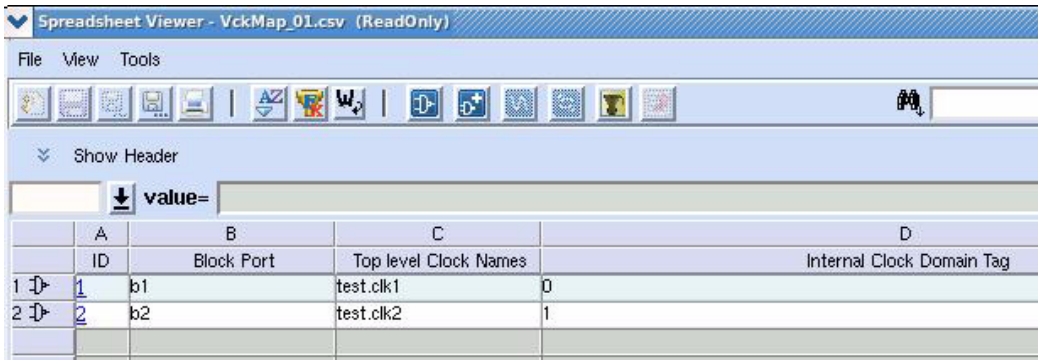
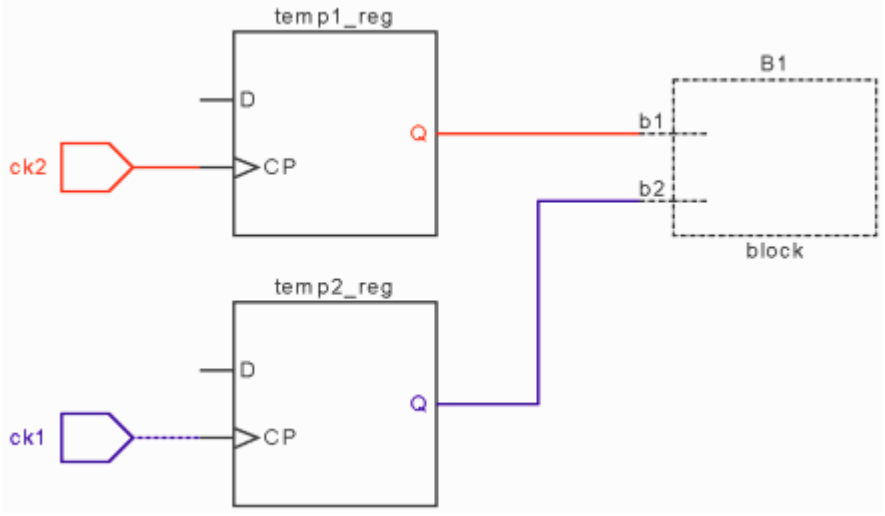


FIGURE 389. Spreadsheet Generated by the SGDC_virtualclock_validation01 Rule

Example Code and/or Schematic

Example 1 - Illustrates When Message 2 is Reported

Consider the following scenario:




```
// block.sgdc
abstract_port -module block -ports b1 -clock vck
abstract_port -module block -ports b2 -clock vck
```

FIGURE 390. Schematic of the SGDC_virtualclock_validation01 Rule Violation

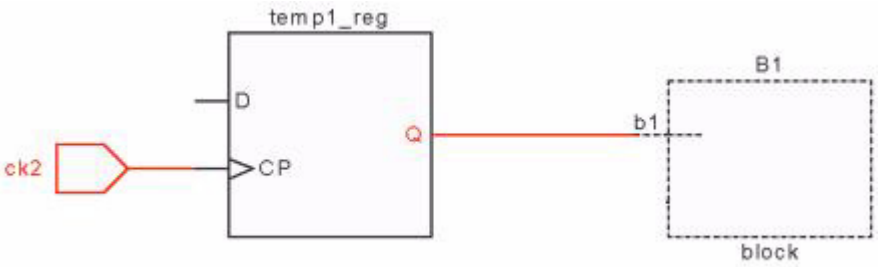
In the above scenario, `B1` is the abstract view in the top module, `test`.

The `b1` and `b2` block ports are constrained to be in the same virtual domain, but are driven by different domain clocks, `ck1` and `ck2`, respectively.

As a result, the *SGDC_virtualclock_validation01* rule reports the *Message 2* violation.

Example 2 - Illustrates When Message 3 is Reported

Consider the following schematic:




```
abstract_port -module block -ports b1 -
clock V1
```

FIGURE 391. Schematic of the SGDC_virtualclock_validation01 Rule Violation

In the above scenario, `B1` is the abstract view in the top module, `test`. The block port `b1` is constrained with two *abstract_port* constraints, one in real clock `ck1` and other in virtual clock `V1`.

However this port is driven by a sequential element in domain `clk1` which matches to the *abstract_port* constraint specified with real clock. As a result no unmapped top-level sequential logic is reaching the `b1` port for mapping with the `V1` virtual clock.

As a result, the *SGDC_virtualclock_validation01* rule reports the *Message 3* violation.

Default and Severity Label

ERROR | WARNING | INFO

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related files

The CKSGDCInfo Report

SGDC_input_validation01

Reports incorrect clock domain specified on block ports by using the input constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *input* constraint.

Description

The *SGDC_input_validation01* rule reports a violation if a sequential element reaching to an input port of a block is from a different domain than that of the clock specified on a port by using the *input* constraint.

NOTE: *The SGDC_input_validation01 rule will be deprecated in a future release. The functionality of the SGDC_input_validation01 rule is covered by the SGDC_abstract_port_validation01 rule. Currently, the SGDC_input_validation01 rule does not report any violation.*

Parameter(s)

None

Constraint(s)

input (Mandatory): Use this constraint to specify clock domain at input ports.

Messages and Suggested Fix

The following message appears if the instance output *<inst-output>* reaches to the input pin *<pin-name>* of the instance *<inst-name>* clocked by *<clock2>*:

[ERROR] `Instance output '<inst-output>' clocked by '<clock1>'`

```
reaches input pin '<pin-name>' of instance '<inst-name>'
(block: '<block-name>') clocked by '<clock2>'
```

Potential Issues

This violation appears if a sequential element reaching to an input port of a block is from a different domain than that of the clock specified on the port by using *input* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during the verification phase of the hierarchical verification flow.

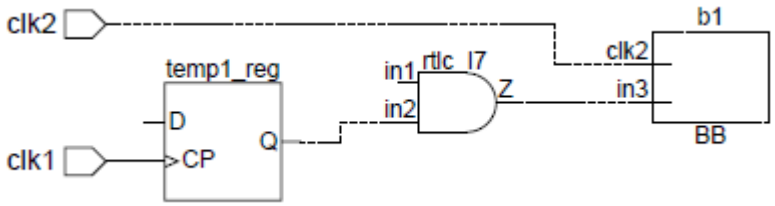
How to Debug and Fix

To debug this violation, perform the following steps:

- Verify the clock domain specified in by the *input* constraint.
- Analyze design connectivity between the top-level sequential element and block input port.

Example Code and/or Schematic

Consider the following figure:




```
// Block-level SGDC file
current_design BB
input -name in3 -clock clk2
```

FIGURE 392. Schematic of the SGDC_input_validation01 Rule Violation

In the above figure, the output of the sequential cell driven by the `clk1` clock reaches the `in3` port of the `BB` block.

Therefore, the *SGDC_input_validation01* rule reports a violation in this case.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_input_validation02

Reports unconstrained abstract-view input ports driven by sequential outputs

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_input_validation02* rule reports a violation if a sequential element from top-level is reaching to a block-level port on which the *input* or *abstract_port* constraint is not defined.

Rule Exceptions

The *SGDC_input_validation02* rule does not report a violation under the following conditions:

- If the input of a block is constrained by using the *quasi_static* or *assume_path* constraint.
- When the clock net driving the clock port specified in the `-clock` field of the *abstract_port* constraint is hanging. However, the violation by the *SGDC_clock_validation02* rule is reported for hanging top-level clock net.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears if the *input* or *abstract_port* constraint is not

specified for the port *<port-name>* of the instance *<inst-name>*:

```
[ERROR] Input/abstract_port constraint is not specified for
port '<port-name>' of instance '<inst-name>' (block:
'<block-name>')
```

Potential Issues

This violation appears if a sequential element from the top-level reaches to a block-level port on which the *input* or *abstract_port* constraint is not defined.

Consequences of Not Fixing

If you do not fix this violation, the following may occur:

- SpyGlass may report incorrect synchronization violations during the verification phase of the hierarchical verification flow.
- Abstract view generation may be incorrect.

How to Debug and Fix

To fix this violation, perform the following steps:

- Specify the *input* or *abstract_port* constraint on the reported block input port.
- Analyze design connectivity between the top-level sequential element and block input port.

Example Code and/or Schematic

Consider the following figure:

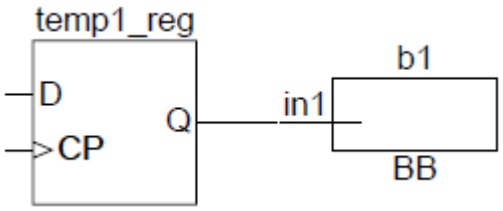


FIGURE 393. Schematic of the SGDC_input_validation02 Rule Violation

In the above figure, the output of the sequential cell hits the `in1` port of the `BB` block. In this case, no *input* or *abstract_port* constraint is defined for that port.

Therefore, the *SGDC_input_validation02* rule reports a violation in this case.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_num_flops_validation01

Reports the same top-level domain reaching to clocks specified in the -from_clk and -to_clk arguments of the num_flops constraint for an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following details before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *num_flops* constraint.

Description

The *SGDC_num_flops_validation01* rule reports a violation if clocks specified by the `-from_clk` and `-to_clk` arguments of the *num_flops* constraint in an SGDC file of an abstract view exist in the same top-level domain.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if you specify the same clock name in the `-from_clock` and the `-to_clock` argument of the *num_flops* constraint:

```
[WARNING] For block instance '<block-inst>' (block:
'<block-name>'), clock domain of clock '<clock-name1>' in -
from_clk and clock '<clock-name2>' in -to_clk is same in
num_flops constraint.
```

Potential Issues

This violation appears if clocks specified by the `-from_clk` and `-to_clk` arguments of the *num_flops* constraint exist in the same domain.

Consequences of Not Fixing

If you do not fix this violation, the following may occur depending upon different situations:

- If the *num_flops* constraint is incorrectly specified in the block-level SGDC file, it may result in inaccurate block-level CDC verification.
- If domains of top-level clocks are not specified correctly, it may result in inaccurate top-level CDC verification.

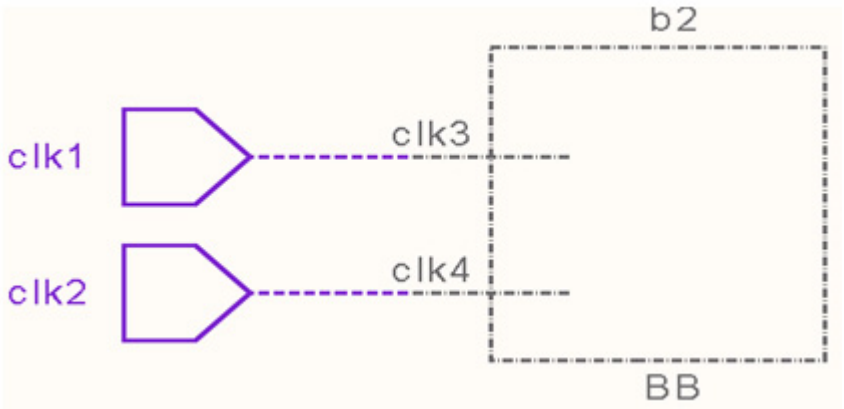
How to Debug and Fix

To fix this violation, perform the following actions:

- Analyze the clock specification in the `-from_clk` and `-to_clk` arguments of the *num_flops* constraint.
- Analyze the specification or propagation of top-level clocks.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// Top-level SGDC file
    current_design top
    clock -name clk1 -domain domain1
    clock -name clk2 -domain domain1

// Block-level SGDC file
    current_design BB
    clock -name clk3 -domain domain1
    clock -name clk4 -domain domain2
    num_flops -from_clk clk3 -to_clk clk4
```

FIGURE 394. Schematic of the SGDC_num_flops_validation01 Rule Violation

In the above example, the top-level clocks `clk1` and `clk2` are in the same domain. However, they are driving different domain block-level clocks `clk3` and `clk4`.

Therefore, this rule reports a violation because the *num_flops* constraint is specified for the same clock domain signals.

Default Severity Label

Warning

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_num_flops_validation02

Reports conflicting values specified in the num_flops constraint of an abstract view and the top-level for the same clock pair

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following details before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *num_flops* constraint.

Description

The *SGDC_num_flops_validation02* rule reports a violation if the value of the *num_flops* constraint defined in the top-level SGDC file for a clock-pair is different from the value of the *num_flops* constraint defined in the block-level SGDC file for the same clock pair.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if conflicting values are specified in the *num_flops* constraint for the abstract-view SGDC file and the top-level SGDC file for the same clock pair:

```
[WARNING] Value '<value1>' defined for top-level num_flops
constraint (-from_clk '<from-clk>' and -to_clk '<to-clk>') does
not match with value '<value2>' for corresponding constraint
defined for block instance '<inst-name>' (block:
```

'<block-name>')

Potential Issues

This violation appears when the value specified by the *num_flops* constraint for a block-level and a top-level SGDC file is different for the same clock pair.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may produce inaccurate CDC verification results.

How to Debug and Fix

To fix this violation, perform the following steps:

1. Analyze the values of the `-from_clk` and `-to_clk` arguments of the *num_flops* constraint defined for an abstract view.
2. Analyze the values of the `-from_clk` and `-to_clk` arguments of *num_flops* constraint defined for the top-level.
3. Analyze the specification or propagation of top-level clocks.

Example Code and/or Schematic

Consider the following SGDC files specified for an abstract view and for the top-level:

```
// Top-level SGDC file

current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d2
num_flops -from_clk clk1 -to_clk clk2 -value 4


// Block-level SGDC file

current_design BB
clock -name clk3 -domain d1
clock -name clk4 -domain d2
num_flops -from_clk clk3 -to_clk clk4 -value 2
```

In this example, the top-level clocks `clk1` and `clk2` drive the block-level clocks `clk3` and `clk4`, respectively.

In this case, *SGDC_num_flops_validation02* rule reports a violation because the value specified in the *num_flops* constraint at block-level is `2`, whereas the value is `4` at the top-level for the same set of clocks.

Default Severity Label

Warning

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related files

SGDC_output_validation01

Reports incorrect clock domains specified on block ports by using the output constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *output* constraint.

Description

The *SGDC_output_validation01* rule reports a violation if the output port of a block at the top-level drives a sequential element from a domain different from the domain of the clock specified on that port by using the *output* constraint.

Parameter(s)

None

Constraint(s)

output (Mandatory): Use this constraint to specify clock domain at output ports.

Messages and Suggested Fix

The following message appears if an incorrect clock domain is specified on a block port by using the *output* constraint:

```
[WARNING] Instance '<inst-name>' (domain: '<domain1>') is
driven by output pin '<pin-name>' of block instance
'<block-inst-name>' (block: '<block-name>') constrained with
clock '<clock-name>' (domain: '<domain2>')
```

Potential Issues

This violation appears if your design contains a sequential instance driven by an output pin, which is constrained with a different clock.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report spurious violations related with clock domain crossing during abstract-view verification.

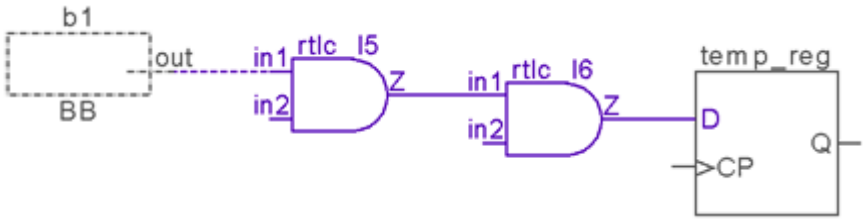
How to Debug and Fix

To fix this violation, perform the following steps:

1. Verify the clock domain specified by the *output* constraint.
2. Analyze design connectivity between the block output port and the top-level sequential element.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
output -name out -clock clk1
```

FIGURE 395. Schematic of the SGDC_output_validation01 Rule Violation

In the above schematic, the `out` port of the `BB` abstract view drives the `temp_reg` sequential cell, which is in the `clk2` domain (`CP` is connected to `clk2`).

In this case, the *SGDC_output_validation01* rule reports a violation because the `out` port is constrained with a different clock domain.

Default Severity Label

Warning

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_output_validation02

Reports unconstrained abstract-view output ports driving sequential inputs

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *output* constraint.

Description

The *SGDC_output_validation02* rule reports a violation if a block-level output port drives a sequential element at the top-level and the *output* constraint is not defined on the block port.

Parameter(s)

None

Constraint(s)

output (Mandatory): Use this constraint to specify clock domain at output ports.

Messages and Suggested Fix

The following message appears if you do not define the output constraint for the block port *<port-name>*:

[WARNING] Output constraint is not defined for port '<port-name>' of block instance '<inst-name>' (block: '<block-name>')

Potential Issues

This violation appears if domain information at the output port of an

abstract view is not present.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report spurious violations related with clock domain crossing during abstract-view verification.

How to Debug and Fix

To fix this violation, perform the following steps:

- Specify the *output* constraint on the reported block output port.
- Analyze design connectivity between the block output port and top-level sequential element.

Example Code and/or Schematic

Consider the following figure:

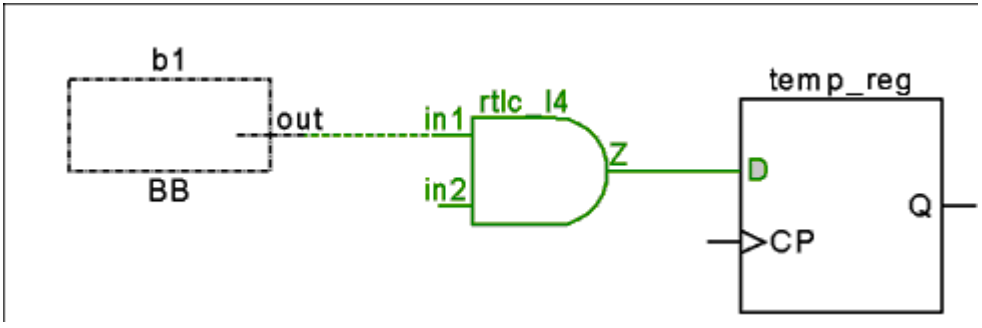


FIGURE 396. Schematic of the SGDC_output_validation02 Rule Violation

In the above example, the `out` port of the abstract view `BB` drives the sequential cell `temp_reg`. However, no *output* constraint is applied on the `out` port.

Therefore, the *SGDC_output_validation02* rule reports a violation.

Default Severity Label

Warning

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_abstract_port_validation01

Reports the incorrect clock domain specified on block ports by using the abstract_port constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *abstract_port* constraint.

Description

The *SGDC_abstract_port_validation01* rule reports a violation if the port of an abstract block is driven from a sequential instance, and there is a mismatch between the clock pin driving this sequential instance and the clock specified in the `-clock` argument of the *abstract_port* or the *input* constraint.

Rule Exceptions

The *SGDC_abstract_port_validation01* rule does not report a violation if an input port specified by the *abstract_port* or the *input* constraint is driven by a quasi-static signal.

Parameter(s)

None

Constraint(s)

- *abstract_port* (Mandatory): Use this constraint to define abstracted information for block ports.
- *input* (Optional): Use this constraint to define the clock information of the block port.

Messages and Suggested Fix

The following message appears if an incorrect clock domain is specified on a block port by using the *abstract_port* or the *input* constraint:

```
[ERROR] Instance output '<inst-output>' clocked by
'<clock-name1>' reaches port '<port-name>' of block instance
'<block-inst>' (block: '<block-name>') clocked by
'<clock-name2>'
```

Potential Issues

This violation appears if the port of an abstract block is driven from a sequential instance, and there is a mismatch between the clock pin driving this sequential instance and the clock specified in the `-clock` argument of the *abstract_port* or the *input* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during verification phase of the hierarchical verification flow.

How to Debug and Fix

To fix this violation, perform the following steps:

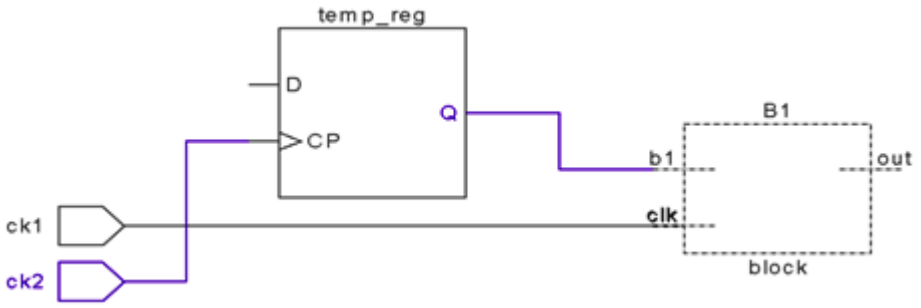
1. Analyze the design connectivity between the top-level sequential element and the block input port.
2. Check if the clock domain specified by the *abstract_port* or the *input* constraint is consistent with the clock domains driving sequential elements identified in the first step.
3. Check sgdc (abstract_port) back-annotation for block abstract_port for which the violation was reported and back-annotation of top level sequential element to indicate the differing clock.

Example code and/or Schematic

Consider a case in which an output of a sequential flip-flop is connected to an input port of an abstract view defined by using the *abstract_port* or the *input* constraint. In this case, the *SGDC_abstract_port_validation01* rule

reports a violation if the clock driving the sequential flip-flop does not match the clock specified by the `clock` argument of the *abstract_port* or the *input* constraint.

For example, consider the following schematic of a violation reported by this rule:




```
//block.sgdc
abstract_port -module block -ports b1 -clock clk
```

FIGURE 397. Schematic of the *S*GDC_abstract_port_validation01 Rule Violation

For the above example, the *SGDC_abstract_port_validation01* rule reports the following violation because of the mismatch in the clock domains:

```
Instance output 'test.temp' clocked by 'test.ck2' reaches port
'b1' of block instance 'test.B1' (block: 'block') clocked by
'test.ck1'
```

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and related files

No report or related file

SGDC_abstract_port_validation02

Reports incorrect usage of the -sync argument of the abstract_port constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *abstract_port* constraint.

Description

The *SGDC_abstract_port_validation02* rule reports a violation if a synchronizer does not reach to an input port of a block for which the `-sync` argument of the *abstract_port* constraint is specified.

Parameter(s)

None

Constraint(s)

- *abstract_port* (Mandatory): Use this constraint to define abstracted information for block ports.

Messages and Suggested Fix

The following message appears if a synchronizer is not reaching to the input pin *<pin-name>* of the block *<block-name>* for which the `-sync` argument of the *abstract_port* constraint is specified:

```
[ERROR] No synchronized signal reaches to input pin '<pin-
name>' of block instance '<block-inst>'(block: '<block-name>')
however -sync option has been provided
```

Potential Issues

This violation appears if your design contains an abstract block port that is not driven by a valid multi-flop or user-specified synchronizing cell when the `-sync` argument of the *abstract_port* constraint is specified for that abstract block.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect synchronization violations during the block verification stage.

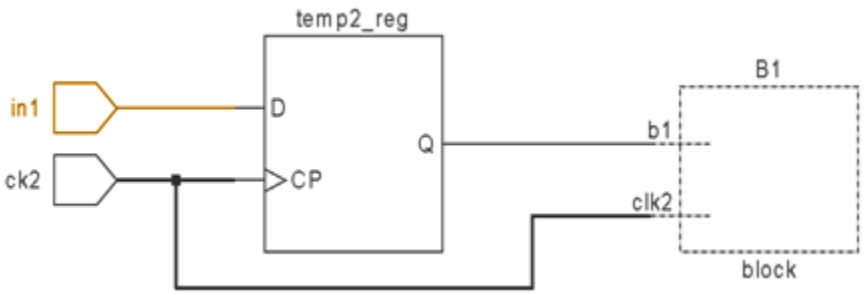
How to Debug and Fix

To fix this violation, perform any of the following actions:

- Analyze the fan-in cone of an input port for the presence of a synchronizer.
- Remove the `-sync` argument from the *abstract_port* constraint.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
//block.sgdc
abstract_port -module block -ports b1 -clock clk2
              -from clk1 -to clk2 -sync active
```

FIGURE 398. Schematic of the SGDC_abstract_port_validation02 Rule Violation

In the above example, the `temp2_reg` synchronizer is not reaching to the input pin of the abstract block port `b1`.

As a result, the *SGDC_abstract_port_validation02* rule reports a violation.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and/or Related Files

No report or related file

SGDC_abstract_port_validation03

Reports invalid clocks specified in the -from or -to arguments of the abstract_port constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *abstract_port* constraint.

Description

The *SGDC_abstract_port_validation03* rule reports a violation if clocks specified by the `-from` or `-to` argument of the *abstract_port* constraint does not match with the source or destination clocks of a synchronizer reaching to an input port.

NOTE: *The SGDC_abstract_port_validation03 rule will be deprecated in a future release. The functionality of the SGDC_abstract_port_validation03 rule is covered by the SGDC_qualifier_validation02 rule. Currently, the SGDC_abstract_port_validation03 rule does not report any violation.*

Rule Exceptions

The *SGDC_abstract_port_validation03* rule does not report a violation if you specify a virtual clock in the `-from` argument of the *abstract_port* constraint, as shown in the following example:

```
abstract_port -ports in1 -clocks clk1 -from VCLK -to clk1
-sync active
```

Parameter(s)

None

Constraint(s)

- *abstract_port* (Mandatory): Use this constraint to define abstracted information for block ports.

Messages and Suggested Fix

Message 1

The following message appears if a clock specified in the `-from` argument of the *abstract_port* constraint does not match with the clock of a synchronizer reaching to an input port:

```
[SApV3_1] [WARNING] Clock domain of source instance '<source-inst>' clocked by '<clock-name>' does not match with block clock port(s) '<block-clock-port-name>' specified in -from field for input port '<input-port>' of block instance '<block-inst>' (block: '<block-name>')
```

Potential issues

This violation appears if there is a mismatch between the clock specified in the `-from` argument of the *abstract_port* constraint and the clock of the synchronizer reaching to an input port.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report spurious synchronization violations during verification phase.

How to Debug and Fix

To fix this violation, ensure that the clock specified in the `-from` argument of the *abstract_port* constraint and the clock of the synchronizer reaching to an input port is same.

Message 2

The following message appears if a clock specified in the `-to` argument of the *abstract_port* constraint does not match with the clock of a synchronizer reaching to an input port:

```
[SApV3_2] [WARNING] Clock domain of destination instance
'<dest-inst>' clocked by '<clock-name>' does not match with
block clock port(s) '<block-clock-port-name>' specified in -to
field for input port '<input-port>' of block instance
'<block-inst>'(block: '<block-name>')
```

Potential issues

This violation appears if there is a mismatch between the clocks specified in the `-to` argument of the *abstract_port* constraint and the clock of the synchronizer reaching to an input port:

Consequences of Not Fixing

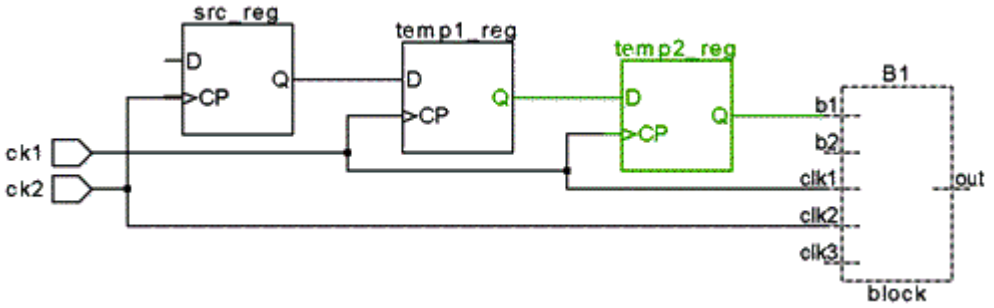
If you do not fix this violation, SpyGlass may report spurious synchronization violations during verification phase.

How to Debug and Fix

To fix this violation, ensure that the clocks specified in the `-to` argument of the *abstract_port* constraint and the clock of the synchronizer reaching to an input port is same.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule in which the `B1` block is abstracted in the design unit `test`:




```
//block.sgdc
abstract_port -module block -ports b1 -clock clk3
              -from clk2 -to clk3 -sync active
```

FIGURE 399. Schematic of the SGDC_abstract_port_validation03 Rule Violation

In the above example, the *SGDC_abstract_port_validation03* rule reports a violation because the `clk3` clock specified by the `-to` argument of the *abstract_port* constraint does not match with the `clk1` clock.

Default Severity label

Warning

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_abstract_port_validation04

Reports if abstract-view ports specified by the -combo no argument of the abstract_port constraint are driven by combinational logic

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *abstract_port* constraint.
- Specify the *sgdc* `-import` constraint.

Description

The *SGDC_abstract_port_validation04* rule reports a violation if a combinational logic exists between a block-level input port and the output of a sequential element at the top-level when the `-combo no` argument of the *abstract_port* constraint is specified for that block in the following cases:

- If at the block level, the abstract_port constraint is defined along with the -combo_ifn argument as shown below:

  ```
  abstract_port  -ports a -clock VCK1 -combo_ifn ck2 -combo no
  ```

  In this case, the SGDC_abstract_port_validation04 rule will report a violation if a sequential element reaches the block port after combinational logic and if the sequential cell has a clock domain that is different from the clock domain of the clock specified in the `-combo_ifn` argument.

- If at the block level, the abstract_port constraint is defined with real clocks as shown below:

  ```
  abstract_port  -ports a -clock clk1 -combo no
  ```

  In this case, the SGDC_abstract_port_validation04 rule will report a

violation if a sequential element reaches the block port after combinational logic and if the sequential cell has a clock domain that is different from the clock domain of the clock specified in the -clock argument.

- If at the block level, the abstract_port constraint is defined with only virtual clock as shown below:

  ```
  abstract_port  -ports a -clock VCK1 -combo no
  ```

  In this case, the SGDC_abstract_port_validation04 rule will report a violation if the sequential element reaches the block port after combinational logic.

Automatically Fixing the abstract_port Constraint of the Reported Port

Set the *autofix_abstract_port* parameter to `yes` to modify the *abstract_port* constraint in the context of SoC for the reported port, and save the modified constraints in the *<module_name>_<instance_name>_cdc_soc_abstract_autoupdate.sgdc* file.

By default, in addition to the modified constraints, this file also contains a copy of all the unmodified input side *abstract_port* constraints present in block-level SGDC file (abstract block). Set the *autofix_dump_allinputs* to `no` to generate only the modified constraints.

The following figure shows the example of using the *autofix_abstract_port* parameter:

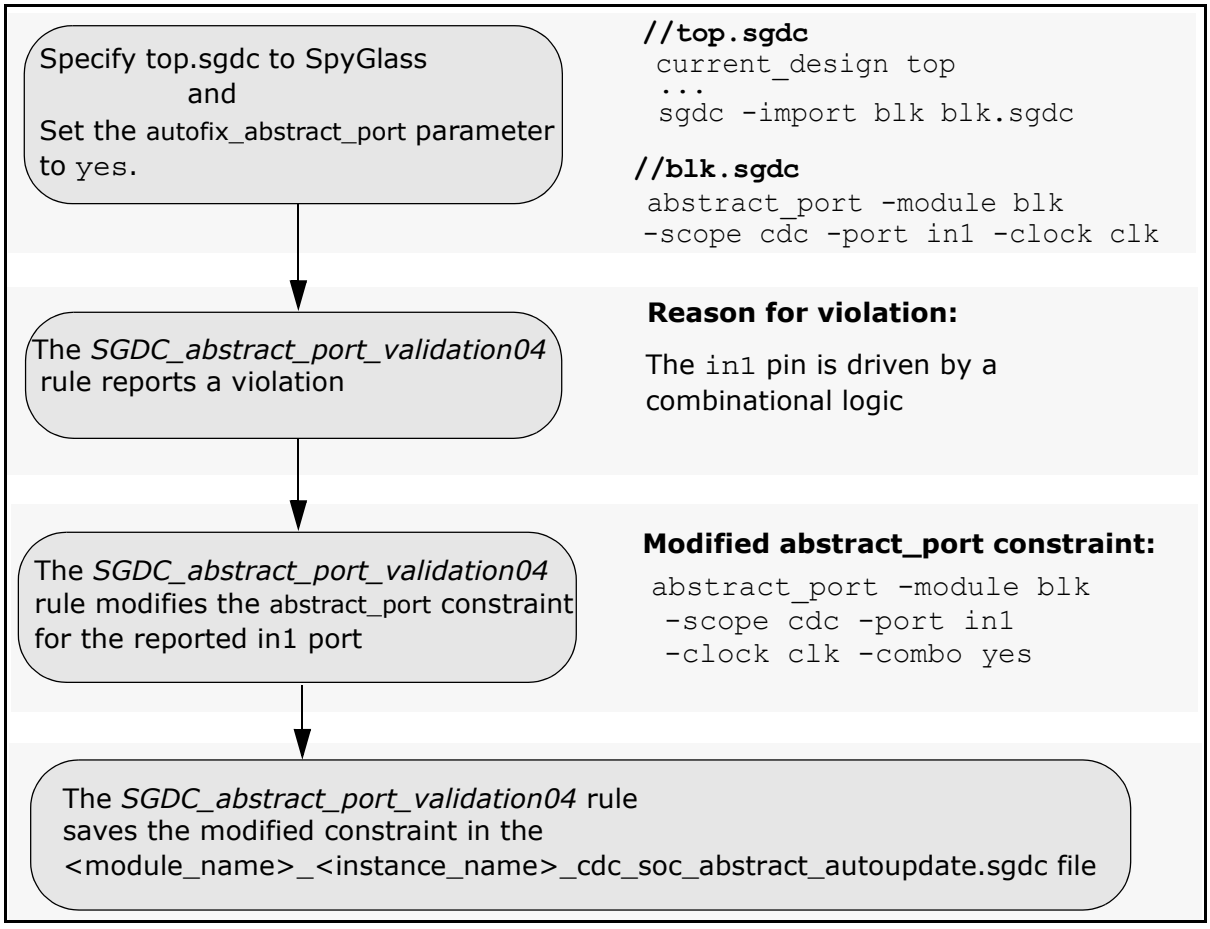


FIGURE 400. Using the autofix_abstract_port parameter

Parameter(s)

- *autofix_abstract_port*: Default value is `yes`. Set this parameter to `no` to disable this rule from modifying the reported *abstract_port* constraints in the context of SoC.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *abstract_port* (Mandatory): Use this constraint to define abstracted information for block ports.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *sgdc* `-import` (Mandatory): Use this constraint to specify a block-level SGDC file to be imported.

Messages and Suggested Fix

The following message appears if a combinational logic exists between a block-level input port and the output of a sequential element when the `-combo no` argument of the *abstract_port* constraint is specified for that block and any of the cases described in the Description section above is true:

[WARNING] Combinational logic exists between instance '<inst-name>' and input port '<port-name>' of block instance '<block-inst>' (block: '<block-name>')

Potential Issues

This violation appears if a combinational logic exists between a block-level input port and the output of sequential element at the top-level when the `-combo no` argument of the *abstract_port* constraint is specified for that block and any of the cases described in the Description section above is true.

Consequences of Not Fixing

If you do not fix this violation, it results in an incorrect setup at the block level.

How to Debug and Fix

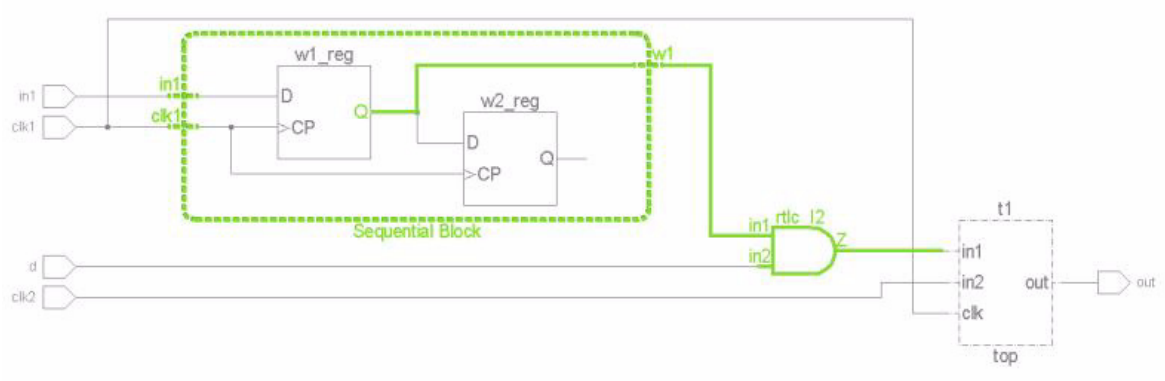
To fix this violation, perform the following actions:

- Remove the `-combo no` argument from the *abstract_port* constraint.
- Analyze the combinational logic between the sequential element and block-level input port.

Example Code and/or Schematic

Example 1

Consider the following schematic of a violation of this rule:



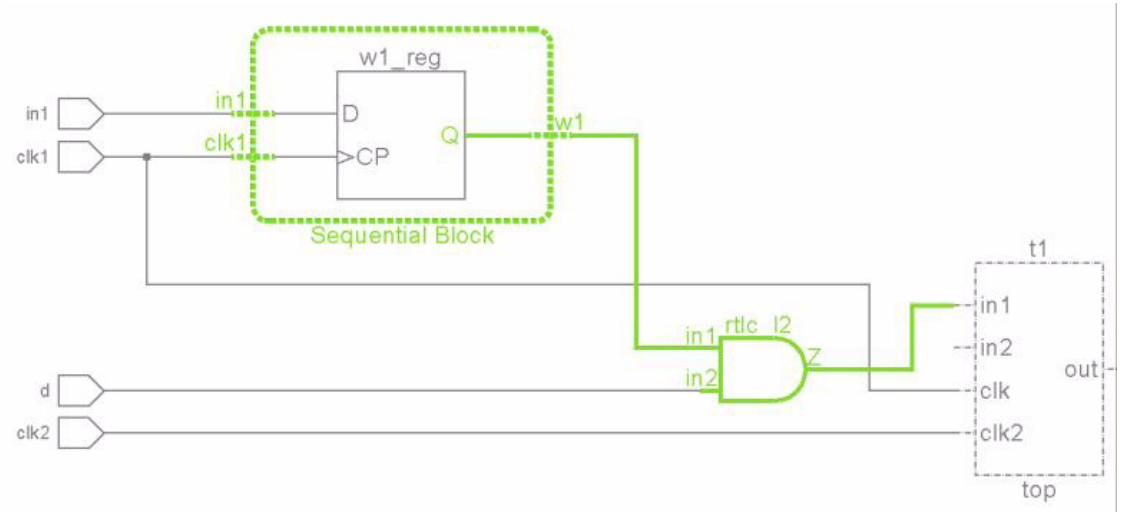

```
//test.sgdc

current_design test
clock -name clk1

clock -name clk2
sgdc -import top top.sgdc


//top.sgdc

current_design top
clock -name clk
clock -name in2
```

FIGURE 401. Schematic of the SGDC_abstract_port_validation04 Rule Violation

In the above example, a combinational logic exists between the output of the `w1_reg` flip-flop and the `in1` input pin of the `t1` block for which the `-combo no` argument of the *abstract_port* constraint is specified. The `w1_reg` flip-flop is driven by `clk1` clock and the `in1` input pin of `t1` block is driven by a combination of `clk` and `in2` clocks. Since the domain of these two clocks are different, the *SGDC_abstract_port_validation04* rule reports a violation.

Example 2

Consider the following schematic of a violation of this rule:




```
//test.sgdc

current_design test
clock -name clk1
clock -name clk2
sgdc -import top top.sgdc


//top.sgdc

current_design top
clock -name clk
clock -name clk2
abstract_port -module top -ports in1 -scope cdc -clock
```

FIGURE 402. Schematic of the SGDC_abstract_port_validation04 Rule Violation

In the above example, a combinational logic exists between the output of the `w1_reg` flip-flop and the `in1` input pin of the `t1` block for which the abstract_port constraint has the `-combo no` argument and the `-clock` argument has a virtual clock. The `w1_reg` flip-flop is driven by the

`clk1` clock and the `in1` input pin of `t1` block is driven by the `clk2` clock as specified in the `-combo_ifn` argument.

Since the domain of these two clocks are different and a combinational logic exists between the two, the *SGDC_abstract_port_validation04* rule reports a violation.

Default and Severity label

Warning

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_qualifier_validation01

Reports same top-level domain reaching to clocks specified in the -from_clk and -to_clk arguments of the qualifier constraint for an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *qualifier* constraint.

Description

The *SGDC_qualifier_validation01* rule reports a violation if clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint for an abstract view exist in the same top-level domain.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message appears when the clocks *<clock-name1>* and *<clock-name2>* specified in the `-from_clk` and `-to_clk` arguments, respectively, of the *qualifier* constraint are from the same domain:

```
[ERROR] For block instance '<block-inst>' (block:
<block-name>), clock domain of clock '<clock-name1>' in
-from_clk and clock '<clock-name2>' in -to_clk is same (domain:
'<domain-name>') in qualifier constraint
```

Potential Issues

This violation appears when the same top-level clock domain propagates to the clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint specified for an abstract view.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report spurious violations for clock domain crossing during abstract-view verification.

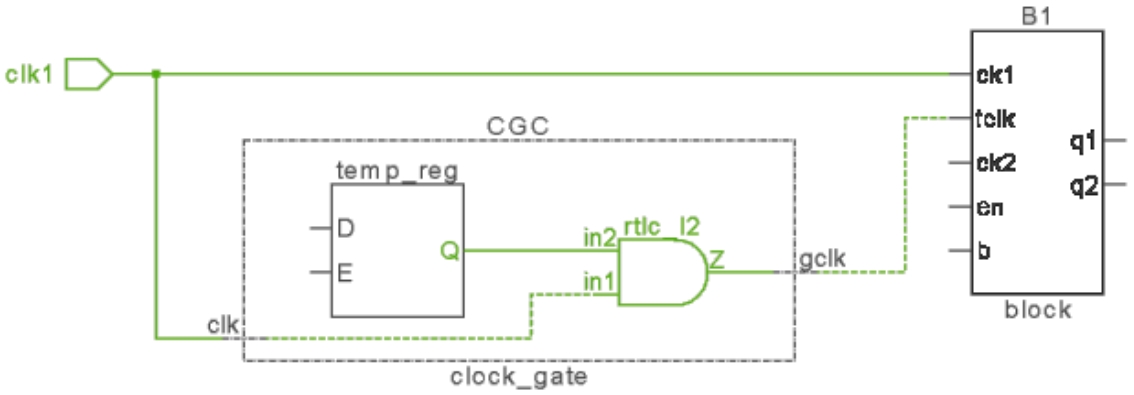
How to Debug and Fix

To debug this violation, perform the following actions:

- Analyze the clocks specified in the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint.
- Analyze specification or propagation of top-level clocks.

Example Code and/or Schematic

Consider the abstract view `B1` in the following schematic:




```
//block.sgdc
qualifier -name en -from_clk ck1 -to_clk tclk
```

FIGURE 403. Schematic of the SGDC_qualifier_validation01 Rule Violation

In the above example, domain of the top-level clock `clk1` propagates to both the clocks `ck1` and `tclk` specified by the `-from_clk` and `-to_clk` arguments, respectively, of the *qualifier* constraint.

Therefore, the *SGDC_qualifier_validation01* rule reports a violation in this case.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No reports or related files

SGDC_qualifier_validation02

Reports unconstrained abstract-view ports driven from a valid qualifier

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_qualifier_validation02* rule reports a violation if a synchronized signal reaches to an input port of a block for which:

- The *qualifier* or *abstract_port* constraint is not defined with the `-sync` argument, or
- The *abstract_port* constraint is defined with the `-sync` argument, but clocks specified by the `-from` or `-to` argument of that `abstract_port` constraint do not match with the source or destination clocks of the synchronizer reaching to that input port.

Rule Exceptions

The *SGDC_qualifier_validation02* rule does not report a violation in the following cases:

- If you specify the *abstract_port* constraint with the `-ignore` argument.
- If you specify the *assume_path* constraint only.

Automatically Fixing the abstract_port Constraint of the Reported Port

The **CDC SoC abstract auto update flow** works if a synchronized signal reaches to a block port and any of the following cases is true:

- *abstract_port* is defined at the block port without -sync and the domain of the synchronized signal matches with the clock specified in abstract_port

- *assume_path* is defined at the block port

Set the *autofix_abstract_port* parameter to `yes` to modify the *abstract_port* constraint in the context of SoC for the reported port, and save the modified constraints in the *<module_name>_<instance_name>_cdc_soc_abstract_autoupdate.sgdc* file.

By default, in addition to the modified constraints, this file also contains a copy of all the unmodified input side *abstract_port* constraints present in block-level SGDC file (abstract block). Set the *autofix_dump_allinputs* to `no` to generate only the modified constraints.

The following figure shows the example of using the *autofix_abstract_port* parameter:

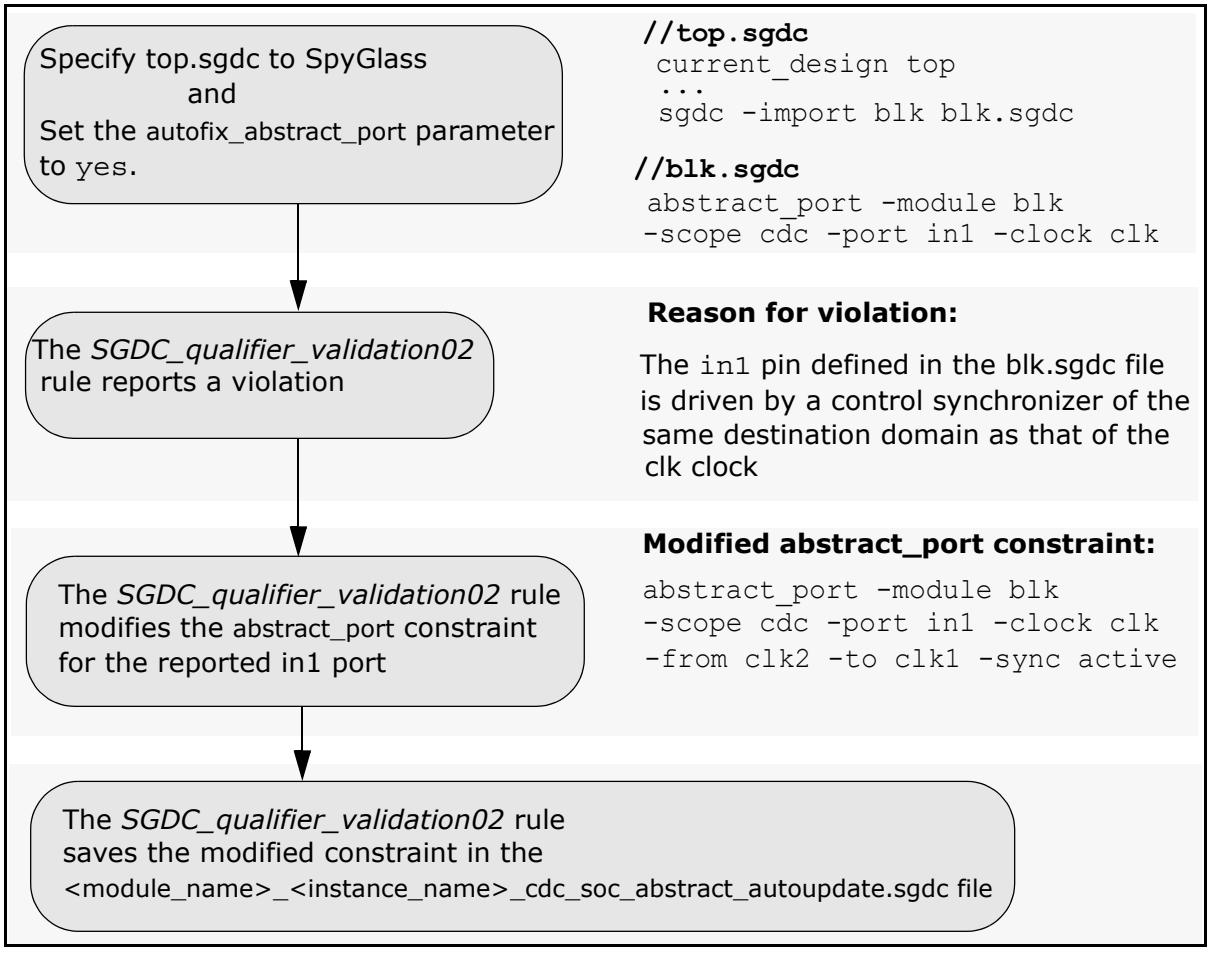


FIGURE 404. Using the autofix_abstract_port parameter

Parameter(s)

autofix_abstract_port: Default value is `yes`. Set this parameter to `no` to disable this rule from modifying the reported *abstract_port* constraints in the context of SoC.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *abstract_port* (Mandatory): Use this constraint to define abstracted information for block ports.

Messages and Suggested Fix

The following message appears when the *qualifier* or *abstract_port* constraint is not defined with the `-sync` argument for the port *<port-name>* of the block instance *<block-inst>* on which the synchronizer *<synchronizer>* is connected:

```
[WARNING] Qualifier/abstract_port not defined for port
'<port-name>' of block instance '<block-inst>' (block: <block-
name>) on which a synchronizer '<synchronizer>' is reaching
```

Potential Issues

This violation appears if a valid synchronizer reaches a port of the abstract view for which the *qualifier* or *abstract_port* constraint is not defined with the `-sync` argument.

Consequences of Not Fixing

If you do not fix this violation, many synchronized clock domain crossings within the abstract view may be reported as unsynchronized during verification.

How to Debug and Fix

To debug and fix this violation, perform the following actions:

- Specify the *qualifier* or *abstract_port* constraint with the `-sync` argument on the reported block port.
- Analyze design connectivity between the synchronized signal and the block input port.

Example Code and/or Schematic

Consider the abstract view `B1` in the following schematic:

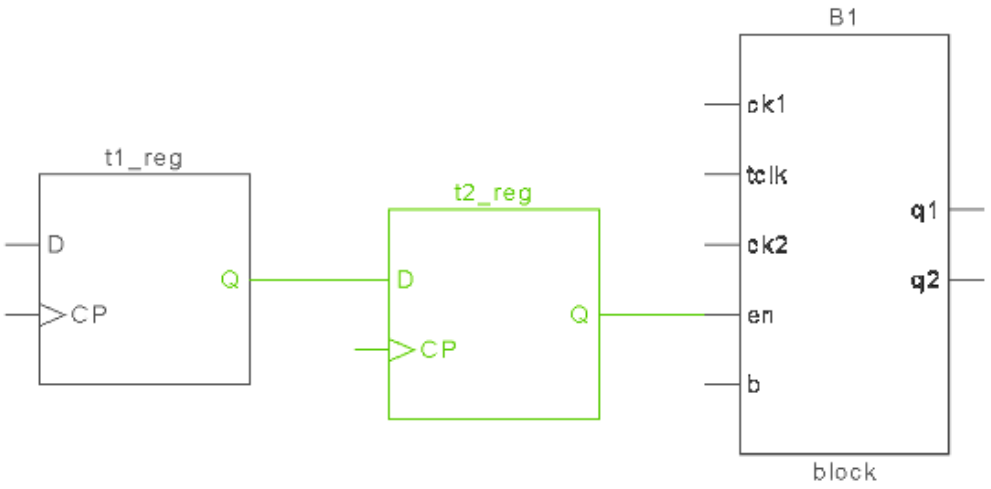


FIGURE 405. Schematic of the *SGDC_qualifier_validation02* Rule Violation

In the above example, the `en` port of the abstract block `B1` is not constrained by the *qualifier* or *abstract_port* constraint.

Therefore, the *SGDC_qualifier_validation02* rule reports a violation in this case.

Default Severity Label

Warning

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_cdc_false_path_validation01

Reports same top-level domain reaching to clocks specified in the -from and -to arguments of the cdc_false_path constraint for an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *cdc_false_path* constraint.
- Specify the *clock* constraint.

Description

The *SGDC_cdc_false_path_validation01* rule reports a violation if the clock domain of clocks specified by the `-from` and `-to` arguments of the *cdc_false_path* constraints in an SGDC file of an abstract view exists in the same top-level domain.

Parameter(s)

None

Constraint(s)

- *cdc_false_path* (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking
- *clock* (Mandatory): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears when the clock domain of clocks specified by the `-from` and `-to` arguments of the *cdc_false_path* constraints in an

SGDC file of an abstract view exists in the same top-level domain:

```
[WARNING] For block instance '<block-inst>' (block: <block-
name>), clock domain of clock '<clock-name1>' in -from and
clock '<clock-name2>' in -to is same (domain: '<domain-name>')
in cdc_false_path constraint
```

Potential Issues

This violation appears if the clock domain of clocks specified by the `-from` and `-to` arguments of the *cdc_false_path* constraints in an SGDC file of an abstract view exists in the same top-level domain.

Consequences of Not Fixing

If you do not fix this violation, false paths for clock domain crossings are not set up as per your expectations. This may also result in increased clock synchronization violations.

How to Debug and Fix

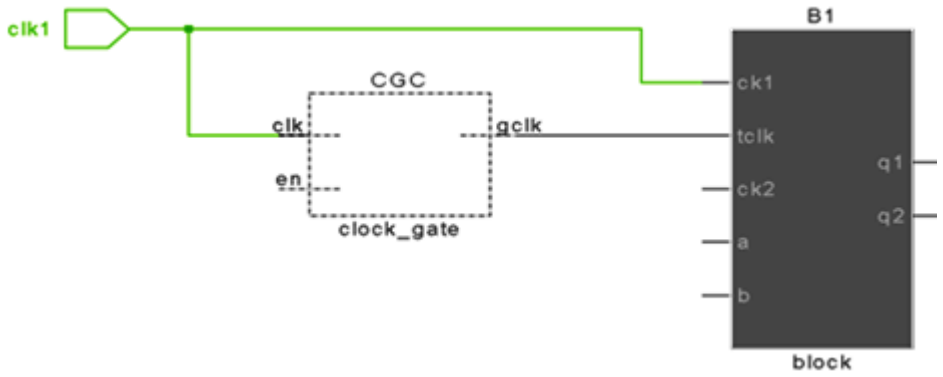
To fix this violation, perform the following actions:

- Analyze specification of clocks in the `-from` and `-to` arguments of the *cdc_false_path* constraint.
- Analyze the specification or propagation of top-level clocks.

  Violations of the *SGDC_clock_domain_validation02* rule appear for the violating clock signals.

Example Code and/or Schematic

Consider the following figure:




```
// Block-level SGDC file
   clock -name ck1
   clock -name tclk
   cdc_false_path -from ck1 -to tclk
```

FIGURE 406. Schematic of the SGDC_cdc_false_path_validation01 Rule Violation

In the above example, clock domain of the ck1 and tck1 clocks is same. Therefore, the *SGDC_cdc_false_path_validation01* rule reports a violation.

Default Severity Label

Warning

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_define_reset_order_validation01

Reports block ports with define_reset_order constraint which are not driven by top-level reset net

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

To run this rule, specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_define_reset_order_validation01* rule reports a violation if the reset specified by the `-from` or `-to` argument of the *define_reset_order* constraint defined for block-level is not driven from a top-level reset net.

Parameter(s)

None

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

This rule reports the following message:

```
[ERROR] Reset '<reset-name>' specified in -from/-to for block
instance '<block-inst>' (block: <block-name>) is not coming
from top-level reset
```

Potential Issues

This violation appears if the reset specified by the `-from` or `-to` argument

of the *define_reset_order* constraint defined for a block is not driven from a top-level reset net.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report incorrect `Ar_resetcross01` violations during block verification. This may generate an incorrect abstract-view SGDC file.

How to Debug and Fix

To debug this violation, perform the following steps:

1. Examine the specification of the *define_reset_order* constraint on block ports.
2. Examine the specification or propagation of top-level resets.

Example Code and/or Schematic

Consider the following constraint specified in a block-level SGDC file:

```
define_reset_order -from scan_rst -to rst
```

In the above case, the *SGDC_define_reset_order_validation01* rule reports a violation if `scan_rst` is not connected to any top-level reset in the SoC design.

Default Severity Label

Error

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_define_reset_order_validation02

Reports the same top-level reset reaching to the resets specified by the -from and -to arguments of the define_reset_order constraint for an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the `set_option sgdc_validate yes` command in the project file.
- Specify the *define_reset_order* constraint.

Description

The *SGDC_define_reset_order_validation02* rule reports a violation if asynchronous resets specified by the `-from` and `-to` arguments of the *define_reset_order* constraint for an abstract view are same in the context of the top-level reset.

Parameter(s)

None

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Mandatory): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears for the abstract view *<block-name>*, when the reset specified by the `-from` and `-to` arguments of the *define_reset_order* constraint come from the same top-level reset

<top-level-reset>:

[ERROR] For block instance '<block-inst>' (block: <block-name>), reset '<reset1>' in -from and reset '<reset2>' in -to are coming from the same top-level reset '<top-level-reset>' in define_reset_order constraint

Potential Issues

This violation appears if the same top-level reset is specified in the `-from` and `-to` arguments of the *define_reset_order* constraint for an abstract view.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass may report false `Ar_resetcross01` violations during the block verification stage.

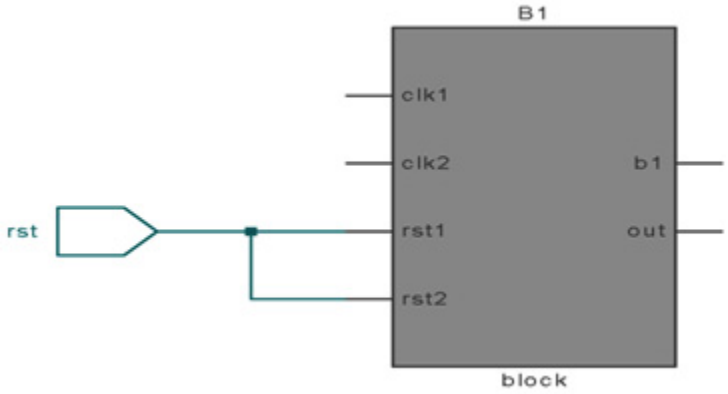
How to Debug and Fix

To fix this violation, perform following steps:

1. Check the reset specification in the `-from` and `-to` arguments of the *define_reset_order* constraint for an abstract port.
2. Ensure that the top-level reset is not same as the resets detected in the first step.

Example Code and/or Schematic

Consider the following figure:




```
// block.sgdc
reset -name rst1
reset -name rst2
define_reset_order -from rst1 -to rst2
```

FIGURE 407. Schematic of the SGDC_define_reset_order_validation02 Rule Violation

In the above figure, the `rst1` and `rst2` asynchronous resets specified in the `-from` and `-to` arguments of the *define_reset_order* constraint are same in the context of the top-level reset.

Therefore, the *SGDC_define_reset_order_validation02* rule reports a violation.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_quasi_static_validation01

Reports unconstrained quasi_static ports of an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_quasi_static_validation01* rule reports a violation if a top-level quasi-static signal reaches a block port on which a *quasi_static* constraint has not been specified.

Rule Exceptions

The SGDC_quasi_static_validation01 rule does not report a violation if you specify the *abstract_port* constraint with the `-ignore` argument on the block port.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears if you do not specify the *quasi_static* constraint on the block port:

[ERROR] Top level quasi-static signal <net-name> reaches to unconstrained port <port-name> of block instance <instance-name> (block: <block-name>)

Potential Issues

This violation appears if a top-level quasi-static signal reaches a block port on which no *quasi_static* constraint has been specified.

Consequences of Not Fixing

If you do not fix this violation, the design may not operate in the desired mode. In addition, the *quasi_static* constraints would not be propagated to block outputs.

How to Debug and Fix

To resolve this violation,

- Specify *quasi_static* constraint on a block port, or
- Analyze the specification or propagation of the top-level quasi-static signal to the block port.

Example Code and/or Schematic

In this example, the *SGDC_quasi_static_validation01* rule reports a violation because the top-level quasi-static signal `top.in2` reaches to the unconstrained port `in` of block instance `top.b1`. Review the following Verilog and SGDC code.

Verilog File	SGDC Files

module top(input in1,in2,clk1,clk2,output reg out);
reg w1;
always@(posedge clk1)
w1<=in1;
 	
assign w2 = w1 & in2;
 	
block b1(w2,clk1,out);
 	
endmodule
 	
module block(input in,clk,output out);
endmodule
	//SGDC file for top
	current_design top
	clock -name clk1
	clock -name clk2
	quasi_static -name top.w1
	quasi_static -name in2
	sgdc -import block block.sgdc
	
	//SGDC for block
	current_design block
	clock -name clk

The following schematic is generated.

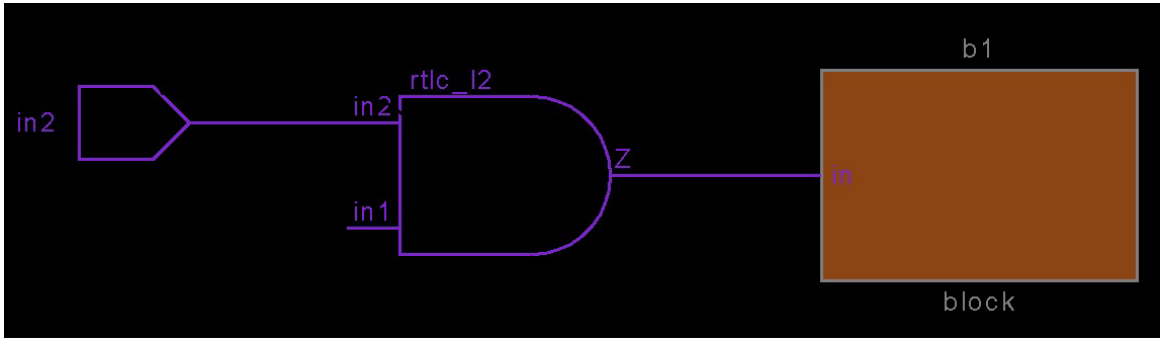


FIGURE 408. Schematic of the SGDC_quasi_static_validation01 Rule Violation

To resolve the violation, apply the *quasi_static* constraint to the unconstrained port `in` of block instance `top.b1`.

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_quasi_static_validation02

Reports quasi-static ports, which are not driven from top-level quasi-static signals, of an abstract view

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the `set_option sgdc_validate yes` command in the project file.

Description

The *SGDC_quasi_static_validation02* rule reports a violation if a *quasi_static* constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port.

Rule Exceptions

The SGDC_quasi_static_validation02 rule does not report a violation if the block port on which the *quasi_static* constraint is defined is being driven by an abstract-view port having the *abstract_port* constraint with the `-ignore` argument.

Parameter(s)

None

Constraint(s)

- *quasi_static* (Mandatory): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears if a *quasi_static* constraint has been specified at a block-level port, however no top-level quasi-static signal is driving the block port:

```
[ERROR] Quasi-static port <port-name> specified for block
instance <instance-name> (block: <block-name>) is not driven
from top level quasi-static signals
```

Potential Issues

The violation message explicitly states the potential issue.

Consequences of Not Fixing

If you do not fix this violation, some crossings in the design may not be detected.

How to Debug and Fix

To resolve this violation, perform the following:

- If the block quasi-static specification is correct, add the missing *quasi_static* at the top-level, else
- Remove the *quasi_static* constraint from the block port.

Example Code and/or Schematic

In this example, the *SGDC_quasi_static_validation02* rule reports a violation because quasi-static port `in` specified for block instance `top.b1` is not driven from top-level quasi-static signals. Review the following Verilog and SGDC code.

Verilog File	SGDC Files

module top(input in1,in2,clk1,clk2,output reg out);
reg w1;
always@(posedge clk1)
w1<=in1;
 	
assign w2 = w1 & in2;
 	
block b1(w2,clk1,out);
 	
endmodule
 	
module block(input in,clk,output out);
endmodule
	//SGDC file for top
	current_design top
	clock -name clk1
	clock -name clk2
	quasi_static -name w1
	sgdc -import block block.sgdc
	
	//SGDC for block
	current_design block
	clock -name clk
	quasi_static -name in

Default Severity Label

Error

Rule Group

SOC_SGDCVALIDATION

Reports and Related Files

No report or related file

SGDC_quasi_static_validation03

Reports top module output ports on which user has defined quasi_static value but none of the quasi_static constraint is propagated to

When to Use

If you have specified the *quasi_static* constraint on some output ports of the top design/module, use this rule to check if any quasi_static value is propagated to those output ports.

Prerequisites

Specify the *quasi_static* constraint on any output port.

Description

The *SGDC_quasi_static_validation03* rule reports a violation when the *quasi_static* constraint is specified on an output port of top module and no quasi_static value is propagated to that port.

Parameter(s)

num_quasi_seq_elem: Default value is `0`. Set this parameter to a positive integer greater than or equal to zero to specify the depth that should be considered when traversing the fanout traversal for the *quasi_static* constraint.

Constraint(s)

quasi_static (Mandatory): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears if the output port of the top module *<module_name>* has the *quasi_static* constraint specified on an output port and no or incorrect quasi_static value is propagated to that port:

[Warning] `Output port of top module <module-name> has quasi_static mismatches`

Potential Issues

This violation appears if the output port is constrained with the *quasi_static* constraint but quasi_static signal is not received from the fanin cone of that port.

Consequences of Not Fixing

If you do not fix this violation, the constrained port will pass incorrect information to other interacting modules when these modules are used in the SoC level.

How to Debug and Fix

To debug and fix this violation, perform the following steps:

- Identify the problematic output ports by referring the rule based spreadsheet for the top module.
- If any quasi_static value is not propagated to an output port on which the *quasi_static* constraint is defined, remove the quasi_static constraint from that port.

Example Code and/or Schematic

Consider the following schematic:

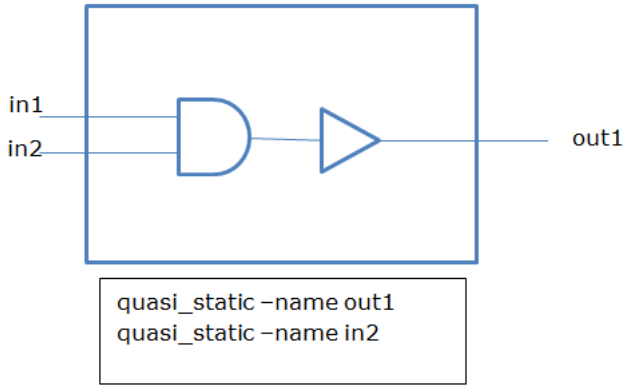


FIGURE 409. Schematic of the SGDC_quasi_static_validation03 Rule Violation

In the above example, though the quasi_static constraint is defined on

`out1` output port, it doesn't propagate quasi_static value to that port because there is no quasi_static defined on `in1`. Therefore, the SGDC_quasi_static_validation03 rule reports a violation.

Default Severity Label

Warning

Rule Group

None

Reports and Related Files

SGDC_quasi_static_validation03.csv

Synchronous Reset Verification Rules

Rules under this category verify user-defined synchronous resets.

These rules work only with the *Advanced_CDC* and *adv_checker* license features.

Following are the rules under this category:

Rule	Description
Ar_syncrstactive01	Reports a mismatch in the polarity on a synchronous reset usage
Ar_syncrstcombo01	Reports a mismatch in the combinational logic in a synchronous reset path
Ar_syncrstload01	Reports if load on a synchronous reset exceeds the specified maximum load
Ar_syncrstload02	Reports if load on a synchronous reset is less than the specified minimum load
Ar_syncrstpragma01	Reports a mismatch in the pragma specification on a synchronous reset usage
Ar_syncrstrtl01	Reports if the usage of a synchronous reset is not detected in a condition of the first `if` statement

Ar_syncrstactive01

Polarity on synchronous reset usage mismatches with -active field in sync_reset_style constraint

When to Use

Use this rule to verify polarity of user-defined synchronous resets.

Description

The *Ar_syncrstactive01* rule reports a violation in the following cases:

- If the `-active` argument of the *sync_reset_style* constraint is set to `low`, and SpyGlass detects at least one synchronous reset usage that is not active low.
- If the `-active` argument of the *sync_reset_style* constraint is set to `high`, and SpyGlass detects at least one synchronous reset usage that is not active high.

Considering Synchronous Reset Usage as Active Low

SpyGlass considers synchronous reset usage as active low when SpyGlass-generated path for the synchronous reset logic contains:

- Only one AND gate.
- Only one inverter followed by one OR gate.

Considering Synchronous Reset Usage as Active High

SpyGlass considers synchronous reset as active high when SpyGlass-generated path for a synchronous reset logic contains:

- Only one OR gate.
- Only one inverter followed by one AND gate

Parameter(s)

None

Constraint(s)

- *sync_reset_style*: (Mandatory) Use this constraint to specify synchronous reset information.
- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] <Un-deterministic | Unexpected> polarity detected in reset tree of synchronous reset '<reset-source>' (Output: '<output-name>')
```

SpyGlass reports the **polarity as un-deterministic** when the *Ar_syncrstactive01* rule could not detect/determine the polarity. For example, such cases occur when the data pin of flip-flop is driven by a NOR gate.

SpyGlass reports a **polarity as unexpected** when the detected polarity mismatches with user-specified polarity. For example, such cases occur when SpyGlass-detected polarity is high and user-specified polarity is low.

Potential Issues

This violation appears if the polarity on the path of synchronous reset signal does not match with the information specified in the `-active` argument of the `sync_reset_style` constraint.

Consequences of Not Fixing

If you do not fix this violation, synchronous reset may not get asserted at the user-specified value. This may result in many uninitialized flip-flops in the design.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message, and check the gates connected to the data pin of a flip-flop.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

//test.v

```
module test(input in, clk1, srst, output reg outa);
    always @ (posedge clk1)
        if (srst) outa = 1'b0;
        else outa = in;
endmodule
```

constr.sgdc

```
current_design test
sync_reset_style -active low
clock -name clk1
reset -name srst -sync
```

For the above example, this rule reports a violation because of unexpected polarity detected in the reset tree of the synchronous reset `srst`.

The following figure shows the schematic of the violation reported in this case:

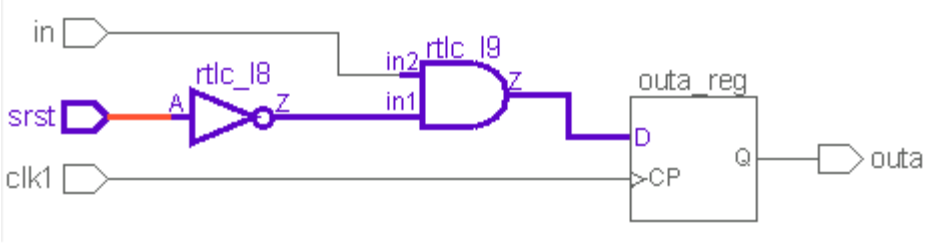


FIGURE 410. Schematic of the Ar_syncrstactive01 Rule Violation

The above schematic highlights the reset path to the flip-flop containing the unexpected polarity.

To fix this violation, modify the `if (srst) outa = 1'b0;` line in the RTL to if `(!srst) outa = 1'b0;`.

Default Severity Label

Warning

Rule Group

Ar_syncrst_validation

Reports and Related Files

A spreadsheet file containing all violations of this rule.

Ar_syncrstcombo01

Combinational logic in synchronous reset path mismatches with -combo field in sync_reset_style constraint

When to Use

Use this rule to verify user-defined synchronous resets.

Description

The *Ar_syncrstcombo01* rule reports a violation when the combinational logic in a synchronous reset path does not match with the combinational logic specified in the `-combo` argument of the `sync_reset_style` constraint.

Parameter(s)

None

Constraint(s)

- *sync_reset_style*: (Mandatory) Use this constraint to specify synchronous reset information.
- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] Combinational logic detected in reset tree of
synchronous reset <reset-source> (Output: <output-name>)
```

Potential Issues

This violation appears if a combinational logic present in the design does not match with the user-specified combination logic in a path.

Consequences of Not Fixing

Presence of combinational elements can introduce glitches and result in delay in a synchronous reset path.

How to Debug and Fix

To debug the violation of this rule, view the *Incremental Schematic* of the violation message and check the logic present in the path with the logic specified in the SGDC file.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
//test.v
module test(input in, clk1, en, srst, output reg outa);
    wire srst2 = srst & en;
    always @ (posedge clk1)
        if (!srst2) outa = 1'b0;
        else outa = in;
endmodule


//constr.sgdc
current_design test
sync_reset_style -combo no
clock -name clk1
reset -name srst -sync
```

For the above example, this rule reports a violation because the design contains a combinational logic that is not specified in the SGDC file.

To fix this violation, remove the following combinational logic generation code from the RTL:

```
wire srst2 = srst & en;
```

Default Severity Label

Warning

Rule Group

Ar_syncrst_validation

Reports and Related Files

A spreadsheet file containing all violations of this rule.

Ar_syncrstload01

Load on synchronous reset exceeds the specified max load

When to Use

Use this rule to verify user-defined synchronous resets.

Description

The *Ar_syncrstload01* rule reports a violation when the load on a synchronous reset exceeds the load specified in the `-max_load` argument of the *sync_reset_style* constraint.

This load is specified in terms of the total number of the following type of terminals driven by a synchronous reset:

sequential element terminals	tristate terminals
mux instance control terminals	black box terminals

Parameter(s)

None

Constraint(s)

- *sync_reset_style*: (Mandatory) Use this constraint to specify synchronous reset information.
- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] Load <current-load> on Synchronous reset '<reset-name>' exceeds the specified maximum load <maximum-allowed-load>
```

Potential Issues

This violation appears if you design contains synchronous reset on which the load is greater than the maximum allowed load.

Consequences of Not Fixing

If you do not fix this violation, synchronous resets will have a very high load compared to the maximum specified load. This will result in a high transition time of a net and possible skew in the design.

In such case, you may need to re-analyze the reset tree to introduce a buffer for balancing the load.

How to Debug and Fix

To debug the violation of this rule, perform any of the following actions depending upon your requirement:

- Analyze and remove the unexpected usage of synchronous reset in a design.
- Increase the value of the `-max_load` argument of the *sync_reset_style* constraint.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v

module test(input [4:0] in, input quasinet, clk1, clk2, srst, srst2,
            output reg [4:0] outa, outc);

    always @ (posedge clk1)
        if (!srst) outa = 5'b00000;
        else outa = in;

    wire l_srst2 = !srst2;
    always@(posedge clk1)
        outc[0] = l_srst2?1'b0:in[0];

endmodule


// constr.sgdc

current_design test
sync_reset_style -max_load 3 -min_load 2 -first_if yes
                 -pragma yes -active low -combo no
clock -name clk1
reset -name srst -sync
reset -name srst2 -sync
```

For the above example, the *Ar_syncrstload01* rule reports a violation because the load on the `srst` synchronous reset exceeds the maximum allowed load limit 3 (`-max_load 3`).

The following figure shows the schematic of this violation:

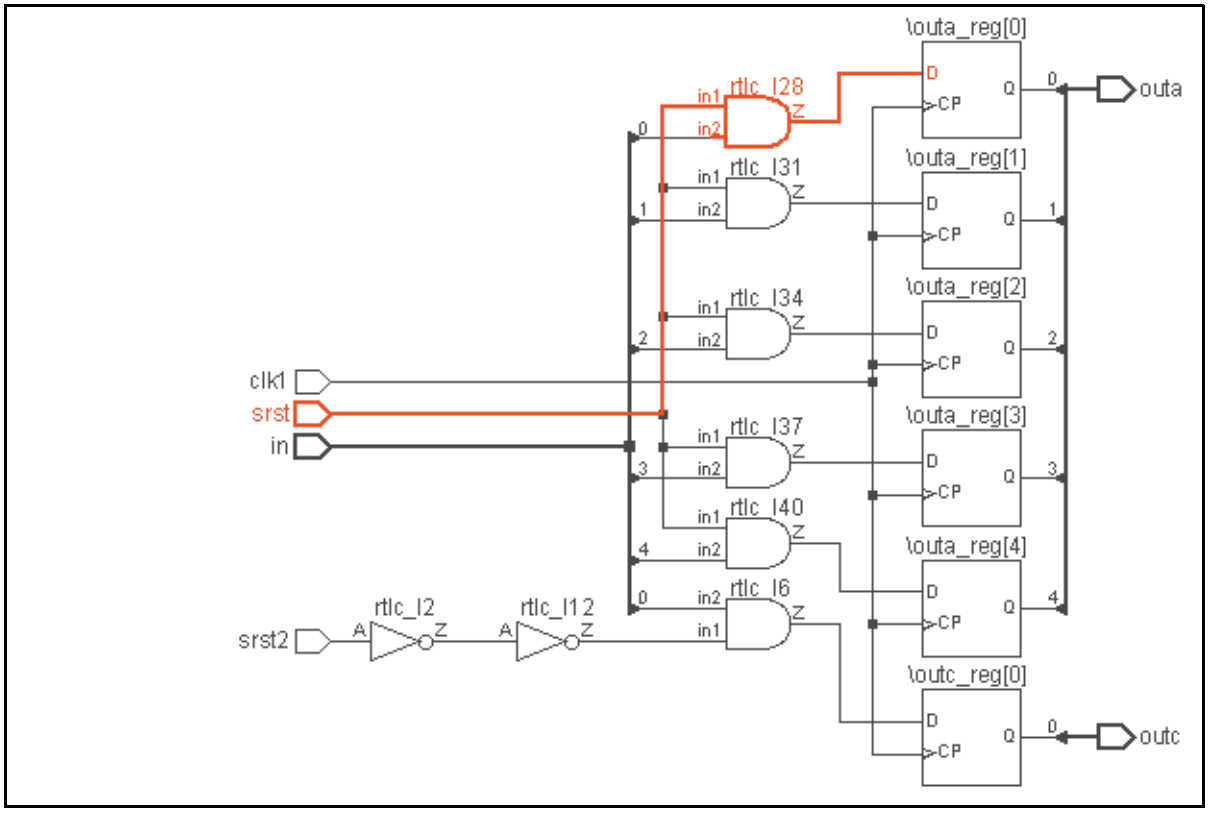


FIGURE 411. Schematic of the Ar_syncrstload01 Rule Violation

To fix this violation, perform any of the following actions depending upon your requirement:

- Remove the reported reset if it is not required in the design.
- Increase the value of the maximum allowed load on resets. That is, specify a value greater than 3 in the `-max_load` argument of the *sync_reset_style* constraint.

Default Severity Label

Warning

Rule Group

Ar_syncrst_validation

Reports and Related Files

A spreadsheet file containing all violations of this rule.

Ar_syncrstload02

Load on synchronous reset less than the specified minimum load

When to Use

Use this rule to verify user-defined synchronous resets.

Description

The *Ar_syncrstload02* rule reports a violation when the load on a synchronous reset is less than the load specified in the `-min_load` argument of *sync_reset_style* constraint.

This load is specified in terms of the total number of the following type of terminals driven by a synchronous reset:

sequential element terminals	tristate terminals
mux instance control terminals	black box terminals

Parameter(s)

None

Constraint(s)

- *sync_reset_style*: (Mandatory) Use this constraint to specify synchronous reset information.
- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] Load <current-load> on Synchronous reset '<reset-name>' is less than the specified minimum load <minimum-allowed-load>
```

Potential Issues

This violation appears if you design contains synchronous resets on which the load is less than the minimum allowed load.

Consequences of Not Fixing

Synchronous resets are generally high load nets and you may have introduced additional logic to cater to the high load, which is unnecessary.

How to Debug and Fix

To debug the violation of this rule, analyze the usage of synchronous reset in your design. It might happen that your synchronous reset is getting blocked in the design and may not allow synchronous reset to initialize your design.

Decreasing the value of the `-min_load` argument of the *sync_reset_style* constraint will remove this violation.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v

module test(input d1, d2, srst, clk, output reg q1, q2);
always @(posedge clk)
    if (srst) begin
      q1 <= 1'b0;
      q2 <= 1'b0;
    end
    else begin
      q1 <= d1;
      q2 <= d2;
    end
endmodule


// constr.sgdc

current_design test
clock -name clk
reset -sync -name srst
sync_reset_style -min_load 3
```

FIGURE 412.

For the above example, the *Ar_syncrstload02* rule reports a violation because the load on the `srst` synchronous reset is less than the minimum load limit 3 (`-min_load 3`).

The following figure shows the schematic of this violation:

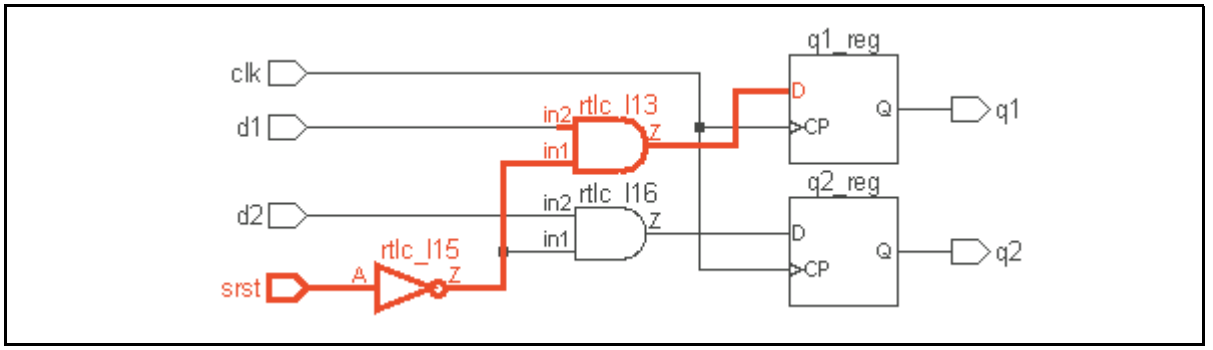


FIGURE 413. Schematic of the Ar_syncrstload02 Rule Violation

To fix this violation, decrease the value of the `-min_load` argument of the *sync_reset_style* constraint.

Default Severity Label

Warning

Rule Group

Ar_syncrst_validation

Reports and Related Files

A spreadsheet file containing all violations of this rule.

Ar_syncrstpragma01

Pragma specification on synchronous reset usage mismatches with -pragma field in sync_reset_style constraint

When to Use

Use this rule to verify user-defined synchronous resets.

Description

The *Ar_syncrstpragma01* rule reports a violation in the following cases:

- If the `-pragma` argument of the *sync_reset_style* constraint is set to `yes` and all the RTL usage of a synchronous reset does not specify the `sync_set_reset` pragma.
- If the `-pragma` argument of the *sync_reset_style* constraint is set to `mixed` and none of the RTL usage of the synchronous reset specifies the `sync_set_reset` pragma.

This rule detects pragma specification on a net (*<net-name>*) when the `// synopsys sync_set_reset` *<net-name>* is detected in the RTL design. See *Example 1 - mixed and yes Speciations of -pragma*.

Parameter(s)

None

Constraint(s)

- *sync_reset_style*: (Mandatory) Use this constraint to specify synchronous reset information.
- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] Reset pragma not detected in <tree-type> reset tree
of synchronous reset '<reset-source>' (Output: '<output-name>')
```

Where, *<tree-type>* is `complete` if the value of the `-pragma` argument of the *sync_reset_style* constraint is `mixed`. Else, this argument is blank.

Potential Issues

This violation appears if your design does not contain expected usage of the `sync_set_reset` pragma on a synchronous reset usage.

Consequences of Not Fixing

If you do not fix this violation, tools using pragma-related information for further optimization will not be able to optimize/use this information.

How to Debug and Fix

To debug the violation of this rule, perform the following steps:

1. Open the *Incremental Schematic* of the violation of this rule.
2. Analyze the usage of synchronous reset in the schematic where pragma is not detected by the tool.

Example Code and/or Schematic

Example 1 - mixed and yes Speciations of -pragma

Consider the following files specified for SpyGlass analysis:

test.v

```
module M1(input ina,
   input clk1, clk2, clk3, clk4,
   input rst, rst1, rst2, rst3, rst4, rst5, rst6, rst7, rst8, rst9, rst10,
   output reg outa, outb, outc, outd);
  // synopsys sync_set_reset rst1
  always@(posedge clk1) if(!rst1)  outa <= 1'b0; else outa <= ina;
endmodule

module M2(input ina,
  input clk1, clk2, clk3, clk4,
  input rst, rst1, rst2, rst3, rst4, rst5, rst6, rst7, rst8, rst9, rst10,
  output reg outa, outb, outc, outd);
  always@(posedge clk1) if(!rst2)  outa <= 1'b0; else outa <= ina;
endmodule

module test(input ina,
  input clk1, clk2, clk3, clk4,
  input rst, rst1, rst2, rst3, rst4, rst5, rst6, rst7, rst8, rst9, rst10,
  output reg outa, outb, outc, outd);
   wire l_rst1 = rst1;
  M1 a_m1(ina, clk1, clk2, clk3, clk4,
    rst, l_rst1, !rst2, rst3, rst4, rst5, rst6, rst7, rst8, rst9, rst10,
    outa, outb, outc, outd);
 M2 a_m2(ina, clk1, clk2, clk3, clk4,
    rst, rst2, rst1, rst3, rst4, rst5, rst6, rst7, rst8, rst9, rst10,
    outa, outb, outc, outd);
endmodule
```

SGDC

```
sync_reset_style -first_if yes -pragma yes -active low -combo no
current_design  test
clock -name clk1
```

For the above example, the *Ar_syncrstpragma01* rule reports the following violation at the line in red:

`Reset pragma not detected in reset tree of synchronous reset 'test.rst1' (Output: 'test.a_m2.outa')`

To fix this violation, specify the following pragma before the line in red in the above example:

```
// synopsys sync_set_reset rst1
```

However, if you modify the constraint in blue of the above example to the following, the *Ar_syncrstpragma01* rule does not report any violation:

```
sync_reset_style -first_if yes -pragma mixed -active low
```

```
-combo no
```

Example 2 - sync_reset_style -pragma set to yes

Consider the following files specified during SpyGlass analysis:

//test.v

```
module test(input in, clk1, en, srst, output reg outa);
    wire srst2 = srst & en;
    always @ (posedge clk1)
        if (!srst2) outa = 1'b0;
        else outa = in;
endmodule
```

//constr.sgdc

```
current_design test
sync_reset_style -pragma yes
clock -name clk1
reset -name srst -sync
```

For the above example, the *Ar_syncrstpragma01* rule reports a violation because pragma is not specified on the `srst` reset. To fix this violation, add the following line in the RTL:

```
// synopsys sync_set_reset srst
```

Default Severity Label

Warning

Rule Group

Ar_syncrst_validation

Reports and Related Files

A spreadsheet file containing all violations of this rule.

Ar_syncrstrtl01

Usage of synchronous reset is not detected in condition of first if statement

When to Use

Use this rule to verify user-defined synchronous resets.

Description

The *Ar_syncrstrtl01* rule reports a violation when the `-first_if` argument of the `sync_reset_style` constraint is set to `yes` and the synchronous reset is not always used in a condition of the first `if` statement.

Parameter(s)

None

Constraint(s)

- *sync_reset_style*: (Mandatory) Use this constraint to specify synchronous reset information.
- *reset* (Mandatory): Use this constraint with the `-sync` argument to specify synchronous reset signals in your design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] Synchronous reset '<reset-source>' at (Output:
'<output-name>') not used in condition of first if statement of
'<always | process>' block
```

Potential Issues

This violation appears if the usage of a synchronous reset is not detected in condition of the first `if` statement.

Consequences of Not Fixing

If you do not fix this violation, it will lead to more logic in the synchronous reset path, which will have delay/timing impact.

How to Debug and Fix

To debug the violation of this rule, analyze the usage of synchronous reset in the `always` block that is used for the generation of a flip-flop.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

//test.v

```
module test(input in, input clk1,srst,srst1,
        output reg outa, outb);
    always @ (posedge clk1)
        if (srst) outa <= 0;
        else outa <= in;
    always @ (posedge clk1) begin
        if (srst1) outb <= 1
        else if (srst) outb <= 1
        else outb <= in;
    end
endmodule
```

//constr.sgdc

```
current_design test
sync_reset_style -first_if yes
reset -name srst -sync
reset -name srst1 -sync
```

For the above example, the *Ar_syncrstrtl01* rule reports a violation because the `srst` reset is not used in first `if` condition of the second `always` block.

To fix this violation, use the `srst` reset in first `if` condition clause.

Default Severity Label

Warning

Rule Group

Ar_syncrst_validation

Reports and Related Files

A spreadsheet file containing all violations of this rule.

Must Rules

The following non-fatal rules are always run whenever the SpyGlass CDC solution is run:

TABLE 8 Non-fatal Must Rules

Rule	Purpose
Ac_abs01	Reports when some functional rules are running and either of the *fa_flopcount* or *fa_seqdepth* parameters are specified
Ac_license01	Checks whether the Advanced_CDC license feature is available when the dependent rules are selected to run
Clock_check07	Reports cases when one clock domain reaches another domain while propagating
Propagate_Clocks	Generates highlight information for clock trees
Propagate_Resets	Generates highlight information for reset trees
SGDC_clockreset02	Sanity checks on the `output` constraint
SGDC_gray_signals01	Reports a violation if you specify on one scalar signal to the *gray_signals* constraint.
SGDC_gray_signals02	Reports a violation if any signal specified by the *gray_signals* constraint is not driven by a clock.
SGDC_gray_signals03	Reports a violation if the signals specified by the *gray_signals* constraint are in multiple clock domains.
SGDC_input02	Reports non-existent ports/nets (non-hierarchical names) specified with the `-clock` argument of the `input` constraint Here, the clock controlling the input port is assumed a virtual clock.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_numflops03a	Reports a violation if a non-hierarchical net name or a port name specified in the `-from_clk` argument of the `num_flops` constraint is not found in the top-level module. Here, the clock controlling the input port is assumed a virtual clock.
SGDC_numflops03c	Reports a violation if an invalid clock is specified in the `-from_clk` argument of the `num_flops` constraint
SGDC_numflops04	Reports a violation if an invalid clock is specified in the `-to_clk` argument of the `num_flops` constraint
SGDC_numflops05	Reports a violation if the domain name specified with the `-from_domain` field of the `num_flops` constraint does not exist as a domain specified to a clock in the SGDC file or as a domain attached to an automatically-inferred clock by SpyGlass. The specific constraint will be ignored in such cases.
SGDC_numflops06	Reports a violation if the domain name specified with the `-to_domain` field of the `num_flops` constraint does not exist as a domain specified to a clock in the SGDC file or as a domain attached to an automatically-inferred clock by SpyGlass. The specific constraint will be ignored in such cases.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_numflops11	Reports a violation if the fields specified with the `num_flops` constraint are logically overlapping. The latest found specification will be used in such cases. Following is an example of such overlapping constraints, where clk1 and clk2 are clocks in domains d1 and d2, respectively. num_flops -from_clk clk1 -to_clk clk2 -value 3 num_flops -from_domain d1 -to_domain d2 -value 4 In this case, the number of flip-flops required for multi-flops synchronization for a crossing between clk1 and clk2 will be taken as 4 (from last found specification).
SGDC_numflops13	Checks the `-lib` argument of the *num_flops* constraint
SGDC_numflops14	Checks the -cell argument of the *num_flops* constraint
SGDC_qualifier02a	Reports a violation if any of the clock names specified with the `-from_clk` field of *qualifier* constraint is not one of the specified clocks or automatically-inferred clock.
SGDC_qualifier02c	Reports a violation if any of the clock names specified with the `-from_clk` field of *qualifier* constraint is a non-hierarchical name that does not exist as a port or net in the current design. This clock is assumed a virtual clock. Here, the clock controlling the input port is assumed a virtual clock.
SGDC_qualifier03a	Reports a violation if any of the clock names specified with the -to_clk field of `qualifier` constraint is not one of the specified clocks or automatically-inferred clock.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_qualifier03c	Reports a violation if any clock specified to the -to_clk argument of *qualifier* constraint is a non-hierarchical name that does not exist as a port or net in the current design.
SGDC_qualifier04	Reports a violation if any of the domain names specified with the -from_domain field of *qualifier* constraint is not one of the domain specified with the `clock` constraint or a domain attached to an automatically-inferred clock.
SGDC_qualifier05	Reports a violation if any of the domain names specified with the -to_domain field of `qualifier` constraint is not one of the domain specified with the `clock` constraint or a domain attached to an automatically-inferred clock.
SGDC_qualifier08	Reports a violation if valid net, hierarchical terminal, port, or sub module port for the wildcard name mentioned with the -net field of the `qualifier` constraint do not exist.
SGDC_qualifier18	Reports a violation if the *qualifier* *-ignore* constraint is specified on a net that is the part of a loop.
Param_clockreset02	Reports illegal values specified with the `num_flops` parameter
Param_clockreset04	Reports incorrectly specified *cdc_reduce_pessimism*, *clock_reduce_pessimism*, and *reset_reduce_pessimism* parameters
Param_clockreset05	Reports a missing or incorrectly specified *simulator_file_name* parameter

TABLE 8 Non-fatal Must Rules

Rule	Purpose
Param_clockreset06	Reports that the parameters, *expected_ckcells_file* and *unexpected_ckcells_file*, are specified together. In this case, the *unexpected_ckcells_file* parameter is ignored.
Param_clockreset07	Reports if the following combination of options are specified with *ac_sync_mode* parameter: • `strict_gate` and `soft_gate` • `strict_qual_logic` and `soft_qual_logic` If any of the above-specified combination is used, default values are used for the `ac_sync_mode` parameter.
Clock_check07	Reports cases when one clock domain reaches another domain while propagating
Reset_check08	Reports reset signals specified as a `set_case_analysis` signal (The *Reset_check03* and *Reset_check04* rules will not be run on that reset signal)
FalsePathSetup	Reports a violation when the *cdc_false_path* constraint does not waive any clock domain crossing in the design
QualifierSetup	Reports a violation when the *qualifier* constraint does not synchronize any clock domain crossing in the design
SignalTypeSetup	Checks for the signal specified by the `-name` argument of the *signal_type* constraint
Ac_initseq01	Reports a violation when all *define_tag* constraints with the `-tag initSeq` argument specified do not have the same length sequence specified with the `-value` argument
Ac_sanity01	Reports issues with the user-specified property files

TABLE 8 Non-fatal Must Rules

Rule	Purpose
Ac_sanity02	Reports non-tristate nets that have multiple drivers
Ac_sanity06	Reports over-constraining
Ac_init01	Reports a violation when none of the clocks in the design are specified using the *clock* constraint and the *use_inferred_clocks* option is also not set.
Ac_multitop01	Reports designs having multiple top-level design units
Ac_initstate01	Reports a valid state of the design from which the formal analysis would actually start
Ac_report01	Reports total number of properties analyzed and number of functional constraints set on a design
SGDC_fifo11	Checks existence of wildcard names in fields of *fifo* constraint.
SGDC_fifo12	Checks existence of memory corresponding to fields of *fifo* constraint.
SGDC_fifo13	Reports if a read pointer width is not equal to the write pointer width for user defined FIFOs.
SGDC_fifo14	Reports if no FIFO could be inferred from the user specified fifo definition.
SGDC_define_reset_order03	Reports when any of the reset names specified in the `-from` field of the *define_reset_order* constraint is not one of the user-defined reset or automatically-inferred reset.
SGDC_define_reset_order04	Reports when any of the reset names specified in the `-to` field of the *define_reset_order* constraint is not one of the user-defined reset or automatically-inferred reset.
SGDC_define_reset_order05	Reports when the resets specified in the *define_reset_order* constraint are conflicting.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_allow_combo_logic02	Reports if a combination of `all`, `none`, or `name` is used in the different specification of the *allow_combo_logic* constraint.
AllowComboLogicSetup	Reports if the modules specified by using the *allow_combo_logic* constraint are not used by any crossing
SGDC_output03	Reports if both *input* and *output* constraints are specified for an inout port. In such cases, the *output* constraint specification is ignored and only the *input* constraint specification is considered.
SGDC_signal_in_domain04	Reports a violation if the object specified to the -name argument of the *signal_in_domain* constraint is not a black box.
SGDC_sgclkgroup01	Reports a violation if an invalid tag name is specified to the `-group1` argument of the *sg_clock_group* constraint.
SGDC_sgclkgroup02	Reports a violation if an invalid tag name is specified to the `-group2` argument of the *sg_clock_group* constraint.
SGDC_sgclkgroup03	Reports a violation if the same tag name is specified to the `-group1` and `-group2` arguments of the *sg_clock_group* constraint.
SGDC_sync_cell02a	Reports if an incorrect non-hierarchical clock name is specified in the `-from_clk` argument of the `sync_cell` constraint. Here, the clock controlling the input port is assumed a virtual clock.
SGDC_sync_cell02c	Reports a violation if the clock name specified by the `-from_clk` argument of the sync_cell constraint is not one of the specified clocks or an automatically-inferred clock.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_sync_cell03b	Reports a violation if the clock name specified by the `-to_clk` argument of the sync_cell constraint is not one of the specified clocks or an automatically-inferred clock.
SGDC_sync_cell04	Reports a violation if the domain of clock name specified with the `-to_clk` argument of the sync_cell constraint matches with domain of clock name specified by the `-from_clk` argument of this constraint.
SGDC_sync_cell05	Reports a violation if the domain name specified by the `-from_domain` argument of the sync_cell constraint does not exist as a domain specified to a clock in an SGDC file or as a domain attached to an automatically-inferred clock by SpyGlass. The specific constraint is ignored in such cases.
SGDC_sync_cell06	Reports a violation if the domain specified by the `-to_domain` argument of the sync_cell constraint does not exist as a domain specified to a clock in an SGDC file or as a domain attached to an automatically-inferred clock by SpyGlass. The specific constraint is ignored in such cases.
SGDC_sync_cell07	Reports a violation if the same domain name is specified by the `-from_domain` and `-to_domain` arguments of the sync_cell constraint.
SGDC_sync_cell08b	Reports a violation if the value specified by the `-from_period` argument of the sync_cell constraint does not match the value specified by the `-period` argument of any clock constraint.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_sync_cell09b	Reports a violation if the value specified by the `-to_period` argument of the sync_cell constraint does not match the value specified by the `-period` argument of any clock constraint.
SGDC_sync_cell10	Reports a violation if multiple sync_cell constraints cover the same crossing. In such cases, all cells specified in these constraints are considered as valid for that crossing.
SyncCellSetup	Reports a violation if the specified sync_cell constraint is not used to synchronize any crossing in design.
SGDC_qualifier11	Reports if the `-crossing` argument is specified with the *qualifier* constraint and the specified qualifier is not defined at the destination output of a clock domain crossing.
SGDC_qualifier12	Reports if the `-crossing` argument is specified with the *qualifier* constraint and the `-to_clk`/`-to_domain` argument does not match with the clock/domain of the destination instance, respectively, of the specified qualifier.
SGDC_qualifier13	Reports if the `-crossing` argument is specified with the *qualifier* constraint and the -from_clk/-from_domain argument does not match the clock/domain of source instance, respectively, of the qualifier.
SGDC_quasi_static01	Reports if the object specified in the `-name` argument of the *quasi_static* constraint does not exist as a net in the current design.
SGDC_quasi_static_style01	Reports if the SGDC file contains multiple specifications of the *quasi_static_style* constraint.

TABLE 8 Non-fatal Must Rules

Rule	Purpose
SGDC_quasi_static_style02	Reports if you do not specify any argument with the *quasi_static_style* constraint.
SGDC_cdc_false_path04	Reports non-existent objects specified (using wildcards) with the `-from/-to/-through` arguments of the *cdc_false_path* constraint
SGDC_cdc_false_path05	Reports if no argument is specified with the *cdc_false_path* constraint
SGDC_cdc_false_path06	Reports type mismatch in the arguments of the *cdc_false_path* constraint
SGDC_reset_synchronizer02	Reports a violation if the synchronizer output specified by the -name argument of the *reset_synchronizer* constraint is not present in the fan-out of the reset specified by the -reset argument of this constraint.
SGDC_reset_synchronizer09	Reports if you have specified the same *reset_synchronizer* constraint multiple times.
SGDC_reset_synchronizer10	Reports if you have specified conflicting arguments in multiple *reset_synchronizer* constraints, that is, different arguments in the -value argument but same arguments in all other arguments.
SGDC_clock_path_wrapper_module01	Reports user-defined wrapper modules in the clock-path
SGDC_clocksense02	Reports for the -tag argument of the *clock_sense* constraint if the tag is not associated with a real clock
SGDC_clocksense03	Reports if a virtual clock is specified in the -tag argument of the *clock_sense* constraint

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_abstract_port01	Reports violation if the module specified by the `-module` argument of the *abstract_port* constraint does not exist in the current design.
SGDC_abstract_port02	Reports violation if the port name specified in the `-ports` argument of the *abstract_port* constraint does not match with any of the ports of the module specified by the `-module` argument of this constraint.
SGDC_abstract_port03	Reports violation if the clock name specified in `-clock` argument of the *abstract_port* constraint is not a port or a net in the current design.
SGDC_abstract_port04	Reports violation if a clock specified in the `-clock` argument of the *abstract_port* constraint is a part-select or full vector of a block port.
SGDC_abstract_port05	Reports violation if the wildcard name specified in the `-clock` argument of the *abstract_port* constraint matches with multiple block ports.
SGDC_abstract_port06	Reports violation if the value specified by the `-combo` argument of the *abstract_port* constraint is other than `yes`, `no`, or `unknown`.
SGDC_abstract_port07	Reports violation if clock names specified in the `-from` argument of the *abstract_port* constraint does not exist as a port or a net in the current design.

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_abstract_port08	Reports violation if the port specified in the -from argument of the *abstract_port* constraint is a part-select or full vector of a block port
SGDC_abstract_port10	Reports violation if the wildcard name specified in the `-from` argument of the *abstract_port* constraint matches with multiple block ports.
SGDC_abstract_port11	Reports violation if the clock specified in the `-to` argument of the *abstract_port* constraint does not exist as a port or net in the current design
SGDC_abstract_port12	Reports violation if the port specified in the `-to` argument of *abstract_port* constraint is a part-select or full vector of a block port
SGDC_abstract_port13	Reports violation if the wildcard name specified in the `-to` argument of the *abstract_port* constraint matches with multiple block ports
SGDC_abstract_port14	Reports violation if the value specified in the `-delay` argument of the *abstract_port* constraint is not an integer or an integer less than two
SGDC_abstract_port15	Reports violation if the value specified in the `-seq` argument of the *abstract_port* constraint is other than `yes` or `no`
SGDC_abstract_port16	Reports violation if the input port specified by the `-related_ports` argument of the *abstract_port* constraint does not exist in module specified by the `-module` argument of this constraint

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_abstract_port18	Reports violation if the value specified in the `-sync` argument of the *abstract_port* constraint is other than `active` or `inactive`
SGDC_abstract_port21	Reports violation if the port specified in *abstract_port* constraint does not exist in the design.
SGDC_abstract_port22	Reports violation if the port specified in the `-combo_ifn` argument of the *abstract_port* constraint does not exist as a port of the module specified in the `-module` argument
SGDC_input01	Reports non-existent objects (port or net) specified with the `-name` argument of the `input` constraint
SGDC_input03	Reports non-existent ports/nets (hierarchical names) specified with the `-clock` argument of the `input` constraint
SGDC_fifo01	Reports a violation if you do not specify any argument with the *fifo* constraint
SGDC_fifo02	Reports a violation if the object specified by the `-rd_data` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design
SGDC_fifo03	Reports a violation if the object specified by the `-wr_data` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design
SGDC_fifo04	Reports a violation if the object specified by the `-rd_ptr` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_fifo05	Reports a violation if the object specified by the `-wr_ptr` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design
SGDC_fifo06	Reports a violation if the value specified by the `-memory` argument of the *fifo* constraint does not match with the name of an existing net, hierarchical terminal, or module in the current design.
SGDC_fifo07	Reports a violation if the `-rd_data` argument of the *fifo* constraint is specified without its corresponding `-wr_data` argument
SGDC_fifo08	Reports a violation if the -wr_data argument of the *fifo* constraint is specified without its corresponding -rd_data argument.
SGDC_fifo09	Reports a violation if the -wr_ptr argument of the *fifo* constraint is specified without its corresponding -rd_ptr argument.
SGDC_fifo10	Reports a violation if the -rd_ptr argument of the *fifo* constraint is specified without its corresponding -wr_ptr argument
SGDC_numflops01	Reports a violation if no field is specified with the `num_flops` constraint. At least one of the fields: `-from_clk`, `-to_clk`, `-to_clock`, `-from_domain`, `-to_domain`, `-to_period`, and `-default` should be specified with this constraint.
SGDC_numflops03b	Reports a violation if an invalid hierarchical net or pin name is specified in the `-from_clk` argument of the `num_flops` constraint

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_numflops07	Reports a violation if the value specified with the `-to_period` field of the `num_flops` constraint is not a float value or a negative float value.
SGDC_numflops08	Reports a violation if the value specified with the `-value` field of the *num_flops* constraint is not an integer or an integer less than 1.
SGDC_numflops09	Reports a violation if the value specified with the `-default` field of the *num_flops* constraint is not an integer or an integer with value less than 2.
SGDC_numflops10	Reports a violation if the value specified with the `-value` field is specified with the `-default` field of the *num_flops* constraint. The `-value` field is ignored in such cases.
SGDC_sync_cell02b	Reports a violation if an incorrect hierarchical clock name is specified in the `-from_clk` argument of the `sync_cell` constraint
SGDC_sync_cell03a	Reports a violation if the clock specified by the `-to_clk` argument of the sync_cell constraint is not found within the module.
SGDC_sync_cell08a	Reports a violation if the value specified by the `-from_period` argument of the sync_cell constraint is not a float value or a negative float value.
SGDC_sync_cell09a	Reports a violation if the value specified by the `-to_period` argument field of the sync_cell constraint is not a float value or a negative float value.
SGDC_qualifier01	Reports a violation if the -name field of the `qualifier` constraint is not specified.

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_qualifier02b	Reports a violation if any of the clock names specified with the `-from_clk` field of `qualifier` constraint is a hierarchical name that does not exist as a port, hierarchical terminal or net in the current design.
SGDC_qualifier03b	Reports a violation if any of the clock names specified with the `-to_clk` field of `qualifier` constraint does not exists as a port, hierarchical terminal, or net in the current design.
SGDC_qualifier06	Reports a violation if the -type field of the `qualifier` constraint is specified a value other than src, des, or both.
SGDC_qualifier09	Reports a violation if neither -from_clk/-to_clk nor -from_domain/-to_domain fields are specified with the `qualifier` constraint.
SGDC_qualifier10	Reports a violation if the clocks specified in each of the fields -from_clk and -to_clk of `qualifier` constraint have same domain.
SGDC_qualifier15	Reports a violation if none of the `-name` or `-enable` arguments are specified to the *qualifier* constraint.
SGDC_qualifier16	Reports a violation if any signal name specified to the `-enable` argument of the *qualifier* constraint is a hierarchical name that does not exist as a port, a hierarchical terminal, or a net in the current design.
SGDC_output01	Reports non-existent objects (port) specified with the `-name` argument of the `output` constraint

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_output02	Reports non-existent objects (port or net) specified with the `-clock` argument of the `output` constraint
SGDC_IP_block01	Reports non-existent objects (module) specified with the `-name` argument of the *ip_block* constraint
SGDC_signal_in_domain01	Reports non-existent objects (module) specified with the `-name` argument of the *signal_in_domain* constraint
SGDC_signal_in_domain02	Reports non-existent objects (pin of module specified with the `-name` argument) specified with the `-domain` argument of the *signal_in_domain* constraint
SGDC_signal_in_domain03	Reports non-existent objects (pin of module specified with the `-name` argument) specified with the `-signal` argument of the *signal_in_domain* constraint
SGDC_cdc_false_path01	Reports non-existent objects (top-level port, net, terminal, module) specified with the `-from` argument of the *cdc_false_path* constraint
SGDC_cdc_false_path02	Reports non-existent objects (top-level port, net, terminal, module) specified with the `-to` argument of the *cdc_false_path* constraint
SGDC_cdc_false_path03	Reports non-existent objects (top-level port, net, terminal, module) specified with the `-through` argument of the *cdc_false_path* constraint

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_network_allowed_cells01	Reports incorrect value of the `-type` argument of the *network_allowed_cells* constraint. Valid values are `clock`, `reset`, and `clock reset`.
SGDC_network_allowed_cells02	Reports non-existent objects (net) specified with the `-from` argument of the *network_allowed_cells* constraint
SGDC_output_not_used01	Reports non-existent objects (port) specified with the `-name` argument of the *output_not_used* constraint
SGDC_define_reset_order01	Reports when any of the arguments specified in the `-from` field of the *define_reset_order* constraint does not exist as a port, hierarchical terminal, or net in the current design.
SGDC_define_reset_order02	Reports when any of the arguments specified in the `-to` field of the *define_reset_order* constraint does not exist as a port, hierarchical terminal, or net in the current design.
SGDC_allow_combo_logic01	Reports if none of the fields are specified in the *allow_combo_logic* constraint.
SGDC_noclockcell01	Existence check for objects (ports/nets) specified with the `-name` argument of the *noclockcell_start* constraint
SGDC_noclockcell02	Existence check for objects (ports/pins/nets) specified with the `-name` argument of the *noclockcell_stop_signal* constraint
SGDC_noclockcell03	Existence check for objects (modules) specified with the `-name` argument of the *noclockcell_stop_module* constraint

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_noclockcell04	Existence check for objects (instances) specified with the `-name` argument of the *noclockcell_stop_instance* constraint
SGDC_deltacheck_start01	Existence check for objects (ports/pins/nets) specified with the `-name` argument of the *deltacheck_start* constraint
SGDC_deltacheck_start02	Non-integer value specified with the `-value` argument of the *deltacheck_start* constraint
SGDC_deltacheck_stop_signal01	Existence check for objects (ports/pins/nets) specified with the `-name` argument of the *deltacheck_stop_signal* constraint
SGDC_deltacheck_stop_module01	Existence check for objects (modules) specified with the `-name` argument of the *deltacheck_stop_module* constraint
SGDC_deltacheck_stop_instance01	Existence check for objects (instances) specified with the `-name` argument of the *deltacheck_stop_instance* constraint
SGDC_deltacheck_ignore_module01	Existence check for objects (modules) specified with the `-name` argument of the *deltacheck_ignore_module* constraint
SGDC_deltacheck_ignore_instance01	Existence check for objects (instances) specified with the `-name` argument of the *deltacheck_ignore_instance* constraint
SGDC_porttimedelay01	Existence check for objects (modules) specified with the `-name` argument of the *port_time_delay* constraint
SGDC_reset_synchronizer01	Reports a violation if the synchronizer output specified in the -name argument of the *reset_synchronizer* constraint does not exist as a port, hierarchical terminal, or a net in the current design.

TABLE 9 FATAL Must Rules

Rule	Purpose
SGDC_reset_synchronizer03	Reports if reset name specified by the -reset argument of the *reset_synchronizer* constraint is a hierarchical name that does not exist as port, hierarchical terminal, or net in the current design.
SGDC_reset_synchronizer04	Reports if reset name specified by the -reset argument of the *reset_synchronizer* constraint is neither the specified reset nor an automatically-inferred reset.
SGDC_reset_synchronizer05	Reports if the synchronizer clock specified by the -clock argument of the *reset_synchronizer* constraint is neither a clock-tag nor a hierarchical name that exists as a port, hierarchical terminal, or a net in the current design.
SGDC_reset_synchronizer06	Reports if the synchronizer clock specified by the -clock argument of the *reset_synchronizer* constraint is not one of the clock tags, user-specified clocks, or automatically-inferred clock.
SGDC_reset_synchronizer07	Reports if the value specified by the -value argument of the *reset_synchronizer* constraint is other than 0 or 1.
SGDC_clocksense01	Reports for an incorrect value in the `-pins` argument of the *clock_sense* constraint.

Ac_abs01

Reports the result of abstraction applied on functional checks

When to Use

Use this rule to check the result of abstraction applied on functional checks.

Description

The *Ac_abs01* rule reports the number of flip-flops considered or the sequential depth used when abstraction is applied on advanced CDC rules based on the value specified in the *fa_flopcount* or *fa_seqdepth* parameter.

It may happen that the number of flip-flops reported by this rule is more than the number specified by the *fa_flopcount* parameter. This is because all flip-flops are considered from clock to reset paths.

Parameter(s)

- *fa_flopcount*: Default value is `-1`. Set this parameter to a positive integer value to specify a maximum number of flip-flops.
- *fa_seqdepth*: Default value is `-1`. Set this parameter to a positive integer value to specify a maximum sequential depth.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears when abstraction is applied to the advanced CDC rules by limiting the cone size to the *<flip-flops-num>* number of flip-flops:

```
[AcAbs1_1] [INFO] Abstraction applied during functional
analysis by limiting cone size to '<flip-flops-num>' flops.
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 2

The following message appears when abstraction is applied on the advanced CDC rules by limiting the cone size to the *<seq-depth>* sequential depth:

[AcAbs1_2] [INFO] Abstraction applied during functional analysis by limiting cone size to '<seq-depth>' sequential depth.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 3

The following message appears when abstraction is applied on the advanced CDC rules by limiting the cone size to *<flip-flops-num>* flops and the *<seq-depth>* sequential depth:

[AcAbs1_3] [INFO] Abstraction applied during functional analysis by limiting cone size to '<flip-flops-num>' flops and '<seq-depth>' sequential depth.

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Not applicable

Default Severity Label

Info

Rule Group

ADV_CLOCKS

Reports and Related Files

No report or related file

Ac_init01

Does initial setup for Advanced SpyGlass CDC Solution rules

When to Use

Use this rule to create initial setup for running advanced CDC rules.

Prerequisites

Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_init01* rule does the following:

- Generates information, such as constraints, initial-state setup, and configuration setup that is required by advanced rules of SpyGlass CDC solution
- Reports constraints specified in a property file, but not found in the design.

NOTE: *This rule is a prerequisite for advanced CDC solution rules that perform functional analysis.*

Parameter(s)

fa_propfile: Default value is `NULL`. Set this parameter to the name of a property file containing properties to be checked.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears when the user-specified property file *<file-name>* cannot open:

```
[ERROR] Could not open property file '<file-name>'
```

Potential Issues

This violation appears if the property file does not open.

Consequences of Not Fixing

If you do not fix this violation, Advanced CDC solution rules may not work properly.

How to Debug and Fix

Check if the property file is available or has required permissions.

Message 2

The following message appears if a syntax error is found in the property file *<file-name>* because of an unrecognized string *<offending-string>* is present at the line number *<line-num>*:

```
[ERROR] Syntax error in property file '<file-name>' at line
number '<line-num>'. Unrecognized string '<offending string>'
```

Potential Issues

This violation appears if the property file has a syntax error, such as an unrecognized string.

Consequences of Not Fixing

If you do not fix this violation, advanced CDC solution rules might not work properly.

How to Debug and Fix

Check the specified line number in the property file, and fix the issue by rectifying the unrecognized string.

Message 3

The following message appears when the user-specified property file *<file-name>* is corrupted:

```
[ERROR] Property file '<file-name>' seems to be corrupted
```

Potential Issues

This violation appears if the specified property file is corrupted.

Consequences of Not Fixing

If you do not fix this violation, the advanced CDC solution rules might not work properly.

How to Debug and Fix

Specify a proper property file.

Message 4

The following message appears when the instance specified by the *simulation_data* constraint does not exist in the VCD file:

```
[ERROR] Instance '<inst-name>' specified by fa_vcdscopename
does not exist in VCD file '<VCD-file>'
```

Potential Issues

This violation appears if the instance specified by the *fa_vcdscopename* parameter does not exist in the VCD file.

Consequences of Not Fixing

If you do not fix this violation, the advanced CDC solution rules might not work properly.

How to Debug and Fix

Specify the correct instance name to the *fa_vcdscopename* parameter or specify a correct VCD file containing that instance.

Example Code and/or Schematic

Not applicable

Default Severity Label

Error

Rule Group

ADV_CLOCKS

Reports and Related Files

No report or related file

Ac_initseq01

Initialization sequences of multiple signals should be of the same length

When to Use

Use this rule to check for any mismatch in the length of initialization sequences for different signals.

Description

The *Ac_initseq01* rule reports a violation if initialization sequences specified by the `define_tag -tag initSeq` constraints are not of the same length.

When you specify the *define_tag* constraint with the `-tag initSeq` argument, the `-value` argument provides initialization sequences for a signal. If two such constraints specify different lengths of initialization sequences, the *Ac_initseq01* rule reports a violation.

Prerequisite(s)

Specify the *define_tag* constraint with the `-tag initSeq` argument.

Parameter(s)

None

Constraint(s)

define_tag (Mandatory): Use this constraint to define initialization sequences for different signals.

Messages and Suggested Fix

The following message appears if the length of initialization sequences for different signals is different:

```
[WARNING] Initialization sequences provided by the 'define_tag
-tag initSeq' constraint are all not of the same length
```

Potential Issues

This violation appears if the design has initialization sequences of different

lengths.

Consequences of Not Fixing

A design is simulated by using the specified initialization sequence to find a valid initial state.

If you specify a different length for initialization sequences on signals, the signals with smaller length of initialization sequences are simulated by using an undefined value for the remaining cycles.

How to Debug and Fix

Check all the specified *define_tag* constraint specifications, and find the signals that have different lengths of initialization sequence.

Fix the issue by making the initialization sequences of the same length.

Example Code and/or Schematic

The following example shows two *define_tag* constraint specifications where the length of the initialization sequence is different:

```
define_tag -tag initSeq -name top.reset1 -value 1 1 1 x x x
define_tag -tag initSeq -name top.reset2 -value x x 1 1
```

For the first *define_tag* constraint specification, the length of sequence is six, and for the second specification, it is four.

In this case, the *Ac_initseq01* rule reports a violation.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and/or Related Files

No reports or related files

Ac_initstate01

Reports initial state of the design

When to Use

Use this rule to know the initial state of a design.

Prerequisites

- Run any of the *CDC Verification Rules* or set the *formal_setup_rules_check* parameter to `yes`.
- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_initstate01* rule reports a valid state of a design from which formal analysis would actually start. This may not be the reset state of the design.

Identifying a Valid State

A valid state can be identified in the following different ways:

- User-defined initial state where the register value assignment is provided using the *define_tag* constraint.
- State value generated by external simulation engine as a VCD/TCl/FSDB file (use the *simulation_data* constraint to provide the file name)
- Initial state detected by applying a user-defined simulation vector using the *define_tag* constraint in a SpyGlass Design Constraints file.
- Initial state determined by SpyGlass CDC solution.

  This search uses the user-specified reset ports (using the *reset* constraint) or auto-detected reset ports and/or may apply proprietary techniques to identify a reachable state of a design.

Parameter(s)

- None

Constraint(s)

- *define_tag* (Optional): Use this constraint to define a named condition for application of certain stimulus at a top port or an internal node.

- *reset* (Optional): Use this constraint to specify reset signals.
- *simulation_data* (Optional): Use this constraint to specify the initial state sequence for a design.
- *set_case_analysis* (Optional): Use this constraint to specify case-analysis signals.

Messages and Suggested Fix

The following message appears to indicate the initial state of the design:

```
[INFO] <num1> percent of sequential outputs are initialized
with sets/resets and <num2> percent sequential outputs are
initialized by data path. Refer file: '<file name>' for
details"
```

The arguments of the above message are explained below:

Argument	Description
<num1>	Percentage of sequential outputs initialized with set/reset
<num2>	Percentage of sequential outputs initialized with data path
<file-name>	Name of the report file

Potential Issues

None

Consequences of Not Running

It is important to initialize your design correctly. Else, it may result in incorrect functional results.

How to Debug and Fix

This rule enables you to analyze the initialization percentage of your design.

Following are some of the points to be taken care of:

- If your design is not 100% initialized, see either the AC_Initstate01 spreadsheets by clicking the violation message or see the adv_cdc.reg file by selecting the *Report ->clock-reset->adv_reg* menu option.

  Consider the following design:

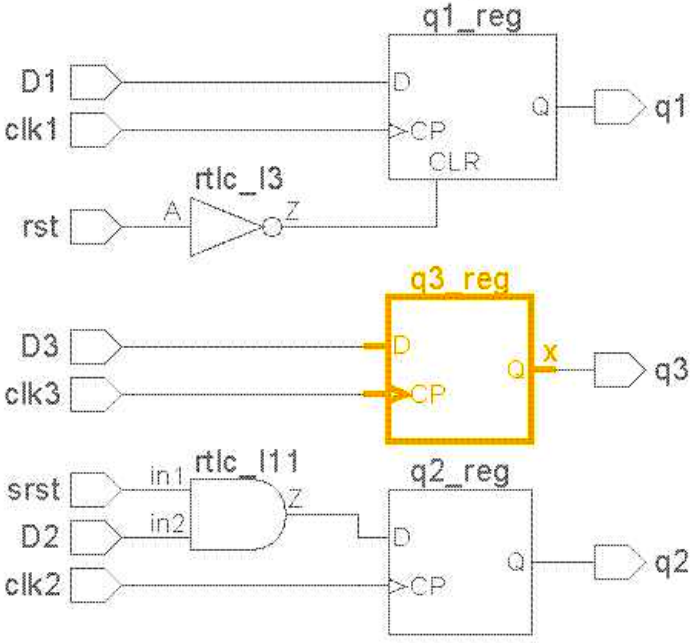


FIGURE 414. Design on Which Ac_initstate01 Rule is Run

The above design has three flip-flops: flip-flop `q1` (that can be reset by an asynchronous reset), flip-flop `q2` (that can be reset by a synchronous reset), and flip-flop `q3` (with no reset).

If clocks and resets are properly defined in an SGDC file, the following `adv_cdc.reg` file is generated:

```
################################################################################
Section A: Clocks in the design
================================
(Clock Name)          ;  (Clock-ID)
--------------------------------------
top.clk3              ;  1
top.clk2              ;  2
top.clk1              ;  3
################################################################################
Section B: Resets in the design
================================
(Reset Name)          ;  (Reset-ID)
--------------------------------------
top.rst               ;  1
srst                  ;  2
################################################################################
Section C: Initial State
================================================================
(Name) ;         (Initial value)  ; (Initialization Phase)     ; (Clock ID) ; (Reset ID) ; (File Name)   ; (Line No.)
----------------------------------------------------------------------------------------------------------------------
top.q1                ; 0              ; After Primary Set/Reset     ; (3)         ; (1)         ; test.v          ; 7
top.q2                ; 0              ; After Primary Set/Reset     ; (2)         ; (3)         ; test.v          ; 14
################################################################################
Section D: Uninitialized Sequential Elements
============================================
(Name) ; (Async Reset) ; (Async Set) ; (Clock) ; (Enable) ; (Data) ; (Others) ; (Clock ID) ; (Reset ID) ; (File Name) ; (Line No.)
----------------------------------------------------------------------------------------------------------------------
top.q3 ;          - ;            - ;         - ;       0 ;       X ;       no ;         (1) ;       (NULL) ;    test.v ;        17
################################################################################
```

FIGURE 415. The adv_cdc.reg File

- Cross-check the clocks and resets specified in *Section-A* and *Section-B* of the adv_cdc.reg report. This information should match with what you provided in your setup.

  Try to provide clocks with periods so that the overall design cycle becomes low. This can be done by specifying clock periods as multiples of each other. This would help you achieve a better initialization percentage.

  For resets, provide active value. Applying active values resets the flip-flop.

- If *Section-A* and *Section-B* contains the expected data, check *Section-D*. This section lists registers that remained uninitialized after set/reset propagation.
- In *Section-D*, if a particular register that you think should have been initialized after applying set/reset propagation still appears as uninitialized, check if resetting that register is possible at all by reset or not.
  - If the register was designed to be reset by an asynchronous reset, check its preset/clear pins to see if active value of its asynchronous reset source is reaching it or not. You can do this with the help of schematic from *Propagate_Resets* rule. If you see the reset source signal as already specified in the SGDC file with the correct active value, this may indicate that this value is getting blocked somewhere down the logic due to some unconstrained signals in the path. Look out for such unconstrained muxes/sequential logic in the path and constraint them properly.

    Consider the following design:

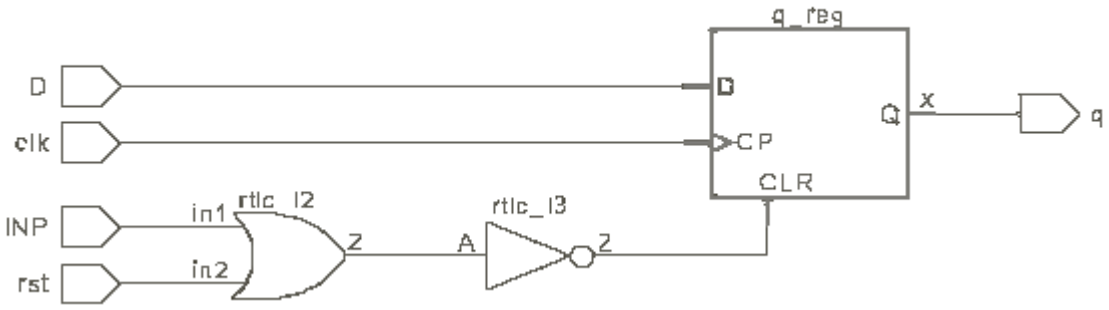


FIGURE 416. Design on Which Ac_initstate01 Rule is Run

In the above design, if `INP` is tied to 1, then the asynchronous reset path would get blocked and the flip-flop would be reported as uninitialized (also denoted by an `X` appearing at its output terminal).

In addition, you should also check if the reset pin is not tied to inactive value in the design or through *set_case_analysis* constraints.

- If the register was designed to be reset by a synchronous reset, then:
  - Check register data path to see if the synchronous reset reaches it or not. Once you find this signal, specify it through *reset -sync* constraint in the SGDC file, if it is not specified.
  - Check if other signals in the reset path are constrained so that active value of the reset can reach to the register for initialization.
  - Check that the register is getting a proper clock as clock is essential for synchronous resets. In addition, check if the clock is not propagating to the register due to unconstrained or blocked clock enables in the path.

  When an event is applied on a clock source, it should reach to the register through simulation so that synchronous reset can be applied on the register.

  Consider the following design:

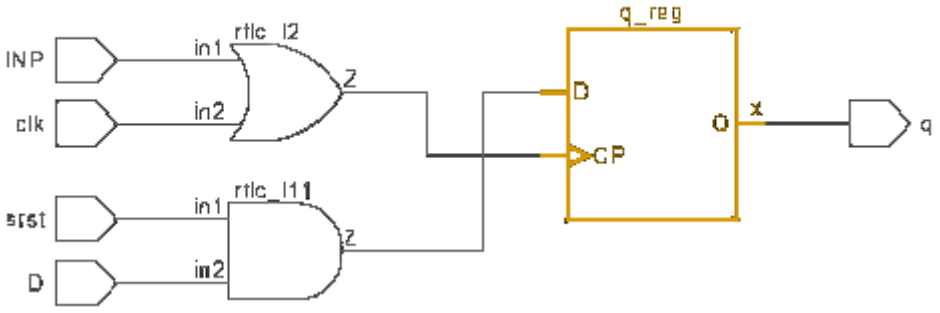


FIGURE 417. Design on Which Ac_initstate01 Rule is Run

In the above design, if `INP` is tied to 1, the clock path would get blocked. This in turn would block the synchronous reset and therefore, the flip-flop would be reported as uninitialized (denoted by an `X` appearing at its output terminal).

- Check if the enable of the register is not tied to an inactive value.
- Check if the clock pin is tied to a constant value in the design or through *set_case_analysis* constraints.

- To summarize the above two points, for flip-flops getting reported as uninitialized in Section-C, you need to ensure that the reset signal for these flip-flops is specified in the SGDC file with its correct active value.

- Memory elements, such as RAM/FIFO are sometimes not required to be initialized. In case they do not have any valid reset (async/sync), they can be left out. However, control signals should ideally be initialized.
- For detailed information on various ways to do initial state setup, see the description of the *Ac_initstate01* rule.

Example Code and/or Schematic

See *How to Debug and Fix*.

Back-annotations

- HDL: Shows the top-level module
- Schematic: Highlights registers that are initialized with value x.

Default Severity Label

Info

Rule Group

ADV_CLOCKS

Report and Related File

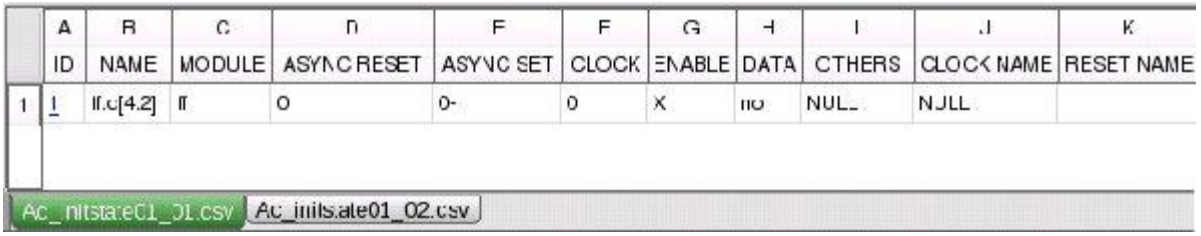
- adv_cdc.reg: This file lists registers that could not be initialized.
- adv_cdc_init_seq.vcd: This is an initial state VCD file, which has simulation vectors applied on primary inputs during the initial state search.
- **The Ac_initstate01 spreadsheet**

  The *Ac_initstate01 Spreadsheet Report* shows information about uninitialized and initialized sequential elements in a design. This information appears under two separate tabs. By default, the tab showing uninitialized elements is selected.

  To open this spreadsheet, double-click on the violation of the *Ac_initstate01* rule. You can click on a sequential element name in the

spreadsheet to back-reference to the RTL code of that element.

The following figure shows the uninitialized sequential elements:

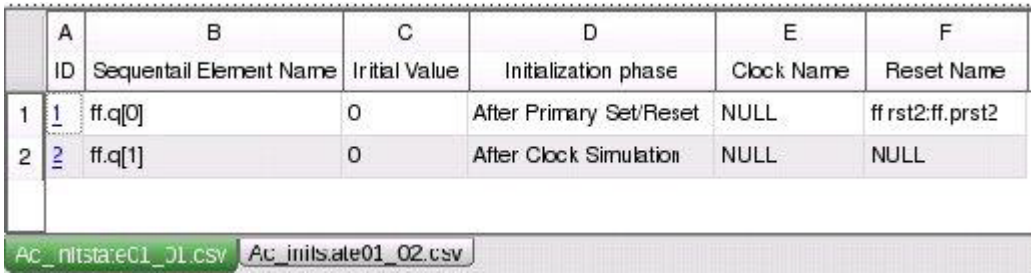


	A	B	C	D	E	F	G	H	I	J	K
	ID	NAME	MODULE	ASYNC RESET	ASYNC SET	CLOCK	ENABLE	DATA	OTHERS	CLOCK NAME	RESET NAME
1	1	ff.q[4:2]	ff	0	0	0	X	no	NULL	NULL	

Ac_nitsta:eC1_01.csv | Ac_inits.ate01_02.csv

FIGURE 418. Default tab showing uninitialized sequential elements

To view the initialized sequential elements, select the second tab. The following spreadsheet shows the initialized sequential elements in a design:



	A	B	C	D	E	F
	ID	Sequentail Element Name	Initial Value	Initialization phase	Clock Name	Reset Name
1	1	ff.q[0]	0	After Primary Set/Reset	NULL	ff rst2:ff.prst2
2	2	ff.q[1]	0	After Clock Simulation	NULL	NULL

Ac_nitsta:eC1_01.csv | Ac_inits.ate01_02.csv

FIGURE 419. Second tab showing the initialized sequential elements

Ac_license01

Reports rules that did not run due to unavailability of the Advanced_CDC or the cdc_dynamic_jitter_analysis license

When to Use

Use this rule to check the availability of the `Advanced_CDC` or the `cdc_dynamic_jitter_analysis` license.

Description

The *Ac_license01* rule reports the unavailability of the `Advanced_CDC`, `cdc_power`, and the `cdc_dynamic_jitter_analysis` license when the rules dependent on this license are run.

Rules that Require the Advanced CDC License

Apart from the Spreadsheet Viewer, the `Advanced_CDC` license is required by the following rules:

Rule Category	Rules		
Must Rules	*Ac_init01*	*Ac_initstate01*	*Ac_multitop01*
	Ac_sanity01	*Ac_sanity02*	*Ac_report01*
Setup Rules	*Setup_quasi_static01*		
CDC Verification Rules	*Ac_cdc01a*	*Ac_cdc01b*	*Ac_cdc01c*
	Ac_cdc08	*Ac_crossing01*	*Ac_sync01*
	Ac_conv02 (Only functional part)	*Ac_datahold01a*	*Ac_sync02*
	Ac_unsync01	*Ac_unsync02*	*Ac_xclock01*
	`Ac_psync01`	`Ac_punsync01`	
Clock Glitch Checking Rules	*Ac_glitch01*	*Ac_glitch02*	*Ac_glitch03*
Formal Setup Rules	*Ac_sanity03*	*Ac_sanity04*	

Rule Category	Rules		
Clock Checking Rules	*Ar_asyncdeassert01*	*Ar_syncdeassert01*	*Ar_sync01*
	Ar_unsync01		
Reset Checking Rules	*Ar_converge01*	*Ar_converge02*	
Delta Delay Rules	*DeltaDelay01*	*DeltaDelay02*	*NoClockCell*
	PortTimeDelay		
Reset Information Rules	*Ar_syncrst_setupcheck01*		

NOTE: *If the Advanced_CDC license is unavailable during the save run, the above rules will not run in the restore mode.*

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears because of the unavailability of the `Advanced_CDC` license:

```
[ERROR] <List-of-rules> not run due to unavailability of
Advanced CDC license/cdc_dynamic_jitter_analysis feature
```

Potential Issues

This message appears if you do not specify the `Advanced_CDC` or the `cdc_dynamic_jitter_analysis` license.

Consequences of Not Fixing

If you do not fix this violation, rules dependent on the license mentioned in

the message are not run. Refer to the *Rules that Require the Advanced CDC License* for the complete list of rules that need the `Advanced_CDC` license to run.

How to Debug and Fix

To fix this violation, specify the `Advanced_CDC` license correctly.

Example Code and/or Schematic

Not applicable

Default Severity Label

Error

Rule Group

`ADV_CLOCKS`

Reports and Related Files

No report or related file

Ac_multitop01

Reports a violation in case of multiple top-level design units

When to Use

Use this rule to detect multiple top-level design units.

Prerequisites

- Run any of the *CDC Verification Rules* or the *Ac_crossing01* rule.
- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_multitop01* rule reports a violation in case of multiple top-level design units.

Parameter(s)

None

Constraint(s)

None

Messages and Suggested Fix

The following message appears in case of multiple top-level design units:

```
[ERROR] Detected '<number-of-top-level-DU>' top level design
units. Please specify a single top level design unit
```

Potential Issues

This violation appears if your design contains multiple top-level design units.

Consequences of Not Fixing

If you do not fix this violation, *CDC Verification Rules* and the *Ac_crossing01* rule does not perform functional checks. These rules work only when a single top-level design unit is present.

How to Debug and Fix

To debug this violation, check the name of multiple top-level design units by referring to the violation of the *DetectTopDesignUnits* rule.

To fix this violation, specify a single top-level design unit by specifying the following command in a project file:

```
set_option top <top-du-name>
```

Example Code and/or Schematic

Consider the following file specified during SpyGlass analysis:

```
module test1 (input d, clk, output reg q);
always @(posedge clk)
  q <= d;
endmodule


module test2 (input d, clk, output reg q);
always @(posedge clk)
  q <= d;
endmodule
```

For the above example, the *Ac_multitop01* rule reports a violation because of the presence of two top-level design units `test1` and `test2`.

Default Severity Label

Error

Rule Group

`ADV_CLOCKS`

Reports and Related Files

No report or related file

Ac_upfsetup02

Reports when appropriate isolation/level shifter strategy on domain element is not specified

When to Use

This is a set-up rule and always runs by default.

Description

The *Ac_upfsetup02* rule reports a violation when proper isolation/level shifter strategy is not defined for all the domain elements using `set_isolation`/`set_level_shifter` commands, respectively.

This rule reports following scenarios:

- Element specified in the `set_isolation`/`set_level_shifter` command is not on any domain boundary or is on an incorrect domain boundary.
- Element specified in the `set_isolation`/`set_level_shifter` command is an INOUT port/pin.
- Multiple isolation/level shifter strategies are specified for the same element or all inputs or all outputs of a domain.
- Element specified in the `set_isolation`/`set_level_shifter` command contains an incorrect power domain.
- Element specified in the `set_isolation` command has multiple source or sink domains.
- Multiple isolation strategies, which have the same precedence, are applicable for the element specified in the `set_isolation` command.
- Incorrect source or sink is specified for the element specified in the `set_isolation` command.

Language

Verilog, VHDL

Parameter(s)

None

Constraint(s)

UPF

- `set_isolation` (Mandatory)
- `set_level_shifter` (Mandatory)

Messages and Suggested Fix

Message 1

The following message appears when the element *<pin-name>* specified in *<cmd-name>* (`set_isolation/set_level_shifter`) command is not on any domain boundary:

[AcUS2_1] [ERROR] Element '<pin-name>' specified in <cmd-name> command [<file-name>:<line-no>] is not on domain boundary. No <chk-type> checks performed on this element

where, *<chk-type>* can be isolation or level shifter.

For debugging information, click *How to Debug and Fix*.

Message 2

The following message appears when an isolation/level shifter strategy is specified on an inout port *<port-name>* using *<cmd-name>* (`set_isolation/set_level_shifter`) command:

[AcUS2_2] [WARNING] Inout port '<port-name>' specified in <cmd-name> command [<file-name>:<line-no>]. No <chktype> checks performed on this element

where, *<chk-type>* can be isolation or level shifter.

For debugging information, click *How to Debug and Fix*.

Message 3

The following message appears when multiple *<chk-type>* (*isolation*/*levelshifter*) strategies are specified for *<reported-element(s)>* of the *<dom-name>* domain:

[AcUS2_4] [ERROR] Multiple <chk-type> strategies '<rulename1>' [<file-name1>:<line-num1>], '<rule-name2>' [<file-name2>:<line-num2>] specified for '<reported-element(s)>' of domain '<dom-name>'. No <chk-type> checks performed on

<reported-element>

where, *<reported-element(s)>* is the hierarchical name of a domain element or *all inputs* or *all outputs* when multiple isolation/level shifter specifications are given for an element or all inputs or all outputs of the domain, respectively.

For debugging information, click *How to Debug and Fix*.

Message 4

The following message appears for the specified *<strategy-name>* strategy when the *<element-hier-name>* element specified in the set_isolation/set_level_shifter command contains an incorrect *<domain-name>* power domain:

[AcUS2_5] [ERROR] Strategy (set_isolation | set_level_shifter) <strategy-name> [<upf-file-name>:<upf-line-number>] is ignored on element <element-hier-name> as element does not belong to specified domain '<domain-name>'

For debugging information, click *How to Debug and Fix*.

Message 5

The following message appears when the *<element-hier-name>* element specified in the set_isolation command has multiple source or sink domains *<domain-name-list>:*

[AcUS2_7] [ERROR] Element '<element-hier-name>' has multiple <source | sink> domains (<domain-name-list>). No isolation strategy is applied on this element

For debugging information, click *How to Debug and Fix*.

Message 6

The following message appears when multiple isolation strategies *<isolation-strategy-list>* that have the same precedence are applicable for the *<element-hier-name>* element specified in the set_isolation command:

[AcUS2_8] [ERROR] Multiple isolation strategies (<isolation-strategy-list>' having same precedence are applicable for element '<element-hier-name>'. No isolation strategy is applied on this element

For debugging information, click *How to Debug and Fix*.

Message 7

The following message appears when an incorrect source or sink is specified for the *<element-hier-name>* element specified in the `set_isolation` command:

[AcUS2_6] [ERROR] `Incorrect <source | sink> '<supply-set-name>' specified for element '<element-hier-name>' in set_isolation command '<upf-isolation-command-name>'. No isolation strategy is applied on this element`

Potential Issues

This violation message explicitly specify the potential issues.

Consequences of Not Fixing

SpyGlass CDC solution does not apply an isolation/level shifter strategy to an element for which this rule is reporting a violation. Since no isolation/level shifter strategy is being applied on an element, subsequent rules (which run after this rule) perform rule checking based assuming there is no isolation/level shifter strategy on that element.

For example, if an isolation cell is inserted for an element on which you have written an isolation strategy and that strategy is reported by this rule. The strategy is not applied to that element. This causes an isolation cell to be redundant and different checks, such as checking its location, its functionality, steady state value on its output etc. are not performed by the rules of SpyGlass CDC solution.

How to Debug and Fix

Ensure that design has properly specified isolation/level shifter strategy.

Example Code and/or Schematic

Example 1

For the following snippet, the *Ac_upfsetup02* rule reports a violation because `u1` is the domain boundary and not `u1/u2`, hence element `u1/u2/out1` is not on domain boundary.

```
create_power_domain PD1 -elements {u1}
set_isolation ISO1 -domain PD1 -elements {u1/u2/out1}…..
```

Therefore, *Message 1* is reported.

Example 2

For the following snippet, the *Ac_upfsetup02* rule reports a violation because `inout1` is an inout port of domain `PD1`.

```
set_isolation ISO1 –domain PD1 –elements {u1/inout1}…..
```

Therefore, *Message 2* is reported.

Example 3

For the following snippet, the *Ac_upfsetup02* rule reports a violation because multiple isolation strategies (ISO1 and ISO2) have been specified for same element {u1/out1}.

```
set_isolation ISO1 –domain PD1 –elements {u1/out1} –
no_isolation

set_isolation ISO2 –domain PD1 –elements {u1/out1} –
clamp_value 0
```

Therefore, *Message 3* is reported.

Example 4

For the following snippet, the *Ac_upfsetup02* rule reports a violation because the isolation strategy `ISO1` is specified for domain `PD1`. However, element `{u2/out1}` specified in the strategy belongs of domain `PD2 (u2)`.

```
create_power_domain PD1 –elements {u1}

create_power_domain PD2 –elements {u2}

set_isolation ISO1 –domain PD1 –elements {u2/out1} –
clamp_value 0
```

Therefore, *Message 4* is reported.

Example 5

Suppose an output port `Y` of domain `PD1_domain` has multiple fan-outs or sink domains: `PD2_domain` and `VD_domain`.

UPF

```
associate_supply_set PD1_domain_set -handle
PD1_domain.primary

associate_supply_set PD2_domain_set -handle
```

```
PD2_domain.primary
```

```
set_isolation ISO2 -domain PD1_domain -isolation_power_net
PD1_domain_supply -source PD1_domain_set -sink
PD2_domain_set
```

In this case, the rule considers `Y` as a non-uniform net. Therefore, no strategy is applied on it and *Message 5* is reported.

Example 6

Suppose an output port `Y` of domain `PD1_domain` has `PD1_domain` as source domain and `PD2_domain` as destination domain.

UPF

```
associate_supply_set PD1_domain_set -handle
PD1_domain.primary
```

```
set_isolation ISO1 -domain PD1_domain -isolation_power_net
PD1_domain_supply -source PD1_domain_set -clamp_value 0
```

```
associate_supply_set PD2_domain_set -handle
PD2_domain.primary
```

```
set_isolation ISO2 -domain PD1_domain -isolation_power_net
PD1_domain_supply -sink PD2_domain_set -clamp_value 0
```

In this case, since both strategies, `ISO1` and `ISO2`, are applicable on output `Y` and they have the same precedence, *Message 6* is reported.

Example 7

Suppose the following isolation strategy is written on the output element A of domain VC. The *Ac_upfsetup02* rule reports a violation if the sink of element A is different from VDDB.

```
set_isolation S1 -domain VC -elements A -sink VDDB
```

This strategy will not be applied to element A and *Message 7* is reported.

Default Severity Label

Error

Rule Group

ADV_CLOCKS

Reports and Related Files

No reports and/or related files

Ac_report01

Reports statistics of properties and functional constraints

When to Use

Use this rule to perform functional analysis and check the number of properties analyzed, failed, passed, and partially proved.

Prerequisites

Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_report01* performs functional analysis and reports the number of properties analyzed, failed, passed, and partially proved when the *fa_audit* parameter is not specified.

NOTE: *The Ac_report01 rule is automatically run when you run any of the CDC Verification Rules.*

Parameter(s)

- *fa_audit*: Default value is `no`. Set this parameter to `yes` to not perform functional analysis.
- *fa_propfile*: Default value is `NULL`. Set this parameter to the name of a property file containing properties to be checked.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears if the *fa_audit* parameter is set to `yes`:

```
[AcRpt1_1] [INFO] Functional analysis not done in audit mode.
Design has '<num>' properties, '<imp-num>' implicit properties,
'<ovl-num>' OVL properties, and '<constr-num>' functional
constraints for top design unit '<du-name>'. Refer file:
'<file-name>' for details
```

The arguments of the above message are explained below:

Argument	Description
<num>	Total number of properties in the design
<imp-num>	Total number of implicit properties
<ovl-num>	Total number of OVL properties
<constr-num>	Total number of functional constraints
<du-name>	Top-level design name
<file-name>	Generated Property file name

Potential Issues

This message appears when you set the *fa_audit* parameter to `yes`.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Message 2

The following message appears if the *fa_audit* parameter is set to `no`:

[AcRpt1_2] [INFO] Implicit: ‘<imp-analyzed-num>’ implicit properties analyzed, '<imp-failed-num>' failed, '<imp-passed-num>' passed, '<imp-partial-num>' partially proved, '<imp-not-analyzed-num>' not analyzed, '<imp-others-num>' others for top design unit '<du-name>'. Refer file: '<file-name>' for details

The arguments of the above message are explained below:

Argument	Description
<imp-analyzed-num>	Number of implicit properties analyzed
<imp-failed-num>	Number of implicit properties failed
<imp-passed-num>	Number of implicit properties passed
<imp-partial-num>	Number of implicit properties partially proved

Argument	Description
<imp-not-analyzed-num>	Number of implicit properties not analyzed
<imp-others-num>	Total number of constraint conflicts
<du-name>	Top-level design name
<file-name>	Generated Property file name

Potential Issues

This message appears when you set the *fa_audit* parameter to `no`.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

Not applicable

Example Code and/or Schematic

Not applicable

Default Severity Label

Info

Rule Group

`ADV_CLOCKS`

Reports and Related Files

- adv_cdc.prp: This file contains the list of *CDC Verification Rules* that have run. For details, see *The Functional Validation Methodology*.
- *The Advanced CDC Report*
- *The adv_cdc Spreadsheet*

Ac_sanity01

Reports an error if there is any issue in the property file.

When to Use

Use this rule to identify missing assertions in a design.

Prerequisites

Specify the following details before running this rule:

- Specify a property file by using the *fa_propfile* parameter.
- Run any of the *CDC Verification Rules* or set the *formal_setup_rules_check* parameter to `yes`.
- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_sanity01* rule reports issues found in the user-specified property files.

Parameter(s)

fa_propfile: Default value is `NULL`. Set this parameter to the name of a property file containing properties to be checked.

Constraint(s)

None

Messages and Suggested Fix

The following message appears when some assertions specified in a property file are not found in the design:

[ERROR] `Some Assertions specified in property file (<prop-file-name>) not found in design. Refer '<file-name>' for more details`

The arguments of the above message are explained below:

Argument	Description
<prop-file-name>	Name of the property file
<file-name>	Name of the error log file

Potential Issues

This violation appears because of an outdated property file.

A property may get outdated because there may be assertions in the property file that do not exist in the design due to the following reasons:

- The design has been modified.
- The design view has been modified due to different set of SpyGlass CDC commands specified by the user.

Consequences of Not Fixing

If you do not fix this violation, some *Formal Setup Rules* may not run.

How to Debug and Fix

To fix this violation, re-run SpyGlass analysis on the design with the *fa_audit* parameter set, and generate a new property file.

Example Code and/or Schematic

Not applicable

Default Severity Label

Error

Rule Group

ADV_CLOCKS

Reports and Related Files

This rule generates the propfile_Assertion_<*rule-name*>.errorlog file. This file contains missing assertions. Here *<rule-name>* refers to the rule that is not run due to invalid assertions.

Ac_sanity02

Reports nets that have multiple drivers

When to Use

Use this rule during functional analysis to detect nets with multiple drivers.

Prerequisites

Following are the prerequisites of running this rule:

- Run any of the *CDC Verification Rules* or set the *formal_setup_rules_check* parameter to `yes`.
- Use the `Advanced_CDC` and `adv_checker` licenses for running this rule.

Description

The *Ac_sanity02* rule reports non-tristate nets that have multiple drivers.

Such nets are considered as primary inputs for functional analysis.

Rule Exception

The *Ac_sanity02* rule does not handle inout ports driven by black boxes correctly and may report incorrect messages in such cases.

Parameter(s)

None

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears when a non-tristate net *<net-name>* is present in a design with multiple drivers:

[WARNING] Net '<net-name>' is not tristate and has multiple simultaneous drivers

Potential Issues

This violation appears if your design contains a non-tristate net with multiple drivers.

Consequences of Not Fixing

Not applicable

How to Debug and Fix

To fix this violation, provide tristate nets in the design.

Example Code and/or Schematic

Consider the following figure:

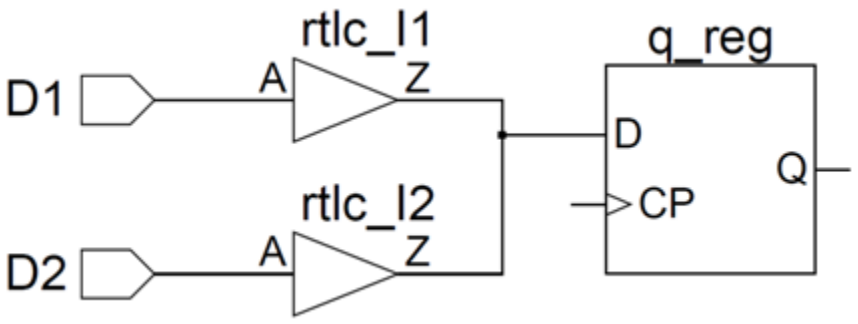


FIGURE 420. Schematic of the Ac_sanity02 Rule Violation

For the above example, the *Ac_sanity02* rule reports a violation because of the presence of a non-tristate net with multiple drivers `D1` and `D2`.

Schematic Details

The *Ac_sanity02* rule highlights the non-tristate net that has multiple

drivers.

Default Severity Label

Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

No related files and reports

Ac_sanity06

Reports any issue found in distributed computing flow

When to Use

Use this rule to detect issues related with distributed runs of the advanced SpyGlass CDC rules.

Prerequisites

Run any of the *CDC Verification Rules* or set the *formal_setup_rules_check* parameter to `yes`.

Description

The *Ac_sanity06* rule reports a violation in the following cases:

- If parse errors are found in the parallel file specified by the *fa_parallelfile* parameter.
- If an error occurs while accessing any of the machines specified in the parallel file.
- If there are insufficient number of advanced SpyGlass CDC solution licenses

Parameter(s)

fa_parallelfile: By default, this parameter is not set to any value. Specify a configuration file to this parameter. This file is used for distributed runs of advanced SpyGlass CDC rules over several machines.

Constraint(s)

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Message Details

Message 1

The following message appears if parse errors are found in the parallel file:

[AcS6_1] [FATAL] Could not open parallel run file '<file-name>'

Potential Issues

This violation appears if there is any error in the parallel run file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a correct parallel run file.

Message 2

The following message appears for an invalid login type:

[AcS6_2] [FATAL] <type> is not a supported login type

Potential Issues

This violation appears if you specify an invalid login type in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a correct login type.

Message 3

The following message appears if the value of the MAX_PROCESSES keyword is equal to or less than 1 or if it is equal to or greater than 500:

[AcS6_3] [FATAL] Value of MAX_PROCESSES should be between 1 and 500

Potential Issues

This violation appears if you specify an incorrect value for the MAX_PROCESSES keyword in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a value between 1 and 500 for the

MAX_PROCESSES keyword.

Message 4

The following message appears for the unsuccessful LSF run because of invalid options in the LSF command:

[AcS6_4] [FATAL] `Lsf run with specified command is not successful`

Potential Issues

This violation appears if you specify invalid options with the LSF command in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify correct options for the LSF command.

Message 5

The following message appears if process count is not a positive integer value in the parallel file:

[AcS6_5] [FATAL] `Process count in parallel file must be a positive integer`

Potential Issues

This violation appears if you specify an invalid integer value to the process count in the parallel file. The process count accepts only a positive integer value.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a positive integer value for the process count in the parallel file.

Message 6

The following message appears if none of the specified machines in the parallel file is accessible:

[AcS6_6] [FATAL] `None of the machines specified in parallel file is accessible`

Potential Issues

This violation appears if none of the machines specified in a parallel file is accessible.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the names of accessible machines in the parallel file.

Message 7

The following message appears to report the machines that are not accessible:

[AcS6_7] [FATAL] `Machines '<machines>' are not accessible`

Potential Issues

This violation appears if none of the machines specified in a parallel file is accessible.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the names of accessible machines in the parallel file.

Message 8

The following message appears if the `LOGIN_TYPE` keyword is not specified in the parallel file:

[AcS6_8] [FATAL] `'LOGIN_TYPE' is not specified in parallel file`

Potential Issues

This violation appears if you do not specify the LOGIN_TYPE keyword in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the LOGIN_TYPE keyword in the parallel file.

Message 9

The following message appears if the MAX_PROCESSES keyword is not specified in the parallel file:

[AcS6_9] [FATAL] 'MAX_PROCESSES' is not specified in parallel file

Potential Issues

This violation appears if you do not specify the MAX_PROCESSES keyword in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the MAX_PROCESSES keyword in the parallel file.

Message 10

The following message appears if the MACHINES keyword is not specified for the rsh/ssh login type in the parallel file:

[AcS6_10] [FATAL] 'MACHINES' not specified for login type rsh/ssh in parallel file

Potential Issues

This violation appears if you do not specify the MACHINES keyword for the rsh/ssh login type in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the `MACHINES` keyword for the `rsh`/`SSE` login type in the parallel file.

Message 11

The following message appears if an error occurs while running the `lsf bsub` command:

[AcS6_11] [FATAL] `Error executing lsf bsub command`

Potential Issues

This violation appears if you specify invalid options, such as `-I`, `-Ip`, and `-Is` with the `bsub` command. These options are not allowed with the `LSF_CMD` keyword in the parallel file.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify valid the options with the `bsub` command.

Message 12

The following message appears to indicate a missing solver executable:

[AcS6_12] [FATAL] `Solver executable '<executable>' not found`

Potential Issues

This violation appears if the solver executable file is not found in the SPYGLASS_HOME/lib/ path of SpyGlass release area. The name of this file is of the format solver.<*platform*>. For example, solver.SunOS5.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, add the missing solver executable file in the SPYGLASS_HOME/lib/ path of SpyGlass release area.

Message 13

The following message appears to indicate missing advanced SpyGlass CDC solution licenses for distributed computing flow:

[AcS6_13] [FATAL] `No Advanced CDC licenses available for Distributed Computing Flow`

Potential Issues

This violation appears if you do not specify advanced SpyGlass CDC licenses.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, provide the advanced SpyGlass CDC licenses for distributed computing flow.

Message 14

The following message appears to indicate inadequate advanced SpyGlass CDC solution licenses available for distributed computing flow:

[AcS6_14] [WARNING] `Only '<num>' Advanced CDC licenses available for Distributed Computing`

Potential Issues

This violation appears if you specify insufficient number of advanced SpyGlass CDC licenses.

Consequences of Not Fixing

If `n` advanced CDC licenses are available, only `n-1` licenses are used for distributed computing as one of the licenses is used by the main process.

How to Debug and Fix

To fix this violation, specify adequate number of advanced SpyGlass CDC licenses for distributed computing flow.

Message 15

The following message appears if you specify an invalid option *<option-name>* in the LSF_CMD keyword in a parallel file:

```
[AcS6_15] [WARNING] Unsupported option '<option-name>'
specified in LSF_CMD field is ignored for Distributed Computing
Flow
```

Potential Issues

This violation appears if you specify an invalid option *<option-name>* in the LSF_CMD keyword in a parallel file.

Consequences of Not Fixing

If you do not fix this violation, distributed computing does not run.

How to Debug and Fix

To fix this violation, specify supported options with the LSF_CMD keyword in a parallel file.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal | Warning

Rule Group

ADV_CLOCKS

Reports and Related Files

No report or related file

AllowComboLogicSetup

Reports if the modules specified by the allow_combo_logic constraint are not used by any crossing

When to Use

Use this rule to check if the *allow_combo_logic* constraint is considered during SpyGlass analysis.

Prerequisites

Specify the *allow_combo_logic* constraint.

Description

The *AllowComboLogicSetup* rule reports a violation if the modules specified by the *allow_combo_logic* constraint are not used by any crossing.

Parameter(s)

None

Constraint(s)

allow_combo_logic (Mandatory): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.

Messages and Suggested Fix

The following message appears when modules specified by the *allow_combo_logic* constraint are not used in any crossing:

```
[WARNING] 'allow_combo_logic' constraint is not used in the design
```

Potential Issues

This violation appears if the modules specified by the *allow_combo_logic* constraint are not used by any crossing.

Consequences of Not Fixing

If you do not fix this violation, the reported modules are not considered in any crossings. This may produce results that might not be as per your expectations.

How to Debug and Fix

To fix this violation, specify correct modules in the *allow_combo_logic* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

Clock_check07

Reports clock domains that reach another clock domain

When to Use

Use this rule to check setup for clocks in a design.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use automatically generated clock signals
- By using a combination of both the above methods

Description

The *Clock_check07* rule reports clock domains that reach another clock domain.

Parameter(s)

- *cdc_express*: Default values is `no`. Set this parameter to `peakmem` to reduce peak memory. Other possible value is `yes`.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

The following message appears when clock propagation of the clock *<clk1-name>* reaches the clock net *<clk2-name>*:

```
[WARNING] Clock domain propagation for clock '<clk1-name>' has
reached clock '<clk2-name>' of another domain. Halting further
propagation of clock '<clk1-name>' on this path
```

Potential Issues

This violation appears when a design contains a clock that reaches to another clock from a different domain in the path.

Consequences of Not Fixing

If you do not fix this violation, clock propagation stops along the path where a clock reaches another clock of a different domain. This may result in an improper clock domain crossing analysis.

How to Debug and Fix

To debug this violation, perform the following steps:

- Check if clock names reported by the message are real clocks.
- Check if the domains of the clocks reported by the message are specified correctly in the SGDC file.
- Open the schematic of the violation and check if clock propagation occurs correctly.
- Make necessary changes to eliminate the conflict.

Example Code and/or Schematic

Consider the following figure:

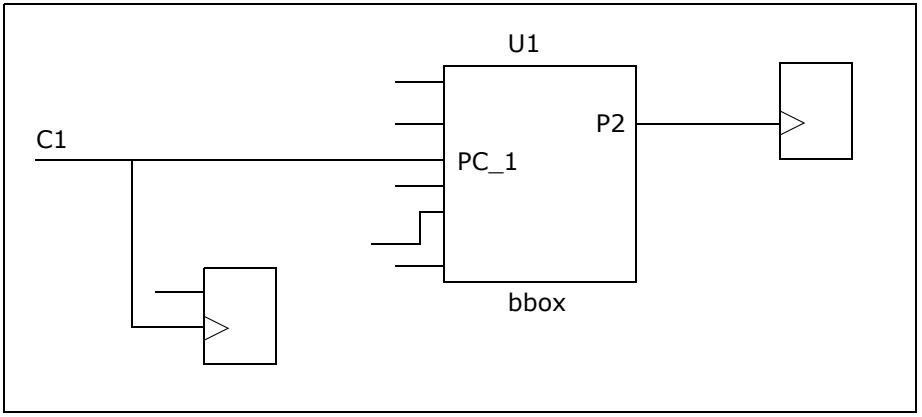


FIGURE 421. Scenario for the Clock_check07 Rule Violation

Also consider the following SGDC file:

```
current_design top
clock -name C1 -domain d1
clock -name top.U1.P2 -domain d2
assume_path -name bbox -input P_C1 -output P2
```

In this case, the top-level clock `C1` from the `d1` domain reaches to `P_C1` and propagates to `P2` because of the *assume_path* constraint. However, the *clock* constraint is also defined on `P2` that has different the domain `d2`.

Therefore, the *Clock_check07* rule reports a violation because of a conflict of domains.

To fix this violation, perform the following actions:

- Do not define the *clock* constraint on `top.U1.P2` as the top-level clock `C1` propagates to `P2` due to the *assume_path* constraint.
- If you want to explicitly specify a clock on the black box output, ensure that the domain defined on `P2` is same as the source clock.

Schematic Details

The *Clock_check07* rule highlights the path from one clock domain to the

point where another clock domain is reached.

Default Severity Label

Warning

Rule Group

VERIFY

Reports and Related Files

The CKSGDCInfo Report

Section A of this report displays clocks with their domain information, which can be used to check if the domains are specified correctly.

Param_clockreset02

Reports if an incorrect value is specified to the num_flops parameter

When to Use

Use this rule to perform sanity checks on the *num_flops* parameter.

Prerequisites

Specify the *num_flops* parameter.

Description

The *Param_clockreset02* rule reports incorrect value specified to the *num_flops* parameter.

Parameter(s)

num_flops: Default value is `2`. Set this parameter to a positive integer value greater than one to specify a minimum number of flip-flops required for synchronizing a signal by using the *Conventional Multi-Flop Synchronization Scheme*.

Constraint(s)

None

Messages and Suggested Fix

The following message appears if an incorrect value *<value>* is specified to the *num_flops* parameter:

[WARNING] Illegal value '<value>' specified with 'num_flops' parameter

Potential Issues

This violation appears if you specify an incorrect value to the *num_flops* parameter.

Consequences of Not Fixing

If you do not fix this violation, the reported *num_flops* parameter is not considered during SpyGlass analysis. In addition, synchronization CDC rules may report false violations.

How to Debug and Fix

To fix this violation, specify a positive integer value greater than 1 to the *num_flops* parameter.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

FalsePathSetup

Reports cases in which the cdc_false_path constraint is not used by any crossing in the design

When to Use

Use this rule to check if the *cdc_false_path* constraint is considered during SpyGlass analysis.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *FalsePathSetup* rule reports a violation if the *cdc_false_path* constraint does not filter any clock domain crossing in a design.

Parameter(s)

None

Constraint(s)

- *cdc_false_path* (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *clock* (Mandatory): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears if the *cdc_false_path* constraint does not filter any clock domain crossing in a design:

```
[WARNING] cdc_false_path constraint is not used to waive any crossing in the design
```

Potential Issues

This violation appears if you do not define the *cdc_false_path* constraint properly.

Consequences of Not Fixing

If you do not fix this violation, you may see certain clock domain crossings that you expected to be filtered by the specified *cdc_false_path* constraint.

How to Debug and Fix

To debug and fix this violation, perform the following actions:

- Review the reported constraints.
- Remove the reported constraint if it is redundant.

  For example, if the same crossings are matched by some other constraint, you can remove the reported constraint.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
clock -name clk1 -tag CK1
clock -name clk2 -domain d2


cdc_false_path -from CK1
cdc_false_path -from clk1 -to clk2
```

In the above example, the last *cdc_false_path* is redundant. Therefore, the *FalsePathSetup* rule reports a violation for this constraint.

Default Severity Label

Warning

Rule Group

Non-Fatal Must Rule

Reports and Related Files

No report or related file

Param_clockreset04

Reports if an incorrect value is specified for the cdc_reduce_pessimism, clock_reduce_pessimism, or reset_reduce_pessimism parameter

When to Use

Use this rule to check if the *cdc_reduce_pessimism*, *clock_reduce_pessimism*, and *reset_reduce_pessimism* parameters are specified correctly.

Prerequisites

Specify the *cdc_reduce_pessimism*, *clock_reduce_pessimism*, or *reset_reduce_pessimism* parameter.

Description

The *Param_clockreset04* rule reports a violation if incorrect values are specified to the *cdc_reduce_pessimism*, *clock_reduce_pessimism*, or *reset_reduce_pessimism* parameter.

Parameter(s)

- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

Constraint(s)

None

Messages and Suggested Fix

The following message appears if incorrect values are specified to the *cdc_reduce_pessimism*, *clock_reduce_pessimism*, or *reset_reduce_pessimism* parameter:

```
[WARNING] Illegal value specified with '<parameter-name>'
parameter. Allowed values are '<valid-values>' only
```

Potential Issues

This violation appears if you specify incorrect values to the *cdc_reduce_pessimism*, *clock_reduce_pessimism*, or *reset_reduce_pessimism* parameter.

Consequences of Not Fixing

If you do not fix this violation, the reported parameter is ignored.

How to Debug and Fix

Specify valid values to the reported parameters, as described in the following table:

Parameter	Valid Values
cdc_reduce_pessimism	bbox, output_not_used, hanging_net, mbit_macro, no_convergence_at_enable, no_convergence_at_syncreset, and all
clock_reduce_pessimism	mux_sel, latch_en, all, all_potential_clocks, and ignore_same_domain
reset_reduce_pessimism	none, all_potential_resets

Example Code and/or Schematic

Consider that you specify the following parameter in a project file:

```
set_parameter  abc
```

In this case, the *Param_clockreset04* rule reports a violation because the value `abc` is not a valid value for this parameter.

Default Severity Label

Warning

Rule Group

Non-Fatal Must Rule

Reports and Related Files

No report or related file

Param_clockreset05

Reports if the simulator_file_name parameter is not specified or an invalid value is specified to this parameter

When to Use

Use this rule to perform sanity checks on the *simulator_file_name* parameter.

Description

The *Param_clockreset05* rule reports a violation in any of the following cases:

- If the simulator mode file specified by the *simulator_file_name* parameter does not exist
- If the *simulator_file_name* parameter is not specified

Parameter(s)

simulator_file_name: Default value is `NULL`. Specify a simulator mode file that contains simulator-specific delta delay information for RTL constructs.

Constraint(s)

None

Messages and Suggested Fix

Message 1

The following message appears if the an incorrect simulator mode file is specified to the *simulator_file_name* parameter:

```
[PClkRst5_2] [FATAL] File '<file-name>' specified through parameter 'simulator_file_name' does not exist
```

Potential Issues

This violation appears if the simulator mode file specified by the *simulator_file_name* parameter does not exist.

Consequences of Not Fixing

If you do not fix this violation, the specified simulator mode file is not considered during SpyGlass analysis and the *DeltaDelay01* and *DeltaDelay02* violations are not reported.

How to Debug and Fix

To fix this violation, specify an existing simulator mode file to the *simulator_file_name* parameter.

The sample simulator mode file, simulator_file.txt, is present at the following path:

SPYGLASS_HOME/policies/clock/

You can directly pass this sample file to the *simulator_file_name* parameter or modify it to specify delay values for different simulators before passing it to this parameter.

Message 2

The following message appears if the *simulator_file_name* parameter is not specified:

```
[PClkRst5_1] [WARNING] Parameter '-simulator_file_name' is not specified
```

Potential Issues

This violation appears the *simulator_file_name* parameter is not specified.

Consequences of Not Fixing

If you do not fix this violation, the *DeltaDelay01* and *DeltaDelay02* violations are not reported.

How to Debug and Fix

To fix this violation, specify a simulator mode file using the *simulator_file_name* parameter.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal / Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

Param_clockreset06

Reports if the unexpected_ckcells_file and expected_ckcells_file parameters are specified together

When to Use

Use this rule to check if the *unexpected_ckcells_file* and *expected_ckcells_file* parameters are specified together.

Description

The *Param_clockreset06* rule reports a violation if you specify the *unexpected_ckcells_file* and *expected_ckcells_file* parameters together.

Parameter(s)

- *unexpected_ckcells_file*: Default value is `NULL`. Specify a comma or space-separated list of files containing a list of cells that are not allowed in clock trees.
- *expected_ckcells_file*: Default value is `NULL`. Specify a comma or space-separated list of files containing a list of cells that are allowed in clock trees.

Constraint(s)

None

Messages and Suggested Fix

The following message appears if you specify the *unexpected_ckcells_file* and *expected_ckcells_file* parameters together:

```
[WARNING] Parameters 'expected_ckcells_file' and
'unexpected_ckcells_file' are specified together. Parameter
'unexpected_ckcells_file' will be ignored
```

Potential Issues

This violation appears if you specify the *unexpected_ckcells_file* and *expected_ckcells_file* parameters together.

Consequences of Not Fixing

If you do not fix this violation, the *unexpected_ckcells_file* parameter is ignored.

How to Debug and Fix

To fix this violation, specify either *unexpected_ckcells_file* or *expected_ckcells_file* parameter.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non Fatal Must rule

Reports and Related Files

No report or related file

Param_clockreset07

Reports conflicting values specified with the ac_sync_mode parameter

When to Use

Use this rule to perform sanity checks on the *ac_sync_mode* parameter.

Prerequisites

Specify the *ac_sync_mode* parameter.

Description

The *Param_clockreset07* rule reports a violation if the following combination of values is specified with the *ac_sync_mode* parameter:

- `strict_gate` and `soft_gate`
- `strict_qual_logic` and `soft_qual_logic`

Parameter(s)

ac_sync_mode: Default value is `strict_gate,strict_qual_logic`. Other possible values are `soft_gate` and `soft_qual_logic`.

Constraint(s)

None

Messages and Suggested Fix

The following message appears if you specify conflicting values with the *ac_sync_mode* parameter:

`[ERROR] Parameter 'ac_sync_mode': Conflicting values '<value1>' and '<value2>' are specified together`

Potential Issues

This violation message appears if you specify conflicting values with the *ac_sync_mode* parameter.

Consequences of Not Fixing

If you do not fix this violation, default value of this parameter is considered, which may not be as per your expectation.

How to Debug and Fix

Specify any value other than the following combinations:

- `strict_gate` and `soft_gate`
- `strict_qual_logic` and `soft_qual_logic`

Example Code and/or Schematic

Consider the following command specified in a project file:

```
set_parameter ac_sync_mode "soft_gate,strict_gate"
```

For the above example, the *Param_clockreset07* rule reports a violation because the combination of values specified with this parameter is not allowed.

Default Severity Label

Error

Rule Group

None

Reports and Related Files

No report or related file

Propagate_Clocks

Propagates clocks and displays a portion of the clock tree

When to Use

Use this rule to generate high-level information for clock trees.

Prerequisites

Specify clock signals in any of the following ways:

- By using the *clock* or *generated_clock* constraint
- By setting the *use_inferred_clocks* parameter to `yes` to use automatically generated clock signals
- By using a combination of both the above methods

Description

The *Propagate_Clocks* rule reports the following:

- Propagation of synchronous and asynchronous clocks in a design

  View the schematic of the rule to see clocks propagation.

  This information is useful to analyze clock domain crossings, synchronization schemes, and other related checks.

NOTE: *Gated and derived clocks are automatically detected and propagated. However, if the enable_generated_clocks parameter is specified, clocks are not propagated beyond sequential elements.*

- Clocks that are not propagated in the design

NOTE: *Propagation of a clock is blocked if a quasi-static signal is encountered in the path of the clock.*

NOTE: *A black box with single input and output pins is considered as a buffer in clock propagation. Therefore, a clock is propagated from its input pin without requiring any constraint from the user.*

Parameter(s)

- *disable_seq_clock_prop*: Default value is `no`. Set this parameter to `yes` to disable propagation of clocks beyond flip-flops.
- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

- *clock_reduce_pessimism*: Default value is `latch_en`, `mux_sel_derived`, `check_enable_for_glitch`. Set the value of this parameter to `mux_sel` to stop traversal of this rule when a clock signal reaches a MUX. Other possible values are `all`, `all_potential_clocks`, and `ignore_same_domain`.
- *cdc_express*: Default values is `no`. Set this parameter to `peakmem` to reduce peak memory. Other possible value is `yes`.
- *filter_named_clocks*: Default value is `rst`, `reset`, `scan`, `set`. Set this parameter to a list of strings.
- *cdc_reduce_pessimism*: Default value is `mbit_macro`, `no_convergence_at_syncreset`, `no_convergence_at_enable`. Set this parameter to an appropriate value to ignore clock domain crossings involving black box instances and clock domain crossings with destinations having unused, hanging, or blocked outputs. For possible values, see *Allowed Values of the cdc_reduce_pessimism Parameter*.
- *enable_generated_clocks*: Default value is `no`. Set this parameter to `yes` to enable spyglass consider the *generated_clock* constraint.
- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in your design.
- *abstract_port* (Optional and applicable for virtual clocks): Use this constraint to define abstracted information for block ports.
- *input* (Optional and applicable for virtual clocks): Use this constraint to specify clock domain at input ports.

- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.
- *generated_clock* (Optional): Use this constraint to specify generated/ derived clocks.
- *quasi_static*: Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

Message 1

The following message appears to indicate about the propagation of clocks:

[PCLK01] [INFO] For *<du-name>*, clock(s) '*<clock-name>*' of domain '*<domain-name>*' propagated

The arguments of the above message are explained below:

Argument	Description
<du-name>	Module name (for Verilog designs) or the design unit name in *<entity-name>*.*<arch-name>* format (for VHDL designs)
<clock-name>	Name of propagated clock. If multiple clocks are defined in single domain, comma separated list of clock names are displayed.
<domain-name>	Name of clock domain

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule. It gives an idea about clock propagation in a design.

Viewing this information may uncover some bugs in clock specifications or

gating and derived clock logic.

Analyze this data before doing any further analysis.

Message 2

The following message appears to indicate about the propagation of clocks:

[PCLK02] [INFO] For *<du-name>*, virtual clock(s) '*<clock-name>*' of domain '*<domain-name>*' propagated

The arguments of the above message are explained below:

Argument	Description
<du-name>	Module name (for Verilog designs) or the design unit name in *<entity-name>*.*<arch-name>* format (for VHDL designs)
<clock-name>	Name of propagated virtual clock. If multiple virtual clocks are defined in single domain, comma separated list of virtual clock names are displayed.
<domain-name>	Name of clock domain

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule. It gives an idea about clock propagation in a design.

Viewing this information may uncover some bugs in clock specifications or gating and derived clock logic.

Analyze this data before doing any further analysis.

Message 3

The following message appears to indicate about the propagation of clocks:

[PCLK04] [WARNING] For *<du-name>*, clock(s) '*<clock-name>*' of domain '*<domain-name>*' not propagated

The arguments of the above message are explained below:

Argument	Description
<du-name>	Module name (for Verilog designs) or the design unit name in *<entity-name>*.*<arch-name>* format (for VHDL designs)
<clock-name>	Name of the clock that did not propagate. If multiple clocks are defined in a single domain, a comma-separated list of clock names is shown.
<domain-name>	Name of clock domain

Potential Issues

This violation appears if clocks get blocked because of constant nets in the path before driving the clock pin of a flip-flop. In addition, the violation is reported if a quasi-static signal is detected and therefore the propagation of the clock is stopped.

Consequences of Not Fixing

If you do not fix this violation, the following can be the consequences:

- Clock domains are not assigned properly to the design resulting in missed SpyGlass CDC violations.
- It may produce a static logic in the design resulting in incorrect simulation results.
- It may affect analysis of lower-level blocks in the SoC flow.

How to Debug and Fix

To fix this violation, analyze and remove the blocked nets in the clock path to ensure correct propagation of clock signals.

No schematic is generated for this violation. When you double-click on the

violation, the line in the SGDC file is highlighted where this clock is defined.

Message 4

The following message appears to indicate about the propagation of clocks:

[PCLK03] [WARNING] For *<du-name>*, virtual clock(s) '*<clock-name>*' of domain '*<domain-name>*' not propagated

The arguments of the above message are explained below:

Argument	Description
<du-name>	Module name (for Verilog designs) or the design unit name in *<entity-name>*.*<arch-name>* format (for VHDL designs)
<clock-name>	Name of the virtual clock that did not propagate. If multiple virtual clocks are defined in a single domain, a comma-separated list of clock names is shown.
<domain-name>	Name of clock domain

Potential Issues

This violation appears if you missed to assign a clock domain to input ports specified in an SGDC file.

Consequences of Not Fixing

If you do not fix this violation, the following can be the consequences:

- Clock domains are not assigned properly to the design resulting in missed SpyGlass CDC violations.
- It may produce a static logic in the design resulting in incorrect simulation results.
- It may affect analysis of lower-level blocks in the SoC flow.

How to Debug and Fix

To fix this violation, assign *abstract_port* or *input* constraint to the input ports to ensure propagation of virtual clocks in a design.

Example Code and/or Schematic

Example 1

Consider the following schematic of a message of this rule:

```
// test.v
module top(d1, d2, clk, out);
  input d1, clk, d2;
  output out;
  reg der_clk;
  reg out;
  always@(posedge clk)
    der_clk <= d1;
  always@(posedge der_clk)
    out <= d2;
endmodule
```

```
// constr.sgdc
current_design top
clock -name "top.clk" -value rtz
```

For the above example, when you run the *Propagate_Clocks* rule, the clocks are propagated in the design. In addition, the schematic shows the clock propagation.

The following figure shows the schematic in this case:

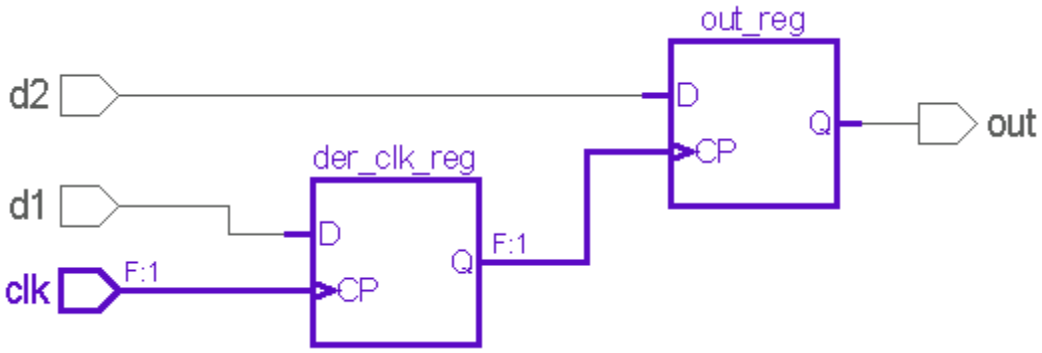


FIGURE 422. Schematic of the Propagate_Clocks Rule Violation

Schematic Highlight

This rule highlights a single sequential element at each logic level in the path of each primary, black box, and derived clock.

NOTE: *In case of undriven source clock, complete source net is highlighted.*

The schematic uses the following naming conventions:

Notation Symbol	Represents
F	Flip-flop
B	Black box
P	Port
L	Latch
C	Library cell

NOTE: *Only a single sequential element is highlighted at each logic level in the path of each primary, black box, and derived clock in the schem1atic. This avoids schematic clutter and is sufficient to show that each clock signal is actually being used as a clock in the design.*

For virtual clocks, the schematic highlights input ports or terminals for which the *abstract_port* and *input* constraints are defined.

Example 2

Consider the following design:

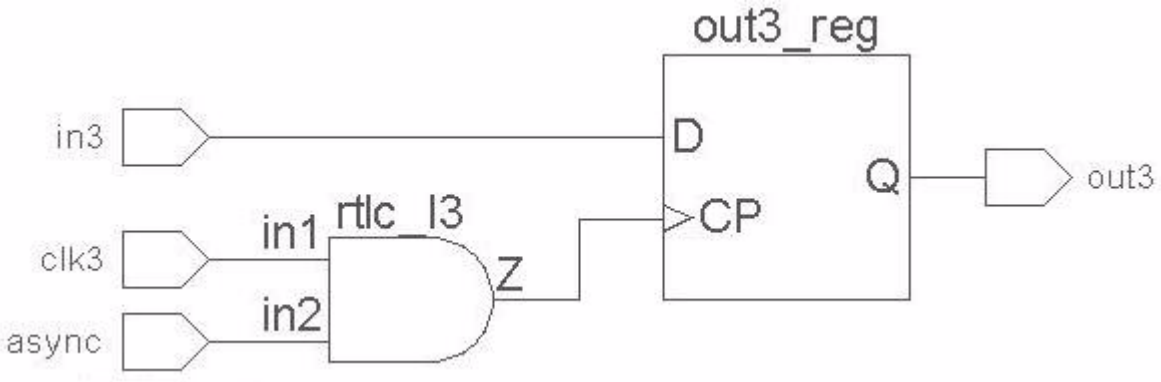


FIGURE 423. Example of the Propagate_Clocks rule

For the above design, the *Propagate_Clocks* rule reports a warning if `async` gets tied to ground in the upper block or you do not assign a proper

set_case_analysis value while changing the mode.

`clk3` will not propagate in such scenarios and the `out3` flip-flop is ignored from SpyGlass CDC checks.

Default Severity Label

Info

Rule Group

`PREREQ`

Reports and Related Files

No report or related file

Propagate_Resets

Propagates resets and displays a portion of the reset tree

When to Use

Use this rule to check reset propagation in a design.

Prerequisites

Specify reset signals in any of the following ways:

- By using the *reset* constraint
- By using the automatically generated resets after setting the *use_inferred_resets* parameter to `yes`
- By using a combination of both the above methods

Description

The *Propagate_Resets* rule reports propagation of asynchronous reset signals in a design.

This information is useful in reset synchronization and deassertion related checks.

NOTE: *Propagation of a reset stops if another reset comes in its path at that point.*

Rule Exceptions

This rule does not report synchronous resets.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.
- *filter_named_resets*: Default value is `clk`, `clock`, `scan`. Specify a list of strings to automatically infer asynchronous resets that do not match the specified strings.
- *reset_reduce_pessimism*: Default value is `filter_unused_synchronizer`, `same_data_reset_flop`. Set the value of this parameter to `all_potential_resets` to detect all the potential resets in a design. For information on the other possible values, see *Possible Values to the reset_reduce_pessimism Parameter*.

- *handle_combo_arc*: Default value is `no`. Set this parameter to `yes` so that the clock/reset propagates from an input pin of a sequential library cell if a combinational timing arc is specified from that pin to any output pin of the cell.
- *ignore_bus_resets*: Default value is `yes`. Set this parameter to `no` to generate reset vector nets, which are not struct nets, in the *autoresets.sgdc* and the `generated_resets.sgdc` file.
- *enable_derived_reset*: Specifies if resets are propagated through derived resets.
- *fa_hybrid_report_hier*: Default value is `no`. Set the value of the parameter to `yes` to enable the supported rules to report the top-level hierarchical names in the SVA Hybrid flow.

Constraint(s)

- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *set_case_analysis* (Optional): Use this constraint to specify case analysis conditions.

Messages and Suggested Fix

Message 1

The following message appears for the propagated reset:

`[INFO] For <du-name>, reset '<reset-name>' propagated`

NOTE: *If a reset is not propagated, Message 2 is reported.*

The arguments of the above message are explained below:

Argument	Description
<du-name>	Module name (for Verilog designs) or the design unit name in *<entity-name>.<arch-name>* format (for VHDL designs)
<reset-name>	Name of the reset

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

This is an informational rule and gives an idea about the propagation of resets.

Viewing reset propagation might uncover some bugs in the reset specifications or their usage.

Analyze this data before performing any further analysis.

Message 2

The following message appears for the reset that is not propagated:

```
[WARNING] For <du-name>, reset '<reset-name>' not propagated
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

The rule generates a Warning message for the resets that are not propagated.

Viewing reset propagation might uncover some bugs in the reset specifications or their usage.

Analyze this data before performing any further analysis.

Example Code and/or Schematic

Consider the following design file specified for SpyGlass analysis:

```
module test( input clk, rst1, rst2, d1, d2, output reg q1, q2);
wire rst, a;

always @(posedge clk or negedge rst1)
  if (!rst1) q1 <= 1'b0;
  else q1 <= d1;
AN2 u1 (rst2, 1'b0, rst);
always @(posedge clk or posedge rst)
  if (rst)   q2 <= 1'b0;
  else   q2 <= d2;
```

In the above example, the `rst1` reset is propagated. However, the `rst2` reset is not propagated because it is blocked at the AND gate.

The following schematic shows the propagation of the `rst1` reset:

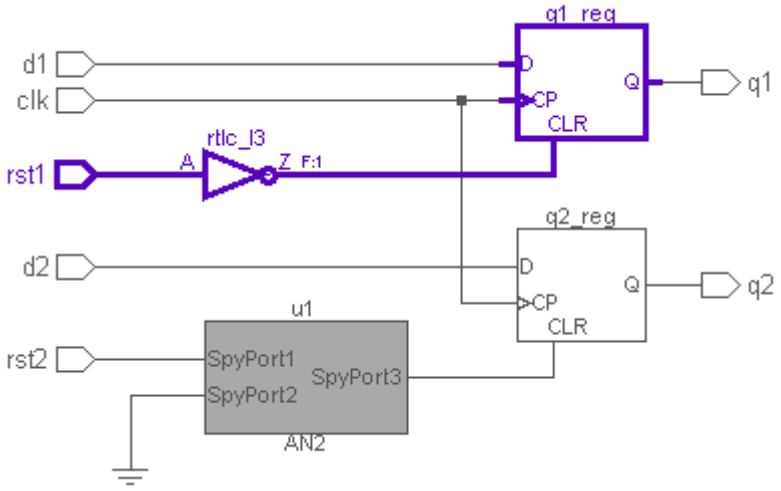


FIGURE 424. Schematic of the Propagate_Resets Rule Violation

Schematic Highlight

This rule highlights a single sequential element at each logic level in the path of each primary and a black box reset, which is propagated.

No schematic information provided for the resets that are not propagated.

The schematic uses the following naming conventions:

Notation Symbol	Represents
F	Flop
B	Black box
P	Port
L	Latch

NOTE: *Only a single sequential element is highlighted at each logic level in the path of each primary and black box clock in the schematic. This avoids schematic clutter and is sufficient to show that each reset signal is actually being used as a reset in the design.*

Default Severity Label

Warning

Rule Group

PREREQ

Reports and Related Files

The Clock-Reset-Summary Report

QualifierSetup

Reports if the qualifier constraint does not synchronize any clock domain crossing in a design

When to Use

Use this rule for qualifier setup check.

Prerequisites

Specify the *qualifier* constraint.

Description

The *QualifierSetup* rule reports a violation if the *qualifier* constraint does not synchronize any crossing in a design.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

Message 1

The following message appears when the *qualifier* constraint does not synchronize any crossing in the current design *<current-design>*:

```
[QS_1] [WARNING] 'qualifier' constraint '<qualifier-name>' is not used to synchronize any crossing in the current design '<current-design>'
```

Potential Issues

This violation appears if your design contains a qualifier signal, specified by the *qualifier* constraint, which does not synchronize any crossing in the design.

Consequences of Not Fixing

If you do not fix this violation, the specified qualifier is ignored.

How to Debug and Fix

Specify a correct qualifier signal in the *qualifier* constraint.

Message 2

The following message appears when the *qualifier* constraint does not ignore any inferred qualifier in the current design *<current-design>*:

```
[QS_2] [WARNING] 'qualifier' constraint '<qualifier-name>' is
not used to ignore any inferred synchronizer in the current
design '<current-design>'
```

Potential Issues

This violation appears if your design contains a qualifier signal, specified by the *qualifier* constraint, which does not ignore any inferred qualifier in the design.

Consequences of Not Fixing

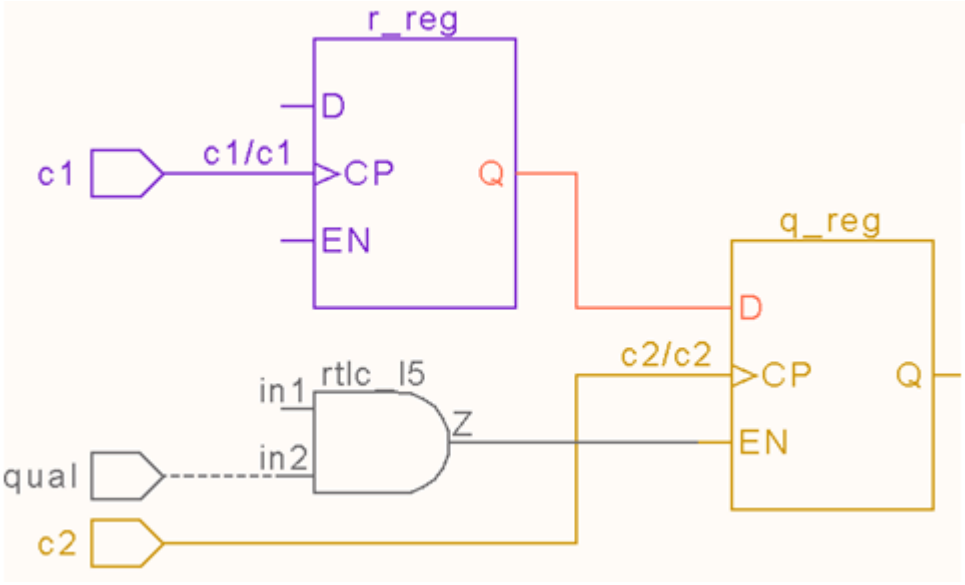
If you do not fix this violation, the specified qualifier is ignored.

How to Debug and Fix

Specify a correct qualifier signal in the *qualifier* constraint.

Example Code and/or Schematic

Consider the following schematic of a design:




```
// constr.sgdc
current_design top
clock -name c1 -domain d1
clock -name c2 -domain d2
qualifier -name qual -from_clk c2 -to_clk c1
```

FIGURE 425. Schematic of the QualifierSetup Rule Violation

In the above design, the `qual` qualifier is used for the clock domain crossing from `c1` to `c2`. However, `qual` is defined in the *qualifier* constraint from `c2` to `c1`.

Therefore, the *QualifierSetup* rule reports a violation in this case. As a result, the `qual` qualifier is not considered by SpyGlass analysis and therefore, it does not synchronize any crossing in the design.

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No related files or reports

ResetSynchronizerSetup

Reset_synchronizer mentioned as -ignore is not used to ignore inferred synchronization

When to Use

Use this rule for reset synchronizer setup check.

Prerequisites

Specify the reset_synchronizer constraint.

Description

The ResetSynchronizerSetup rule reports a violation if the *reset_synchronizer* constraint with `-ignore` argument is not used to ignore any inferred or user-defined synchronizer.

Parameter(s)

None

Constraint(s)

reset_synchronizer (Mandatory): Use this constraint to specify a reset synchronizer for synchronizing a reset domain crossing.

Messages and Suggested Fix

The following message appears when the *reset_synchronizer* constraint does not ignore any inferred or user-defined reset synchronizer in the current design <current-design>:

```
[RS_2] [WARNING] 'Reset_synchronizer' constraint '<object-name>' is not used to ignore any inferred synchronizer in the current design '<current-design>'
```

Potential Issues

This violation appears if your design contains a reset synchronizer, specified by the *reset_synchronizer* constraint, which does not ignore any inferred or user-defined reset synchronizer in the design.

Consequences of Not Fixing

If you do not fix this violation, the specified reset synchronizer is ignored.

How to Debug and Fix

Specify a correct reset synchronizer in the reset_synchronizer constraint.

Example Code and/or Schematic

Consider the following schematic of a design:

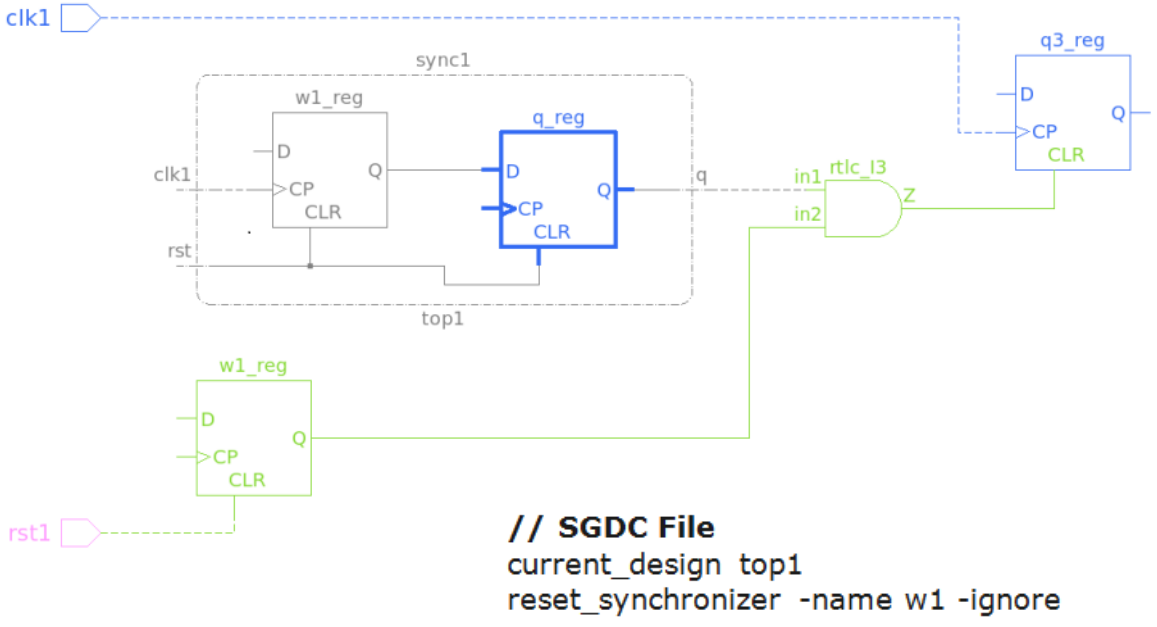


FIGURE 426. Schematic for the ResetSynchronizerSetup Rule Violation

In the above design, the reset_synchronizer - ignore constraint is defined at sync1.w1. The ResetSynchronizerSetup rule reports a warning message in this case because there is no valid synchronizer before the sync1.w1 net.

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No related files or reports

Reset_check08

Reports reset signals that are constrained by using the set_case_analysis constraint

When to Use

Use this rule during the RTL or pre-layout phase to detect reset signals that are constrained by using the *set_case_analysis* constraint.

Prerequisites

Perform the following actions before running this rule:

- Specify reset signals by using the *set_case_analysis* constraint
- Run the *Reset_check03* or *Reset_check04* rule.

Description

The *Reset_check08* rule reports reset signals that are specified by the `-name` argument of the *set_case_analysis* constraint.

NOTE: *This rule is run automatically when the Reset_check03 or Reset_check04 rules are run, and reset signals are constrained by using the set_case_analysis constraint.*

Parameter(s)

None

Constraint(s)

- *set_case_analysis* (Mandatory): Use this constraint to specify case analysis conditions.
- *clock* (Mandatory): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the reset signal *<signal-name>* is specified by the *set_case_analysis* constraint:

```
[WARNING] Reset signal '<signal-name>' also specified as
'set_case_analysis', hence not checked by rule <rule-name>
```

Potential Issues

This violation appears if your design contains reset signals that are specified by the *set_case_analysis* constraint.

Consequences of Not Fixing

If you do not fix this violation, reset signals get constrained as static nets, which may impact design functionality.

How to Debug and Fix

To fix this violation, review the *set_case_analysis* constraint applied on the reset signal and remove this constraint or the reset definition.

If the *set_case_analysis* constraint is intentional (for example, to configure the design in a particular mode), disable or waive this rule.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v
module test( input clk, rst1, srst, d1, d2, d3, output reg q1, q2, q3);
always @(posedge clk or negedge rst1)
  if (!rst1) q1 <= 1'b0;
  else q1 <= d1;
always @(posedge clk)
  if (rst1) q2 <= 1'b0;
  else q2 <= d2;
always @(posedge clk)
  if (srst) q3 <= 1'b0;
  else q3 <= d3;
endmodule

// constr.sgdc
current_design test
reset -name rst1
reset -name srst -sync
set_case_analysis -name srst -value 1
```

For the above example, the *Reset_check08* rule reports a violation because the `srst` reset signal is specified by the *set_case_analysis* constraint.

Default Severity Label

Warning

Rule Group

`VERIFY`

Reports and Related Files

No reports or related file

SGDC_allow_combo_logic01

Reports if no argument is specified in the allow_combo_logic constraint

When to Use

Use this rule to perform sanity checks on the *allow_combo_logic* constraint specification.

Prerequisites

Use the *allow_combo_logic* constraint.

Description

The *SGDC_allow_combo_logic01* rule reports a violation if you do not specify any argument with the *allow_combo_logic* constraint.

Parameter(s)

None

Constraint(s)

allow_combo_logic (Mandatory): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.

Messages and Suggested Fix

The following message appears when no argument is specified in the *allow_combo_logic* constraint:

```
[FATAL] 'allow_combo_logic': No field specified in the constraint
```

Potential Issues

This violation appears if no argument is specified with the *allow_combo_logic* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the correct syntax for the *allow_combo_logic* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal Must Rules

Reports and Related Files

No reports or related files

SGDC_allow_combo_logic02

Reports if different arguments are used in different specifications of the allow_combo_logic constraint

When to Use

Use this rule to perform sanity checks on the *allow_combo_logic* constraint specification.

Prerequisites

Use the *allow_combo_logic* constraint.

Description

The *SGDC_allow_combo_logic02* rule reports a violation if different arguments, such as `-all`, `-none`, and/or `-name` are specified in different specifications of the *allow_combo_logic* constraint.

Parameter(s)

None

Constraint(s)

allow_combo_logic (Mandatory): Use this constraint to allow combinational logic between crossings only if the logic is within the modules specified by using this constraint.

Messages and Suggested Fix

The following message appears if different arguments, such as `-all`, `-none`, and/or `-name` are specified in different specifications of the *allow_combo_logic* constraint

```
[WARNING] Combination of all, none or specific names has been used in allow_combo_logic constraint
```

Potential Issues

This violation appears when you specify multiple specifications of the *allow_combo_logic* constraint with different arguments.

Consequences of Not Fixing

If the *allow_combo_logic* constraint is specified multiple times, the following order of preference, starting from the highest priority, is applied which may differ from your requirement:

1. `allow_combo_logic -all`
2. `allow_combo_logic -none`
3. `allow_combo_logic -name <list>`

How to Debug and Fix

To fix the violation, remove multiple *allow_combo_logic* constraint specifications containing conflicting arguments.

Example Code and/or Schematic

Consider the following *allow_combo_logic* constraint specifications in an SGDC file:

```
allow_combo_logic -all
allow_combo_logic -none
```

In this case, the *SGDC_allow_combo_logic02* rule reports a violation and only the following specification is considered:

```
allow_combo_logic -all
```

Default Severity Label

Warning

Rule Group

Non-Fatal Must Rules

Reports and Related Files

No reports or related files

SGDC_cdc_false_path01

Reports a violation if non-existent objects are specified in the -from argument of the cdc_false_path constraint

When to Use

Use this rule to perform sanity checks on the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path01* rule reports a violation if the object specified in the `-from` argument of the *cdc_false_path* constraint does not exist as any of following object in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	Clock tag

Parameter(s)

None

Constraint(s)

- *cdc_false_path* (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *clock* `-tag` (Optional): Use this constraint to specify a logical clock name.

Messages and Suggested Fix

The following message appears if you specify non-existent objects in the `-from` argument of the *cdc_false_path* constraint:

[FATAL] '<name>'[TopPort + SubModulePort + Net + HierTerminal + SubModule] not found on/within module '<module-name>'

Potential Issues

This violation appears if the object specified in the `-from` argument of *cdc_false_path* constraint does not exist as any of following object in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	Clock tag

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing object, such as a top-level port, net, terminal, or module in the `-from` argument of the *cdc_false_path* constraint.

Example Code and/or Schematic

Consider an example in which the design `top` does not contain any object by the name `ck1`.

In this case, the *SGDC_cdc_false_path01* rule reports a violation if you specify the following *cdc_false_path* constraint in the SGDC file:

```
current_design top
clock -name ck2 -domain d2
cdc_false_path -from ck1 -to ck2
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path02

Reports a violation if non-existent objects are specified in the -to argument of the cdc_false_path constraint

When to Use

Use this rule to perform sanity checks on the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path02* rule reports a violation if the object specified in the `-to` argument of the *cdc_false_path* constraint does not exist as any of following object in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	Clock tag

Parameter(s)

None

Constraint(s)

- *cdc_false_path* (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.
- *clock* `-tag` (Optional): Use this constraint to specify a logical clock name.

Messages and Suggested Fix

The following message appears if you specify non-existent objects in the `-to` argument of the *cdc_false_path* constraint:

```
[FATAL] '<name>'[TopPort + SubModulePort + Net + HierTerminal + SubModule] not found on/within module '<module-name>'
```

Potential Issues

This violation appears if the object specified in the `-to` argument of *cdc_false_path* constraint does not exist as any of following object in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	Clock tag

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing object, such as a top-level port, net, terminal, or module in the `-to` argument of the *cdc_false_path* constraint.

Example Code and/or Schematic

Consider an example in which the design `top` does not contain any object by the name `ck2`.

In this case, the *SGDC_cdc_false_path02* rule reports a violation if you specify the following *cdc_false_path* constraint in the SGDC file:

```
current_design top
clock -name ck1 -domain d2
cdc_false_path -from ck1 -to ck2
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path03

Reports a violation if non-existent objects are specified in the -through argument of the cdc_false_path constraint

When to Use

Use this rule to perform sanity checks on the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path03* rule reports a violation if the object specified in the `-through` argument of the *cdc_false_path* constraint does not exist as any of following object in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

The following message appears if you specify non-existent objects in the `-through` argument of the *cdc_false_path* constraint:

```
[FATAL] '<name>'[TopPort + SubModulePort + Net + HierTerminal + SubModule] not found on/within module '<module-name>'
```

Potential Issues

This violation appears if the object specified in the `-through` argument of

cdc_false_path constraint does not exist as any of following object in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing object, such as a top-level port, net, terminal, or module in the `-through` argument of the *cdc_false_path* constraint.

Example Code and/or Schematic

Consider an example in which the design `top` does not contain any object by the name `n1`.

In this case, the *SGDC_cdc_false_path03* rule reports a violation if you specify the following *cdc_false_path* constraint in the SGDC file:

```
current_design top
cdc_false_path -through top.n1
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path04

Reports a violation if wildcard names specified in the -from, -to, or -through argument of the cdc_false_path constraint does not match with the name of any object in a design

When to Use

Use this rule to perform sanity checks on the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path04* rule reports a violation if wildcard names specified in the `-from`, `-to`, or `-through` argument of the *cdc_false_path* constraint does not match with the names of any of following objects in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	

NOTE: *Non-wildcard names are checked by the SGDC_cdc_false_path01, SGDC_cdc_false_path02, and SGDC_cdc_false_path03 rules.*

Rule Exceptions

This rule does not consider wildcard names for hierarchical pins and intermediate nets. Therefore, it reports false violations if you specify wildcard names for such pins or nets.

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

The following message appears if wildcard names specified in the `-from`, `-to`, or `-through` argument of the *cdc_false_path* constraint does not match with the name of any object in the design:

```
[ERROR] "<object-name>" specified in the "<argument-name>" field of cdc_false_path constraint could not be found in the design
```

Potential Issues

This violation appears if wildcard names specified in the `-from`, `-to`, or `-through` argument of the *cdc_false_path* constraint does not match with the name of any of the following types of objects in the design:

Top level port	Net	Hierarchical Terminal
Module	Module port	

Consequences of Not Fixing

If you do not fix this violation, the reported *cdc_false_path* constraint specification is ignored and no crossing is filtered from this constraint.

How to Debug and Fix

To fix this violation, specify correct wildcard names in the `-from`, `-to`, or `-through` argument of the *cdc_false_path* constraint so that they match with the names of existing objects in the design.

Example Code and/or Schematic

Consider the design `top` in which the name of none of the object starts with the string `ck`.

In this case, the *SGDC_cdc_false_path04* rule reports a violation if you specify the following *cdc_false_path* constraint in the SGDC file:

```
current_design top
cdc_false_path -from "ck*"
```

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path05

Reports a violation if no argument is specified with the cdc_false_path constraint

When to Use

Use this rule to perform sanity checks on the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path05* rule reports a violation if no argument is specified with the *cdc_false_path* constraint.

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

The following message appears if no argument is specified with the *cdc_false_path* constraint:

```
[WARNING] 'cdc_false_path': No field specified in the constraint
```

Potential Issues

This violation appears if no argument is specified with the *cdc_false_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored during

SpyGlass analysis.

How to Debug and Fix

To fix this violation, perform any of the following actions:

- Remove the reported constraint.
- Specify at least one argument with this constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path06

Checks type mismatch in the arguments of the cdc_false_path constraint

When to Use

Use this rule to perform sanity checks on the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path06* rule reports a violation if the signal type specified in the `-from_type` or `-to_type` arguments of the *cdc_false_path* constraint does not match with the signal specified in the `-from` or `-to` arguments, respectively, of this constraint.

The rule also reports a violation if the virtual clocks are not specified properly in the `-from` and `-to` arguments of the *cdc_false_path* constraint.

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

Message 1

The following message appears if there is a type mismatch in the arguments of the *cdc_false_path* constraint:

```
[ERROR] Type specified in the "<-from_type | -to_type>" field
does not match with the signal "<signal-name>" specified in the
"<-from | -to>" field of cdc_false_path constraint
```

Potential Issues

This violation appears if the signal type specified in the `-from_type` or `-to_type` arguments of the *cdc_false_path* constraint does not match with the type of the signal specified in the `-from` or `-to` arguments, respectively.

Consequences of Not Fixing

If you do not fix this violation, the reported *cdc_false_path* constraint specification is ignored and no crossing is filtered from this constraint.

How to Debug and Fix

To fix this violation, specify the correct type in the `-from_type` or `-to_type` arguments of the *cdc_false_path* constraint.

Message 2

The following message appears if a virtual clock is specified in either of the `-from` or `-to` arguments of the *cdc_false_path* constraint and a non-clock object is specified in the other argument:

```
[ERROR] Virtual clock specified in <field_name> field and non-clock specified in <field_name> field. Ignoring the constraint
```

Potential Issues

This violation appears if a virtual clock is specified in either of the `-from` or `-to` arguments of the *cdc_false_path* constraint and a non-clock object is specified in the other argument.

Consequences of Not Fixing

If you do not fix this violation, the reported *cdc_false_path* constraint specification is ignored and no crossing is filtered from this constraint.

How to Debug and Fix

To fix this violation, ensure that if a virtual clock is specified in the `-from` or the `-to` arguments of the *cdc_false_path* constraint, a clock is specified for the other argument as well.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

// test.v

```
module top(input d,clk1,clk2,clk3,output q1,q2);
  reg q1,q2;
  wire w1,tmp1,tmp2;
  assign merg_clk = clk1 & clk2;
  block1 bb1 (d,clk1,tmp1);
  block2 bb2 (tmp1,clk2, tmp2);
endmodule

module block1(input in,clk1,output reg out);
  always@(posedge clk1)
      out<=in;
endmodule

module block2(input in,clk,output reg out);
    always@(posedge clk)
        out<=in;
endmodule
```

// test.sgdc

```
current_design top
clock -name clk1
clock -name clk2
clock -name clk3

cdc_false_path -from clk1 clk2
  -to clk2 -from_type data
  -to_type data
```

For the above example, the *SGDC_cdc_false_path06* rule reports violations for the `-from_type` and `-to_type` arguments of the *cdc_false_path* constraint because the types specified in these arguments do not match with the type of signals specified in the `-from` and `-to` arguments, respectively.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path07

Checks for existence of the -from_type and -to_type arguments of the cdc_false_path constraint

When to Use

Use this rule to check for the existence of the `from_type` and `to_type` arguments of the *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path07* rule reports a violation if the `-to` or the `-from` arguments of the *cdc_false_path* constraint is specified without specifying the corresponding `-from_type` or the `-to_type` arguments constraint.

For example, if the `-to` type argument of the constraint is specified without the `-to_type` argument, the rule reports a violation.

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

The following message appears if the `-from_type` or the `-to_type` arguments are missing for a *cdc_false_path* constraint:

```
[ERROR] From_type or to_type are missing for at least one cdc_false_path constraint
```

Potential Issues

This violation appears if the `-from` or the `-to` arguments of the

cdc_false_path constraint are specified without the corresponding `-from_type` or `-to_type` arguments.

Consequences of Not Fixing

If you do not fix this violation, the `-from_type` or `-to_type` arguments of the *cdc_false_path* constraint uses the default value '`all`', which increases the analysis effort and run time.

How to Debug and Fix

To fix this violation, specify the `-from_type` and the `-to_type` arguments corresponding to the `-from` and `-to` arguments of the *cdc_false_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Error

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path08

Reports if a terminal is specified with -from or -to field and connected net of that terminal connects to multiple possible sources or destinations

When to Use

Use this rule to check whether unexpected crossings have been ignored due to *cdc_false_path* constraint.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

SpyGlass CDC uses connected nets of terminals for internal processing. If these nets connect to multiple destinations/sources, additional crossings may be ignored. The *SGDC_cdc_false_path08* rule reports violations if more than one possible source or destination is connected to these nets.

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

The following message appears if the connected net of a terminal connects to more than one possible sources or destinations:

```
[Warning] Connected net <net-name> of terminal <terminal-name>
specified with <field-name> field may connect to multiple
<sources/destinations>
```

Potential Issues

This violation appears if multiple possible crossings are ignored by the *cdc_false_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, many important violations might not be reported because the *cdc_false_path* constraint impacts many rules.

How to Debug and Fix

To fix this violation, check if additional crossings are ignored by the *cdc_false_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_cdc_false_path09

Reports if objects specified with fields -from_obj/to_obj are not driven by clocks specified with -from_clk/-to_clk fields

When to Use

Use this rule to check if objects specified with arguments `-from_obj/-to_obj` are not driven by clocks specified with `-from_clk/-to_clk` arguments.

Prerequisites

Specify the *cdc_false_path* constraint.

Description

The *SGDC_cdc_false_path09* rule reports violations if objects specified with the `-from_obj/-to_obj` arguments of the *cdc_false_path* constraint are not driven by clocks specified in the `-from_clk/-to_clk` arguments.

Parameter(s)

None

Constraint(s)

cdc_false_path (Mandatory): Use this constraint to specify false paths so that clock-domain crossings along these paths are ignored for rule checking.

Messages and Suggested Fix

The following message appears if the objects specified with the `-from_obj/to_obj` fields are not driven by clocks specified with the `-from_clk/-to_clk` fields:

```
[Warning] Objects <object name> specified with <from_obj |-
to_obj> field are not driven by clocks <clock name> specified
with <-from_clk | -to_obj> field
```

Potential Issues

This violation appears if objects specified with fields `-from_obj/-to_obj`

are not driven by clocks specified with `-from_clk`/`-to_clk` arguments of the *cdc_false_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the *cdc_false_path* constraint.

How to Debug and Fix

To fix this violation, check the `-from_obj`/`-to_obj` and the `-from_clk`/`-to_clk` arguments of the *cdc_false_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_clockreset02

Reports a violation if an invalid clock is specified in the -clock argument of the input or output constraint

When to Use

Use this rule to perform sanity checks on the *input* or *output* constraint.

Prerequisites

Specify the *input* or *output* constraint.

Description

The *SGDC_clockreset02* rule reports a violation if the clock specified by the `-clock` argument of the *input* or *output* constraint for ports is not specified by any *clock* constraint.

Parameter(s)

None

Constraint(s)

- *input* (Mandatory): Use this constraint to specify clock domain at input ports.
- *output* (Mandatory): Use this constraint to specify a clock domain at output ports.

Messages and Suggested Fix

Message 1

The following message appears if the signal *<signal-name>* specified by the `-clock` argument of the *input* constraint for ports *<port-names>* is not specified by the *clock* constraint:

```
[SClkRst2_3] [WARNING] Signal <signal-name> in -clock field in
'input' constraint for port(s) <port-names> has not been
specified as a clock via 'clock' constraint
```

Potential Issues

This violation appears if the clock specified by the `-clock` argument of the *input* constraint is not a valid clock.

Consequences of Not Fixing

If you do not fix this violation, clock-domain crossing analysis does not happen for the reported ports.

How to Debug and Fix

To fix this violation, update the `-clock` argument of the reported *input* constraint to specify a clock specified by the *clock* constraint.

Message 2

The following message appears if the signal *<signal-name>* specified by the `-clock` argument of the *output* constraint for ports *<port-names>* is not specified by the *clock* constraint:

```
[SClkRst2_1] [WARNING] Signal <signal-name> in -clock field in
'output' constraint for port(s) <port-names> has not been
specified as a clock via 'clock' constraint
```

Potential Issues

This violation appears if the clock specified by the `-clock` argument of the *output* constraint is not a valid clock.

Consequences of Not Fixing

If you do not fix this violation, clock-domain crossing analysis does not happen for the reported ports.

How to Debug and Fix

To fix this violation, update the `-clock` argument of the reported *output* constraint to specify a clock specified by the *clock* constraint.

Message 3

The following message appears if the `-clock` argument of the *output* constraint is not specified for the ports *<port-names>*:

```
[SClkRst2_2] [WARNING] '-clock' field is not specified in
'output' constraint for port(s) <port-names>
```

Potential Issues

This violation appears if no `-clock` argument of the *output* constraint is specified for some ports.

Consequences of Not Fixing

If you do not fix this violation, clock-domain crossing analysis does not happen for the reported ports.

How to Debug and Fix

To fix this violation, specify the `-clock` argument of the reported ports of the *output* constraint.

Example Code and/or Schematic

Consider following constraints specified in an SGDC file:

```
current_design top

clock -name CK1 -domain d1
clock -name CK2 -domain d2
output -name P1 -clock CC
```

In the above example, the `CC` clock specified by the `-clock` argument of the *output* constraint is not specified by any *clock* constraint.

Therefore, the *SGDC_clockreset02* rule reports a violation.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_clocksense01

Reports for an incorrect value in the -pins argument of the clock_sense constraint

When to Use

Use this rule to perform sanity checks on the *clock_sense* constraint.

Prerequisites

Specify the *clock_sense* constraint.

Description

The *SGDC_clocksense01* rule reports a violation if a non existing terminal is specified in the `-pins` argument of the *clock_sense* constraint.

Parameter(s)

None

Constraint(s)

- *clock_sense* (Mandatory): Use this constraint to stop propagation of clocks from the specified pins.

Messages and Suggested Fix

The following message appears if a non existing terminal is specified in the `-pins` argument of the *clock_sense* constraint:

```
[FATAL] Constraint 'clock_sense':
'<terminal-name>'[HierTerminal] not found on/within module
'<module-name>'
```

Potential Issues

This violation appears if a non existing terminal is specified in the `-pins` argument of the *clock_sense* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a valid terminal name in the `-pins` argument of the *clock_sense* constraint.

Example Code and/or Schematic

Consider the following files specified in SpyGlass analysis:

// top.v

```
module top(in, clk, out);
input in, clk;
output out;
reg out;
BBOX b1(clk, clk_out);
always@(posedge clk_out)
    out <= in;
endmodule
module BBOX(input in, output out);
endmodule
```

// constr.sgdc

```
current_design top
clock -name clk -domain d1
clock_sense -pins b2.in
```

For the above example, the *SGDC_clocksense01* rule reports a violation because the `b2.in` terminal is not present in the `top` module.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_clocksense02

Reports for the -tag argument of the clock_sense constraint if the tag is not associated with a real clock

When to Use

Use this rule to perform sanity checks on the *clock_sense* constraint.

Prerequisites

Specify the *clock_sense* constraint.

Description

The *SGDC_clocksense02* rule reports a violation if the clock tag specified in the `-tag` argument of the *clock_sense* constraint is not associated with any real clock in the design.

Parameter(s)

None

Constraint(s)

- *clock_sense* (Mandatory): Use this constraint to stop propagation of clocks from the specified pins.

Messages and Suggested Fix

The following message appears for an invalid tag specified in the `-tag` argument of the *clock_sense* constraint:

```
[WARNING] Constraint 'clock_sense': Clock name '<tag-name>' specified in field '-tag' is not a valid clock tag
```

Potential Issues

This violation appears if the clock tag specified in the `-tag` argument of the *clock_sense* constraint is not associated with any real clock in the design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint specification.

How to Debug and Fix

To fix this violation, update the `-tag` argument to specify valid tags that are associated with real clocks in the design.

Example Code and/or Schematic

Consider the following constraints specification:

```
current_design top
clock -name clk -domain d1
clock_sense -pins b1.in -tag C1
```

For the above example, the *SGDC_clocksense02* rule reports a violation because the `C1` tag is not associated with any clock in the design.

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_clocksense03

Reports if a virtual clock is specified in the -tag argument of the clock_sense constraint

When to Use

Use this rule to perform sanity checks on the *clock_sense* constraint.

Prerequisites

Specify the *clock_sense* constraint.

Description

The *SGDC_clocksense03* rule reports a violation if a virtual clock is specified in the `-tag` argument of the *clock_sense* constraint.

Parameter(s)

None

Constraint(s)

- *clock_sense* (Mandatory): Use this constraint to stop propagation of clocks from the specified pins.

Messages and Suggested Fix

The following message appears if a virtual clock is specified in the -tag argument of the *clock_sense* constraint:

```
[WARNING] Constraint 'clock_sense': Clock Name '<clk-name>'
specified in field '-tag' is virtual clock'
```

Potential Issues

This violation appears if a virtual clock is specified in the `-tag` argument of the *clock_sense* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint

specification.

How to Debug and Fix

To fix this violation, update the `-tag` argument to specify a tag that is associated with a real clock in the design.

Example Code and/or Schematic

Consider the following constraints specification:

```
current_design top
clock -name clk -domain d1
clock -tag C2
clock_sense -pins b1.in -tag C2
```

For the above example, the *SGDC_clocksense03* rule reports a violation for the `C2` tag because `C2` is a virtual clock.

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_define_reset_order01

Reports a violation if an invalid object is specified in the -from argument of the define_reset_order constraint

When to Use

Use this rule to perform sanity checks on the *define_reset_order* constraint.

Prerequisites

Specify the *define_reset_order* and *reset* constraints.

Description

The *SGDC_define_reset_order01* rule reports a violation if the object specified in the `-from` argument of the *define_reset_order* constraint does not exist as port, hierarchical terminal, or net in the current design.

Parameter(s)

None

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Mandatory): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if an invalid object is specified in the `-from` argument of the *define_reset_order* constraint:

```
[FATAL] Constraint 'define_reset_order': '<signal-name>'
[TopPort + Net + HierTerminal] not found on/within module
'<top-design>'
```

Potential Issues

This violation appears if your design does not contain the port, hierarchical terminal, or net specified by the `-from` argument of the *define_reset_order*

constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port, hierarchical terminal, or net in the `-from` argument of the *define_reset_order* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the `rst1` port or net. Now consider that you specify the following constraint:

```
define_reset_order -from rst1 -to rst2
```

In the above case, *SGDC_define_reset_order01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_define_reset_order02

Reports a violation if an invalid object is specified in the -to argument of the define_reset_order constraint

When to Use

Use this rule to perform sanity checks on the *define_reset_order* constraint.

Prerequisites

Specify the *define_reset_order* and *reset* constraints.

Description

The *SGDC_define_reset_order02* rule reports a violation if the object specified in the `-to` argument of the *define_reset_order* constraint does not exist as port, hierarchical terminal, or net in the current design.

Parameter(s)

None

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Mandatory): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if an invalid object is specified in the `-to` argument of the *define_reset_order* constraint:

```
[FATAL] 'define_reset_order': '<signal-name>'[TopPort + Net + HierTerminal] not found on/within module '<top-design>'
```

Potential Issues

This violation appears if your design does not contain the port, hierarchical terminal, or net specified by the `-to` argument of the *define_reset_order* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port, hierarchical terminal, or net in the `-to` argument of the *define_reset_order* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the `rst2` port or net. Now consider that you specify the following constraint:

```
define_reset_order -from rst1 -to rst2
```

In the above case, *SGDC_define_reset_order02* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_define_reset_order03

Reports a violation if an invalid reset is specified in the -from argument of the define_reset_order constraint

When to Use

Use this rule to perform sanity checks on the *define_reset_order* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *define_reset_order* constraint.
- Run the `Ar_resetcross01` rule to enable the *SGDC_define_reset_order03* rule because the *define_reset_order* constraint is used by the `Ar_resetcross01` rule.

Description

The *SGDC_define_reset_order03* rule reports a violation if the reset specified by the `-from` argument of the *define_reset_order* constraint is not any of the following reset:

- A reset specified by the *reset* constraint
- An automatically inferred reset when the *use_inferred_resets* parameter is set to `yes`.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Mandatory): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if the reset specified by the `-from`

argument of the *define_reset_order* constraint is not a user-specified reset or it is not an automatically inferred reset:

```
[WARNING] Constraint 'define_reset_order': Reset name '<reset-name>' specified in '-from' field is not a valid reset
```

Potential Issues

This violation appears if your design does not contain the reset specified by the `-from` argument of the *define_reset_order* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following reset in the `-from` argument of the *define_reset_order* constraint:

- A reset specified by the *reset* constraint
- An automatically inferred reset when the *use_inferred_resets* parameter is set to `yes`.

Example Code and/or Schematic

Consider that the `rst` reset is neither specified by the *reset* constraint nor it is an automatically inferred reset.

Now consider that you specify the following constraint:

```
define_reset_order -from rst -to rst1
```

In the above case, the *SGDC_define_reset_order03* rule reports a violation.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_define_reset_order04

Reports a violation if an invalid reset is specified in the -to argument of the define_reset_order constraint

When to Use

Use this rule to perform sanity checks on the *define_reset_order* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *define_reset_order* constraint.
- Run the `Ar_resetcross01` rulesto enable the *SGDC_define_reset_order04* rule because the *define_reset_order* constraint is used by the `Ar_resetcross01` rule.

Description

The *SGDC_define_reset_order04* rule reports a violation if the reset specified by the `-to` argument of the *define_reset_order* constraint is not any of the following reset:

- A reset specified by the *reset* constraint
- An automatically inferred reset when the *use_inferred_resets* parameter is set to `yes`.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Mandatory): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if the reset specified by the `-to` argument

of the *define_reset_order* constraint is not a user-specified reset or it is not an automatically inferred reset:

```
[WARNING] Constraint 'define_reset_order': Reset name
'<reset-name>' specified in '-to' field is not a valid reset
```

Potential Issues

This violation appears if your design does not contain the reset specified by the `-to` argument of the *define_reset_order* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following reset in the `-to` argument of the *define_reset_order* constraint:

- A reset specified by the *reset* constraint
- An automatically inferred reset when the *use_inferred_resets* parameter is set to `yes`.

Example Code and/or Schematic

Consider that the `rst1` reset is neither specified by the *reset* constraint nor it is an automatically inferred reset.

Now consider that you specify the following constraint:

```
define_reset_order -from rst -to rst1
```

In the above case, the *SGDC_define_reset_order04* rule reports a violation.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_define_reset_order05

Checks for bidirectional reset ordering specified by the define_reset_order constraint

When to Use

Use this rule to perform sanity checks on the *define_reset_order* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *define_reset_order* constraint.
- Run the `Ar_resetcross01` rule to enable the *SGDC_define_reset_order05* rule because the *define_reset_order* constraint is used by the `Ar_resetcross01` rule.

Description

The *SGDC_define_reset_order05* rule reports a violation if the reset specified by the `-to` or `-from` argument of the *define_reset_order* constraint matches with the reset specified by the `-from` or `-to` argument, respectively, of another *define_reset_order* constraint.

Parameter(s)

- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *define_reset_order* (Mandatory): Use this constraint to specify a reset order, which determines the flow of data from one reset to another reset.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if bidirectional reset ordering is specified by the *define_reset_order* constraint:

```
[INFO] Constraint 'define_reset_order': Bi-directional reset
ordering specified between '<reset-specification1>' to
```

'<reset-specification2>'. Ignoring reset-crossings between them

Potential Issues

This violation appears if the reset specified in the `-to` or `-from` argument of the *define_reset_order* constraint matches with the reset specified in the `-from` or `-to` argument, respectively, of another *define_reset_order* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported bidirectional reset ordering is considered valid. As a result, reset crossings between such resets are ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, analyze the reported reset ordering.

If the ordering is intentional, ignore this violation. Else, specify a correct set of resets in the `-from` and `-to` arguments of the *define_reset_order* constraint specifications to allow the flow of data from one reset to another.

Example Code and/or Schematic

In this example, the *SGDC_define_reset_order05* rule reports a violation because the `rst*` expression specified by the `-from` argument of the first specification matches with the `rst1` value specified by the `-to` argument of the second specification.

```
define_reset_order -from "rst*" -to reset
define_reset_order -from reset -to rst1
```

Default Severity Label

Info

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_ignore_instance01

Reports a violation if an invalid instance is specified in the -name argument of the deltacheck_ignore_instance constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_ignore_instance* constraint.

Prerequisites

Specify the *deltacheck_ignore_instance* constraint.

Description

The *SGDC_deltacheck_ignore_instance01* rule reports a violation if the instance specified by the `-name` argument of the *deltacheck_ignore_instance* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

deltacheck_ignore_instance (Mandatory): Use this constraint to specify instances to be ignored for delta delay value checking.

Messages and Suggested Fix

The following message appears if the instance *<instance>* specified by the `-name` argument of the *deltacheck_ignore_instance* constraint does not exist in the design *<top-module>*:

```
[FATAL] '<instance-name>' [Instance] not found on/within module
'<top-module>'
```

Potential Issues

This violation appears if the current design does not contain the instance specified by the `-name` argument of the *deltacheck_ignore_instance* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing instance in the `-name` argument of the *deltacheck_ignore_instance* constraint.

Example Code and/or Schematic

Consider the design `top` that does not contain the `U_DF1` instance. Now consider the following constraints specified in an SGDC file:

```
current_design top
deltacheck_ignore_instance -name U_DF1
```

In the above case, the *SGDC_deltacheck_ignore_instance01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_ignore_module01

Reports a violation if an invalid module is specified in the -name argument of the deltacheck_ignore_module constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_ignore_module* constraint.

Prerequisites

Specify the *deltacheck_ignore_module* constraint.

Description

The *SGDC_deltacheck_ignore_module01* rule reports a violation if the module specified by the `-name` argument of the *deltacheck_ignore_module* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

deltacheck_ignore_module (Mandatory):

Messages and Suggested Fix

The following message appears if the module *<module>* specified by the `-name` argument of the *deltacheck_ignore_module* constraint does not exist in the design *<top-module>*:

```
[FATAL] '<module>' [SubModule] is never instantiated within environment '<top-module>'
```

Potential Issues

This violation appears if the current design does not contain the module specified by the `-name` argument of the *deltacheck_ignore_module* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing module in the `-name` argument of the *deltacheck_ignore_module* constraint.

Example Code and/or Schematic

Consider the design `top` that does not contain the `mod1` module. Now consider the following constraints specified in an SGDC file:

```
current_design top
deltacheck_ignore_module -name mod1
```

In the above case, the *SGDC_deltacheck_ignore_module01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_start01

Reports a violation if an invalid object is specified in the -name argument of the deltacheck_start constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_start* constraint.

Prerequisites

Specify the *deltacheck_start* constraint.

Description

The *SGDC_deltacheck_start01* rule reports a violation if the object specified by the `-name` argument of the *deltacheck_start* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

deltacheck_start (Mandatory): Use this constraint to specify start points, such as clock ports, clock pins, or clock nets for *DeltaDelay01* rule-checking.

Messages and Suggested Fix

The following message appears if an invalid object *<object-name>* is specified in the `-name` argument of the *deltacheck_start* constraint:

[FATAL] <object-name> [TopPort + Net + HierTerminal] not found on/within module 'TOP'

Potential Issues

This violation appears if the design does not contain the object specified by the `-name` argument of the *deltacheck_start* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing object in the `-name` argument of the *deltacheck_start* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the clock net `top.clk1`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
deltacheck_start -name top.clk1
```

For the above example, the *SGDC_deltacheck_start01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_start02

Reports a violation if a non-integer value is specified in the -value argument of the deltacheck_start constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_start* constraint.

Prerequisites

Specify the *deltacheck_start* constraint.

Description

The *SGDC_deltacheck_start02* rule reports a violation if a non-integer value is specified in the `-value` argument of the *deltacheck_start* constraint.

The `-value` argument is an optional argument used for clock-pin balancing.

Parameter(s)

None

Constraint(s)

deltacheck_start (Mandatory): Use this constraint to specify start points, such as clock ports, clock pins, or clock nets for *DeltaDelay01* rule-checking.

Messages and Suggested Fix

The following message appears if a non-integer value *<value>* is specified in the `-value` argument of the *deltacheck_start* constraint:

```
[FATAL] Invalid [Pre-defined-range] specification '<value>' for
'-value' field of constraint 'deltacheck_start'
```

Potential Issues

This violation appears if a non-integer value is specified in the `-value` argument of the *deltacheck_start* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an integer value in the `-value` argument of the *deltacheck_start* constraint.

Example Code and/or Schematic

Consider the following constraints:

```
current_design top
deltacheck_start -name top.clk1 -value "-4"
```

For the above example, the *SGDC_deltacheck_start02* rule reports a violation because the non-integer value `"-4"` is specified in the `-value` argument of the *deltacheck_start* constraint.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_stop_instance01

Reports a violation if an invalid instance is specified in the -name argument of the deltacheck_stop_instance constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_stop_instance* constraint.

Prerequisites

Specify the *deltacheck_stop_instance* constraint.

Description

The *SGDC_deltacheck_stop_instance01* rule reports a violation if the instance specified by the `-name` argument of the *deltacheck_stop_instance* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

deltacheck_stop_instance (Mandatory): Use this constraint to specify instances where the *DeltaDelay01* rule should stop further traversal along the clock tree.

Messages and Suggested Fix

The following message appears if an invalid instance is specified in the `-name` argument of the *deltacheck_stop_instance* constraint:

```
[FATAL] '<inst-name>' [Instance] not found on/within module '<top_module_name>'
```

Potential Issues

This violation appears if the design does not contain the instance specified by the `-name` argument of the *deltacheck_stop_instance* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing instance in the `-name` argument of the *deltacheck_stop_instance* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain any instance of the name `I21`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
deltacheck_stop_instance -name top.I21
```

For the above example, the *SGDC_deltacheck_stop_instance01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_stop_module01

Reports a violation if an invalid module is specified in the -name argument of the deltacheck_stop_module constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_stop_module* constraint.

Prerequisites

Specify the *deltacheck_stop_module* constraint.

Description

The *SGDC_deltacheck_stop_module01* rule reports a violation if the module specified by the `-name` argument of the *deltacheck_stop_module* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

deltacheck_stop_module (Mandatory): Use this constraint to specify design units where the *DeltaDelay01* rule should stop further traversal along the clock tree.

Messages and Suggested Fix

The following message appears if an invalid module is specified in the `-name` argument of the *deltacheck_stop_module* constraint:

```
[FATAL] '<submod-name>' [SubModule] is never instantiated
within environment '<top-mod-name>'
```

Potential Issues

This violation appears if the design does not contain the module specified by the `-name` argument of the *deltacheck_stop_module* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing module in the `-name` argument of the *deltacheck_stop_module* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain any module of the name `m1`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
deltacheck_stop_module -name m1
```

For the above example, the *SGDC_deltacheck_stop_module01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_deltacheck_stop_signal01

Reports a violation if an invalid object is specified in the -name argument of the deltacheck_stop_signal constraint

When to Use

Use this rule to perform sanity checks on the *deltacheck_stop_signal* constraint.

Prerequisites

Specify the *deltacheck_stop_signal* constraint.

Description

The *SGDC_deltacheck_stop_signal01* rule reports a violation if the object specified by the `-name` argument of the *deltacheck_stop_signal* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

deltacheck_stop_signal (Mandatory): Use this constraint to specify design points, such as ports, pins, or nets where the *DeltaDelay01* rule should stop further traversal.

Messages and Suggested Fix

The following message appears if an invalid object is specified in the `-name` argument of the *deltacheck_stop_signal* constraint:

```
[FATAL] '<submod-name>' [SubModule] is never instantiated
within environment '<top-mod-name>'
```

Potential Issues

This violation appears if the design does not contain the object specified by the `-name` argument of the *deltacheck_stop_signal* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing object in the `-name` argument of the *deltacheck_stop_signal* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the pin `top.I31.d1`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
deltacheck_stop_signal -name top.I31.d1
```

For the above example, the *SGDC_deltacheck_stop_signal01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo01

Reports a violation if no argument is specified with the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo01* rule reports a violation if you do not specify any argument with the *fifo* constraint.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if you do not specify any argument with the *fifo* constraint:

```
[FATAL] It is Mandatory to Specify At Least One Value for
combination of fields '-rd_data -wr_data -rd_ptr -wr_ptr
-memory ' of constraint 'fifo'
```

Potential Issues

This violation appears if you do not specify any argument with the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify at least one argument with the *fifo* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo02

Reports if an incorrect object is specified in the -rd_data argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo02* rule reports a violation if the object specified by the `-rd_data` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if an incorrect object *<object-name>* is specified in the `-rd_data` argument of the *fifo* constraint:

```
[FATAL] '<object-name>' [Net + HierTerminal] not found on/
within module '<top-module>'
```

Potential Issues

This violation appears if the current design does not contain a net or a hierarchical terminal specified by the `-rd_data` argument of the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net or a hierarchical terminal in the `-rd_data` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider that the current design `top` does not contain a net or a hierarchical terminal of the name `ram_req`. In this case, the *SGDC_fifo02* rule reports a violation if you specify the following constraints:

```
current_design top
-rd_data "ram_req" -wr_data "i_req"
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo03

Reports if an incorrect object is specified in the -wr_data argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo03* rule reports a violation if the object specified by the `-wr_data` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if an incorrect object *<object-name>* is specified in the `-wr_data` argument of the *fifo* constraint:

```
[FATAL] '<object-name>' [Net + HierTerminal] not found on/
within module '<top-module>'
```

Potential Issues

This violation appears if the current design does not contain a net or a hierarchical terminal specified by the `-wr_data` argument of the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net or a hierarchical terminal in the `-wr_data` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider that the current design `top` does not contain a net or a hierarchical terminal of the name `i_req`. In this case, the *SGDC_fifo03* rule reports a violation if you specify the following constraints:

```
current_design top
-rd_data "ram_req" -wr_data "i_req"
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo04

Reports if an incorrect object is specified in the -rd_ptr argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo04* rule reports a violation if the object specified by the `-rd_ptr` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if an incorrect object *<object-name>* is specified in the `-rd_ptr` argument of the *fifo* constraint:

```
[FATAL] '<object-name>' [Net + HierTerminal] not found on/
within module '<top-module>'
```

Potential Issues

This violation appears if the current design does not contain a net or a hierarchical terminal specified by the `-rd_ptr` argument of the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net or a hierarchical terminal in the `-rd_ptr` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider that the current design `top` does not contain a net or a hierarchical terminal of the name `adr`. In this case, the *SGDC_fifo04* rule reports a violation if you specify the following constraints:

```
current_design top
fifo -rd_ptr "adr" -wr_ptr "q_wt_adr"
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo05

Reports if an incorrect object is specified in the -wr_ptr argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo05* rule reports a violation if the object specified by the `-wr_ptr` argument of the *fifo* constraint does not exist as a net or a hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if an incorrect object *<object-name>* is specified in the `-wr_ptr` argument of the *fifo* constraint:

```
[FATAL] '<object-name>' [Net + HierTerminal] not found on/
within module '<top-module>'
```

Potential Issues

This violation appears if the current design does not contain a net or a hierarchical terminal specified by the `-wr_ptr` argument of the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net or a hierarchical terminal in the `-wr_ptr` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider that the current design `top` does not contain a net or a hierarchical terminal of the name `adr`. In this case, the *SGDC_fifo05* rule reports a violation if you specify the following constraints:

```
current_design top
fifo -rd_ptr "q_rd_adr" -wr_ptr "adr"
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo06

Reports if an incorrect value is specified in the -memory argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo06* rule reports a violation if the value specified by the `-memory` argument of the *fifo* constraint does not match with the name of an existing net, hierarchical terminal, or module in the current design.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if an incorrect value *<value>* is specified in the `-memory` argument of the *fifo* constraint:

```
[FATAL] '<value>' [Net + Instance + HierTerminal + SubModule]
not found on/within module '<top-module>'
```

Potential Issues

This violation appears if the current design does not contain a net, hierarchical terminal, or module specified by the `-memory` argument of the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net, hierarchical terminal, or module in the `-memory` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider that the current design `top` does not contain a net, hierarchical terminal, or module of the name `ram_r`. In this case, the *SGDC_fifo06* rule reports a violation if you specify the following constraints:

```
current_design top
fifo -memory ram_r
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo07

Reports if the -wr_data argument is not specified with the -rd_data argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo07* rule reports a violation if the `-rd_data` argument of the *fifo* constraint is specified without its corresponding `-wr_data` argument.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if the `-rd_data` argument of the *fifo* constraint is specified without its corresponding `-wr_data` argument:

```
[FATAL] Constraint 'fifo': Field '-rd_data' can/should be specified with field(s) '-wr_data' only
```

Potential Issues

This violation appears if the `-rd_data` argument of the *fifo* constraint is specified without its corresponding `-wr_data` argument.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the `-wr_data` argument along with the `-rd_data` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
fifo -rd_data "ram_req"
```

For the above constraint, the *SGDC_fifo07* rule reports a violation because the `-wr_data` argument is not specified with the `-rd_data` argument.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo08

Reports if the -rd_data argument is not specified with the -wr_data argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo08* rule reports a violation if the `-wr_data` argument of the *fifo* constraint is specified without its corresponding `-rd_data` argument.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if the `-wr_data` argument of the *fifo* constraint is specified without its corresponding `-rd_data` argument:

```
[FATAL] Constraint 'fifo': Field '-wr_data' can/should be specified with field(s) '-rd_data' only
```

Potential Issues

This violation appears if the `-wr_data` argument of the *fifo* constraint is specified without its corresponding `-rd_data` argument.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the `-rd_data` argument along with the `-wr_data` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
fifo -wr_data "ram_req"
```

For the above constraint, the *SGDC_fifo08* rule reports a violation because the `-rd_data` argument is not specified with the `-wr_data` argument.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo09

Reports if the -rd_ptr argument is not specified with the -wr_ptr argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo09* rule reports a violation if the `-wr_ptr` argument of the *fifo* constraint is specified without its corresponding `-rd_ptr` argument.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if the `-wr_ptr` argument of the *fifo* constraint is specified without its corresponding `-rd_ptr` argument:

```
[FATAL] Constraint 'fifo': Field '-rd_ptr' can/should be specified with field(s) '-wr_ptr' only
```

Potential Issues

This violation appears if the `-wr_ptr` argument of the *fifo* constraint is specified without its corresponding `-rd_ptr` argument.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the `-rd_ptr` argument along with the `-wr_ptr` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
fifo -wr_ptr "wr_adr"
```

For the above constraint, the *SGDC_fifo09* rule reports a violation because the `-rd_ptr` argument is not specified with the `-wr_ptr` argument.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo10

Reports if the -wr_ptr argument is not specified with the -rd_ptr argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo10* rule reports a violation if the `-rd_ptr` argument of the *fifo* constraint is specified without its corresponding `-wr_ptr` argument.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if the `-rd_ptr` argument of the *fifo* constraint is specified without its corresponding `-wr_ptr` argument:

```
[FATAL] Constraint 'fifo': Field '-wr_ptr' can/should be specified with field(s) '-rd_ptr' only
```

Potential Issues

This violation appears if the `-rd_ptr` argument of the *fifo* constraint is specified without its corresponding `-wr_ptr` argument.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the `-wr_ptr` argument along with the `-rd_ptr` argument of the *fifo* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
fifo -rd_ptr "rd_adr"
```

For the above constraint, the *SGDC_fifo10* rule reports a violation because the `-wr_ptr` argument is not specified with the `-rd_ptr` argument.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo11

Reports a violation if the wildcard name specified by an argument of the fifo constraint does not match with any object in the design

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo11* rule reports a violation if the wildcard expression specified by any of the following arguments of the *fifo* constraint does not match with the name of any object in the design:

-rd_data	-wr_data	-rd_ptr	-wr_ptr	-memory

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if the object *<object-name>* specified by the argument *<argument-name>* of the *fifo* constraint does not exist in the design:

```
[ERROR] "<object-name>" specified in the "<argument-name>"
field of fifo constraint could not be found in the design
```

Potential Issues

This violation appears if the design does not contain the object specified by the wildcard expressions of any of the following arguments of the *fifo* constraint:

-rd_data	-wr_data	-rd_ptr	-wr_ptr	-memory

Consequences of Not Fixing

If you do not fix this violation, FIFO specified by the reported *fifo* constraint will remain undetected. As a result, such FIFOs will not be analyzed.

How to Debug and Fix

To fix this violation, specify the name of existing objects in the arguments of the *fifo* constraint.

Example Code and/or Schematic

Consider that a design does not contain the net whose name is prefixed with `raddr1`. Now consider that you specify the following constraint in an SGDC file:

```
fifo -rd_ptr "raddr1*" -wr_ptr wptr1
```

For the above example, the *SGDC_fifo11* rule reports a violation.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo12

Reports a violation if no FIFO memory could be inferred from the object specified by an argument of the fifo constraint

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo12* rule reports a violation if no memory instance could be inferred from the object specified by any of the following arguments of the *fifo* constraint:

-rd_data	-wr_data	-rd_ptr	-wr_ptr

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if no memory instance could be inferred from the object *<object-name>* specified by the argument *<argument-name>* of the *fifo* constraint:

[ERROR] No memory could be inferred from "<object-name>" specified in the "<argument-name>" field of fifo constraint

Potential Issues

This violation appears if the design does not contain the FIFO memory corresponding to the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, FIFO specified by the reported *fifo* constraint will remain undetected. As a result, such FIFOs will not be analyzed.

How to Debug and Fix

To fix this violation, perform the following actions:

- Check the arguments of the *fifo* constraint.

  If any of the `-rd_data`, `-wr_data`, `-rd_ptr`, and `-wr_ptr` arguments is assigned an incorrect value for RTL nets, specify a correct value.
- If all the bits of read/write pointers are not used in fifo, specify a correct index.

Example Code and/or Schematic

Consider a scenario in which a design contains a FIFO with the `rptr` and `wptr` pointers. However, the user specifies the following constraint in an SGDC file:

```
fifo -rd_ptr raddr -wr_ptr wptr
```

In this case, the *SGDC_fifo12* rule reports a violation for the incorrect read pointer specified by the `-rd_ptr` argument of the above *fifo* constraint.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo13

Reports a violation in case of a width mismatch of read and write pointers of a user-defined FIFO

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo13* rule reports a violation if the width of read and write pointers present in a design does not match, and these pointers are specified by the *fifo* constraint.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if there is a width mismatch of read and write pointers of a user-defined FIFO:

```
[ERROR] Read pointer '<read-ptr>' width (<read-ptr-width>) does not match write pointer '<write-ptr>' width (<write-ptr-width>)
```

Potential Issues

This violation appears if your design contains read and write pointers of different widths, and such pointers are specified by the *fifo* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported *fifo* constraint is ignored during SpyGlass analysis. As a result, such FIFOs remain undetected, and

therefore, are not analyzed.

How to Debug and Fix

To debug and fix this violation, perform the following actions:

- Check the arguments specified by the *fifo* constraint.
- Ensure that both the read and write pointers of this constraint follow the same coding format, such as one-hot coding or gray-coding as that of the corresponding pointers in the design.

Example Code and/or Schematic

Consider a scenario in which the design contains a FIFO with the following pointers:

- The `rptr[0:3]` read pointer that is used in the decoded format
- The `wptr[0:15]` write pointer that is used in the one-hot format

Now consider that you specify the following *fifo* constraint:

```
fifo -rd_ptr rptr -wr_ptr wptr
```

In this case, the *SGDC_fifo13* rule reports a violation because the width of the read and write pointers specified by the *fifo* constraint does not match.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_fifo14

Reports invalid user-defined FIFOs

When to Use

Use this rule to perform sanity checks on the *fifo* constraint.

Prerequisites

Specify the *fifo* constraint.

Description

The *SGDC_fifo14* rule reports a violation if no FIFO could be inferred from the *fifo* constraint.

Parameter(s)

None

Constraint(s)

fifo (Mandatory): Use this constraint to specify FIFO information.

Messages and Suggested Fix

The following message appears if no FIFO could be inferred from the *fifo* constraint:

```
[ERROR] Invalid user defined FIFO
```

Potential Issues

This violation appears if SpyGlass cannot not infer any FIFO from the *fifo* constraint.

This occurs when data, pointers, and memory specified by the *fifo* constraint does not belong to the same FIFO.

Consequences of Not Fixing

If you do not fix this violation, the reported *fifo* constraint is ignored during SpyGlass analysis.

How to Debug and Fix

To debug and fix this violation, perform the following actions:

- Check the arguments specified by the *fifo* constraint.
- If any of the `-rd_ptr`/`-wr_ptr` or `-rd_data`/`-wr_data` arguments is assigned an incorrect value for RTL nets, specify a correct value.

Example Code and/or Schematic

Consider a scenario in which `raddr`/`waddr` and `rptr`/`wptr` are pointers for two different FIFOs in a design. However, the user has specified the following constraint in an SGDC file:

```
fifo -rd_ptr raddr -wr_ptr wptr
```

In this case, the *SGDC_fifo14* rule reports a violation because `raddr` and `wptr` are not the read/write pointers of the same fifo memory.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_generated_clock03

Either of the -divide_by or -multiply_by argument of generated_clock should be specified

When to Use

Use this rule to ensure that either of the `-divide_by` or `-multiply_by` argument is specified for the *generated_clock* constraint.

Prerequisites

Use the *generated_clock* constraint to specify generated clocks.

Description

The *SGDC_generated_clock03* rule reports a violation if none of the `-divide_by` or `-multiply_by` argument is specified to the *generated_clock* constraint.

Parameter(s)

None

Constraint(s)

generated_clock (Mandatory): Use this constraint to specify derived clocks.

Messages and Suggested Fix

The following message appears if none of the `-divide_by` or `-multiply_by` argument is specified to the *generated_clock* constraint:

```
[FATAL] It is Mandatory to Specify At Least One Value for combination of fields '-multiply_by -divide_by ' of constraint 'generated_clock'
```

Potential Issues

This violation appears if you do not specify one of the `-divide_by` or `-multiply_by` argument of the *generated_clock* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify either of the `-divide_by` or `-multiply_by` argument of the *generated_clock* constraint.

Example Code and/or Schematic

The *SGDC_generated_clock03* rule reports a violation in the following case because none of the `-divide_by` or `-multiply_by` argument is specified for the *generated_clock* constraint:

```
current_design test
clock -name clk1 -domain d1 -tag t1 -add

generated_clock -name gen_clk -source test.gen_clk_reg.CP
-master_clock t1 -tag g1
```

Rule Group

Fatal must rule

Default Severity Label

Fatal

Reports and Related Files

No report or related file

SGDC_generated_clock04

Incorrect value for the -multiply_by argument of the generated_clock constraint

When to Use

Use this rule to check for the correctness of the value specified to the `-multiply_by` argument of the *generated_clock* constraint.

Prerequisites

Use the *generated_clock* constraint to specify generated clocks.

Description

The *SGDC_generated_clock04* rule reports a violation if the value specified to the `-multiply_by` argument of the *generated_clock* constraint is not a float value or a negative float value.

Parameter(s)

None

Constraint(s)

generated_clock (Mandatory): Use this constraint to specify derived clocks.

Messages and Suggested Fix

The following message appears if the value *<value>* specified to the `-multiply_by` argument of the *generated_clock* constraint is not a float value or a negative float value:

[FATAL] `Invalid [Pre-defined-range] specification '<value>' for '-multiply_by' field of constraint 'generated_clock'`

Potential Issues

This violation appears if the value specified to the `-multiply_by` argument of the *generated_clock* constraint is not a float value or a negative float value.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a float value to the `-multiply_by` argument of the *generated_clock* constraint.

Example Code and/or Schematic

The *SGDC_generated_clock04* rule reports a violation in the following case because a non float value is specified to the `-multiply_by` argument of the *generated_clock* constraint:

```
generated_clock -name gen_clk -source test.gen_clk_reg.CP
-multiply_by "-1" -master_clock t1 -add -tag g1
```

Rule Group

Fatal must rule

Default Severity Label

Fatal

Reports and Related Files

No report or related file

SGDC_generated_clock05

Incorrect value for the -divide_by argument of the generated_clock constraint

When to Use

Use this rule to check for the correctness of the value specified to the `-divide_by` argument of the *generated_clock* constraint.

Prerequisites

Use the *generated_clock* constraint to specify generated clocks.

Description

The *SGDC_generated_clock05* rule reports a violation if the value specified to the `-divide_by` argument of the *generated_clock* constraint is not a float value or a negative float value.

Parameter(s)

None

Constraint(s)

generated_clock (Mandatory): Use this constraint to specify derived clocks.

Messages and Suggested Fix

The following message appears if the value *<value>* specified to the `-divide_by` argument of the *generated_clock* constraint is not a float value or a negative float value:

[FATAL] `Invalid [Pre-defined-range] specification '<value>' for '-divide_by' field of constraint 'generated_clock'`

Potential Issues

This violation appears if the value specified to the `-divide_by` argument of the *generated_clock* constraint is not a float value or a negative float value.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a float value to the `-divide_by` argument of the *generated_clock* constraint.

Example Code and/or Schematic

The *SGDC_generated_clock05* rule reports a violation in the following case because a non float value is specified to the `-divide_by` argument of the *generated_clock* constraint:

```
generated_clock -name gen_clk -source test.gen_clk_reg.CP
-divide_by "-1" -master_clock t1 -add -tag g1
```

Rule Group

Fatal must rule

Default Severity Label

Fatal

Reports and Related Files

No report or related file

SGDC_generated_clock06

Sanity checks for the generated_clock constraint

When to Use

Use this rule to perform sanity checks on the *generated_clock* constraint.

Prerequisites

Use the *generated_clock* constraint to specify derived clocks.

Description

The *SGDC_generated_clock06* rule reports a violation in any of the following cases:

- The *clock* constraint is specified on a generated clock specified by the *generated_clock* constraint. See *Message 1*.
- Same tag names are assigned to different generated clocks specified by the *generated_clock* constraints. See *Message 2*.
- The source object specified in the *generated_clock* constraint is not in the fan-in of the generated clock. See *Message 3*.
- No clock is reaching to the source of the generated clock specified by the *generated_clock* constraint. See *Message 4*.
- The master clock specified by the *generated_clock* constraint is not reaching the source of the generated clock. See *Message 5*.
- The *enable_generated_clocks* parameter is not set to yes when the *generated_clock* constraint is specified. See *Message 6*.

Parameter(s)

None

Constraint(s)

generated_clock (Mandatory): Use this constraint to specify derived clocks.

Messages and Suggested Fix

Message 1

The following message appears if you specify the *clock* constraint on a generated clock:

```
[ERROR] 'clock' constraint specified on '<generated-clock>'.
Ignoring the constraint
```

Potential Issues

This violation appears if you specify the *clock* constraint on a generated clock specified by the *generated_clock* constraint.

Consequences of Not Fixing

If you do not fix this violation, the specified *generated_clock* constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, remove the *clock* constraint for the reported generated clock.

Message 2

The following message appears if you specify the same tag for multiple generated clocks:

```
[ERROR] Tag '<tag-name>' specified already on clock
'<generated-clock>'. Ignoring the constraint
```

Potential Issues

This violation appears if you specify a duplicate tag on the same generated clock specified by the *generated_clock* constraints.

Consequences of Not Fixing

If you do not fix this violation, the *generated_clock* constraint having the

same tag as the previous *generated_clock* constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify unique tag names to the generated clocks specified by the *generated_clock* constraints.

Message 3

The following message appears if the source clock specified for a generated clock in the *generated_clock* constraint is not in the fan-in of the generated clock:

```
[ERROR] Source object '<source-object>' specified is not in
fanin of generated clock '<generated-clock>'. Ignoring the
constraint
```

Potential Issues

This violation appears if the source clock specified for the generated clock in the *generated_clock* constraint is not in the fan-in of the generated clock.

Consequences of Not Fixing

If you do not fix this violation, the *generated_clock* constraint containing the reported clock is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, modify the *generated_clock* constraint to specify a source clock that exists in the fan-in of the generated clock.

Message 4

The following message appears if there is no master clock for the source clock specified in the *generated_clock* constraint:

```
[ERROR] No clock is reaching on the source '<source-clock>' of
generated clock. Ignoring the constraint
```

Potential Issues

This violation appears if the source clock specified in the *generated_clock* constraint does not have a master clock.

Consequences of Not Fixing

If you do not fix this violation, the *generated_clock* constraint containing the reported source clock is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, modify the design by connecting a master clock to the source clock specified in the *generated_clock* constraint.

Message 5

The following message appears if there is the master clock specified in the *generated_clock* constraint does not reach the source clock specified in this constraint:

```
[ERROR] Master clock '<master-clock>' is not reaching on the
source '<source-clock>' of generated clock. Ignoring the
constraint
```

Potential Issues

This violation appears if the master clock specified in the *generated_clock* constraint does not reach the source clock specified in this constraint.

Consequences of Not Fixing

If you do not fix this violation, the *generated_clock* constraint containing the reported master clock is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, modify the *generated_clock* constraint to specify a master clock that reaches the specified source clock.

Message 6

The following message appears if the *generated_clock* constraint is specified but the *enable_generated_clocks* parameter is not set to `yes`:

```
[ERROR] Parameter 'enable_generated_clocks' not set to yes.
Ignoring 'generated_clock' constraints
```

Potential Issues

This violation appears if the *generated_clock* constraint is specified but the *enable_generated_clocks* parameter is not set to `yes`.

Consequences of Not Fixing

If you do not fix this violation, the *generated_clock* constraints are ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, set the *enable_generated_clocks* parameter to yes while specifying the *generated_clock* constraint.

Example Code and/or Schematic

Consider the following constraints:

```
clock -name gen_clk -domain d3
```

```
generated_clock -name gen_clk -source test.gen_clk_reg.CP
-divide_by 2 -master_clock t1 -add -tag g1
```

```
generated_clock -name gen_clk -source
test.gen_clk_new_reg.CP -divide_by 2 -master_clock t2 -add
-tag g2
```

For the above example, the *SGDC_generated_clock06* rule reports a violation (*Message 1*) because the *clock* constraint is specified for the generated clock `gen_clk`.

Rule Group

Non Fatal must rule

Default Severity Label

Error

Reports and Related Files

No report or related file

SGDC_gray_signals01

Checks presence of multiple scalar signals in gray_signals constraint

When to Use

Use this rule to check the signals specified to the *gray_signals* constraint.

Prerequisites

Use the *gray_signals* constraint to specify the signals that should be gray encoded.

Description

The *SGDC_gray_signals01* rule reports a violation if you specify only one scalar signal to the *gray_signals* constraint.

Parameter(s)

None

Constraint(s)

gray_signals (Mandatory): Use this constraint to specify signals that should be gray encoded.

Messages and Suggested Fix

The following message appears if you specify only one scalar signal to the `gray_signals` constraint:

```
[ERROR] Constraint 'gray_signals': Only one scalar signal
'<signal-name>' specified. Multiple scalar signals or a bus
should be specified
```

Potential Issues

This violation appears if you specify only one scalar signal to the *gray_signals* constraint.

Consequences of Not Fixing

If you do not fix this violation, no gray encoding is performed on the reported signal.

How to Debug and Fix

To fix this violation, modify the *gray_signals* constraint to specify any of the following:

- More than one scalar signal
- A bus

Example Code and/or Schematic

Consider the following constraint:

```
gray_signals -name src2
```

In the above case, the *SGDC_gray_signals01* rule reports a violation because only one scalar signal `src2` is specified in this constraint.

Rule Group

Non fatal must rule

Default Severity Label

Error

Reports and Related Files

No report or related file

SGDC_gray_signals02

Checks whether the signals specified by the gray_signals constraint are driven by a clock

When to Use

Use this rule to check the signals specified to the *gray_signals* constraint.

Prerequisites

Use the *gray_signals* constraint to specify the signals that should be gray encoded.

Description

The *SGDC_gray_signals02* rule reports a violation if any signal specified by the *gray_signals* constraint is not driven by a clock.

Parameter(s)

None

Constraint(s)

gray_signals (Mandatory): Use this constraint to specify signals that should be gray encoded.

Messages and Suggested Fix

The following message appears if the signals <signal-names> specified by the `gray_signals` constraint are not driven by a clock:

[ERROR] `Constraint 'gray_signals': No clock domain detected for signals '<signal-names>'`

Potential Issues

This violation appears if the signals specified by the `gray_signals` constraint are not driven by a clock.

Consequences of Not Fixing

If you do not fix this violation, no gray encoding check is performed on the

reported signals.

How to Debug and Fix

To fix this violation, correct the clock definition in the design so that the reported signals receive a clock.

Example Code and/or Schematic

Consider the following schematic of a design and the SGDC file specified for that design:

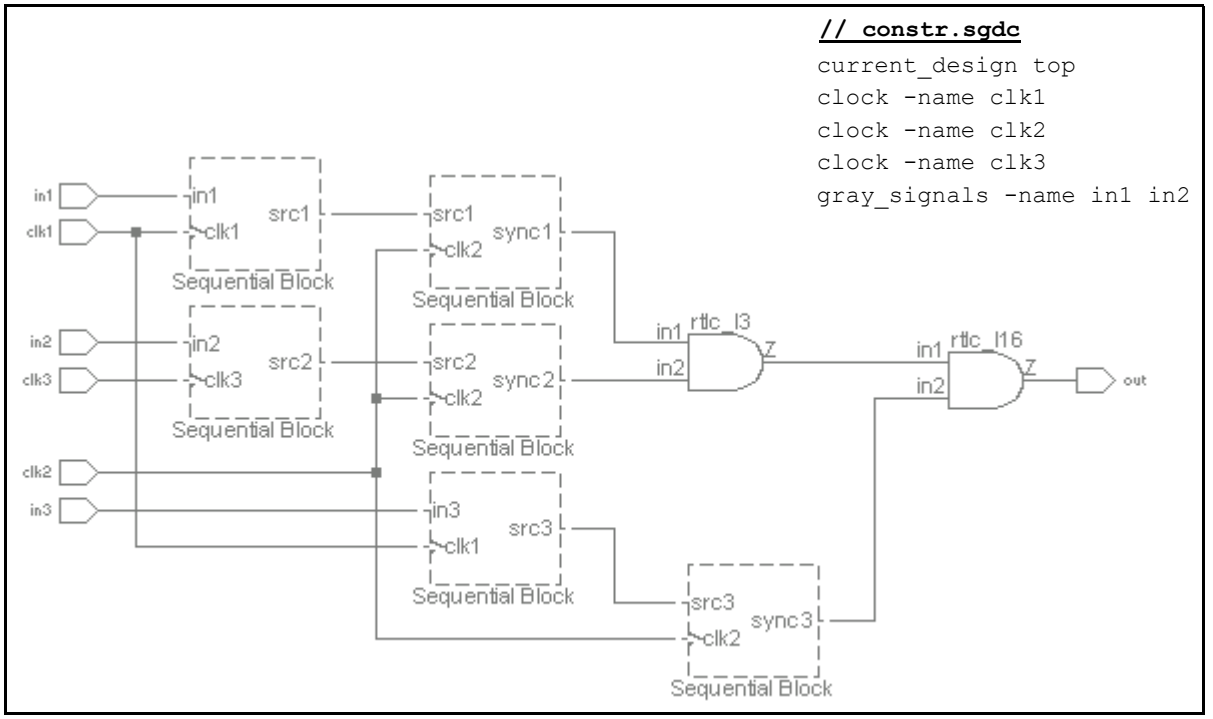


FIGURE 427. Schematic of the SGDC_gray_signals02 Rule Violation

In the above example, the `in1` and `in2` signals are not driven by any clock. Therefore, the *SGDC_gray_signals02* rule reports a violation for these signals.

NOTE: *The SGDC_gray_signals02 rule checks the fan-in cone of each signal. If any*

sequential cell is found in the fan-in cone, this rule checks if any clock signal drives that sequential cell. If no such sequential is found, this rule reports a violation.

Rule Group

Non fatal must rule

Default Severity Label

Error

Reports and Related Files

No report or related file

SGDC_gray_signals03

Checks if the signals specified by the gray_signals constraint are in the same clock domain

When to Use

Use this rule to check the signals specified to the *gray_signals* constraint.

Prerequisites

Use the `gray_signals` constraint to specify the signals that should be gray encoded.

Description

The *SGDC_gray_signals03* rule reports a violation in the following cases:

- If the signals specified by the *gray_signals* constraint are not in the same clock domain.
- If a signal specified by the *gray_signals* constraint is a part of multiple clock domains.

Parameter(s)

None

Constraint(s)

gray_signals (Mandatory): Use this constraint to specify signals that should be gray encoded.

Messages and Suggested Fix

Message 1

The following message appears if the signal *<signal-name>* is a part of multiple clock domains:

```
[SGs1_1] [ERROR] Constraint 'gray_signals': signal '<signal-
name>' is sampled by multiple clocks '<clock1>' and '<clock2>'
```

Potential Issues

This violation appears if the signal specified by the *gray_signals* constraint is

a part of multiple clock domains.

Consequences of Not Fixing

If you do not fix this violation, the reported signal is not checked for gray encoding.

How to Debug and Fix

To fix this violation, modify the design so that the reported signal is a part of a single clock domain.

Message 2

The following message appears if the signals specified by the *gray_signals* constraint are not in the same clock domain:

```
[SGs1_2] [ERROR] Constraint 'gray_signals': Multiple clock
domains found: signal '<signal-name1>' clocked by '<clock1>'
and signal '<signal-name2>' clocked by '<clock2>'
```

Potential Issues

This violation appears if the signals specified by the *gray_signals* constraint are not in the same clock domain

Consequences of Not Fixing

If you do not fix this violation, the reported signals are not checked for gray encoding.

How to Debug and Fix

To fix this violation, modify the design so that the reported signals are in the same clock domain.

Example Code and/or Schematic

Consider the following schematic of a design and the SGDC file specified for that design:

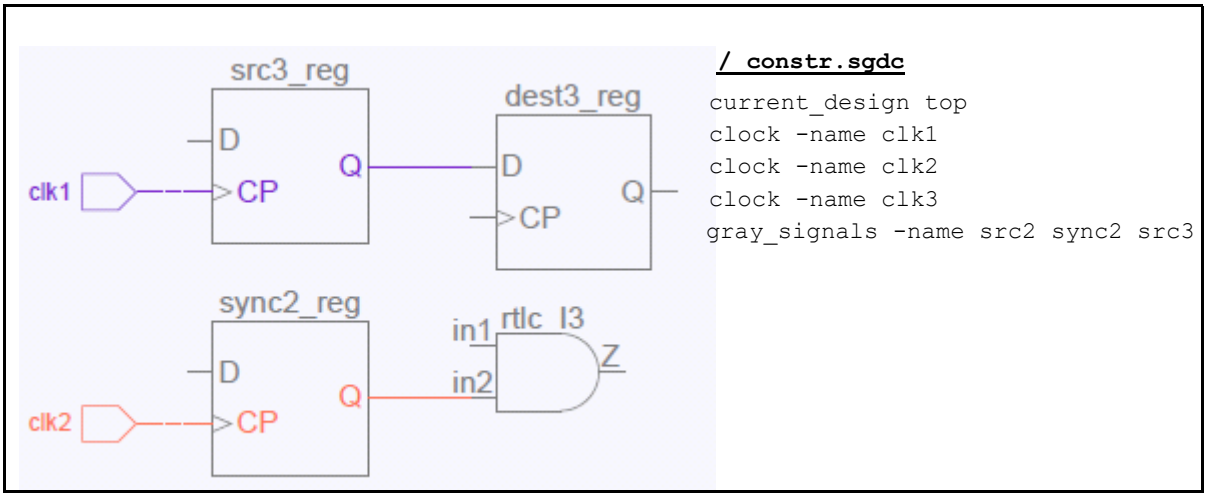


FIGURE 428. Schematic of the SGDC_gray_signals03 Rule Violation

In the above example, the `src3` signal is driven by the `clk1` clock and the `sync2` signal is driven by the `clk2` clock. Therefore, the *SGDC_gray_signals03* rule reports a violation for these signals.

Rule Group

Non fatal must rule

Default Severity Label

Error

Reports and Related Files

No report or related file

SGDC_input01

Reports a violation if a non-existing object is specified in the -name argument of the input constraint

When to Use

Use this rule to perform sanity checks on the *input* constraint.

Prerequisites

Specify the *input* constraint.

Description

The *SGDC_input01* rule reports a violation if the object specified in the `-name` argument of the *input* constraint does not exist as a port, net, or hierarchical pin in the current design.

Parameter(s)

None

Constraint(s)

input (Mandatory): Use this constraint to specify clock domain at input ports.

Messages and Suggested Fix

The following message appears if the object *<signal-name>* specified in the `-name` argument of the *input* constraint does not exist as a port, net, or hierarchical pin in the current design *<design-name>*:

[FATAL] '<signal-name>'[TopPort + Net + HierTerminal] not found on/within module '<design-name>'

Potential Issues

This violation appears if the current design does not contain the object specified by the `-name` argument of the *input* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, update the value of the `-name` argument of the *input* constraint to specify a name that exists as a port, net, or hierarchical pin in the design.

Example Code and/or Schematic

Consider an example in which the design `top` does not contain any port, net, or hierarchical pin by the name `in3`.

Now consider that you specify the following constraints in an SGDC file:

```
current_design top

clock -name clk1
input -name in3 -clock clk1
```

In this case, the *SGDC_input01* rule reports a violation because the nonexistent object `in3` is specified in the `-name` argument of the *input* constraint.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_input02

Reports a violation if a non-existent port or net is specified in the -clock argument of the input constraint

When to Use

Use this rule to perform sanity checks on the *input* constraint.

Prerequisites

Specify the *input* constraint.

Description

The *SGDC_input02* rule reports a violation if the clock name specified in the `-clock` argument of the *input* constraint is a non-hierarchical name, and it does not exist as a port or a net in the current design.

Parameter(s)

None

Constraint(s)

input (Mandatory): Use this constraint to specify a clock domain at input ports.

Messages and Suggested Fix

The following message appears if the object *<clock-name>* specified by the `-clock` argument of the *input* constraint does not exist in the design *<design-name>*:

```
[INFO] '<clock-name>' not found on/within module
'<design-name>'. Considering it as virtual clock
```

Potential Issues

This violation appears if the design does not contain a port or net specified by the `-clock` argument of the *input* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported clock is considered as a virtual clock, and cross-domain crossing checks are done with the same assumption. This may not be as per your expectations.

How to Debug and Fix

To debug and fix this violation, analyze the *input* constraint specification for ports driven from virtual clocks. If required, update the `-clock` argument of the *input* constraint to specify an existing clock.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
clock -tag clk3
input -name in1 -clock clk3
```

In the above example, `clk3` is a virtual clock. In this case, synchronization checks are performed for the `in1` port with respect to `clk3` domain.

Default Severity Label

Info

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_input03

Reports if an invalid port or net is specified in the -clock argument of the input constraint

When to Use

Use this rule to perform sanity checks on the *input* constraint.

Prerequisites

Specify the *input* constraint.

Description

The *SGDC_input03* rule reports a violation if the clock name specified by the `-clock` argument of the *input* constraint is a hierarchical name that does not exist as a port or a net in the current design.

Parameter(s)

None

Constraint(s)

input (Mandatory): Use this constraint to specify a clock domain at input ports.

Messages and Suggested Fix

The following message appears if the signal *<signal-name>* specified by the `-clock` argument of the *input* constraint is a hierarchical name that does not exist in the module *<module-name>*:

```
[FATAL] '<signal-name>' not found on/within module
'<module-name>'
```

Potential Issues

This violation appears if your design does not contain a port or net specified by the `-clock` argument of the *input* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port or net in the `-clock` argument of the *input* constraint.

Example Code and/or Schematic

Consider the design `top` that does not contain the `top1.clk2` port/net. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
input -name "in[1]" -clock "top1.clk2"
```

In the above case, the *SGDC_input03* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_input05

Conflicting input constraints specified for a path

When to Use

Use this rule to detect cases in which an *input* constraint overrides the domain of a net.

Prerequisites

Specify the *input* constraint.

Description

The *SGDC_input05* rule reports a violation if multiple *input* constraints of different clock domains are specified on internal nets present on the same path.

Parameter(s)

None

Constraint(s)

input (Mandatory): Use this constraint to specify a clock domain at input ports.

Messages and Suggested Fix

The following message appears if

```
[WARNING] Constraint 'input': Conflicting specifications
between '<net1>' (Clock: '<clock1>') and '<net2>' ('<clock2>')
present in the same path
```

The arguments of the above message are explained below:

Argument	Description
<net1>	Refers to the name of the intermediate net for which the *input* constraints are applied.
<clock1>	Refers to the name of the clock with which *<net1>* is constrained.

Argument	Description
<net2>	Refers to the name of the stop point for which all the following conditions are true: • It is a terminal of a sequential element, black box, library cell, port, abstract port, or any other net where a different clock domain is found. • It is present in the fan-in of <*net1*>.
<clock2>	Refers to the name of the clock reaching <*net2*> when the domain of this clock is different from that of <*clock1*>

Potential Issues

This violation appears if your design contains nets (on the same path) for which *input* constraints of different clock domains are specified.

Consequences of Not Fixing

If you do not fix this violation, the clock domain crossings reported by *The Ac_sync_group Rules* contains incorrect clock names.

How to Debug and Fix

To fix this violation, remove the conflicting constraint specifications from the SGDC file.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design "test"
clock -name clk1 -tag C1  -domain d1
clock -name clk2 -tag C2  -domain d2
input -name "test.q2_tmp" -clock "clk2"
input -name "test.q3_tmp" -clock "clk1"
```

For the above example, the *SGDC_input05* rule reports a violation when `test.q2_tmp` is in the fan-out cone of `test.q3_tmp` (or vice-versa), as shown in the following schematic:

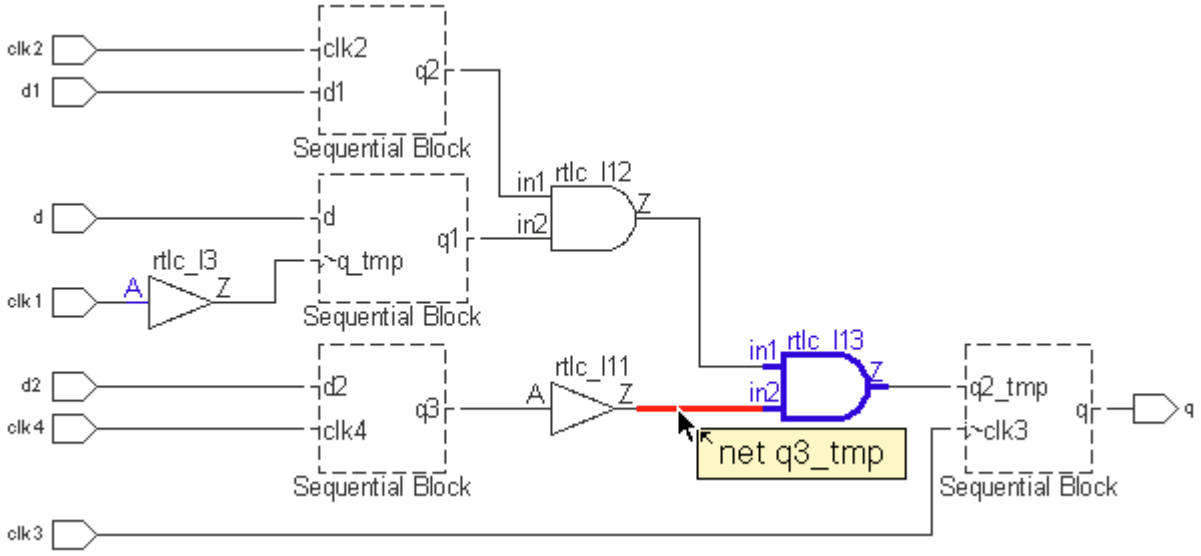


FIGURE 429. SGDC_input05 - Example

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_inputoutput01

The input / output constraint is defined on internal nets

When to Use

Use this rule to check invalid specification of *input* or *output* constraint.

Description

The *SGDC_inputoutput01* rule reports a violation if:

- The *input* constraint is defined on an internal net or an output port.
- The *output* constraint is defined on an internal net or an input port.

Parameter(s)

None

Constraint(s)

- *input* (Optional): Use this constraint to specify clock domain at input ports.
- *output* (Optional): Use this constraint to specify a clock domain at output ports.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] '<input | output>' constraint not specified on <input
| output> or inout ports. Ignoring the constraint
```

Potential Issues

This violation appears if:

- The *input* constraint is defined on an internal net or an output port.
- The *output* constraint is defined on an internal net or an input port.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from

SpyGlass analysis.

How to Debug and Fix

To fix this violation:

- Specify the *input* constraint on an input or inout port.
- Specify the *output* constraint on an output or inout port.

Example Code and/or Schematic

Consider the following files specified for SpyGlass analysis:

// test.v

```
module test (d,d1,d2,clk1,clk2,clk3,
             clk4,q);
input d,d1,d2,clk1,clk2,clk3,clk4;
output q;
wire d,clk1,clk2,q_tmp,q1_tmp,
       q2_tmp,q3_tmp;
reg q1,q2,q3,q;
assign q_tmp=clk1;
always@(posedge q_tmp)
begin
  q1 <= d;
end
always@(negedge clk2)
begin
  q2 <=d1;
end
always@(negedge clk4)
begin
  q3 <= d2;
end
assign q3_tmp = q3;
assign q1_tmp = q2 & q1;
assign q2_tmp = q1_tmp & q3_tmp;
always@(posedge clk3)
begin
  q <= q2_tmp;
end
endmodule
```

// constr.sgdc

```
current_design test

clock -name clk1 -tag C1 -domain d1
clock -name clk2 -tag C2 -domain d2
clock -name clk3 -tag C3 -domain d3
clock -name clk4 -tag C4 -domain d4

current_design "test"
input -name "test.q2_tmp" -clock "clk2"
input -name "test.q3_tmp" -clock "clk1"
input -name "test.q_tmp" -clock "clk4"

current_design "test"
abstract_port -module test -ports clk1
              -clock "clk3"

current_design "test"
input -name "test.q2" -clock "clk4"
```

In the above example, none of the *input* constraints is specified on an input or inout port. Therefore, the *SGDC_inputoutput01* rule reports a violation for each *input* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_network_allowed_cells01

Reports a violation if an invalid value is specified in the -type argument of the network_allowed_cells constraint

When to Use

Use this rule to perform sanity checks on the *network_allowed_cells* constraint.

Prerequisites

Specify the *network_allowed_cells* constraint.

Description

The *SGDC_network_allowed_cells01* rule reports a violation if the value specified in the `-type` argument of the *network_allowed_cells* constraint is other than `clock`, `reset`, or `clock reset`.

Parameter(s)

None

Constraint(s)

network_allowed_cells (Mandatory): Use this constraint to specify cells that should be allowed or disallowed in clock trees.

Messages and Suggested Fix

The following message appears if an invalid value is specified in the `-type` argument of the *network_allowed_cells* constraint:

```
[FATAL] Invalid [Pre-defined] specification '<name>' for '-type' field of constraint 'network_allowed_cells'
```

Potential Issues

This violation appears if the value specified in the `-type` argument of the *network_allowed_cells* constraint is other than `clock`, `reset`, or `clock reset`.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify `clock`, `reset`, or `clock reset` in the `-type` argument of the *network_allowed_cells* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
network_allowed_cells -name NR2 -type dummy
```

For the above constraint, the *SGDC_network_allowed_cells01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_network_allowed_cells02

Reports a violation if a non-existing net is specified in the -from argument of the network_allowed_cells constraint

When to Use

Use this rule to perform sanity checks on the *network_allowed_cells* constraint.

Prerequisites

Specify the *network_allowed_cells* constraint.

Description

The *SGDC_network_allowed_cells02* rule reports a violation if a non-existing net is specified in the `-from` argument of the *network_allowed_cells* constraint.

Parameter(s)

None

Constraint(s)

network_allowed_cells (Mandatory): Use this constraint to specify cells that should be allowed or disallowed in clock trees.

Messages and Suggested Fix

The following message appears if the net *<net-name>* specified in the `-from` argument of the *network_allowed_cells* constraint does not exist in the design *<design-name>*:

[FATAL] '<net-name>'[TopPort + Net] not found on/within module '<design-name>'

Potential Issues

This violation appears if the design does not contain the net specified by the `-from` argument of the *network_allowed_cells* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port or net in the `-from` argument of the *network_allowed_cells* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the `w1` net.

Now consider that you specify the following constraint:

```
current_design top
network_allowed_cells -name A1234 -from w1 -allow
```

In this case, the *SGDC_network_allowed_cells02* rule reports a violation because the non-existing net w1 is specified in the `-from` argument of the *network_allowed_cells* constraint.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_noclockcell01

Reports a violation if an invalid object is specified in the -name argument of the noclockcell_start constraint

When to Use

Use this rule to perform sanity checks on the *noclockcell_start* constraint.

Prerequisites

Specify the *noclockcell_start* constraint.

Description

The *SGDC_noclockcell01* rule reports a violation if the object specified in the `-name` argument of the *noclockcell_start* constraint does not exist as port or a hierarchical net in the current design.

Parameter(s)

None

Constraint(s)

noclockcell_start (Optional): Use this constraint to specify start points, such as ports or nets for rule-checking.

Messages and Suggested Fix

The following message appears of the port or net *<object-name>* specified by the `-name` argument of the *noclockcell_start* constraint does not exist in the current design *<current-design>*:

[FATAL] '<object-name> '[TopPort + Net] not found on/within module '<current-design>'

Potential Issues

This violation appears if the current design does not contain the port or net specified by the `-name` argument of the *noclockcell_start* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing port or net in the `-name` argument of the *noclockcell_start* constraint.

Example Code and/or Schematic

Consider that the design top does not contain the port/net `clk1`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
noclockcell_start -name top.clk1
```

For the above example, the *SGDC_noclockcell01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_noclockcell03

Reports a violation if a non-existing module is specified in the -name argument of the noclockcell_stop_module constraint

When to Use

Use this rule to perform sanity checks on the *noclockcell_stop_module* constraint.

Prerequisites

Specify the *noclockcell_stop_module* constraint.

Description

The *SGDC_noclockcell03* rule reports a violation if the module specified by the `-name` argument of the *noclockcell_stop_module* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

noclockcell_stop_module (Optional): Use this constraint to specify a 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.

Messages and Suggested Fix

The following message appears if the module *<module-name>* specified by the `-name` argument of the *noclockcell_stop_module* constraint does not exist in the current design *<current-design>*:

```
[FATAL] '<module-name>' [SubModule] is never instantiated
within environment '<current-design>
```

Potential Issues

This violation appears if your design does not contain the module specified by the `-name` argument of the *noclockcell_stop_module* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing module in the `-name` argument of the *noclockcell_stop_module* constraint.

Example Code and/or Schematic

Consider the design `top` that does not contain the module `DF1`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
noclockcell_stop_module -name DF1
```

In the above case, the *SGDC_noclockcell03* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_noclockcell04

Reports a violation if a non-existing instance is specified in the -name argument of the noclockcell_stop_instance constraint

When to Use

Use this rule to perform sanity checks on the *noclockcell_stop_instance* constraint.

Prerequisites

Specify the *noclockcell_stop_instance* constraint.

Description

The *SGDC_noclockcell04* rule reports a violation if the instance specified by the `-name` argument of the *noclockcell_stop_instance* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

noclockcell_stop_instance (Optional): Use this constraint to specify an instance where the *NoClockCell* rule should stop further traversal along the clock tree when the clock pin of the specified instance is hit.

Messages and Suggested Fix

The following message appears if the instance *<inst-name>* specified by the `-name` argument of the *noclockcell_stop_instance* constraint does not exist in the current design *<current-design>*:

```
[FATAL] '<inst-name>' [Instance] not found on/within module
'<current-design>
```

Potential Issues

This violation appears if your design does not contain the instance specified by the `-name` argument of the *noclockcell_stop_instance* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing instance in the `-name` argument of the *noclockcell_stop_instance* constraint.

Example Code and/or Schematic

Consider the design `top` that does not contain the instance `U_DF1`. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
noclockcell_stop_instance -name U_DF1
```

In the above case, the *SGDC_noclockcell04* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops01

Reports a violation if no argument is specified in the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops01* rule reports a violation if you do not specify any argument to the *num_flops* constraint.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if you do not specify any argument to the *num_flops* constraint:

```
[FATAL] It is Mandatory to Specify At Least One Value for
combination of fields '-from_clk -to_clk -from_domain -
to_domain -to_period -default ' of constraint 'num_flops'
```

Potential Issues

This violation appears if you do not specify any argument to the *num_flops* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify at least one of the following arguments for the *num_flops* constraint:

- `-from_clk` and `-to_clk`
- `-to_clock`
- `-from_domain` and `-to_domain`
- `-to_period`
- `-default`

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops03a

Existence check for non-hierarchical name with '-from_clk' field of constraint 'num_flops'

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops03a* rule reports a violation if a non-hierarchical net name or a port name specified in the `-from_clk` argument of the *num_flops* constraint is not found in the top-level module.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

This rule reports the following message:

```
[INFO] Constraint 'num_flops':clock '<net-name>' specified with
'-from_clk' not found on/within module '<module-name>'.
Considering it as a virtual clock
```

Potential Issues

This violation appears if a non-hierarchical net name or a port name specified in the `-from_clk` argument of the *num_flops* constraint is not found in the top-level module.

Consequences of Not Fixing

If you do not fix this violation, the specified net name or port name is considered as a virtual clock. As a result, SpyGlass may report spurious synchronization violations.

How to Debug and Fix

To fix this violation, perform the following actions:

- Analyze the *num_flops* constraint specification.
- Specify virtual clock names only when input signals are triggered from a virtual clock domain outside the design scope.

Example Code and/or Schematic

Consider the following constraints specifications:

```
input -name in1 -clock vclk1
num_flops -from_clk vclk1 -to_clk clk2 -value 3
```

In this example, all clock-domain crossings from the `in1` port to the `clk2` clock require at least three flip-flops for synchronization results.

In this case, the *SGDC_numflops03a* rule reports a violation for *num_flops* constraint.

Default Severity Label

Info

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

The CKSGDCInfo Report

SGDC_numflops03b

Reports a violation if an invalid hierarchical net or pin name is specified in the -from_clk argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops03b* rule reports a violation if the hierarchical net or pin name specified by the `-from_clk` argument of the *num_flops* constraint does not exist in the design.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if the hierarchical net name specified by the `-from_clk` argument of the *num_flops* constraint is missing in the module *<module-name>*:

```
[FATAL] Constraint 'num_flops':clock 'hierarchical net name' specified with '-from_clk' not found on/within module '<module-name>'
```

Potential Issues

This violation appears if the hierarchical net or pin name specified by the `-from_clk` argument of the *num_flops* constraint does not exist in the design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net or a pin in the `-from_clk` argument of the *num_flops* constraint.

Example Code and/or Schematic

Consider that your design does not contain any clock by the name `ck3`.

In this case, the *SGDC_numflops03b* rule reports a violation if you specify the following *num_flops* constraint specification in the SGDC file:

```
num_flops -from_clk top.ck3 -to_clk ck1 -value 3
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops03c

Reports a violation if an invalid clock is specified in the -from_clk argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops03c* rule reports a violation if the clock specified in the `-from_clk` argument of the *num_flops* constraint is not one of the specified clock nor it is an automatically-inferred clock.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if you specify an invalid clock in the `-from_clk` argument of the *num_flops* constraint:

```
[WARNING] Constraint 'num_flops': Clock name '<clock-name>'
specified in field '-from_clk' is not a valid clock
```

Potential Issues

This violation appears if the clock specified in the `-from_clk` argument of the *num_flops* constraint is not one of the specified clock nor it is an automatically-inferred clock.

Consequences of Not Fixing

If you do not fix this violation, the reported *num_flops* constraint specification is ignored.

How to Debug and Fix

To fix this violation, specify any of the following clocks in the `-from_clk` argument of the *num_flops* constraint:

- Clocks specified by the *clock* constraint
- Automatically-inferred clocks when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Consider that the `ck1` clock is not specified by the *clock* constraint nor it is an automatically-inferred clock.

In this case, the *SGDC_numflops03c* rule reports a violation if you specify the following *num_flops* constraint specification in the SGDC file:

```
current_design top
clock -name ck2 -domain d2
num_flops -from_clk ck1 -to_clk ck2 -value 3
```

Default Severity Label

Warning

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_numflops04

Reports a violation if an invalid clock is specified in the -to_clk argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops03c* rule reports a violation if the clock specified in the `-to_clk` argument of the *num_flops* constraint is not one of the specified clock nor it is an automatically-inferred clock.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if you specify an invalid clock in the `-to_clk` argument of the *num_flops* constraint:

```
[WARNING] Constraint 'num_flops': Clock name '<clock-name>' specified in field '-to_clk' is not a valid clock
```

Potential Issues

This violation appears if the clock specified in the `-to_clk` argument of the *num_flops* constraint is not one of the specified clock nor it is an automatically-inferred clock.

Consequences of Not Fixing

If you do not fix this violation, the reported *num_flops* constraint specification is ignored.

How to Debug and Fix

To fix this violation, specify any of the following clocks in the `-to_clk` argument of the *num_flops* constraint:

- Clocks specified by the *clock* constraint
- Automatically-inferred clocks when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Consider that the `ck2` clock is not specified by the *clock* constraint nor it is an automatically-inferred clock.

In this case, the *SGDC_numflops04* rule reports a violation if you specify the following *num_flops* constraint specification in the SGDC file:

```
current_design top
clock -name ck1 -domain d1
num_flops -from_clk ck1 -to_clk ck2 -value 3
```

Default Severity Label

Warning

Rule Group

`SOC_SGDCVALIDATION`

Reports and Related Files

No report or related file

SGDC_numflops05

Reports if the domain specified by the -from_domain argument of the num_flops constraint does not exist

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *num_flops* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By setting the *use_inferred_clocks* parameter to `yes` to use automatically generated clock signals
  - By using a combination of both the above methods

Description

The *SGDC_numflops05* rule reports a violation if the domain specified by the `-from_domain` argument of the *num_flops* constraint does not exist as a domain of a clock specified by the *clock* constraint or an automatically-inferred clock.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *num_flops* (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if you specify an incorrect domain in the `-from_domain` argument of the *num_flops* constraint:

```
[WARNING] Constraint 'num_flops': Incorrect domain
'<domain-name>' specified in field '-from_domain'
```

Potential Issues

This violation appears if the domain specified by the `-from_domain` argument of the *num_flops* constraint does not exist as a domain of any clock in a design.

Consequences of Not Fixing

If you do not fix this violation, the reported *num_flops* constraint is ignored during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify a correct domain in the `-from_domain` argument of the *num_flops* constraint so that the specified domain matches with the domain of a clock in the design.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design top
clock -name clk1 -domain D1 -period 15
clock -name clk2 -domain D2 -period 20
num_flops -from_domain D! -to_domain D2 -value 10
```

In the above example, the `D!` domain specified in the `-from_domain` argument of the *num_flops* constraint does not exist as a domain of any clock in the design. Therefore, the *SGDC_numflops05* rule reports a violation.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops06

Reports if the domain specified by the -to_domain argument of the num_flops constraint does not exist

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *num_flops* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By setting the *use_inferred_clocks* parameter to `yes` to use automatically generated clock signals
  - By using a combination of both the above methods

Description

The *SGDC_numflops06* rule reports a violation if the domain specified by the `-to_domain` argument of the *num_flops* constraint does not exist as a domain of a clock specified by the *clock* constraint or an automatically-inferred clock.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *num_flops* (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if you specify an incorrect domain in the `-to_domain` argument of the *num_flops* constraint:

```
[WARNING] Constraint 'num_flops': Incorrect domain
'<domain-name>' specified in field '-to_domain'
```

Potential Issues

This violation appears if the domain specified by the `-to_domain` argument of the *num_flops* constraint does not exist as a domain of any clock in a design.

Consequences of Not Fixing

If you do not fix this violation, the reported *num_flops* constraint is ignored during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify a correct domain in the `-to_domain` argument of the *num_flops* constraint so that the specified domain matches with the domain of a clock in the design.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design top
clock -name clk1 -domain D1 -period 15
clock -name clk2 -domain D2 -period 20
num_flops -from_domain D1 -to_domain D -value 10
```

In the above example, the `D` domain specified in the `-to_domain` argument of the *num_flops* constraint does not exist as a domain of any clock in the design. Therefore, the *SGDC_numflops05* rule reports a violation.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops07

Reports if an incorrect value is specified in the -to_period argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops07* rule reports a violation if the value specified in the `-to_period` argument of the *num_flops* constraint is not a float value or it is a negative float value.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if an incorrect value *<value>* is specified in the `-to_period` argument of the *num_flops* constraint:

[FATAL] `Invalid [Pre-defined-range] specification '<value>' for '-to_period' field of constraint 'num_flops'`

Potential Issues

This violation appears if the value specified in the `-to_period` argument of the *num_flops* constraint is not a float value or it is a negative float value.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a float value in the `-to_period` argument of the *num_flops* constraint.

Example Code and/or Schematic

Consider the following *num_flops* constraint specification:

```
num_flops -to_period TEN -value 10
```

For the above constraint, the *SGDC_numflops07* rule reports a violation because the value `TEN` specified in the `-to_period` argument is not a float value.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops08

Reports if an incorrect value is specified in the -value argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops08* rule reports a violation if the value specified by the `-value` argument of the *num_flops* constraint is not an integer value or it is an integer value less than 1.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if the value specified by the `-value` argument of the *num_flops* constraint is not an integer value or it is an integer value less than 1:

```
[FATAL] Constraint 'num_flops': Number of required flip-flops specified with '-value' can only be greater than 0
```

Potential Issues

This violation appears if the value specified by the `-value` argument of the *num_flops* constraint is not an integer value or it is an integer value less than 1.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an integer value greater or equal to 1 in the `-value` argument of the *num_flops* constraint.

Example Code and/or Schematic

Consider the following *num_flops* constraint specification:

```
num_flops -from_clk clk1 -to_clk clk2 -value 0
```

For the above constraint, the *SGDC_numflops08* rule reports a violation because the value specified in the `-value` argument is less than 1.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops09

Reports if an incorrect value is specified in the -default argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops09* rule reports a violation if the value specified by the `-default` argument of the *num_flops* constraint is not an integer value or it is an integer value less than 2.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if the value *<value>* specified by the `-default` argument of the *num_flops* constraint is not an integer value or it is an integer value less than 2:

[FATAL] `Invalid [Pre-defined-range] specification '<value>' for '-default' field of constraint 'num_flops'`

Potential Issues

This violation appears if the value specified by the `-default` argument of the *num_flops* constraint is not an integer value or it is an integer value less than 2.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an integer value greater or equal to 2 in the `-default` argument of the *num_flops* constraint.

Example Code and/or Schematic

Consider the following *num_flops* constraint specification:

```
num_flops -default 1
```

For the above constraint, the *SGDC_numflops09* rule reports a violation because the value specified in the `-default` argument is less than 2.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops10

Reports a violation if the -value and -default arguments of the num_flops constraint are specified together

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops10* reports a violation if the `-value` and `-default` arguments of the *num_flops* constraint are specified together.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears if the `-value` and `-default` arguments of the *num_flops* constraint are specified together.

```
[WARNING] Constraint 'num_flops': -value field will be ignored due to -default specification
```

Potential Issues

This violation appears if the `-value` and `-default` arguments of the *num_flops* constraint are specified together.

Consequences of Not Fixing

If you do not fix this violation, the `-value` argument is ignored. This may not be as per your expectation.

How to Debug and Fix

To fix this violation, specify either `-value` or `-default` argument with the *num_flops* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
num_flops -default 2 -value 4
```

For the above constraint, the *SGDC_numflops10* reports a violation because both the `-value` and `-default` arguments are specified with this constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops11

Reports a violation for overlapping specifications of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops11* reports a violation if *num_flops* constraint specifications are logically overlapping.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears to indicate overlapping specifications of the *num_flops* constraint:

```
[WARNING] Constraint 'num_flops': Overlapping Constraint specifications found. Latest specification will be used for rule-checking
```

Potential Issues

This violation appears if you specify overlapping specifications of the *num_flops* constraint.

Consequences of Not Fixing

If you do not fix this violation, the last *num_flops* constraint specification

overrides the previous specification. This may not be as per your expectations.

How to Debug and Fix

To fix this violation, ensure to specify only one value for a given clock pair and remove all the other overlapping specifications.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
clock -name clk1 -domain d1
clock -name clk2 -domain d2
num_flops -from_clk clk1 -to_clk clk2 -value 3
num_flops -from_domain d1 -to_domain d2 -value 4
```

In the above example, the `clk1` and `clk2` clocks belong to the `d1` and `d2` domain, respectively.

Therefore, the *num_flops* constraint specifications are logically overlapping in this case.

As a result, SpyGlass considers the last *num_flops* constraint specification. Therefore, the number of flip-flops required for multi-flops synchronization for a crossing between `clk1` and `clk2` is considered as 4.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_numflops13

Checks the -lib argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops13* rule reports a violation message when the library name specified with the `-lib` argument of the *num_flops* constraint is not available in the current design run.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears when you specify a library name *<lib-name>* that is not available in the current SpyGlass run:

```
[WARNING] Constraint 'num_flops': library '<lib-name>' not passed in current run
```

Potential Issues

This violation appears if you specify library names that are not available in the current SpyGlass run.

Consequences of Not Fixing

If you do not fix this violation, the constraint is ignored. This may not be as per your expectation.

How to Debug and Fix

To fix this violation, specify a valid library name in the `-lib` argument of

the *num_flops* constraint.

Example Code and/or Schematic

For the following *num_flops* constraint specification, the *SGDC_numflops13* rule reports a violation if the LIB1 library specified in the `-lib` argument is not available in the current SpyGlass run.

```
num_flops -to_period 50 -value 3 -lib LIB1
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

The CKSGDCInfo Report

SGDC_numflops14

Checks the -cell argument of the num_flops constraint

When to Use

Use this rule to perform sanity checks on the *num_flops* constraint.

Prerequisites

Specify the *num_flops* constraint.

Description

The *SGDC_numflops14* rule reports a violation message when the library cell specified in the `-cell` argument of the *num_flops* constraint is not located in the libraries list specified in the `-lib` argument.

Parameter(s)

None

Constraint(s)

num_flops (Mandatory): Use this constraint to specify a minimum number of flip-flops required in a synchronizer chain.

Messages and Suggested Fix

The following message appears when you specify a cell name *<cell-name>* that is not present in the specified library:

```
[WARNING] Constraint 'num_flops': Cell '<cell-name>' not found in the specified library
```

Potential Issues

This violation appears if you specify cell names that are not present in the specified library.

Consequences of Not Fixing

If you do not fix this violation, the constraint is ignored. This may not be as per your expectation.

How to Debug and Fix

To fix this violation, check the values specified in the `-cells` and `-lib`

arguments of the *num_flops* constraint. The cells must exist in the specified library list.

Example Code and/or Schematic

For the following *num_flops* constraint specification, the *SGDC_numflops14* rule reports a violation if the `FD1` cell is not present in the `LIB1` library.

```
num_flops -to_period 50 -value 3 -lib LIB1 -cells FD1
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

The CKSGDCInfo Report

SGDC_noclockcell02

Reports a violation if invalid objects are specified by the -name argument of the noclockcell_stop_signal constraint

When to Use

Use this rule to perform sanity checks on the *noclockcell_stop_signal* constraint.

Prerequisites

Specify the *noclockcell_stop_signal* constraint.

Description

The *SGDC_noclockcell02* rule reports a violation if the object specified by the `-name` argument of the *noclockcell_stop_signal* constraint does not exist as a port or a net in the current design.

Parameter(s)

None

Constraint(s)

noclockcell_stop_signal (Mandatory): Specifies design points, such as ports, pins, or nets at which the *NoClockCell* rule-traversal should stop along a clock tree.

Messages and Suggested Fix

The following message appears if the object specified by the `-name` argument of the *noclockcell_stop_signal* constraint does not exist as a port or a net in the current design:

```
[FATAL] '<object-name>' [top port + net + hier terminal] not found on/within module '<current-design>'
```

Potential Issues

This violation appears if the current design does not contain any object specified by the `-name` argument of the *noclockcell_stop_signal* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, update the value of the `-name` argument of the *noclockcell_stop_signal* constraint to specify objects that exist as a port or a net.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.v
module flop(d,clk,q);
 input d,clk;
 output q;
 reg q;
 wire d;
 always @(posedge clk)
  q<=d;
endmodule

module top(d,clk,q);
 input d,clk;
 output [0:1]q;
 wire clk1;
 wire [0:1]w;
 assign clk1 = clk;
 flop F1(d,clk,w[0]);
 flop F2(w[0],clk1,q[0]);
 flop F3(d,clk,w[1]);
 flop F4(w[1],clk1,q[1]);
endmodule
```

```
// constr.sgdc
current_design top
noclockcell_stop_signal -name w[2]
```

For the above example, the *SGDC_noclockcell02* rule reports a violation because the `w[2]` object specified by the *noclockcell_stop_signal* constraint does not exist in the `top` module.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_output01

Reports a violation if a non-existent object is specified in the -name argument of the output constraint

When to Use

Use this rule to perform sanity checks on the *output* constraint.

Prerequisites

Specify the *output* constraint.

Description

The *SGDC_output01* rule reports a violation if the object specified in the `-name` argument of the *output* constraint does not exist as a port or net in the current design.

Parameter(s)

None

Constraint(s)

output (Mandatory): Use this constraint to specify clock domains at output ports.

Messages and Suggested Fix

The following message appears if the object *<object-name>* specified by the `-name` argument of the *output* constraint does not exist in the design *<design-name>*:

[FATAL] '<object-name>'[TopPort + Net] not found on/within module '<design-name>'

Potential Issues

This violation appears if your design does not contain a port or net specified by the `-name` argument of the *output* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port or net in the `-name` argument of the *output* constraint.

Example Code and/or Schematic

Consider the design `top` that does not contain any port or net by the name `out1`. Now consider that you specify the following constraints:

```
current_design top
output –name out1 –clock clk2
```

For the above example, the *SGDC_output01* rule reports a violation because the non-existing object `out1` is specified in the `-name` argument of the *output* constraint.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_output02

Reports invalid non-hierarchical names specified to the -clock argument of the output constraint

When to Use

Use this rule to perform sanity checks on the *output* constraint.

Prerequisites

Specify the *output* constraint.

Description

The *SGDC_output02* rule reports a violation if the object specified to the `-clock` argument of the *output* constraint is a non hierarchical name and that object does not exist as a port or net in the current design.

Parameter(s)

None

Constraint(s)

output (Mandatory): Use this constraint to specify clock domains at output ports.

Messages and Suggested Fix

The following message appears if an invalid non-hierarchical name (port name or net name) is specified to the `-clock` argument of the *output* constraint:

`[INFO] '<object-name>' not found on/within module '<module-name>'.Considering it as virtual clock"`

Potential Issues

This violation appears if the design does not contain a port or net specified by the `-clock` argument of the *output* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported clock is considered as a virtual clock, and cross-domain crossing checks are done with the same assumption. This may not be as per your expectations.

How to Debug and Fix

To debug and fix this violation, analyze the *output* constraint specification for ports driven from virtual clocks. If required, update the `-clock` argument of the *output* constraint to specify an existing clock.

Example Code and/or Schematic

Consider the design `top` in which the `ck1` clock does not exist. In this case, the *SGDC_output02* rule reports a violation if you specify the following *output* constraint specification in the SGDC file:

```
current_design top
clock -tag ck1
output -name OUT1 -clock ck1
```

Default Severity Label

Info

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_output03

Reports inout ports for which both input and output constraints are specified

When to Use

Use this rule to check for any incorrect usage of the *input* and *output* constraints.

Prerequisites

Specify the *input* and *output* constraints.

Description

The *SGDC_output03* rule reports a violation if both the *input* and *output* constraints are specified for an inout port.

Parameter(s)

None

Constraint(s)

- *output* (Mandatory): Use this constraint to specify clock domains at output ports.
- *input* (Mandatory): Use this constraint to specify clock domain at input ports.

Messages and Suggested Fix

The following message appears if both the *input* and *output* constraints are specified for an inout port:

```
[WARNING] Both input and output constraints specified for inout port '<inout-port>'. Ignoring output constraint
```

Potential Issues

This violation appears if your design contains an inout port that is specified by both the *input* and *output* constraints.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the *output* constraint during CDC verification.

How to Debug and Fix

To fix this violation, review the *input* and *output* constraints specified for the same inout port. Retain only one constraint that you want to use for CDC verification.

Example Code and/or Schematic

Consider the following constraints in an SGDC file:

```
current_design top
input -name P1 -clock ck1
output -name P1 -clock ck2
```

For the above example, the *SGDC_output03* rule reports a violation because both the *input* and *output* constraints specified for the same inout port `P1`.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_output04

Reports invalid hierarchical names specified to the -clock argument of the output constraint

When to Use

Use this rule to perform sanity checks on the *output* constraint.

Prerequisites

Specify the *output* constraint.

Description

The *SGDC_output04* rule reports a violation if the object specified to the `-clock` argument of the *output* constraint is a hierarchical name and that object does not exist as a port or net in the current design.

Parameter(s)

None

Constraint(s)

output (Mandatory): Use this constraint to specify clock domains at output ports.

Messages and Suggested Fix

The following message appears if an invalid hierarchical name (port name or net name) is specified to the `-clock` argument of the *output* constraint:

```
[FATAL] '<object-name>' not found on/within module
'<module-name>'
```

Potential Issues

This violation appears if a hierarchical object name (port name or net name) is specified to the `-clock` argument of the *output* constraint, and that object does not exist in the design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing object (port or net) in the `-clock` argument of the *output* constraint.

Example Code and/or Schematic

Consider the design `top` in which the `ck1` clock does not exist. In this case, the *SGDC_output02* rule reports a violation if you specify the following *output* constraint specification in the SGDC file:

```
current_design top
output -name OUT1 -clock top.ck1
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_output_not_used01

Existence check for '-name' field of constraint 'output_not_used'

When to Use

Use this rule to perform sanity checks on the *output_not_used* constraint specification.

Prerequisites

Specify the *output_not_used* constraint.

Description

The *SGDC_output_not_used01* rule reports a violation if a port specified by the `-name` argument of the *output_not_used* constraint do not exist in the design.

Parameter(s)

None

Constraint(s)

output_not_used (Mandatory): Use this constraint to specify a primary output port.

Messages and Suggested Fix

The following message appears at the top-level design module when you specify non-existent ports with the *output_not_used* constraint:

```
[FATAL] 'port-name' [<top-port>] not found on/within module
'<top-design-unit>'
```

Potential Issues

This violation appears if the port specified by the `-name` argument of the *output_not_used* constraint does not exist in the design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, analyze the current design and specify an existing output port in the `-name` argument of the *output_not_used* constraint.

Example Code and/or Schematic

Consider that a design does not contain any port by the name `q2`.

In this case, the *SGDC_output_not_used01* rule reports a violation if you specify the following constraint:

```
output_not_used -name q2
```

Default Severity Label

Fatal

Rule Group

FATAL must rule

Reports and Related Files

No report or related file

SGDC_porttimedelay01

Reports if an invalid design unit is specified in the -name argument of the port_time_delay constraint

When to Use

Use this rule to perform sanity checks on the *port_time_delay* constraint specifications.

Prerequisites

Specify the *port_time_delay* constraint.

Description

The *SGDC_porttimedelay01* rule reports a violation if the design unit specified by the `-name` argument of the *port_time_delay* constraint does not exist in the current design.

Parameter(s)

None

Constraint(s)

port_time_delay (Mandatory): Use this constraint to specify design units to be checked by the *PortTimeDelay* rule.

Messages and Suggested Fix

The following message appears if the module *<module-name>* specified by the `-name` argument of the *port_time_delay* constraint does not exist in the current design:

```
[FATAL] '<module-name>' [SubModule] is never instantiated in the design
```

Potential Issues

This violation appears if your design does not contain the design unit specified by the `-name` argument of the *port_time_delay* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing design unit in the `-name` argument of the *port_time_delay* constraint

Example Code and/or Schematic

Consider that the current design `top` does not contain the design unit `ent`. In this case, the *SGDC_porttimedelay01* rule reports a violation in the following case:

```
current_design top
porttimedelay -name ent -ignore_ports "d*" "q*"
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier01

Reports a violation if a non-existent object is specified in the -name argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint specifications.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier01* rule reports a violation if the qualifier specified in the `-name` argument of the *qualifier* constraint does not exists as a net, port, or hierarchical terminal in a design.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message appears if an invalid object is specified in the `-name` argument of the *qualifier* constraint:

```
[FATAL] Constraint 'qualifier': '<signal-name>'[TopPort + SubModulePort + Net + HierTerminal] not found on/within module '<top-design>'
```

Potential Issues

This violation appears if your design does not contain the net, port, or hierarchical terminal specified by the `-name` argument of the *qualifier* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net, port, or hierarchical terminal in the `-name` argument of the *qualifier* constraint.

Example Code and/or Schematic

Consider that the current design does not contain the `qual1` qualifier. Now consider that you specify the following constraint:

```
qualifier -name qual1 -from_clk c1 -to_clk
```

In the above case, the *SGDC_qualifier01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier02a

Reports a violation if an invalid clock is specified in the -from_clk argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint specifications.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier02a* rule reports a violation if the clock specified in the `-from_clk` argument of the *qualifier* constraint is not one of the specified clocks or an automatically-inferred clock.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears when you specify an invalid clock in the `-from_clk` argument of the *qualifier* constraint:

```
[WARNING] Constraint 'qualifier': Clock name '<clock-name>' specified in field '-from_clk' is not a valid clock
```

Potential Issues

This violation appears if the clock specified in the `-from_clk` argument of the *qualifier* constraint is not one of the specified clock or an automatically-

inferred clock.

Consequences of Not Fixing

If you do not fix this violation, the *qualifier* constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, perform the following actions:

- Check if the reported clock is considered as a virtual clock by SpyGlass. If yes, ignore this violation.
- Else, specify any of the following clocks in the `-from_clk` argument of the *qualifier* constraint:
  - Clocks specified by the *clock* constraint.
  - Clocks inferred in a design when the *use_inferred_clocks* parameter is set.

Example Code and/or Schematic

Consider the port `scanen` that is not specified by the *clock* constraint, nor it is inferred as a valid clock when the *use_inferred_clocks* parameter is set.

Now consider that you specify the following constraint specification:

```
qualifier -name qual1 -from_clk scanen -to_clk clk2
```

In this case, the *SGDC_qualifier02a* rule reports `scanen` as an invalid clock.

Default Severity Label

Warning

Rule Group

Non-Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier02b

Reports a violation if a non-existent hierarchical object is specified in the -from_clk argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint specifications.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier02b* rule reports a violation if the clock name specified in the `-from_clk` argument of the *qualifier* constraint is a hierarchical name that does not exist as a port, hierarchical terminal, or net in the current design.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message if the hierarchical clock name *<hierarchical-name>* specified by the `-from_clk` argument of the *qualifier* constraint is not found in the current design *<module-name>*:

```
[FATAL] Constraint 'qualifier': Hierarchical name
'<hierarchical-name>' not found within module '<module-name>'
```

Potential Issues

This violation appears if the hierarchical clock name specified in the `-from_clk` argument of the *qualifier* constraint is a hierarchical name that does not exist as a port, hierarchical terminal, or net in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass analysis does not proceed further.

How to Debug and Fix

To fix this violation, specify existing hierarchical signals in the `-from_clk` argument of the *qualifier* constraint.

Example Code and/or Schematic

Consider the module `bbox` that does not have any port by the name `ck1`.

In this case, the *SGDC_qualifier02b* rule reports a violation if you specify the following constraint:

```
qualifier -name qual1 -from_clk test.bbox.ck1 -to_clk clk2
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier02c

Reports a violation if a non-existent object is specified in the -from_clk argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier02c* rule reports a violation if the clock name specified in the `-from_clk` argument of the *qualifier* constraint is a non-hierarchical name that does not exist as a port or a net in the current design.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message appears if the clock name specified in the `-from_clk` argument of the *qualifier* constraint is a non-hierarchical name that does not exist as a port or a net in the current design:

```
[INFO] Constraint 'qualifier': Non-hierarchical name '<name>' not found within module '<module-name>'. Considering it as a virtual clock
```

Potential Issues

This violation appears if the clock name specified in the `-from_clk` argument of the *qualifier* constraint is a non-hierarchical name that does not exist as a port or a net in the current design.

Consequences of Not Fixing

If you do not fix this violation, the port or net specified in the `-from_clk` argument of the *qualifier* constraint is considered as a virtual clock.

How to Debug and Fix

To fix this violation, specify an existing port or net name in the `-from_clk` argument of the *qualifier* constraint.

Example Code and/or Schematic

Consider the case in which the current design does not contain the clock `ck1`. In this case, the *SGDC_qualifier02c* rule reports a violation if you specify the following constraint:

```
qualifier -name qual1 -from_clk ck1 -to_clk ck2
```

Default Severity Label

Info

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier03a

Reports a violation if invalid clock names are specified in the -to_clk argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier03a* rule reports a violation if any of the clock names specified by the `-to_clk` argument of the *qualifier* constraint is not one of the clocks specified by the *clock* constraint or it is not one of the automatically inferred clock.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears if the clock specified by the `-to_clk` argument of the *qualifier* constraint is not one of the specified clocks:

```
[WARNING] Constraint 'qualifier': Clock name '<clock-name>' specified in field '-to_clk' is not a valid clock
```

Potential Issues

This violation appears if you specify an invalid clock in the `-to_clk` argument of the *qualifier* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported *qualifier* constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following clocks in the `-to_clk` argument of the *qualifier* constraint:

- Clock specified by the *clock* constraint
- Automatically inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier03b

Reports a violation if a non-existent hierarchical object is specified in the -to_clk argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint specifications.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier03b* rule reports a violation if the clock name specified in the `-from_clk` argument of the *qualifier* constraint is a hierarchical name that does not exist as a port, hierarchical terminal, or net in the current design.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message if the hierarchical clock name *<hierarchical-name>* specified by the `-to_clk` argument of the *qualifier* constraint is not found in the current design *<module-name>*:

```
[FATAL] Constraint 'qualifier': Hierarchical name
'<hierarchical-name>' not found within module '<module-name>'
```

Potential Issues

This violation appears if the hierarchical clock name specified in the `-to_clk` argument of the *qualifier* constraint is a hierarchical name that does not exist as a port, hierarchical terminal, or net in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass analysis does not proceed further.

How to Debug and Fix

To fix this violation, specify existing hierarchical signals in the `-to_clk` argument of the *qualifier* constraint.

Example Code and/or Schematic

Consider the module `bbox` that does not contain any port by the name `clk2`.

In this case, the *SGDC_qualifier03b* rule reports a violation if you specify the following constraint:

```
qualifier -name qual1 -from_clk ck1 -to_clk test.bbox.clk2
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier03c

Reports a violation if a non-existent object is specified in the -to_clk argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier03c* rule reports a violation if the clock name specified in the `-to_clk` argument of the *qualifier* constraint is a non-hierarchical name that does not exist as a port or a net in the current design.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message appears if the clock name specified in the `-to_clk` argument of the *qualifier* constraint is a non-hierarchical name that does not exist as a port or a net in the current design:

```
[INFO] Constraint 'qualifier': Non-hierarchical name '<name>' not found within module '<module-name>'. Considering it as a virtual clock
```

Potential Issues

This violation appears if the clock name specified in the `-to_clk` argument of the *qualifier* constraint is a non-hierarchical name that does not exist as a port or a net in the current design.

Consequences of Not Fixing

If you do not fix this violation, the port or net specified in the `-to_clk` argument of the *qualifier* constraint is considered as a virtual clock.

How to Debug and Fix

To fix this violation, specify an existing port or net name in the `-to_clk` argument of the *qualifier* constraint.

Example Code and/or Schematic

Consider the case in which the current design does not contain the clock `ck2`. In this case, the *SGDC_qualifier03c* rule reports a violation if you specify the following constraint:

```
qualifier -name qual1 -from_clk ck1 -to_clk ck2
```

Default Severity Label

Info

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier04

Reports a violation if an invalid domain is specified in the -from_domain argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier04* rule reports a violation if a domain specified by the `-from_domain` argument of the *qualifier* constraint is not one of the domains specified by the *clock* constraint or a domain attached to an automatically inferred clock.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears if an invalid domain is specified in the `-from_domain` argument of the *qualifier* constraint:

```
[WARNING] Constraint 'qualifier': Domain name '<domain-name>'
specified in field '-from_domain' is not a valid domain
```

Potential Issues

This violation appears if you specify an invalid domain in the `-from_domain` argument of the *qualifier* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported *qualifier* constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following domains in the `-from_domain` argument of the *qualifier* constraint:

- Domain specified by the *clock* constraint
- Domain of an automatically inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier05

Reports a violation if an invalid domain is specified in the -to_domain argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier05* rule reports a violation if a domain specified by the `-to_domain` argument of the *qualifier* constraint is not one of the domains specified by the *clock* constraint or a domain attached to an automatically inferred clock.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in your design.

Messages and Suggested Fix

The following message appears if an invalid domain is specified in the `-from_domain` argument of the *qualifier* constraint:

```
[WARNING] Constraint 'qualifier': Domain name '<domain-name>'
specified in field '-to_domain' is not a valid domain
```

Potential Issues

This violation appears if you specify an invalid domain in the `-to_domain` argument of the *qualifier* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported *qualifier* constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following domains in the `-to_domain` argument of the *qualifier* constraint:

- Domain specified by the *clock* constraint
- Domain of an automatically inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier06

Reports a violation if an incorrect value is specified in the -type argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier06* rule reports a violation if the value specified in the `-type` argument of the *qualifier* constraint is other than `src` or `des`.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the value *<value>* specified in the `-type` argument of the *qualifier* constraint is other than `src` or `des`:

```
[FATAL] Constraint 'qualifier': Invalid [Pre-defined]
specification '<value>' for '-type' field of constraint
'qualifier'
```

Potential Issues

This violation appears if the value specified in the `-type` argument of the *qualifier* constraint is other than `src` or `des`.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify `src` or `des` in the `-type` argument of the *qualifier* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
qualifier -from_domain d1 d2 d3 -to_domain d4 d5 d6
-name qual -type srcdes
```

For the above constraint, the *SGDC_qualifier06* rule reports a violation because the value specified in the `-type` argument is neither `src` nor `des`.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier08

Reports a violation if the wildcard name specified by the -name argument of the qualifier constraint does not match with any object in the design

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier08* rule reports a violation if the wildcard expression specified by the `-name` argument of the *qualifier* constraint does not match with the name of any net, hierarchical terminal, port, or sub module port in the design.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid net, hierarchical terminal, port, or sub module port is specified in the `-name` argument of the *qualifier* constraint:

```
[WARNING] Constraint 'qualifier': '<object-name>'[TopPort + Net
+ HierTerminal + SubmodulePort] does not exist in the current
design '<current-design'
```

Potential Issues

This violation appears if the wildcard name specified by the `-name` argument of the *qualifier* constraint does not match with any net, hierarchical terminal, port, or sub module port in the design.

Consequences of Not Fixing

If you do not fix this violation, the reported *qualifier* constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify a correct wildcard expression in the `-name` argument of the *qualifier* constraint so that the expression matches with an existing net, hierarchical terminal, port, or sub module port of the design.

Example Code and/or Schematic

Consider that your design does not contain any object starting with the string `ql`. In this case, the *SGDC_qualifier08* rule reports a violation if you specify the following constraint:

```
qualifier -from_clk c1 -to_clk c2 -name "ql*" -type src
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier09

Reports a violation if none of the -from_clk, -from_domain, -from_obj, or -ignore arguments of the qualifier constraint are specified

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier09* rule reports a violation if none of the following arguments of the *qualifier* constraint are specified:

- `-from_clk`
- `-from_domain`
- `-from_obj`
- `-ignore`

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if neither `-from_clk`/`-to_clk` nor `-from_domain`/`-to_domain` arguments of the *qualifier* constraint are specified:

```
[FATAL] Constraint 'qualifier': fields -from_clk or -
from_domain or -ignore or -from_obj not specified
```

Potential Issues

This violation appears if none of the following arguments of the *qualifier* constraint are specified:

- `-from_clk`
- `-from_domain`
- `-from_obj`
- `-ignore`

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify at least one of the following arguments to the *qualifier* constraint:

- `-from_clk`
- `-from_domain`
- `-from_obj`
- `-ignore`

Example Code and/or Schematic

Consider the following constraint:

```
qualifier -name qual -type src
```

For the above constraint, the *SGDC_qualifier09* rule reports a violation because none of the `-from_clk`, `-from_domain`, `-from_obj`, `-ignore`

arguments are specified.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier10

Reports a violation if the domain specified by the -from_clk/ from_domain and -to_clk/to_domain arguments of the qualifier constraint are same

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier10* rule reports a violation in any of the following cases:

- If clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint are from the same domain
- If same domain is specified in the `-from_domain` and `-to_domain` arguments of the *qualifier* constraint

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *report_clock_names_sgdc_qualifier10*: Default value is `yes`. Set this parameter to `no` to enable the *SGDC_qualifier10* rule not include the clock/domain names in the violation message.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint are from the same domain, or same domain is specified in the `-from_domain` and `-to_domain` arguments of this constraint:

```
[ERROR] Constraint 'qualifier': Domain specified in '-from_clk/-from_domain' '<name>' and '-to_clk/-to_domain' '<name>' matches
```

Potential Issues

This violation appears in any of the following cases:

- If clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint are of the same domain
- If same domain is specified in the `-from_domain` and `-to_domain` arguments of the *qualifier* constraint

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, check the clock specifications in the SGDC file and ensure the following:

- Clocks specified by the `-from_clk` and `-to_clk` arguments of the *qualifier* constraint should be from different domains.
- Different domains should be specified in the `-from_domain` and `-to_domain` arguments of the *qualifier* constraint.

Example Code and/or Schematic

Example 1

Consider the following constraints specified in an SGDC file:

```
current_design top
clock -name c1 -domain D1
```

```
clock -name c2 -domain D1
qualifier -from_clk c1 -to_clk c2 -name qual -type src
```

For the above example, the *SGDC_qualifier10* rule reports a violation because the `c1` and `c2` clocks specified by the `-from_clk` and `-to_clk` arguments, respectively, of the *qualifier* constraint are from the same domain `D1`.

Example 2

Consider the following constraint:

```
qualifier -from_domain D1 -to_domain D1 -name qual -type src
```

For the above constraint, the *SGDC_qualifier10* rule reports a violation because the same domain `D1` is specified in the `-from_domain` and `-to_domain` arguments of the *qualifier* constraint.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier11

Reports a violation if a qualifier is not defined at the destination output of a clock domain crossing

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier11* rule reports a violation if the *qualifier* constraint is specified with the `-crossing` argument, but the qualifier is not defined at the destination output of a clock domain crossing.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the qualifier net *<net-name>* is not the output of a clock-domain crossing:

```
[ERROR] Constraint 'qualifier': '<net-name>' is not an output of a clock domain crossing
```

Potential Issues

This violation appears if your design contains a qualifier for which the `-crossing` argument is specified in the *qualifier* constraint, but this qualifier is not present in the destination output of a clock domain crossing.

Consequences of Not Fixing

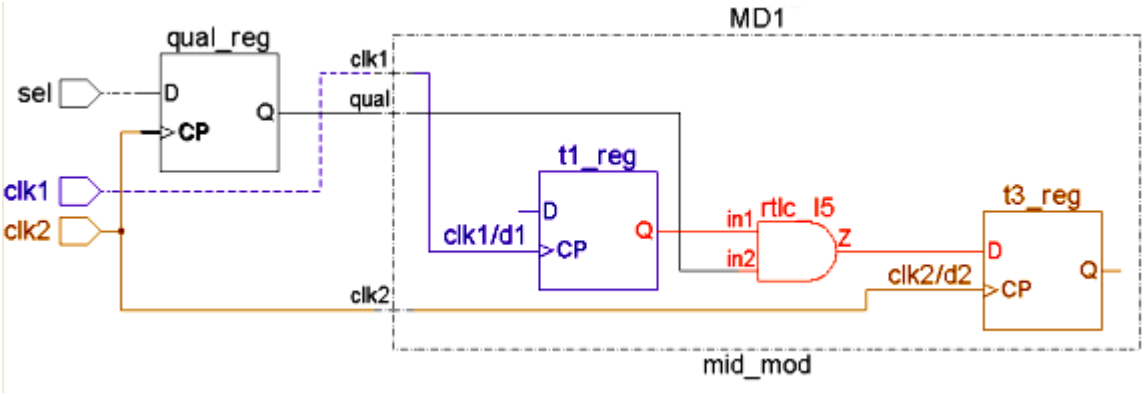
If you do not fix this violation, the reported qualifier does not synchronize clock domain crossings.

How to Debug and Fix

To fix this violation, analyze the reported qualifier signal, and specify the `-crossing` argument only if that qualifier defines a crossing output that contains a single destination flip-flop.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:




```
// Constr.sgdc
qualifier -name test.MD1.qual -from_clk test.clk1 -to_clk clk2 -crossing
```

FIGURE 430. Schematic of the SGDC_qualifier11 Rule Violation

In the above case, *SGDC_qualifier11* rule reports a violation because the

qualifier net is driven from the output net of a standalone flip-flop.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier12

Reports a violation if the clock/domain specified by the -to_clk or -to_domain argument of the qualifier constraint does not match with the clock/domain of the destination instance of the qualifier

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier12* rule reports a violation if the *qualifier* constraint is defined with the `-crossing` argument, but the `-to_clk` or `-to_domain` argument does not match with the clock or domain, respectively, of the destination instance of the qualifier.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

Message 1

The following message appears if the clock specified by the `-to_clk` argument does not match with clock of the destination instance of the

qualifier:

```
[SQual12_1] [ERROR] Constraint 'qualifier': Clock specified in
-to_clk field does not match with clock of destination instance
of the qualifier
```

Potential Issues

This violation appears if your design contains a qualifier specified by the *qualifier* constraint with the `-crossing` argument, and the clock specified by the `-to_clk` argument of this constraint does not match with the clock of the destination instance of the qualifier.

Consequences of Not Fixing

If you do not fix this violation, the reported qualifier does not synchronize the clock domain crossings.

How to Debug and Fix

To fix this violation, ensure that the clock specified by the `-to_clk` argument of the *qualifier* constraint matches with the clock of the destination instance of the qualifier.

Message 2

The following message appears if the domain specified by the `-to_domain` argument does not match with the domain of the destination instance of the qualifier:

```
[SQual12_2] [ERROR] Constraint 'qualifier': Domain specified in
-to_domain field does not match with domain of destination
instance of the qualifier
```

Potential Issues

This violation appears if your design contains a qualifier specified by the *qualifier* constraint with the `-crossing` argument, and the domain specified by the `-to_domain` argument of this constraint does not match with the clock domain of the destination instance of the qualifier.

Consequences of Not Fixing

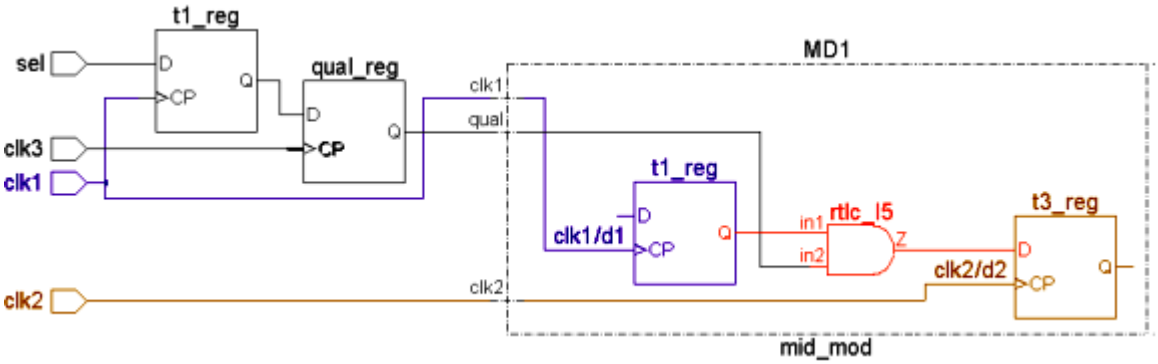
If you do not fix this violation, the reported qualifier does not synchronize the clock domain crossings.

How to Debug and Fix

To fix this violation, ensure that the domain specified by the `-to_domain` argument of the *qualifier* constraint matches with the domain of the destination instance of the qualifier.

Example Code and/or Schematic

Consider the following schematic of a violation of this rule:




```
// constr.sgdc
clock -name clk1 -domain d1
clock -name clk2 -domain d2
clock -name clk3 -domain d3
qualifier -name test.MD1.qual -from_clk test.clk1 -to_clk clk2 -crossing
```

FIGURE 431. Schematic of the SGDC_qualifier12 Rule Violation

In the above example, the destination instance of the `qual_reg` qualifier is driven by the `clk1` clock. This clock does not match with the destination clock `clk3` specified in the SGDC file. Therefore, the *SGDC_qualifier12* rule reports a violation.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier13

Reports if an incorrect clock or domain is specified in the -from_clk or -from_domain argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *qualifier* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_qualifier13* rule reports a violation if the *qualifier* constraint is specified with the `-crossing` argument, but the clock or domain specified by the `-from_clk` or `-from_domain` argument of this constraint does not match with the clock or domain of the source instance of the qualifier.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.

Constraint(s)

- *qualifier* (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

Message 1

The following message appears if the clock specified by the `-from_clk` argument of the *qualifier* constraint does not match with the clock of the

source instance of the qualifier:

```
[SQual13_1] [ERROR] Constraint 'qualifier': Clock specified in
-from_clk field does not match with clock of source instance of
the <qualifier>
```

Potential Issues

This violation appears if the *qualifier* constraint is specified with the `-crossing` argument and the clock specified by the `-from_clk` argument does not match the clock of the source instance of the qualifier.

Consequences of Not Fixing

If you do not fix this violation, your design does not synchronize the clock-domain crossing as intended.

How to Debug and Fix

To fix this violation, ensure that the clocks specified by the `-from_clk` argument of the *qualifier* constraint matches with the clock of the source instance of the qualifier.

Message 2

The following message appears if the domain specified by the `-from_domain` argument of the *qualifier* constraint does not match with the domain of the source instance of the qualifier:

```
[SQual13_2] [ERROR] Constraint 'qualifier': Domain specified in
-from_domain field does not match with domain of source
instance of the <qualifier>
```

Potential Issues

This violation appears if the *qualifier* constraint is specified with the `-crossing` argument and the domain specified by the `-from_domain` argument does not match the domain of the source instance of the qualifier.

Consequences of Not Fixing

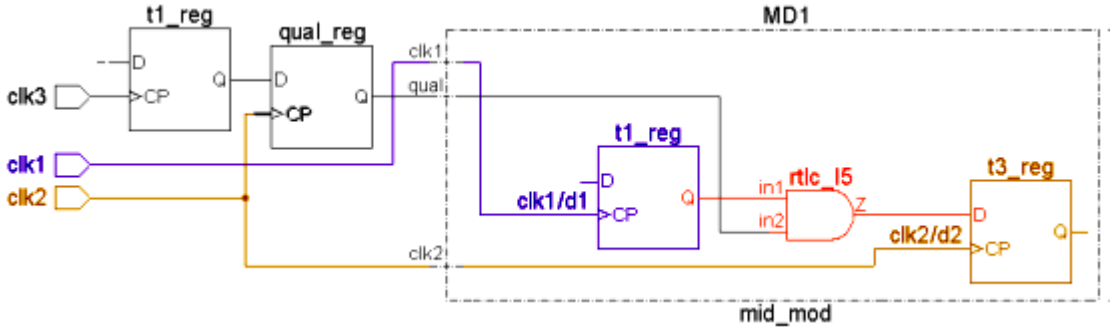
If you do not fix this violation, your design does not synchronize the clock-domain crossing as intended.

How to Debug and Fix

To fix this violation, ensure that the domains specified by the `-from_domain` argument of the *qualifier* constraint matches with the domain of the source instance of the qualifier.

Example Code and/or Schematic

Consider the following schematic of a violation reported by this rule:




```
// constr.sgdc
clock -name clk1 -domain d1
clock -name clk2 -domain d2
clock -name clk3 -domain d3
qualifier -name test.MD1.qual -from_clk test.clk1 -to_clk clk2 -crossing
```

FIGURE 432. Schematic of the SGDC_qualifier13 Rule Violation

In the above example, the source of the qualifier signal `MD1.qual` is triggered from the `clk3` clock and not the `clk1` clock, as mentioned in the constraint file. Therefore, the *SGDC_qualifier13* rule reports a violation.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier15

Existence check for the -name or -enable arguments of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier15* rule reports a violation if none of the `-name` or `-enable` arguments are specified to the *qualifier* constraint.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message appears if none of the `-name` or `-enable` arguments are specified to the *qualifier* constraint:

```
[FATAL] It is Mandatory to Specify At Least One Value for
combination of fields '-name -enable ' of constraint
'qualifier'
```

Potential Issues

This violation appears if none of the `-name` or `-enable` arguments are specified to the *qualifier* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify at least one of the `-name` or `-enable` arguments to the *qualifier* constraint.

Example Code and/or Schematic

The *SGDC_qualifier15* rule reports a violation in the following case as none of the `-name` or `-enable` arguments are specified to the *qualifier* constraint:

```
qualifier -from_obj  top.src.q  -to_obj  top.des.q
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier16

Existence check for valid signal names specified to the -enable argument of the qualifier constraint

When to Use

Use this rule to perform sanity checks on the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier16* rule reports a violation if any signal name specified to the `-enable` argument of the *qualifier* constraint is a hierarchical name that does not exist as a port, a hierarchical terminal, or a net in the current design.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

The following message appears if an invalid signal name is specified to the `-enable` argument of the *qualifier* constraint:

[FATAL] `Constraint 'qualifier': '<enable> '[TopPort + Net + HierTerminal] does not exist in the current design '<current-design>'`

Where, *<enable>* refers to the signal name specified to the `-enable` argument of the *qualifier* constraint.

Potential Issues

This violation appears if any signal name specified to the `-enable` argument of the *qualifier* constraint is a hierarchical name that does not

exist as a port, a hierarchical terminal, or a net in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a valid signal name to the `-enable` argument of the *qualifier* constraint such that the name is a hierarchical name that exists as a port, a hierarchical terminal, or a net in the current design.

Example Code and/or Schematic

The *SGDC_qualifier16* rule reports a violation in the following case if there is no object by the name `en` in the current design:

```
qualifier -enable "top.en" -from_obj  top.src.q  -to_obj
top.des.q
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_qualifier18

qualifier -ignore specified on a net that is the part of a loop

When to Use

Use this rule to perform sanity checks related to the *qualifier* constraint.

Prerequisites

Specify the *qualifier* constraint.

Description

The *SGDC_qualifier18* rule reports a violation if you specify the *qualifier -ignore* constraint on a net that is the part of a loop.

Parameter(s)

None

Constraint(s)

qualifier (Mandatory): Use this constraint to specify a qualifier for synchronizing a clock domain crossing.

Messages and Suggested Fix

This rule reports the following message:

```
[ERROR] Constraint qualifier -ignore prevents propagation of
all qualifiers reaching to the loop
```

Potential Issues

This violation appears if you specify the *qualifier -ignore* constraint on a net that is the part of a loop.

Consequences of Not Fixing

If you do not fix this violation, the *qualifier -ignore* constraint gets applied to all the nets present in the loop. This may not be the intended behavior.

How to Debug and Fix

To fix this violation, apply the *qualifier* *-ignore* constraint outside the loop and in the fan-in cone of the loop.

Example Code and/or Schematic

Consider the example shown in the following figure:

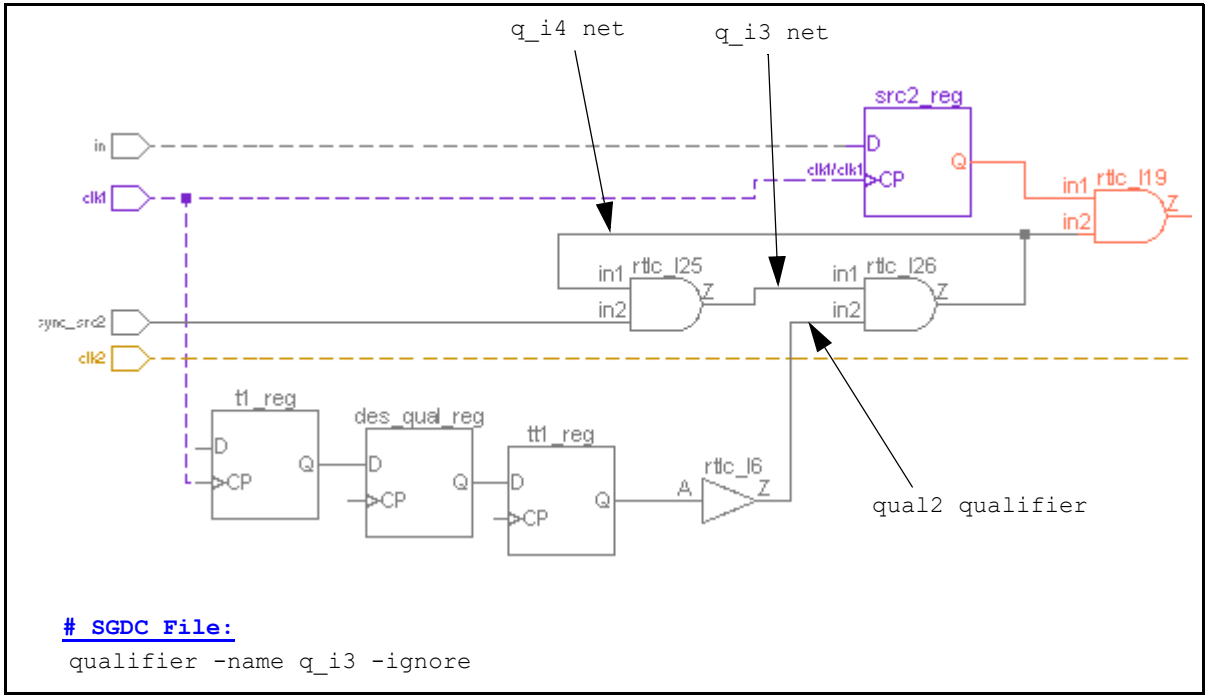


FIGURE 433.

In the above example, the *qualifier* *-ignore* constraint is specified on the `q_i3` net. Since this net is the part of a loop, the *qualifier* *-ignore* constraint gets applied on all the nets (`q_i4` in this case) of that loop. As a result, propagation of the `qual2` qualifier stops beyond the `rtlc_I26` AND gate.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_quasi_static01

Reports a violation if an invalid net is specified in the -name argument of the quasi_static constraint

When to Use

Use this rule to perform sanity checks on the *quasi_static* constraint.

Prerequisites

Specify the *quasi_static* constraint.

Description

The *SGDC_quasi_static01* rule reports a violation if the object specified in the `-name` argument of the *quasi_static* constraint does not exist as a net in the current design.

Parameter(s)

None

Constraint(s)

quasi_static (Mandatory): Use this constraint to specify signals whose value is predominantly static.

Messages and Suggested Fix

The following message appears if the net *<net-name>* specified by the `-name` argument of the *quasi_static* constraint does not exist in the current design *<current-design>*:

```
[FATAL] '<net-name>' [Net] not found on/within module
'<current-design>'
```

Potential Issues

This violation appears the current design does not contain the net specified by the `-name` argument of the *quasi_static* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing net in the `-name` argument of the *quasi_static* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the `in1` net. Now consider the following constraints specified in an SGDC file:

```
current_design top
quasi_static -name "in1"
```

For the above example, the *SGDC_quasi_static01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_quasi_static_style01

Reports multiple specifications of the quasi_static_style constraint.

When to Use

Use this rule to perform sanity checks on the *quasi_static_style* constraint.

Prerequisites

Specify the *quasi_static_style* constraint.

Description

The *SGDC_quasi_static_style01* rule reports a violation if the SGDC file contains multiple specifications of the *quasi_static_style* constraint.

Parameter(s)

None

Constraint(s)

quasi_static_style (Mandatory): Use this constraint to specify a criterion based on which SpyGlass infers quasi-static signals in a design.

Messages and Suggested Fix

The following message appears if multiple specifications of the *quasi_static_style* constraint exist in the SGDC file:

```
[WARNING] Multiple specification of quasi_static_style
constraint detected in constraint file. Latest specification
will be used
```

Potential Issues

This violation appears if the SGDC file contains multiple specifications of the *quasi_static_style* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass considers the last specification of the *quasi_static_style* constraint and ignores the rest.

How to Debug and Fix

To fix this violation, specify only one specification of the *quasi_static_style* constraint that matches your requirement.

Example Code and/or Schematic

Consider the constraints specified in the following SGDC file:

```
current_design test

clock -name clk1
input -name in_cfg -clock clk1
reset -sync -name cfg1 -value 1

quasi_static_style -min_domain_fanouts 2 -min_seq_fanouts 5
-names
"*cfg*"
quasi_static_style -min_domain_fanouts 2 -min_seq_fanouts 5
```

In the above example, two specifications of the *quasi_static_style* constraint are present. In this case, SpyGlass only considers the last specification.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_quasi_static_style02

Reports if no argument is specified in the quasi_static_style constraint.

When to Use

Use this rule to perform sanity checks on the *quasi_static_style* constraint.

Prerequisites

Specify the *quasi_static_style* constraint.

Description

The *SGDC_quasi_static_style02* rule reports a violation if you do not specify any argument with the *quasi_static_style* constraint.

Parameter(s)

None

Constraint(s)

quasi_static_style (Mandatory): Use this constraint to specify a criterion based on which SpyGlass infers quasi-static signals in a design.

Messages and Suggested Fix

The following message appears if you do not specify any argument with the *quasi_static_style* constraint:

```
[WARNING] quasi_static_style': No field specified in the constraint
```

Potential Issues

This violation appears if you do not specify any argument with the *quasi_static_style* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass considers the default argument values of the *quasi_static_style* constraint while inferring quasi-static signals

in a design.

How to Debug and Fix

To fix this violation, specify at least one argument with the *quasi_static_style* constraint.

Example Code and/or Schematic

Consider the constraints specified in the following SGDC file:

```
current_design test
clock -name clk1

quasi_static_style
```

For the above example, the *SGDC_quasi_static_style02* rule reports a violation because no argument is specified with the *quasi_static_style* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path01

Reports if no argument is specified to the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path01* rule reports a violation if no argument is specified to the *reset_filter_path* constraint.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[FATAL] Constraint 'reset_filter_path': fields -from_rst or
-to_rst or -from_obj or -to_obj or -clock not specified
```

Potential Issues

This violation appears if no argument is specified to the *reset_filter_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify at least one of the following arguments to the *reset_filter_path* constraint:

-from_rst <source-rst-list>	-to_rst <dest-rst-list>	-clock <clk-name>
-from_obj <src-list>	-to_obj <des-list>	

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path02a

Reports if a non-existent object is specified to the -clock argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path02a* rule reports a violation if the object specified to the `-clock` argument of the *reset_filter_path* constraint does not exist as a port, hierarchical terminal, net, or virtual clock in the current design.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': clock name '<clock-name>' specified in field '-clock' does not exist either as a net, port, hierarchical terminal or as a virtual clock
```

Potential Issues

This violation appears if you the object specified to the `-clock` argument of the *reset_filter_path* constraint is none of the port, hierarchical terminal, net, or virtual clock in the current design.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify an existing port, hierarchical terminal, net, or a virtual clock to the `-clock` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path02b

Reports a violation if an invalid clock is specified to the -clock argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path02b* rule reports a violation if the clock specified to the `-clock` argument of the *reset_filter_path* constraint is not one of the following clock in the current design:

- Clock specified by the *clock* constraint
- Clock inferred after setting the *use_inferred_clocks* parameter to yes
- Virtual clock

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': clock name '<clock-name>' specified in field '-clock' is found but it is not a valid clock
```

Potential Issues

This violation appears if the clock specified to the `-clock` argument of the *reset_filter_path* constraint is not one of the following clock in the current

design:

- Clock specified by the *clock* constraint
- Clock inferred after setting the *use_inferred_clocks* parameter to yes
- Virtual clock

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following clock to the `-clock` argument of the *reset_filter_path* constraint:

- Clock specified by the *clock* constraint
- Clock inferred after setting the *use_inferred_clocks* parameter to yes
- Virtual clock

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path02c

Reports if a virtual clock is specified to the -clock argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path02c* rule reports a violation if a virtual clock is specified to the `-clock` argument of the *reset_filter_path* constraint.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': clock '<clock-name>' specified with '-clock' within module '<module-name>' is a virtual clock
```

Potential Issues

This violation appears if a virtual clock is specified to the `-clock` argument of the *reset_filter_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify a clock other than the virtual clock to the `-clock` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path03a

Reports if a non-existent object is specified to the -from_rst argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path03a* rule reports a violation if the object specified to the `-from_rst` argument of the *reset_filter_path* constraint does not exist as a net, port, hierarchical terminal, or virtual reset in the current design.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': reset name
'<reset-name>' specified in field '-from_rst' does not exist
either as a net, port, hierarchical terminal or as a virtual
reset
```

Potential Issues

This violation appears if the object specified to the `-from_rst` argument of the *reset_filter_path* constraint does not exist as a net, port, hierarchical terminal, or virtual reset in the current design.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify an existing net, port, hierarchical terminal, or virtual reset to the `-from_rst` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path03b

Reports if an invalid reset is specified to the -from_rst argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Following are the prerequisites for running this rule:

- Specify the *reset_filter_path* constraint.
- Run the `Ar_resetcross01` rule.

Description

The *SGDC_reset_filter_path03b* rule reports a violation if the reset specified to the `-from_rst` argument of the *reset_filter_path* constraint is not one of the following reset in the current design:

- Reset specified by the *reset* constraint
- Reset inferred after setting the *use_inferred_resets* parameter to `yes`
- Virtual reset

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': reset name '<reset-name>' specified in '-from_rst' field is found but it is not a valid reset
```

Potential Issues

This violation appears if the reset specified to the `-from_rst` argument of the *reset_filter_path* constraint is not one of the following reset in the current design:

- Reset specified by the *reset* constraint
- Reset inferred after setting the *use_inferred_resets* parameter to yes
- Virtual reset

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following reset to the `-from_rst` argument of the *reset_filter_path* constraint:

- Reset specified by the *reset* constraint
- Reset inferred after setting the *use_inferred_resets* parameter to yes
- Virtual reset

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path03c

Reports if a virtual reset is specified to the -from_rst argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path03c* rule reports a violation if a virtual reset is specified to the `-from_rst` argument of the *reset_filter_path* constraint.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': reset '<reset-name>' specified with '-from_rdc' within module '<module-name>' is a virtual reset
```

Potential Issues

This violation appears if a virtual reset is specified to the `-from_rst` argument of the *reset_filter_path* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify a reset other than the virtual reset to the `-from_rst` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path04a

Reports if a non-existent object is specified to the -to_rst argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path03a* rule reports a violation if the object specified to the `-to_rst` argument of the *reset_filter_path* constraint does not exist as a port, net, or hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[FATAL] Constraint 'reset_filter_path': reset name
'<reset-name>' specified in field '-to_rst' does not exist
either as a port, hierarchical terminal, or net
```

Potential Issues

This violation appears if the object specified to the `-to_rst` argument of the *reset_filter_path* constraint does not exist as a port, net, or hierarchical terminal in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass analysis does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port, net, or hierarchical terminal to the `-to_rst` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path04b

Reports if an invalid reset is specified to the -to_rst argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Following are the prerequisites for running this rule:

- Specify the *reset_filter_path* constraint.
- Run the `Ar_resetcross01` rule.

Description

The *SGDC_reset_filter_path04b* rule reports a violation if the reset specified to the `-to_rst` argument of the *reset_filter_path* constraint is not one of the following reset in the current design:

- Reset specified by the *reset* constraint
- Reset inferred after setting the *use_inferred_resets* parameter to `yes`

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'reset_filter_path': reset name '<reset-name>' specified in '-to_rst' field is found but it is not a valid reset
```

Potential Issues

This violation appears if the reset specified to the `-to_rst` argument of

the *reset_filter_path* constraint is not one of the following reset in the current design:

- Reset specified by the *reset* constraint
- Reset inferred after setting the *use_inferred_resets* parameter to yes

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following reset to the `-to_rst` argument of the *reset_filter_path* constraint:

- Reset specified by the *reset* constraint
- Reset inferred after setting the *use_inferred_resets* parameter to yes

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path05a

Reports if a non-existent object is specified to the -from_obj argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path05a* rule reports a violation if the object specified to the `-from_obj` argument of the *reset_filter_path* constraint does not exist as a port, net, or hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[FATAL] Constraint 'reset_filter_path': reset name
'<reset-name>' specified in field '-from_obj' does not exist
either as a port, hierarchical terminal, or net
```

Potential Issues

This violation appears if the object specified to the `-from_obj` argument of the *reset_filter_path* constraint does not exist as a port, net, or hierarchical terminal in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass analysis does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port, net, or hierarchical terminal to the `-from_obj` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path06a

Reports if a non-existent object is specified to the -to_obj argument of the reset_filter_path constraint

When To Use

Use this rule to perform sanity checks on the *reset_filter_path* constraint.

Prerequisites

Specify the *reset_filter_path* constraint.

Description

The *SGDC_reset_filter_path06a* rule reports a violation if the object specified to the `-to_obj` argument of the *reset_filter_path* constraint does not exist as a port, net, or hierarchical terminal in the current design.

Parameter(s)

None

Constraint(s)

reset_filter_path (Mandatory): Use this constraint to specify false paths so that reset crossings along these paths are ignored from rule checking.

Messages and Suggested Fix

This rule reports the following violation:

```
[FATAL] Constraint 'reset_filter_path': reset name
'<reset-name>' specified in field '-to_obj' does not exist
either as a port, hierarchical terminal, or net
```

Potential Issues

This violation appears if the object specified to the `-to_obj` argument of the *reset_filter_path* constraint does not exist as a port, net, or hierarchical terminal in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass analysis does not proceed further.

How to Debug and Fix

To fix this violation, specify an existing port, net, or hierarchical terminal to the `-to_obj` argument of the *reset_filter_path* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_filter_path_validation01

Reports block-level reset_filter_path constraints which do not have a matching top-level reset_filter_path constraint

When to Use

Use this rule during the hierarchical CDC verification flow to validate user-specified constraints for an abstract block in the context of a higher-level hierarchy.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_filter_path* constraint
- Specify the `clock` constraint for objects given in `-from_clock`/`-to_clock`/`-clock` arguments of the *reset_filter_path* constraint
- Specify the `reset` constraint for objects given in `-from_rst`/`-to_rst` arguments of the *reset_filter_path* constraint

Description

The `SGDC_cdc_false_path_validation01` rule reports a violation if the *reset_filter_path* constraint specified on an abstracted block does not have an equivalent *reset_filter_path* constraint at the top level. the rule matches the `-from_rst`, `-to_rst`, `-from_clock`, `-to_clock`/`-clock`, and `-type` arguments.

Parameter(s)

None

Constraint(s)

- *reset_filter_path* (Mandatory): Use this constraint to specify reset paths so that the reset domain crossings across these paths are ignored from SpyGlass analysis.
- *clock* (Optional): Use this constraint to specify clock signals in a design.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The rule reports the following message when the `reset_filter_path` constraint given in an SGDC file of an abstract view does not have an equivalent *reset_filter_path* constraint at the top level.

```
[ERROR] For block instance '<block-inst>' (block: <block
name>), Constraint reset_filter_path specified at the block
level with fields -from_rst '<from-reset>', -to_rst '<to-
reset>', -from_clock '<from-clock>', -to_clock/-clock '<to-
clock/clock>' and -type '<type>' has no equivalent constraint
at the top level
```

Potential Issues

This violation appears if the *reset_filter_path* constraint specified in an SGDC file of an abstract view does not have an equivalent *reset_filter_path* constraint at the top level.

Consequences of Not Fixing

If you do not fix this violation, ignored paths by reset_filter_path are inconsistent between top and abstracted block. This may result in an inconsistency in the violations reported for the top and the abstracted block by the Ar_resetcross01 and Ar_sync_group rules.

How to Debug and Fix

To fix this violation, first analyze the following:

- The specification of *reset_filter_path* constraint in block level as well as top level.
- The specification or propagation of top-level clocks.
- The specification or propagation of top-level resets.

Next, if the block-level constraints are incorrect, specify the correct block-level constraints and run block-level verification again. If the top-level constraints are incorrect with respect to the block, specify the correct top-level constraints and run top-level verification again.

Example Code and/or Schematic

Consider the following schematic:

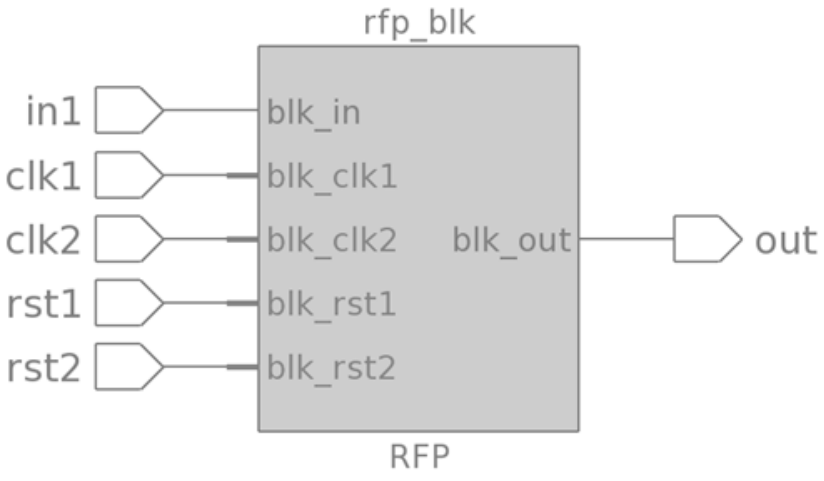


FIGURE 434.

In addition, consider the following SGDC files:

//Block-level SGDC file

```
current_design "RFP"
clock -name blk_clk1 -domain clk1
clock -name blk_clk2 -domain clk2
reset -name blk_rst1 -value 0
reset -name blk_rst2 -value 0
reset_filter_path -from_rst blk_rst1 -to_rst blk_rst2 -
from_clock blk_clk1 -to_clock blk_clk2 -type rdc
```

//Top-level SGDC file

```
current_design top
clock -name clk1
clock -name clk2
reset -name rst1 -value 0
reset -name rst2 -value 0
sgdc -import RFP RFP.sgdc
```

In the above example, only one *reset_filter_path* constraint is specified in the block-level SGDC file. However, no matching *reset_filter_path* constraint is specified in the top-level SGDC file.

Therefore, the `SGDC_reset_filter_path_validation01` rule reports a violation in this case.

Default Severity Label

Error

Rule Group

None

Reports and Related Files

No report or related file

SGDC_reset_synchronizer01

Reports a violation if the net/port/hierarchical terminal specified by the -name argument of the reset_synchronizer constraint is not found

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer01* rule reports a violation if the synchronizer output specified by the `-name` argument of the *reset_synchronizer* constraint does not exist as a port, hierarchical terminal, or a net in the current design.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the signal *<sig-name>* specified by the `-name` argument of the *reset_synchronizer* constraint does not exist in the module *<module-name>*:

```
[FATAL] Constraint 'reset_synchronizer': '<sig-name>' [TopPort
+ Net + HierTerminal] not found on/within module
'<module-name>'
```

Potential Issues

This violation appears if the current design does not contain the port, hierarchical terminal, or net specified by the `-name` argument of the *reset_synchronizer* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run cannot proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing port, hierarchical terminal, or net in the `-name` argument of the *reset_synchronizer* constraint.

Example Code and/or Schematic

Consider the following example if `out1` is not the pin of the reset synchronizer instance `S1`:

```
reset_synchronizer -name top.S1.out1 -reset rst -clock clk1
-value 0
```

In this case, the *SGDC_reset_synchronizer01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer02

Reports if the synchronized output specified by the -name argument of the reset_synchronizer constraint is not present in the path of reset specified the -reset argument

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer02* rule reports a violation if the synchronizer output specified by the `-name` argument of the *reset_synchronizer* constraint is not present in the fan-out of the reset specified by the `-reset` argument of this constraint.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the synchronizer output *<output-name>* specified by the `-name` argument of the *reset_synchronizer* constraint is not present in the fan-out of the reset *<reset-name>* specified by the `-reset` argument of this constraint:

```
[WARNING] Constraint 'reset_synchronizer': Sync output
'<output-name>' (specified with '-name' field) is not present
in the fanout of reset '<reset-name>' (specified with '-reset'
field)
```

Potential Issues

This violation appears if the synchronizer output specified by the `-name` argument of the *reset_synchronizer* constraint is not present in the fan-out of the reset specified by the `-reset` argument of this constraint.

Consequences of Not Fixing

If you do not fix this violation, signal or pin name specified by the `-name` argument of the *reset_synchronizer* constraint is not considered as a valid synchronizer in such reset paths.

How to Debug and Fix

To fix this violation, update the value of the `-name` argument of the *reset_synchronizer* constraint to specify a synchronizer output that is present in the fan-out of the reset specified by the `-reset` argument of this constraint.

Example Code and/or Schematic

Consider the following example in which the `-name` argument is specified

through local scoping in the *reset_synchronizer* constraint:

```
reset_synchronizer -name reset_sync::sync_rst -reset rst
-clock clk1 -value 0
```

In this case, this rule reports a violation if all instantiation of the `reset_sync` module is not present in the propagated path of the `rst` reset.

The following figure shows the propagated paths of the `rst` reset in this case:

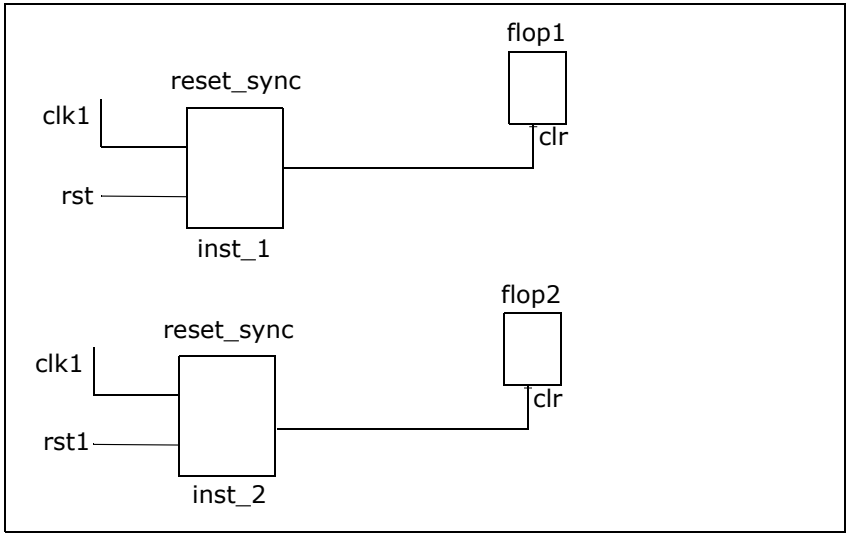


FIGURE 435. Scenario of the SGDC_reset_synchronizer02 Rule Violation

In the above case, the *SGDC_reset_synchronizer02* rule reports the following violation:

```
Constraint 'reset_synchronizer': Sync output
'top.inst_2.sync_rst' (specified with '-name' field) is not
present in the fanout of reset 'rst' (specified with '-reset'
field)
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer03

Reports a violation if the net/port/hierarchical terminal specified by the -reset argument of the reset_synchronizer constraint does not exist

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer03* rule reports a violation if the reset name specified by the `-reset` argument of the *reset_synchronizer* constraint is a hierarchical name that does not exist as a port, hierarchical terminal, or net in the current design.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the signal name *<sig-name>* specified by the `-reset` argument of the *reset_synchronizer* constraint does not exist in the module *<module-name>*:

```
[FATAL] Constraint 'reset_synchronizer': '<sig-name>' [TopPort
+ Net + HierTerminal] not found on/within module
'<module-name>'
```

Potential Issues

This violation appears if the current design does not contain the signal specified by the `-reset` argument of the *reset_synchronizer* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run cannot proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing signal in the `-reset` argument of the *reset_synchronizer* constraint.

Example Code and/or Schematic

For the following example, this rule reports a violation if `rst` is not the pin of the black box instance B1:

```
reset_synchronizer -name test.S1.out -reset test.B1.rst
-clock clk1 -value 0
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer04

Reports if an invalid reset is specified by the -reset argument of the reset_synchronizer constraint

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer04* rule reports a violation if the reset specified by the `-reset` argument of the *reset_synchronizer* constraint is none of the following resets:

- Reset specified by the *reset* constraint
- Automatically-inferred reset when the *use_inferred_resets* parameter is set to `yes`

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the reset specified by the `-reset` argument of the *reset_synchronizer* constraint is neither specified by the *reset* constraint nor it is an automatically-inferred reset:

```
[FATAL] Constraint 'reset_synchronizer': Reset name
'<reset-name>' specified in field '-reset' is not a valid reset
```

Potential Issues

This violation appears if the reset specified by the `-reset` argument of the *reset_synchronizer* constraint is none of the following resets:

- Reset specified by the *reset* constraint
- Automatically-inferred reset when the *use_inferred_resets* parameter is set to `yes`

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run cannot proceed further.

How to Debug and Fix

To fix this violation, perform the following actions:

- Specify relevant *reset* constraints
- Set the *use_inferred_resets* parameter to `yes` to detect reset signals automatically.

Example Code and/or Schematic

For the following example, this rule reports a violation if `rst` is not a user-specified or automatically-inferred reset:

```
reset_synchronizer -name test.S1.srst -reset rst -clock clk1
-value 0
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer05

Reports if the net/port/hierarchical terminal specified by the -clock argument of the reset_synchronizer constraint does not exist in the design

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer05* rule reports a violation if the synchronizer clock specified by the `-clock` argument of the *reset_synchronizer* constraint is neither a clock-tag nor a hierarchical name that exists as a port, hierarchical terminal, or a net in the current design.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the signal *<sig-name>* specified by the `-clock` argument of the *reset_synchronizer* constraint does not exist in the module *<module-name>*:

```
[FATAL] Constraint 'reset_synchronizer': '<sig-name>'[TopPort + Net + HierTerminal] not found on/within module '<module-name>'
```

Potential Issues

This violation appears if the current design does not contain the port, hierarchical terminal, or a net specified by the `-clock` argument of the *reset_synchronizer* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run cannot proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing port, hierarchical terminal, or a net specified in the `-clock` argument of the *reset_synchronizer* constraint.

Example Code and/or Schematic

For the following example, this rule reports a violation if `ck` is not a pin of the `B1` black box:

```
reset_synchronizer -name test.S1.out -reset rst -clock test.B1.ck -value 0
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer06

Reports if an invalid clock is specified by the -clock argument of the reset_synchronizer constraint

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer06* rule reports a violation if the synchronizer clock specified by the `-clock` argument of the *reset_synchronizer* constraint is none of the following:

- One of the clock tags
- One of the clock specified by the *clock* constraint
- One of the automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the clock *<synchronizer-clock>* specified by the `-clock` argument of the *reset_synchronizer* constraint is not one of the clock tags, user-specified clocks, or an automatically-inferred clock:

```
[FATAL] Constraint 'reset_synchronizer': Clock name
'<synchronizer-clock>' specified in field '-clock' is not a
valid clock
```

Potential Issues

This violation appears if the clock specified by the `-clock` argument of the *reset_synchronizer* constraint is none of the following:

- One of the clock tags
- One of the clock specified by the *clock* constraint
- One of the automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run cannot proceed further.

How to Debug and Fix

To fix this violation, perform the following actions:

- Specify relevant *clock* constraints for your design.
- Set the *use_inferred_clocks* parameter to `yes` automatically detect clock signals.

Example Code and/or Schematic

For the following example, this rule reports a violation if `clk1` is not a user-specified clock/tag or automatically-inferred clock:

```
reset_synchronizer -name test.S1.srst -reset rst -clock clk1
-value 0
```

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer07

Reports if an invalid value is specified in the -value argument of the reset_synchronizer constraint

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer06* rule reports a violation if the value specified by the `-value` argument of the *reset_synchronizer* constraint is other than 0 or 1.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the value *<value>* specified by the `-value` argument of the *reset_synchronizer* constraint is other than 0 or 1:

[FATAL] `Invalid [Pre-defined-range] specification '<value>' for '-value' field of constraint 'reset_synchronizer'`

Potential Issues

This violation appears if you specify a value other than 0 or 1 in the `-value` argument of the *reset_synchronizer* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run cannot proceed further.

How to Debug and Fix

To fix this violation, specify `0` or `1` in the `-value` argument of the *reset_synchronizer* constraint.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer08

Checks if the synchronized output specified by the -name argument of reset_synchronizer constraint is unused.

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`

Description

The *SGDC_reset_synchronizer08* rule reports a violation if a net, pin, or port specified by the `-name` argument of the *reset_synchronizer* constraint is not used to synchronize any reset in the current design.

Parameter(s)

use_inferred_resets: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

Messages and Suggested Fix

The following message appears if the net *<net-name>* specified by the `-name` argument of the *reset_synchronizer* constraint is not used to synchronize any reset in the current design:

```
[WARNING] Constraint 'reset_synchronizer': Sync output
```

```
'<net-name>' (specified with '-name' field) is not used in the
design
```

Potential Issues

This violation appears in any of the following cases:

- If the *reset_synchronizer* constraint was specified for a different clock domain
- The net specified by the `-name` argument of the *reset_synchronizer* constraint did not reach the reset/set terminal of any flip-flop. As a result, this constraint was left unused.

Consequences of Not Fixing

If you do not fix this violation, the specified *reset_synchronizer* constraint is not considered for SpyGlass analysis. In addition, SpyGlass may report extra *Ar_unsync01* and/or *Ar_asyncdeassert01* violations.

How to Debug and Fix

To debug and fix this violation, check the arguments of the *reset_synchronizer* constraint to ensure the following:

- These arguments refer to a correct reset-clock combination
- The net, pin, and port specified by the `-name` argument is reaching to the reset/set terminal of a flip-flop of the corresponding clock domain.

However, if you do not want the *reset_synchronizer* constraint to synchronize any reset in the current design, ignore this violation.

Example Code and/or Schematic

Consider the following files specified during SpyGlass analysis:

```
// test.vhd
entity flop is
    port( d, clk, rst : in bit;
          q : out bit);
end flop;
architecture flop_arch of flop is
begin
    process(clk, rst)
    begin
        if(rst = '0 ') then
            q <= '0';
        elsif(clk'event and clk = '1') then
            q <= d;
        end if;
    end process;
end flop_arch;

entity top is
    port( in0, in1, clk1, clk2, rst : in bit;
          out0, out1 : out bit);
end top;

architecture top_arch of top is
    component flop is
        port( d, clk, rst : in bit;
              q : out bit);
    end component;
    signal w1, w2, w3, w4 : bit;
begin
    w1 <= rst;
    w2 <= w1;
    w3 <= w1;
    w4 <= w2;
    f1 : flop port map( d => in0, clk => clk1, rst => w4, q => out0 );
    f2 : flop port map( d => in1, clk => clk2, rst => w3, q => out1 );
end top_arch;
```

```
// constr.sgdc
current_design top
clock -name clk1
clock -name clk2
reset -name rst -value 1
reset_synchronizer -name w2
  -reset rst -clock clk2 -value 1
```

In the above example, the *SGDC_reset_synchronizer08* rule reports a violation because of the *reset_synchronizer* constraint specification.

To resolve this violation, modify the *reset_synchronizer* constraint specification to any of the following:

```
reset_synchronizer -name w2 -reset rst -clock clk1 -value 1
```

OR

```
reset_synchronizer -name w1 -reset rst -clock clk1 -value 1
```

```
reset_synchronizer -name w1 -reset rst -clock clk2 -value 1
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_reset_synchronizer09

Reports duplicate reset_synchronizer constraint specifications

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer09* rule reports a violation if you specify duplicate *reset_synchronizer* constraint specifications.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if you specify duplicate *reset_synchronizer* constraint specifications:

```
[WARNING] Ignoring duplicate 'reset_synchronizer' constraint for '<sig-name>'
```

Where, *<sig-name>* refers to the signal name specified by the `-name` argument.

Potential Issues

This violation appears if you specify duplicate *reset_synchronizer* constraint specifications.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the duplicate *reset_synchronizer* constraint specification.

How to Debug and Fix

To debug this violation, check the previous *reset_synchronizer* constraint specification for the reported signal.

Example Code and/or Schematic

Consider the following example:

```
reset_synchronizer -name rst_synch::rst -reset rst -clock clk1 -value 0

reset_synchronizer -name test.S1.sync_reset -reset rst -clock clk1_tag -value 0
```

In the above example, this rule reports a violation if `sync_reset` is connected to the output pin `rst` of the `rst_synch` module.

Default Severity Label

Warning

Rule Group

Fatal must rule

Reports and Related Files

The CKSGDCInfo Report

SGDC_reset_synchronizer10

Reports conflicting reset_synchronizer constraint specifications

When to Use

Use this rule to perform sanity checks on the *reset_synchronizer* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *reset_synchronizer* constraint.
- Specify reset signals in any of the following ways:
  - By using the *reset* constraint
  - By using the automatically-generated resets after setting the *use_inferred_resets* parameter to `yes`
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_reset_synchronizer10* rule reports a violation if you have specified conflicting arguments in multiple *reset_synchronizer* constraints, that is, different arguments in the `-value` argument but same arguments in all other arguments.

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use auto-generated clock information.
- *use_inferred_resets*: Default value is `no`. Set this parameter to `yes` to use auto-generated reset information.

Constraint(s)

- *reset_synchronizer* (Mandatory): Use this constraint to specify a reset synchronizer signal along with its asserted reset value.
- *reset* (Optional): Use this constraint to specify reset signals in a design.

- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

This rule reports the following message:

```
[WARNING] Ignoring 'reset_synchronizer' constraint for
'<sig-name>', that conflicts with previous specification
```

Where, *<sig-name>* refers to the signal name specified by the `-name` argument.

Potential Issues

This violation appears if you specify conflicting arguments in different *reset_synchronizer* constraint specifications.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the conflicting *reset_synchronizer* constraint specification.

How to Debug and Fix

To fix this violation, check previous *reset_synchronizer* constraint specification for the reported signal.

Example Code and/or Schematic

Consider the following example:

```
reset_synchronizer -name top_en.w2 -reset top_en.rstb
-clock top_en.clk2 -value 0
```

```
reset_synchronizer -name top_en.w2 -reset top_en.rstb
-clock top_en.clk2 -value 1
```

For the above example, this rule reports a violation because the values of all the arguments except the `-value` argument are same in both the *reset_synchronizer* constraint specifications.

Default Severity Label

Warning

Rule Group

Fatal must rule

Reports and Related Files

The CKSGDCInfo Report

SGDC_signal_in_domain01

Reports a violation if a non-existent module is specified in the -name argument of the signal_in_domain constraint

When to Use

Use this rule to perform sanity checks on the *signal_in_domain* constraint.

Prerequisites

Specify the *signal_in_domain* constraint.

Description

The *SGDC_signal_in_domain01* rule reports a violation if the module specified by the `-name` argument of the *signal_in_domain* constraint is not instantiated in the current design.

Parameter(s)

None

Constraint(s)

signal_in_domain (Mandatory): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

The following message appears if the module *<module-name>* specified by the `-name` argument of the *signal_in_domain* constraint does not exist in the current design:

```
[FATAL] '<module name>' [SubModule] is never instantiated in the design
```

Potential Issues

This violation appears if the current design does not contain the module specified by the `-name` argument of the *signal_in_domain* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing module in the `-name` argument of the *signal_in_domain* constraint.

Example Code and/or Schematic

Consider that the design `top` does not contain the module `mod1`. Now consider the following constraints specified in an SGDC file:

```
current_design top
signal_in_domain -name mod1 -domain rd_clk -signal out1
```

For the above example, the *SGDC_signal_in_domain01* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_signal_in_domain02

Reports a violation if a non-existent pin is specified in the -domain argument of the signal_in_domain constraint

When to Use

Use this rule to perform sanity checks on the *signal_in_domain* constraint.

Prerequisites

Specify the *signal_in_domain* constraint.

Description

The *SGDC_signal_in_domain02* rule reports a violation if the pin specified by the `-domain` argument of the *signal_in_domain* constraint does not exist on the module specified by the `-name` argument of this constraint.

Parameter(s)

None

Constraint(s)

signal_in_domain (Mandatory): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

The following message appears if the pin *<pin-name>* specified by the `-domain` argument of the *signal_in_domain* constraint does not exist in the module *<module-name>*:

```
[FATAL] '<pin-name>'[TopPort] not found on/within module
'<module-name>'
```

Potential Issues

This violation message if the design module specified by the `-name` argument of the *signal_in_domain* constraint does not contain the pin specified by the `-domain` argument of this constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing pin in the `-domain` argument of the *signal_in_domain* constraint.

Example Code and/or Schematic

Consider the design module `mod1` that does not contain the `p1` pin. Now consider that you specify the following constraint:

```
signal_in_domain -name mod1 -signal ouddt1 in -domain p1
```

For the above constraint, the *SGDC_signal_in_domain02* rule reports a violation because the `p1` pin does not exist in the `mod1` module.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_signal_in_domain03

Reports a violation if a non-existent pin is specified in the -signal argument of the signal_in_domain constraint

When to Use

Use this rule to perform sanity checks on the *signal_in_domain* constraint.

Prerequisites

Specify the *signal_in_domain* constraint.

Description

The *SGDC_signal_in_domain03* rule reports a violation if the pin specified by the `-signal` argument of the *signal_in_domain* constraint does not exist on the module specified by the `-name` argument of this constraint.

Parameter(s)

None

Constraint(s)

signal_in_domain (Mandatory): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

The following message appears if the pin *<pin-name>* specified by the `-signal` argument of the *signal_in_domain* constraint does not exist in the module *<module-name>*:

```
[FATAL] '<pin-name>'[TopPort] not found on/within module
'<module-name>'
```

Potential Issues

This violation appears if the design module specified by the `-name` argument of the *signal_in_domain* constraint does not contain the pin specified by the `-signal` argument of this constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing pin in the `-signal` argument of the *signal_in_domain* constraint.

Example Code and/or Schematic

Consider the design module `mod1` that does not contain the `p2` pin. Now consider that you specify the following constraint:

```
signal_in_domain -name mod1 -signal p2 in -domain p1
```

For the above constraint, the *SGDC_signal_in_domain03* rule reports a violation because the `p2` pin does not exist in the `mod1` module.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_signal_in_domain04

The object specified in the -name argument of the signal_in_domain constraint is not a black box.

When to Use

Use this rule to perform sanity checks on the *signal_in_domain* constraint.

Prerequisites

Specify the *signal_in_domain* constraint.

Description

The *SGDC_signal_in_domain04* rule reports a violation if the object specified in the `-name` argument of the *signal_in_domain* constraint is not a black box.

Parameter(s)

None

Constraint(s)

signal_in_domain (Mandatory): Use this constraint to specify a domain for output pins of black box instances.

Messages and Suggested Fix

The following message appears if the object specified in the `-name` argument of the *signal_in_domain* constraint is not a black box:

```
[WARNING] Constraint 'signal_in_domain':Name '<object-name>' specified in field '-name' is not a blackbox. Ignoring the constraint
```

Potential Issues

This violation appears if the object specified in the `-name` argument of the *signal_in_domain* constraint is not a black box.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is not considered for SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify the name of a black box to the `-name` argument of the *signal_in_domain* constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sgclkgroup01

Invalid tag specified to the -group1 argument of the sg_clock_group constraint

When to Use

Use this rule to perform sanity checks on the *sg_clock_group* constraint.

Prerequisites

Specify the *sg_clock_group* constraint.

Description

The *SGDC_sgclkgroup01* rule reports a violation if an invalid tag name is specified to the `-group1` argument of the *sg_clock_group* constraint.

Parameter(s)

None

Constraint(s)

sg_clock_group (Mandatory): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'sg_clock_group': Clock name
'<clock-name>' specified in field '-group1' is not a valid
clock tag
```

Potential Issues

This violation appears if an invalid tag name is specified to the `-group1` argument of the *sg_clock_group* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from domain computation.

How to Debug and Fix

To fix this violation, specify the correct tag name to the `-group1` argument of the *sg_clock_group* constraint.

Example Code and/or Schematic

Consider the following SGDC file:

```
current_design top
clock -name clk1 -domain d1 -tag T1
clock -name clk2 -domain d2 -tag T2
sg_clock_group -asynchronous -group1 {T3 T4} -group2 T1
```

For the above *sg_clock_group* constraint specification, the *SGDC_sgclkgroup01* rule reports a violation because of the invalid tag names `T3` and `T4` specified in the `-group1` argument.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

None

SGDC_sgclkgroup02

Invalid tag specified to the -group2 argument of the sg_clock_group constraint

When to Use

Use this rule to perform sanity checks on the *sg_clock_group* constraint.

Prerequisites

Specify the *sg_clock_group* constraint.

Description

The *SGDC_sgclkgroup02* rule reports a violation if an invalid tag name is specified to the `-group2` argument of the *sg_clock_group* constraint.

Parameter(s)

None

Constraint(s)

sg_clock_group (Mandatory): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'sg_clock_group': Clock name
'<clock-name>' specified in field '-group2' is not a valid
clock tag
```

Potential Issues

This violation appears if an invalid tag name is specified to the `-group2` argument of the *sg_clock_group* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from domain computation.

How to Debug and Fix

To fix this violation, specify the correct tag name to the `-group2` argument of the *sg_clock_group* constraint.

Example Code and/or Schematic

Consider the following SGDC file:

```
current_design top
clock -name clk1 -domain d1 -tag T1
clock -name clk2 -domain d2 -tag T2
sg_clock_group -asynchronous -group1 {T1 T2} -group2 T3
```

For the above *sg_clock_group* constraint specification, the *SGDC_sgclkgroup02* rule reports a violation because of the invalid tag name `T3` specified in the `-group2` argument.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

None

SGDC_sgclkgroup03

Same tag specified to the -group1 and -group2 arguments of the sg_clock_group constraint

When to Use

Use this rule to perform sanity checks on the *sg_clock_group* constraint.

Prerequisites

Specify the *sg_clock_group* constraint.

Description

The *SGDC_sgclkgroup03* rule reports a violation if the same tag name is specified to the `-group1` and `-group2` arguments of the *sg_clock_group* constraint.

Parameter(s)

None

Constraint(s)

sg_clock_group (Mandatory): Use this constraint to define asynchronous relationship between clocks.

Messages and Suggested Fix

This rule reports the following violation:

```
[WARNING] Constraint 'sg_clock_group': Same clock tags have
been specified in group1 and group2 fields
```

Potential Issues

This violation appears if the same tag name is specified to the `-group1` and `-group2` arguments of the *sg_clock_group* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported constraint is ignored from domain computation.

How to Debug and Fix

To fix this violation, specify the correct unique tag names to the `-group1` and `-group2` arguments of the *sg_clock_group* constraint.

Example Code and/or Schematic

Consider the following SGDC file:

```
current_design top
clock -name clk1 -domain d1 -tag T1
clock -name clk2 -domain d2 -tag T2
sg_clock_group -asynchronous -group1 T1 -group2 T1
```

For the above *sg_clock_group* constraint specification, the *SGDC_sgclkgroup03* rule reports a violation because the same tag `T1` is specified to the `-group1` and `-group2` arguments.

Default Severity Label

Error

Rule Group

Non-fatal must rule

Reports and Related Files

None

SGDC_sync_cell02a

Reports if an incorrect non-hierarchical clock name is specified in the -from_clk argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell02a* rule reports a violation if the clock name specified in the `-from_clk` argument of the *sync_cell* constraint is a non-hierarchical name, and that clock does not exist in the specified module.

Parameter(s)

None

Constraint(s)

sync_cell (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.

Messages and Suggested Fix

The following message appears if the clock *<clk-name>* does not exist in the module *<module-name>*:

```
[INFO] Constraint 'sync_cell':clock '<clk-name>' specified with
'-from_clk' not found on/within module '<module-name>'.
Considering it as a virtual clock
```

Potential Issues

This violation appears if a design module does not contain the clock specified by the `-from_clk` argument of the *sync_cell* constraint.

Consequences of Not Fixing

If you do not fix this violation, the reported clock is considered as a virtual clock.

How to Debug and Fix

To fix this violation, specify the name of existing clock in the `-from_clk` argument of the *sync_cell* constraint.

Example Code and/or Schematic

Consider that the module `top` does not contain the `clk3` clock. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
sync_cell -name FD1P -from_clk clk3 -to_clk clk1
```

For the above example, the *SGDC_sync_cell02a* rule reports a violation.

Default Severity Label

Info

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell02b

Reports if an incorrect hierarchical clock name is specified in the -from_clk argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell02b* rule reports a violation if the clock name specified in the `-from_clk` argument of the *sync_cell* constraint is a hierarchical name, and that clock does not exist in the specified module.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the clock *<clk-name>* does not exist in the module *<module-name>*:

```
[FATAL] Constraint 'sync_cell':clock '<clk-name>' specified
with '-from_clk' not found on/within module '<module-name>'
```

Potential Issues

This violation appears if a design module does not contain the clock specified by the `-from_clk` argument of the *sync_cell* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing clock in the `-from_clk` argument of the *sync_cell* constraint.

Example Code and/or Schematic

Consider that the module instance `mod1` does not contain the `clk` clock. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
sync_cell -name FD1 -from_clk top.mod1.clk -to_clk
top.clk2
```

For the above example, the *SGDC_sync_cell02b* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell02c

Reports invalid clocks specified by the -from_clk argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell02C* rule reports a violation if the clock specified by the `-from_clk` argument of the *sync_cell* constraint is none of the following clocks:

- A clock specified by the *clock* constraint
- An automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

sync_cell (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.

Messages and Suggested Fix

The following message appears if the clock *<clk-name>* specified by the `-from_clk` argument of the *sync_cell* constraint is not a user-specified clock or an automatically-inferred clock:

```
[WARNING] Constraint 'sync_cell': Clock name '<clk-name>'
specified in field '-from_clk' is not a valid clock
```

Potential Issues

This violation appears if the clock specified by the `-from_clk` argument of the *sync_cell* constraint is none of the following clocks:

- A clock specified by the *clock* constraint
- An automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`.

Consequences of Not Fixing

If you do not fix this violation, the *sync_cell* constraint is not considered during SpyGlass analysis.

How to Debug and Fix

To fix this violation, specify any of the following clocks in the `from_clk` argument of the *sync_cell* constraint:

- A clock specified by the *clock* constraint
- An automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`.

Example Code and/or Schematic

Consider that the `scanin` clock is neither an automatically-inferred clock nor it is specified by the *clock* constraint.

In this case, the *SGDC_sync_cell02C* rule reports a violation if you specify the following constraint:

```
sync_cell -name FD1 -from_clk scanin -to_clk top.clk2
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell03a

Reports if an incorrect clock name is specified in the -to_clk argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell03a* rule reports a violation if the clock specified by the `-to_clk` argument of the *sync_cell* constraint does not exist in the specified module.

Parameter(s)

None

Constraint(s)

sync_cell (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.

Messages and Suggested Fix

The following message appears if the clock *<clk-name>* does not exist in the module *<module-name>*:

Potential Issues

This violation appears if a design module does not contain the clock specified by the `-to_clk` argument of the *sync_cell* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify the name of an existing clock in the `-to_clk` argument of the *sync_cell* constraint.

Example Code and/or Schematic

Consider that the module `top` does not contain the `clk` clock. Now consider that you specify the following constraints in an SGDC file:

```
current_design top
sync_cell -name FD1 -from_clk top.clk2 -to_clk top.clk
```

For the above example, the *SGDC_sync_cell03a* rule reports a violation.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell03b

Reports invalid clocks specified by the -to_clk argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell03b* rule reports a violation if the clock specified by the `-to_clk` argument of the *sync_cell* constraint is none of the following clocks:

- A clock specified by the *clock* constraint
- An automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the clock *<clk-name>* specified by the `-to_clk` argument of the *sync_cell* constraint is not a user-specified clock or an automatically-inferred clock:

```
[WARNING] Constraint 'sync_cell': Clock name '<clk-name>'
specified in field '-to_clk' is not a valid clock
```

Potential Issues

This violation appears if the clock specified by the `-to_clk` argument of the *sync_cell* constraint is none of the following clocks:

- A clock specified by the *clock* constraint
- An automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, specify any of the following clocks in the `-to_clk` argument of the *sync_cell* constraint:

- A clock specified by the *clock* constraint
- An automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`.

Example Code and/or Schematic

Consider that the `scanin` clock is neither an automatically-inferred clock nor it is specified by the *clock* constraint.

In this case, the *SGDC_sync_cell03b* rule reports a violation if you specify the following constraint:

```
sync_cell -name FD1 -from_clk clk1 -to_clk top.scanin
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell04

Reports if same domain clocks are specified in the -from_clk and -to_clk arguments of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint.
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`.

Description

The *SGDC_sync_cell04* rule reports a violation if the clocks specified by the `-to_clk` and `-from_clk` arguments of the *sync_cell* constraint are from the same domain.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the clocks specified by the `-to_clk` and `-from_clk` arguments of the *sync_cell* constraint are from the same domain:

```
[WARNING] Constraint 'sync_cell': Domain of clock
'<from-clock>' specified in field '-from_clk' matches with
domain of clock '<to-clock>' specified in field '-to_clk'
```

Potential Issues

This violation appears if the clocks specified by the `-to_clk` and `-from_clk` arguments of the *sync_cell* constraint are from the same domain.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, ensure that the clocks specified by the `-to_clk` and `-from_clk` arguments of the *sync_cell* constraint are from different domains.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design top

clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d1 -period 20
clock -name top.clk3 -domain d3 -period 30


sync_cell -name SYNC1 -from_clk clk1 -to_clk clk2
```

For the above example, the *SGDC_sync_cell04* rule reports a violation because the `clk1` and `clk2` clocks specified by the `-from_clk` and `-to_clk` arguments, respectively, of the *sync_cell* constraint are from the same domain `d1`.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell05

Reports a violation if an invalid domain is specified in the -from_domain argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell05* rule reports a violation if the domain specified by the `-from_domain` argument of the *sync_cell* constraint does not match with any of the following domains:

- The domain of a clock specified by the *clock* constraint
- The domain of an automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid domain *<domain-name>* is specified in the `-from_domain` argument of the *sync_cell* constraint:

```
[WARNING] Constraint 'sync_cell': Incorrect domain
'<domain-name>' specified in field '-from_domain'
```

Potential Issues

This violation appears if the domain specified by the `-from_domain` argument of the *sync_cell* constraint does not match with any of the following domains:

- The domain of a clock specified by the *clock* constraint
- The domain of an automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, specify any of the following domains in the `-from_domain` argument of the *sync_cell* constraint:

- The domain of a clock specified by the *clock* constraint
- The domain of an automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design top

clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d2 -period 20


sync_cell -name FD2 -from_domain d -to_domain d2
```

In this example, the design `top` does not contain any clock (user-specified or automatically-inferred) that belongs to the domain `d`.

Therefore, the *SGDC_sync_cell05* rule reports a violation in this case because the `d` domain is specified in the `-from_domain` argument of the *sync_cell* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell06

Reports a violation if an invalid domain is specified in the -to_domain argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the *sync_cell* constraint.

Description

The *SGDC_sync_cell06* rule reports a violation if the domain specified by the `-to_domain` argument of the *sync_cell* constraint does not match with any of the following domains:

- The domain of a clock specified by the *clock* constraint
- The domain of an automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid domain *<domain-name>* is specified in the `-to_domain` argument of the *sync_cell* constraint:

```
[WARNING] Constraint 'sync_cell': Incorrect domain
'<domain-name>' specified in field '-to_domain'
```

Potential Issues

This violation appears if the domain specified by the `-to_domain` argument of the *sync_cell* constraint does not match with any of the following domains:

- The domain of a clock specified by the *clock* constraint
- The domain of an automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, specify any of the following domains in the `-to_domain` argument of the *sync_cell* constraint:

- The domain of a clock specified by the *clock* constraint
- The domain of an automatically-inferred clock when the *use_inferred_clocks* parameter is set to `yes`

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design top

clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d2 -period 20


sync_cell -name FD2 -from_domain d1 -to_domain d
```

In this example, the design `top` does not contain any clock (user-specified or automatically-inferred) that belongs to the domain `d`.

Therefore, the *SGDC_sync_cell06* rule reports a violation in this case because the `d` domain is specified in the `-to_domain` argument of the *sync_cell* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell07

Reports if the same domain is specified in the -to_domain and -from_domain arguments of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_sync_cell07* rule reports a violation if the same domain is specified in the `-to_domain` and `-from_domain` arguments of the *sync_cell* constraint.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the same domain *<domain-name>* is specified in the `-from_domain` and `-to_domain` arguments of the *sync_cell* constraint:

```
[WARNING] Constraint 'sync_cell': Domain '<domain-name>'
specified in field '-from_domain' is same as the domain
specified in field '-to_domain'
```

Potential Issues

This violation appears if the same domain is specified by the `-from_domain` and `-to_domain` arguments of the *sync_cell* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, specify different domains in the `-from_domain` and `-to_domain` arguments of the *sync_cell* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
sync_cell -name SYNC2 -from_domain d1 -to_domain d1
```

For the above example, the *SGDC_sync_cell07* rule reports a violation because the `same` domain `d1` is specified in the `-from_domain` and `-to_domain` arguments of the *sync_cell* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell08a

Reports if an incorrect value is specified in the -from_period argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_sync_cell08a* rule reports a violation if the value specified by the `-from_period` argument of the *sync_cell* constraint is not a float value or it is a negative float value.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid value *<value>* is specified in the `-from_period` argument of the *sync_cell* constraint:

[FATAL] `Invalid [Pre-defined-range] specification '<value>' for`

'-from_period' field of constraint 'sync_cell'

Potential Issues

This violation appears if the value specified by the `-from_period` argument of the *sync_cell* constraint is not a float value or it is a negative float value.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a non-negative float value in the `-from_period` argument of the *sync_cell* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
sync_cell -name FD1P -from_period "-10" -to_period 10
```

For the above example, the *SGDC_sync_cell08a* rule reports a violation because the `-from_period` argument of the above constraint is assigned a negative float value.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell08b

Reports a violation if an invalid period value is specified in the -from_period argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_sync_cell08b* rule reports a violation if the period specified by the `-from_period` argument of the *sync_cell* constraint does not match with the period of any clock specified by the *clock* constraint.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid period is specified in the `-from_period` argument of the *sync_cell* constraint:

```
[WARNING] Constraint 'sync_cell': Incorrect period specified in
```

```
field '-from_period'
```

Potential Issues

This violation appears if the period specified by the `-from_period` argument of the *sync_cell* constraint does not match with the period of any clock specified by the *clock* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, ensure that the period specified by the `-from_period` argument of the *sync_cell* constraint matches with the period of a clock specified by the *clock* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
current_design top

clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d2 -period 30

sync_cell -name FD2 -from_period 20 -to_period 30
```

For the above example, the *SGDC_sync_cell08b* rule reports a violation because the period `20` specified by the `-from_period` argument of the *sync_cell* constraint does not match with the period of any clock specified by the *clock* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell09a

Reports if an incorrect value is specified in the -to_period argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_sync_cell09a* rule reports a violation if the value specified by the `-to_period` argument of the *sync_cell* constraint is not a float value or it is a negative float value.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid value *<value>* is specified in the `-to_period` argument of the *sync_cell* constraint:

```
[FATAL] Invalid [Pre-defined-range] specification '<value>' for
```

'-to_period' field of constraint 'sync_cell'

Potential Issues

This violation appears if the value specified by the `-to_period` argument of the *sync_cell* constraint is not a float value or it is a negative float value.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run does not proceed further.

How to Debug and Fix

To fix this violation, specify a non-negative float value in the `-to_period` argument of the *sync_cell* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
sync_cell -name FD1 -to_period xyz
```

For the above example, the *SGDC_sync_cell09a* rule reports a violation because the `-to_period` argument of the above constraint is not assigned a float value.

Default Severity Label

Fatal

Rule Group

Fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell09b

Reports a violation if an invalid period value is specified in the -to_period argument of the sync_cell constraint

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_sync_cell09b* rule reports a violation if the period specified by the `-to_period` argument of the *sync_cell* constraint does not match with the period of any clock specified by the *clock* constraint.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if an invalid period is specified in the `-to_period` argument of the *sync_cell* constraint:

```
[WARNING] Constraint 'sync_cell': Incorrect period specified in
```

```
field '-from_period'
```

Potential Issues

This violation appears if the period specified by the `-to_period` argument of the *sync_cell* constraint does not match with the period of any clock specified by the *clock* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To fix this violation, ensure that the period specified by the `-to_period` argument of the *sync_cell* constraint matches with the period of a clock specified by the *clock* constraint.

Example Code and/or Schematic

Consider the following constraint:

```
current_design top

clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d2 -period 30

sync_cell -name FD2 -from_period 10 -to_period 20
```

For the above example, the *SGDC_sync_cell09b* rule reports a violation because the period `20` specified by the `-to_period` argument of the *sync_cell* constraint does not match with the period of any clock specified by the *clock* constraint.

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_sync_cell10

Reports sync_cell constraint specifications that cover the same clock-domain crossing

When to Use

Use this rule to perform sanity checks on the *sync_cell* constraint.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SGDC_sync_cell10* rule reports a violation if multiple *sync_cell* constraint specifications cover the same clock-domain crossing.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if

```
[INFO] Constraint 'sync_cell': Overlapping constraint specification found. All the specifications will be used for rule-checking
```

Potential Issues

This violation appears if the same clock-domain crossing is covered by multiple *sync_cell* constraint specifications.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass considers all the synchronizer cells of the reported *sync_cell* constraints as valid for clock-domain crossings.

How to Debug and Fix

To debug this violation, analyze the *sync_cell* constraint specifications that cover the same crossing.

If these overlapping specifications are intentional, ignore this violation. However, if they are not intentional, modify these specifications so that they do not cover the same crossing.

Example Code and/or Schematic

Consider the following constraints specified in an SGDC file:

```
current_design top

clock -name top.clk1 -domain d1 -period 10
clock -name top.clk2 -domain d2 -period 20

sync_cell -name SYNC1 -from_clk clk1 -to_clk clk2
sync_cell -name SYNC2 -from_domain d1 -to_domain d2
```

In the above example, the *sync_cell* constraint specifications are overlapping as they cover the common clock domain crossing from the `clk1` clock to the `clk2` clock.

Default Severity Label

Info

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_virtualclock01

Reports a virtual clock specified in combination with other real or virtual clock in abstract_port constraint

When to Use

Use this rule to perform sanity checks on the *abstract_port* constraint specifications.

Prerequisites

Specify the *abstract_port* constraint before running this rule.

Description

A violation is reported if any of the following combination is specified in the `-clock`, `-from`, or `-to` arguments of the *abstract_port* constraint:

- A virtual clock with a real clock
- A virtual clock with another virtual clock

Parameter(s)

None

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.

Messages and Suggested Fix

The following message appears if a virtual clock is specified in combination with other real or virtual clocks in the *abstract_port* constraint.

```
[WARNING] Virtual clock in combination with other real or
virtual clock specified in '<field-name>' field of
abstract_port constraint. Ignoring the constraint
```

Potential Issues

This violation appears if any of the following combination is specified in the `-clock`, `-from`, or `-to` arguments of the *abstract_port* constraint.

- A virtual clock with a real clock
- A virtual clock with another virtual clock

Consequences of Not Fixing

If you do not fix this violation, SpyGlass run will ignore the *abstract_port* constraint.

How to Debug and Fix

To debug and fix this violation, specify a signal virtual clock with unique domain.

Example Code and/or Schematic

Consider the following code snippets:

Example 1

```
abstract_port -module MOD -ports P1 -clock VCLK1 VCLK2
```

In the above code, two virtual clocks are specified in the abstract_port constraint, so SpyGlass will report the SGDC_virtualclock01 violation and will ignore this constraint.

Example 2

```
abstract_port -module MOD -ports P2 -clock VCLK1 CLK1
```

In the above code, a real clock is specified along with virtual clock in the abstract_port constraint, so SpyGlass will report the *SGDC_virtualclock01* violation and will ignore this constraint.

SGDC_virtualclock02

Reports same virtual clock in abstract_port constraint specified on both input & output port of a module

When to Use

Use this rule to check if both the input and output port/pin of a module are associated with same virtual clock by using the *abstract_port* constraint.

Prerequisites

Specify the *abstract_port* constraint before running this rule.

Description

A violation is reported if both the input and output port of a module are associated with the same virtual clock by using the *abstract_port* constraint.

Parameter(s)

None

Constraint(s)

- *abstract_port* (Optional): Use this constraint to define abstracted information for block ports.

Messages and Suggested Fix

The following message appears if both the input and output port or pin of a module are associated with the same virtual clock using abstract_port.

`[WARNING] Same Virtual clock <virt-clk-name> is used for both input <input-port-name> and output <output-port-name> ports of the module <module-name>`

Potential Issues

This violation appears if both the input and output port or pin of a module are associated with the same virtual clock by using the *abstract_port* constraint.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass considers the domains of both the input and output ports as the same domain. In addition, the virtual clock mapping is not performed for validation.

How to Debug and Fix

To debug and fix this violation, specify a single virtual clock association.

Example Code and/or Schematic

Consider the following `abstract_port` constraints:

Example 1

top.sgdc

```
current_design top


clock -name clk1
clock -name clk2
sgdc -import block block.sgdc
```

block.sgdc

```
current_design block
abstract_port -ports IN1 -clock VCLK1
abstract_port -ports OUT1 -clock VCLK1
```

In the above example, the `IN1` and the `OUT1` input and output ports of the abstracted block are associated with the same virtual clock and therefore the SGDC_virtualclock02 rule reports a violation.

Note that if you specify a verification virtual clock for both input and output ports, the `SGDC_virtualclock02` rule does not report a violation. For example, consider the following scenario where the verification clock is specified for both input and output ports:

```
current_design block
abstract_port -ports IN1 -clock VCLK1 -start
abstract_port -ports OUT1 -clock VCLK1
```

In the above scenario, the `SGDC_virtualclock02` rule does not report a violation.

SGDC_virtualclock03

Reports virtual clocks that have the same name as the domain name specified in the -domain argument of the clock constraint

When to Use

Use this rule to report virtual clocks that have the same name as the domain name specified in the `-domain` argument of the *clock* constraint for a design object.

Description

The SGDC_virtualclock03 rule reports violation if a virtual clock has the same name as domain name specified in the `-domain` field of the *clock* constraint specified on a real design object.

Parameter(s)

None

Constraint(s)

- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if a virtual clock has the same name as domain name specified in the `-domain` field of the *clock* constraint.

```
[WARNING] Virtual clock name <virtual-clock-name> is specified in the -domain field of clock constraint
```

Potential Issues

This violation appears if a virtual clock has the same name as domain name specified in the `-domain` field of the *clock* constraint.

Consequences of Not Fixing

If you do not fix this violation, the design intent might be lost.

How to Debug and Fix

To debug and fix this violation, specify a unique name for virtual clock.

Example Code and/or Schematic

Consider the following SGDC file:

```
clock -name clk1 -domain d1

clock -tag d1

abstract_port -ports p1 -clock d1
```

In the above scenario, the domain name and the virtual clock names are the same and therefore the rule reports a violation.

SignalTypeSetup

Checks the signal specified by the -name argument of the signal_type constraint

When to Use

Use this rule to perform existence checks for signals specified by the *signal_type* constraint.

Description

The *SignalTypeSetup* rule reports a violation if the signal specified by the `-name` argument of the *signal_type* constraint is not any of the following:

- Output of a clock-domain crossing
- Source of a clock-domain crossing
- Input port acting as the source of a clock-domain crossing

Parameter(s)

None

Constraint(s)

- *signal_type* (Mandatory): Use this constraint to specify the signal type (control or data).

Messages and Suggested Fix

The following message appears when the signal specified by the `-name` argument of the *signal_type* constraint does not exist in any clock-domain crossing:

```
[WARNING] 'signal_type' constraint <signal-name> of type
<control | data> is not applied in the current design
<current_design>
```

Potential Issues

This violation appears if the signal specified by the `-name` argument of the *signal_type* constraint is not any of the following:

- Output of a clock-domain crossing

- Source of a clock-domain crossing
- Input port acting as the source of a clock-domain crossing

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the *signal_type* constraint specified for the reported signal. As a result, the signal is checked for data as well as control synchronization schemes by `Ac_sync_group` rules.

For details on these rules, see *Working With the Ac_sync_group Rules*.

How to Debug and Fix

To fix this violation, update the `-name` argument of the *signal_type* constraint to specify a signal that is one of the following:

- Output of a clock-domain crossing
- Source of a clock-domain crossing
- Input port acting as the source of a clock-domain crossing

Example Code and/or Schematic

Consider that the `top.des` signal is not the part of any clock-domain crossing in a design.

In this case, if you specify the following constraint, the *SignalTypeSetup* rule reports a violation:

```
signal_type -name top.des -type data
```

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SyncCellSetup

Reports a violation if the sync_cell constraint does not synchronize any crossing in the current design

When to Use

Use this rule to check cases in which the *sync_cell* constraint does not synchronize any crossing in a design.

Prerequisites

Specify the following information before running this rule:

- Specify the *sync_cell* constraint.
- Specify clock signals in any of the following ways:
  - By using the *clock* constraint
  - By using the automatically-generated clocks after setting the *use_inferred_clocks* parameter to `yes`

Description

The *SyncCellSetup* rule reports a violation if the *sync_cell* constraint does not synchronize any crossing in the current design.

Parameter(s)

use_inferred_clocks: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.

Constraint(s)

- *sync_cell* (Mandatory): Use this constraint to specify synchronizer cells that should be considered valid for crossings that contain specified frequencies, source/destination clocks, or domains.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears if the *sync_cell* constraint does not synchronize any crossing in the design *<current-design>*:

```
[WARNING] 'sync_cell' constraint is not used to synchronize any
```

```
crossing in the current design '<current-design>'
```

Potential Issues

This violation appears the *sync_cell* constraint does not synchronize any crossing in the current design.

Consequences of Not Fixing

If you do not fix this violation, SpyGlass ignores the reported constraint.

How to Debug and Fix

To debug this violation, perform the following actions:

- Analyze the clocks specified by the -from_clk/-from_domain and -to_clk/-to_domain arguments of the *sync_cell* constraint.
- Check the specification of the module specified by the -name argument of the *sync_cell* constraint and ensure that it is instantiated as a destination module.
- If a combinational logic exists in a crossing, specify an appropriate value for the allow_combo_logic.
- Remove the reported constraint if it is redundant.

  For example, if the same crossings are matched by some other constraint, you can remove the reported constraint.

Example Code and/or Schematic

Not applicable

Default Severity Label

Warning

Rule Group

Non-fatal must rule

Reports and Related Files

No report or related file

SGDC_clock_path_wrapper_module01

Reports user-defined wrapper modules in the clock-path

When to Use

Use this rule to check the clock path hierarchy.

Prerequisites

Specify the wrapper modules by using the *clock_path_wrapper_modules* constraint.

Description

The *SGDC_clock_path_wrapper_module01* rule reports all user-defined wrapper modules present in the propagated clock path

Parameter(s)

- *use_inferred_clocks*: Default value is `no`. Set this parameter to `yes` to use automatically generated clock information.
- *same_domain_at_gate*: Default value is `no`. Set the value of the parameter to an allowed value to optimize the inference of domains on clock merging gates.

Constraint(s)

- *clock_path_wrapper_modules* (Mandatory): Use this constraint to exclude modules from the checks performed by the *Clock_hier01*, *Clock_hier02*, and *Clock_hier03* rules.
- *clock* (Optional): Use this constraint to specify clock signals in a design.

Messages and Suggested Fix

The following message appears when a user-specified wrapper module is detected:

```
[INFO] User-specified wrapper module '<mod-
name>'(instance:'<hierarchy>') detected at net '<net-name>'
```

Potential Issues

Not applicable

Consequences of Not Fixing

Not applicable

How to Debug and Fix

View the **Incremental Schematic** of the violation message. The schematic highlights the clock-path to the wrapper module.

Example Code and/or Schematic

In this example, the wrapper module is defined as:

```
clock_path_wrapper_module -names cgc_lib
```

The *SGDC_clock_path_wrapper_module01* rule reports a message for this wrapper, as follows:

```
[INFO] User-specified wrapper module 'cgc_lib'(instance:'CG4')
detected at net 'test.gclk'
```

The following schematic illustrates the instantiation of this wrapper module.

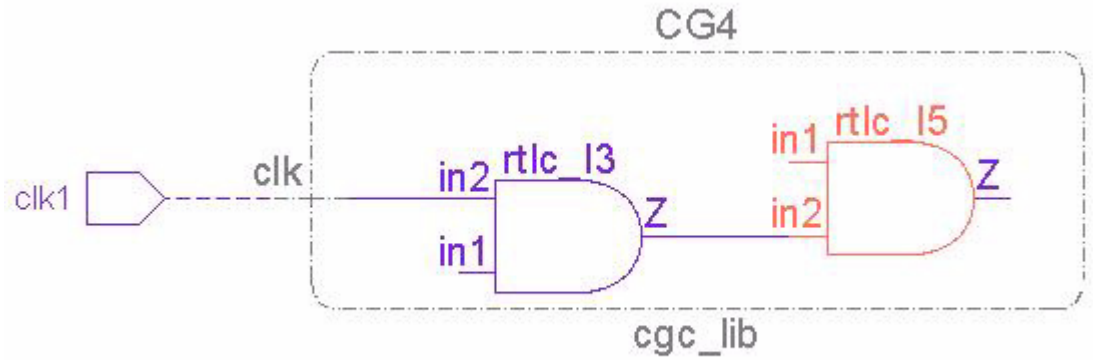


FIGURE 436. Schematic for the SGDC_clock_path_wrapper_module01 Rule

Default Severity Label

Info

Rule Group

Non-fatal must rule

Reports and Related Files

The CKSGDCInfo Report

Rule Grouping in SpyGlass CDC

The rules in SpyGlass CDC are divided into the following rule groups:

Rule Group	Rules
SETUP	Setup_Clock01, Setup_port01
FIND	Clock_info01, Clock_info02, Reset_info01, Reset_info02
INFORMATION	Clock_info03a, Clock_info03b, Clock_info03c, Clock_info05, Clock_info05b, Clock_info06, Clock_info07, Clock_info14, Clock_info16, Clock_info17, Clock_Reset_info01, Propagate_Clocks, Propagate_Resets, Reset_info09, Setup_clockreset01
VERIFY	Clock_check01, Clock_check02, Clock_check03, Clock_check04, Clock_check05, Clock_check06a, Clock_check06b, Clock_check07, Reset_check01, Reset_check02, Reset_check03, Reset_check04, Reset_check05, Reset_check06, Reset_check07, Reset_check09, Reset_check10, Clock_Reset_check01, Clock_Reset_check02, Clock_Reset_check03, Clock_glitch01, Clock_glitch02, Clock_glitch03, Clock_glitch04, Clock_converge01
SYNCHRONIZATION	Clock_sync03a, Clock_sync03b, Clock_sync05, Clock_sync06, Clock_sync08a, Clock_sync09, Reset_sync01, Reset_sync02, Reset_sync03, Reset_sync04
DELTADELAY	Clock_delay01, Clock_delay02, DeltaDelay01, DeltaDelay02, NoClockCell, PortTimeDelay
ADV_CLOCKS	Ac_cdc01a, Ac_cdc01b, Ac_cdc01c, Ac_cdc08, Ac_crossing01, Ac_fifo01, Ac_glitch01, Ac_glitch02, Ac_handshake01, Ac_handshake02, Ac_init01, Ac_initstate01, Ac_license01, Ac_multitop01, Ac_sanity01, Ac_sanity02, Ac_sanity03, Ac_sanity04, Ac_report01, Clock_sync03a

Terminologies in SpyGlass CDC

The following terminologies are used in SpyGlass CDC:

Properties	*Assertions*	*Clock Cycle Count and Sequential Depth*
Design Period	*Initial State*	*Functional Flip-Flop*
Reset Flip-Flop	*Reset Cone*	*Synchronous Clocks*
Repeaters	*Derived Resets*	*Control Signals*
Design Assumptions		

Properties

A property means the expected design behavior to be tested.

Following are some examples of properties in the context of ASIC designs:

- There can be only one active driver on a tristate bus.
- There should be no floating busses.
- There should be no data loss while moving from fast to slow clock domain.
- A request should be acknowledged within five clock cycles.

Assertions

Assertions are directives to the formal verification tool to verify the given *Properties*.

In other words, when it is asserted that a property holds true, the *Properties* becomes an assertion. Therefore, in the formal verification flow, these two terms are used interchangeably.

Clock Cycle Count and Sequential Depth

Whenever a rule-violation message is reported by SpyGlass, a sequential depth is reported that indicates the number of clock cycles it takes to start from an initial state and reach the location of rule-violation.

In case of a single clock that is active only at posedge or only at negedge, this number is straightforward. However, in a multi-clock environment with the clocks active at posedge, negedge, or both, the cycle count can be interpreted differently.

In order to provide an accurate idea of the number of cycles, SpyGlass reports two numbers for the sequential depth. These two numbers are the same for a design with a single clock active at a single edge. For all other cases, these two numbers are defined as follows:

1. Number of cycles of fastest clock in the cone of influence of the property being checked

   If a property is applied to a set of nets of a design, then the fastest clock of the relevant nets is extracted and if a message is occurring, then the number of cycles of the fastest clock at the time of rule-violation is reported. In the simple case of a single clock system, this number will be the number of clock cycles of the given clock. In this scheme, a clock cycle is accounted for as soon as one of the edges has occurred. As a result, a half cycle is considered as a full cycle.

2. Number of non-overlapping edges

   This number represents both the positive edge count and the negative edge count from the initial state to the rule-violation where if two edges of two clocks occurred at the exact same time, the counter is incremented only by one. SpyGlass generates a vector signal named `verification_cycle` to represent the counter value. This value is displayed in the Waveform Viewer when an assertion failure occurs.

   Note that both edges are counted regardless if registers are triggered at posedge, negedge, or both. In particular, in a single clock system, this number will be equal to twice the number of clock cycles (~+1 due to the fact that a half cycle is accounted as a full cycle).

   For example, the following figure shows three clocks waveforms:

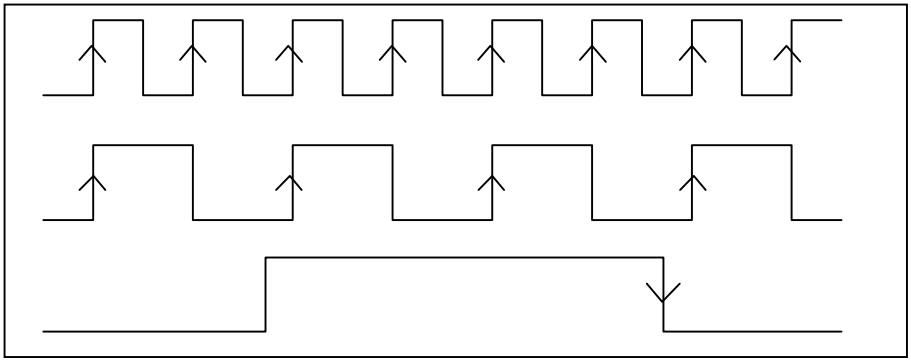


FIGURE 1. Clock cycle count and sequential depth

In the above figure, for a given window of time, there are:

- 8 clock cycles (number of cycles of the fastest clock)
- 17 edges (15 edges for the top waveform, 0 for the middle since all the edges are covered by the top waveform, and 2 edges for the last clock since it is not overlapping with any edges of the previous clocks)

NOTE: *Although not all edges are active, the counting includes all edges - active or inactive.*

Design Period

Functional analysis complexity increases with the number of asynchronous clocks in a design. To understand how clock frequencies affect the functional analysis process, consider two clocks running with 17 ns period and 13 ns period respectively. If the rising edges of the two clocks are aligned at time 0 ns, the next time the rising edges will again be aligned corresponds to 221 ns (LCM of two clock periods). This means that the design behaves asynchronously for 221 ns.

Any functional analysis process that would exploit repetition (for proving a property, for example) would have to analyze the design at least for this period of time, which may correspond to many evaluations of logic in the design. SpyGlass CDC refers to this period as the **Design Period**.

The number of non-overlapping edges of all clocks covered by the Design Period is known as the **Design Cycle**. A smaller Design Cycle will lead to better formal QOR. For more details, refer the *High Design Cycle* section.

Design Cycle Exceeding the Threshold Value

For cases in which design cycle exceeds the threshold value of 65535, SpyGlass does not report the actual value of the design cycle. Instead, it reports the following message:

```
Design Cycle: <Exceeds threshold value of 65535>
```

Initial State

The initial state is a register-value assignment from which functional analysis will start its analysis. Example: Given a 4-bit counter and a property asserting that the counter will eventually reach 15, this assertion passes in 5 cycles if the counter is initialized with 10, whereas it will pass in 15 cycles if the counter is initialized to 0.

An initial state may or may not be a reset state of a design. A reset state of a design is a register-value assignment obtained by resetting a design using a reset signal (may be user-specified). SpyGlass can obtain an initial state in four different ways:

1. Direct register-value assignment using the *reset* constraint.
2. State value generated by an external simulation engine as a VCD file and read in to SpyGlass using the *simulation_data* constraint.
3. Initialization vector that can reset registers using the *define_tag* constraint.
4. Find an initial state automatically. If you do not provide an initial state and/or do not provide an initialization vector, then SpyGlass determines an initial state using the following approach:
   - ❒ Use reset port to reset registers.
   - ❒ Random analysis by applying stimulus to the inputs and simulating to find *valid register value assignment*. Although not a reset state, the assignment is guaranteed to be reached by stimulating the inputs of a design.

     Use the *fa_ieffort* parameter to specify the effort that SpyGlass puts in while searching the initial state of a design during simulation.
   - ❒ Functional analysis for a valid register-value assignment.

Automatic initial state search (case 4) cannot be combined with other cases. Therefore, any register not initialized by the user-specified vector or the initial state remains at “x”, which can impact the outcome of functional analysis.

You must validate the initial state of a design before running any functional analysis. SpyGlass provides various reports as well as RTL back-annotation and schematic highlight for the initial state exploration.

Functional Flip-Flop

Refers to any flip-flop that is not feeding a reset of any other flip-flop, even after layers of sequential logic.

The following figure shows a functional flip-flop:

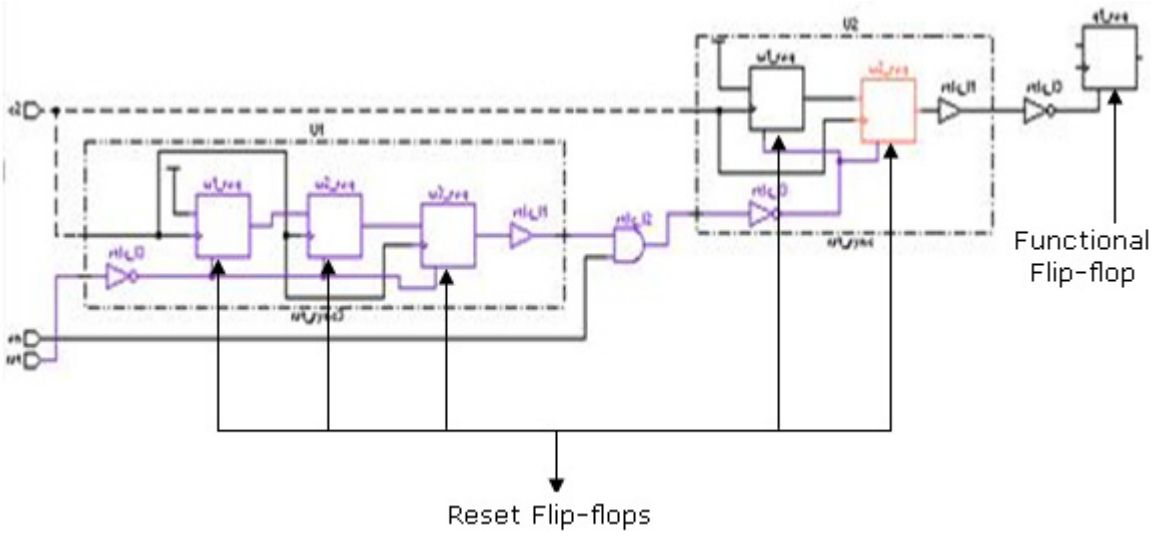


FIGURE 2. Functional Flip-Flop

Reset Flip-Flop

Refers to any flip-flop receiving a user-defined asynchronous reset as a control to its asynchronous reset pin.

The following figure shows the example of a functional flip-flop and a reset flip-flop:

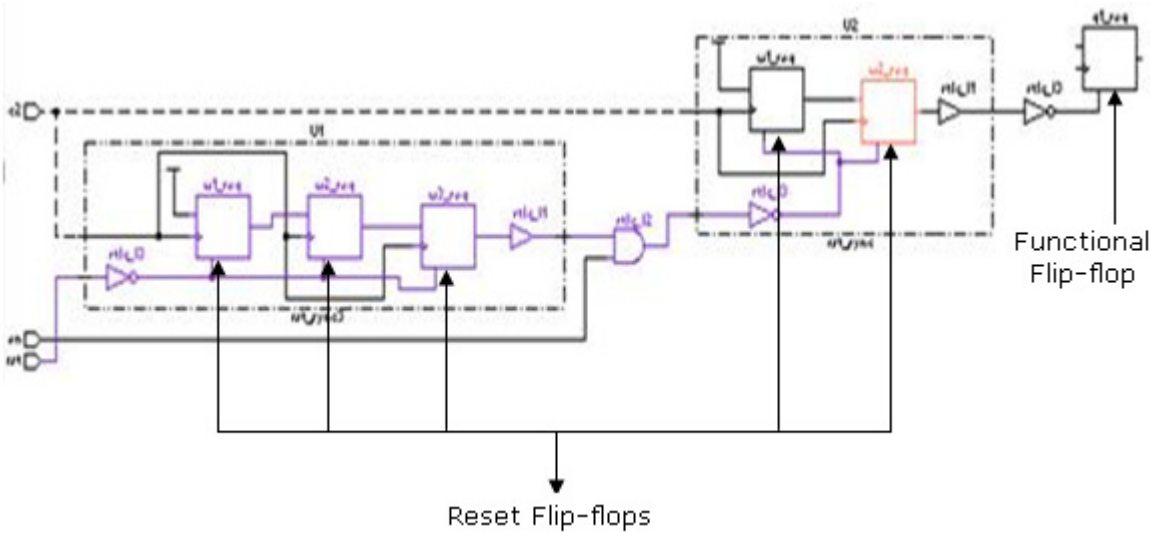


FIGURE 3. Reset Flip-Flops

Reset Cone

Consider the following figure:

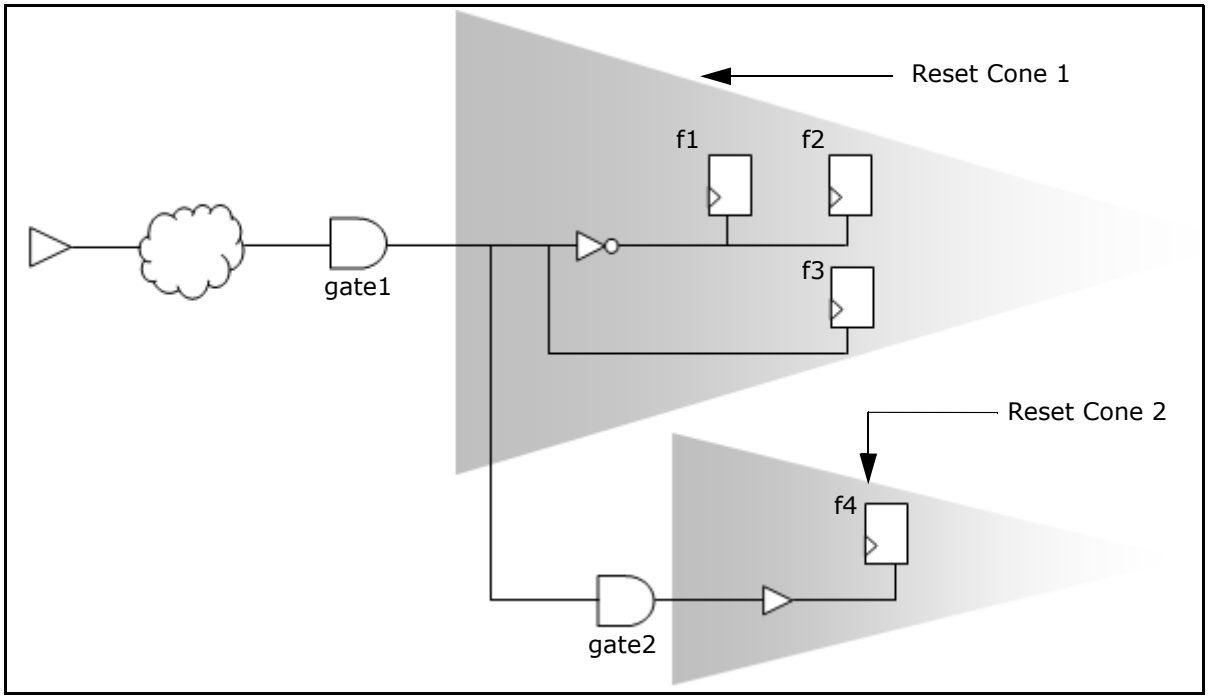


FIGURE 4. Reset Cones

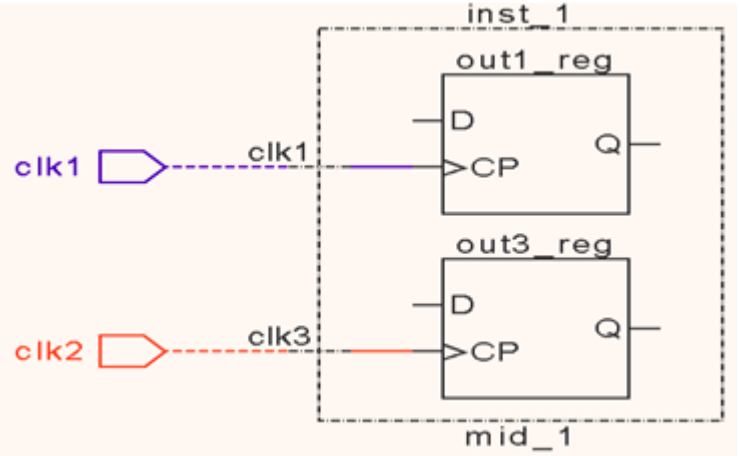
In the above figure, the area covered after the output of the gate `gate1` is one reset cone. Similarly, the area covered after the output of the gate `gate2` is another reset cone.

Within each reset cone, a violation is reported for only one flip-flop. For example, for the reset cone `Reset Cone 1`, either of the `f1`, `f2`, or `f3` flip-flop is reported.

Synchronous Clocks

Refers to the clocks that have the same domain.

The following figure shows the example of synchronous clocks `clk1` and `clk2` that belong to the `d1` domain:




```
// constr.sgdc

current_design top
clock -name clk1 -domain d1
clock -name clk2 -domain d1
```

FIGURE 5. Synchronous clocks

Repeaters

Repeaters are sequential elements that are inserted in a path to meet certain timing requirements. These are usually inserted manually by designers at the RTL stage to add additional cycles of delay based on clock frequencies of source and destination clocks.

Derived Resets

These are the resets that come from the output of a flip-flop that is receiving a primary reset or another derived reset.

In the following figure, `temp3_reg.Q` is the derived reset:

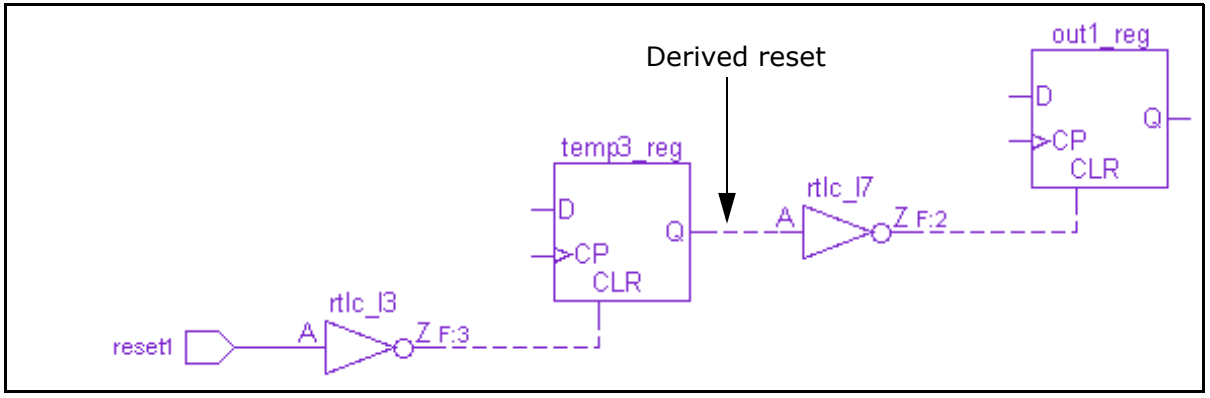


FIGURE 6. Example of a derived reset

Control Signals

A source signal is called as a control signal if it is:

- A scalar source signal, such as `in2` in *Figure 6*.
- A source specified through the *signal_type* `-type control` constraint.

For example, consider the scenario shown in the following figure:

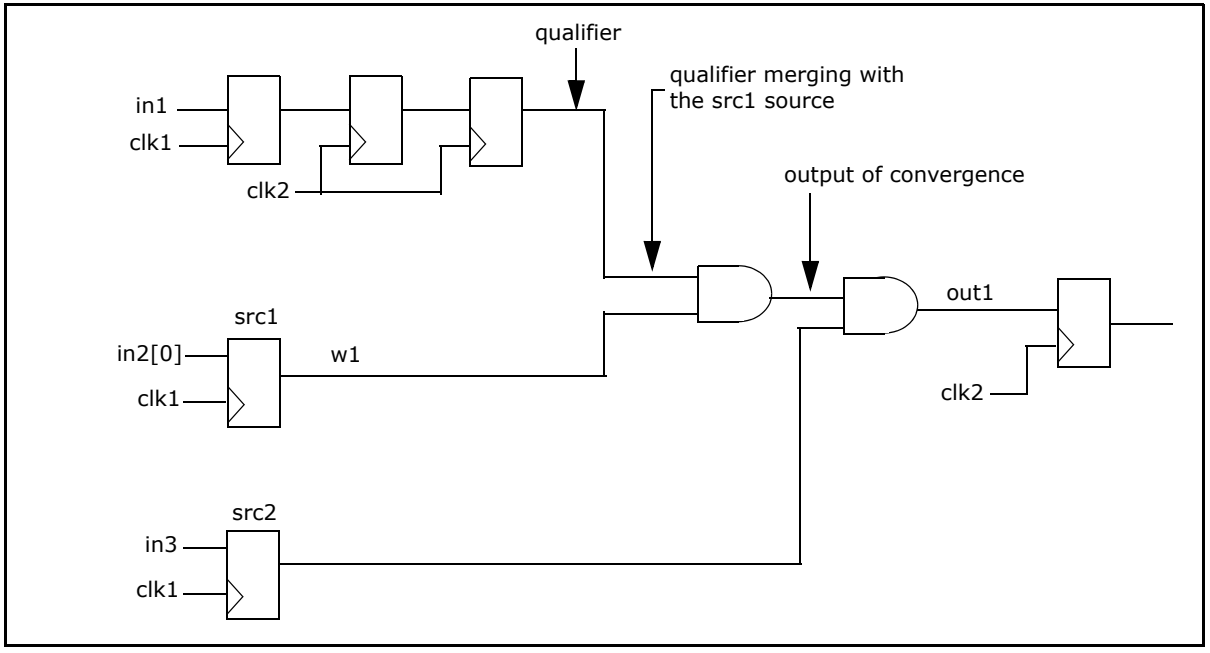


FIGURE 7. Qualifier Merging with a Source

In the above scenario, specify the following constraint to consider w1 as a control signal:

```
signal_type -name w1 -type control
```

Design Assumptions

Design assumptions are the constraints that you specify in an SGDC file to capture the design intent.

For example, you can specify the assumption that a design signal is static by applying the *quasi_static* constraint on that signal.

Appendix: SGDC Constraints

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

In addition, you can restrict SpyGlass analysis to certain objects in a design by specifying these objects by using SGDC commands.

SpyGlass Design Constraints Used by SpyGlass CDC

The following table lists the SGDC commands used by SpyGlass CDC solution:

SpyGlass CDC Solution		
abstract_file	*abstract_port*	*allow_combo_logic*
assume_path	*breakpoint*	*cdc_attribute*
cdc_false_path	*cdc_filter_path*	*cdc_filter_coherency*
cdc_matrix_attributes	*clock*	*clock_sense*
define_reset_order	*define_tag*	*deltacheck_ignore_instance*
deltacheck_ignore_module	*deltacheck_start*	*deltacheck_stop_instance*
deltacheck_stop_module	*deltacheck_stop_signal*	*fifo*
generated_clock	*gray_signals*	*input*
ip_block	*meta_design_hier*	*meta_inst*
meta_module	*meta_monitor_options*	*monitor_time*
network_allowed_cells	*num_flops*	*noclockcell_start*
noclockcell_stop_instance	*noclockcell_stop_module*	*noclockcell_stop_signal*
no_convergence_check	*output*	*output_not_used*
port_time_delay	*quasi_static*	*quasi_static_style*
qualifier	*repeater*	*reset*
reset_filter_path	*reset_synchronizer*	*sdc_data*
set_case_analysis	*sg_clock_group*	*sgdc*
simulation_data	*signal_in_domain*	*signal_type*
sync_cell	*watchpoint*	*validation_filter_path*

List of Topics