Conformal® Constraint Designer User Guide

Product Version 23.2
October 2023

Contents

1

Introduction to the Conformal Constraint Designer Solution

Overview

Conformal Constraint Designer automates the validation and refinement of constraints to ensure that timing constraints are valid throughout the entire design process, helping designers achieve rapid timing closure.

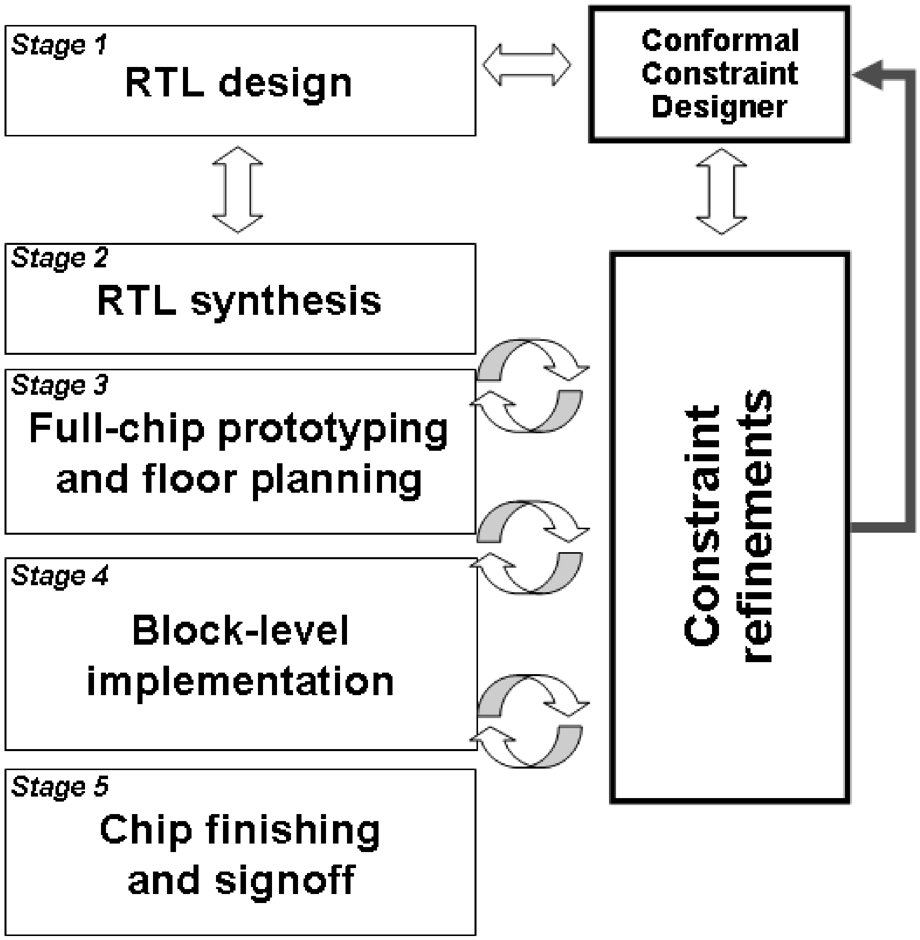
- Shortens design cycles: comprehensive analysis environment enables checking the creation and integration of block-level and top-level constraints
- Improves quality of silicon in terms of area, timing, and power by using higher-quality constraints
- Reduces risk of re-spins through formal validation of exceptions
- Speeds convergence for timing closure by quickly validating failing timing paths as functionally false
- Creates initial constraints effortlessly with the SDC Advisor directly from RTL (XL configuration)
- Links formal validation with simulation through PSL and SVA assertions, making for easy adoption (XL configuration)

Conformal Constraint Designer at Critical Stages

Conformal Constraint Designer offers a complete, functional constraint validation solution that enables rapid timing closure when working with implementation tools. Conformal Constraint Designer fits into the design cycle, at critical stages (as illustrated in Figure 1-1 on page 17):

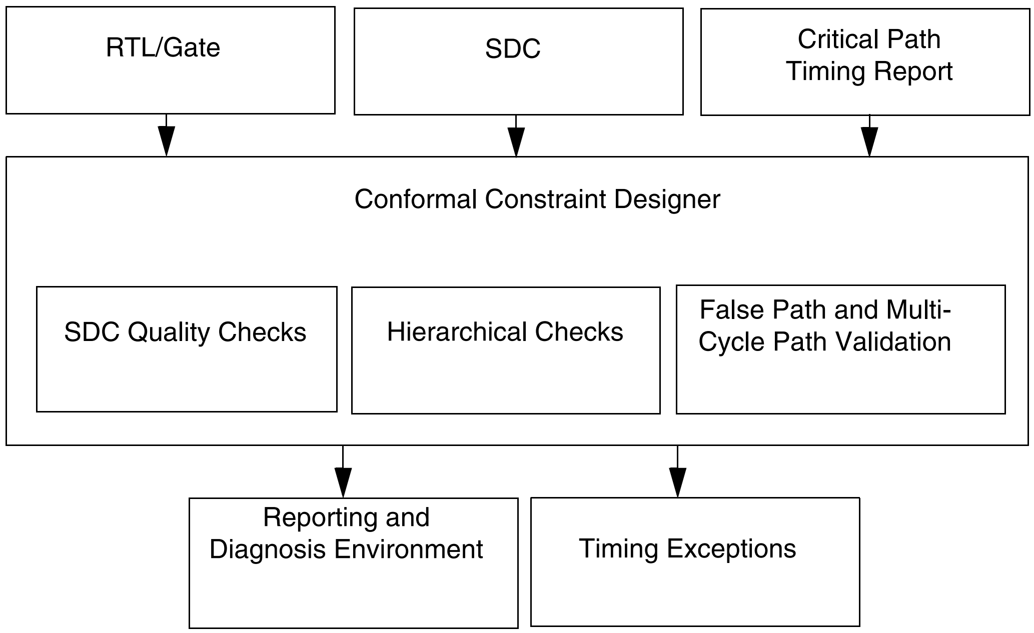
- Stage 1—After RTL designers create the initial constraints, Conformal Constraint Designer can start at the RTL level, and verify the validity of the syntax, structure, and methodology of the initial constraints.
- Stage 2—After synthesis translates the SDC, you can use the Conformal Constraint Designer on timing netlists or modified constraints for the same purpose.
- Stage 3—Conformal Constraint Designer refines constraints by identifying false paths from critical paths.
- Stages 4 and 5—Conformal Constraint Designer can perform refinement, similar to stages 2–3, on the netlists and revised constraints outputted by place and route.

Figure 1-1 Conformal Constraint Designer in a Design Cycle



Key Features of Conformal Constraint Designer

The following figure illustrates the Conformal Constraint Designer features.



- *SDC quality checks* report exception inconsistencies or conflicts.

  Conformal Constraint Designer can check design constraints, specifically SDCs. Timing constraint files give major constraint input to synthesis and layout tools. Due to the current trend of SoC design methodology, constraint files of different modules can come from different sources (such as IP vendors or other departments). It is possible that constraint files have inconsistent specification or contain conflicting constraints. Conformal Constraint Designer verifies SDC files against these inconsistencies using SDC quality checks. You can read in an SDC file, view its rule violations, and view the source code or schematics associated with a rule violation. SDC quality checks in Conformal Constraint Designer saves you time by identifying basic SDC issues that relate to structure and methodology early in the design cycle.

- *Hierarchical checks* report conflicts and inconsistencies between block and chip-level SDCs.

- *False path and multi-cycle path validation*—Formally validates FP/MCP exceptions.

  Conformal Constraint Designer provides a *Validation Manager* that functionally validates the FP constraints and timing reports. This automated, formal verification feature saves you from having to manually validate false paths

Note: While you diagnose exception properties, you will also have access to the schematic display, the waveform viewer, and the source code windows.

Conformal Constraint Designer provides an integrated debugging and analysis environment, which you can use to debug and analyze errors that are associated with constraints. This environment includes the following tools:

- SDC Rule Manager—You can use this tool to examine individual rule violations for diagnosis.
- Validation Manager—The Validation Manager functionally validates FP/TRV constraints. You can use this tool to add, delete, report, and validate exception checks, and to diagnose failed exception checks.
- Source Code Viewer, Waveform Display, and the Schematic Viewer—You can use these tools to further debug problems using RTL and netlist information.

Design Constraints Overview

Design constraints are typically described in a design constraint format. One of the most widely-used formats is the SDC format, which describes the design intent and surrounding constraints for synthesis, clocking, timing, power, test, environmental, and operating conditions.

Note: SDC conforms to Tcl syntax. For a complete list of the available constraints in SDC, contact Synopsys.

Key Components of SDC

This section describes the key functions of SDC.

Defining Clocks

A *clock* is an element that helps define circuit behavior. For synchronous inputs (such as, input `D`), you have to specify setup and hold times, with respect to the `CLOCK` input. Setup and hold times specify that data input must remain stable for a specified interval, before and after the clock input changes.

Use the `create_clock` command in SDC to define a clock.

Generated Clocks

A generated clock is a clock that was created from a clock source. This capability models clock dividers and multipliers to create a new clock.

Use the `create_generated_clock` command in SDC to define a generated clock.

Virtual Clocks

A *virtual clock* is a named waveform that is not attached to any clock source in the design. You can use virtual clocks as references when you need to specify input and output delays relative to a clock.

Use the `create_clock` command with `-name`, without specifying any source objects, in SDC to define a virtual clock.

Clock Latency

Clock Source Latency is the time a clock signal takes to propagate from its origin point in the ideal waveform to the clock port in the design.

Use the `set_clock_latency` command in SDC to define clock source latency.

Clock Transition

Clock transition is the transition time of register clock pins.

Use the `set_clock_transition` command in SDC to define clock transition.

Clock Uncertainty

Clock uncertainty (also known as, *skew*) specifies the allowable time delay between two signals. Exceeding this time can cause devices to behave unexpectedly. The skew of a clock tree is the difference between the minimum and maximum insertion delays of the tree.

Use the `set_clock_uncertainty` command in SDC to define clock uncertainty.

Defining Boundary Constraints

Boundary constraints define delay constraints that are set on a design's input and output ports.

Set Input Delay

Set input delay defines the arrival time relative to a clock. For bidirectional ports, you can specify the path delays for both input and output modes. Use the `-add_delay` option to capture multi-clock delay relations.

Use the `set_input_delay` command in SDC to set input delays.

Set Output Delay

Similar to `set_input_delay`, you can use the `set_output_delay` command to define the delay induced by the logic driven by the output port.

Defining Exceptions

By default, static timing tools assume that all timing paths are single cycle paths (given that there is at least one defined clock). However, there are exceptions to this assumption where the path can either be a multi-cycle path or a false path.

Multi-cycle path

A *multi-cycle path* is a path that requires more than one clock period for execution.

Use the `set_multicycle_path` command in SDC to set multi-cycle paths.

False Paths

A *false path* is a path that can never be sensitized in the actual circuit. These paths are logically and functionally impossible. Specify false paths to synthesis and static timing analysis tools so that timing analysis is performed only on all true timing paths.

Use the `set_false_path` command in SDC to set false paths.

Specifying Modes

Designs can run in different modes, such as the functional and test mode. In the test mode, the logic related to test is turned on. In the functional mode, this logic is turned off.

This command specifies that a port or pin is at a constant logic value of one or zero. This method specifies the design mode, without altering a netlist. For a constant imposed by `set_case_analysis` on a port or pin, it is propagated through the network.

Typical Problems with SDC Files

Given the SDC components explained in Key Components of SDC on page 19, this section gives examples of some of problems that the Conformal Constraint Designer can flag.

Defining Clocks

```
create_clock -period 7 -waveform {0 3.5} [get_ports {clk_fpci66m}]
create_clock -period 14 -waveform {3.1 10.1} [get_ports {clk_ref25m}]
set_input_delay 0.75 -min -clock clk_ref39p {IN2}
set_clock_transition  0.75 -min clk_fpci66m
create_generated_clock -edges {1 2 3} -duty_cycle 50 -source \
   [get_ports clk] [get_pins div_reg/Q]
```

In this code, the following problems can occur:

- `set_input_delay` is set on a pin with respect to an undefined clock
- `set_clock_transition` might be outside the characterization range
- `create_generated_clock` might have arguments that cannot appear together, such as `-edges` and `-duty_cycle`

Defining Boundary Constraints

```
set_output_delay 3 -clock "clk_pci" [get_ports {decalfipmi_fllen_z}]
set_input_delay 2.5 -clock "clk_ref25m" [get_ports {ipmitxbfr_rdata[0]}]
```

In this code, the following problems can occur:

- There could be an input pin without a corresponding `set_input_delay`
- Input and output delay values might exceed the clock period for a combinational path (*input_delay* + *output_delay* > *clock_period*)

Defining Exceptions

```
set_multicycle_path -through [get_nets pci_am_block/mul*] -setup 2
set_multicycle_path 2 -setup -start -through \
   [get_pins {pci_if_top1/pci_datapath1/pci64_xfer_reg}]
```

In this code, the following problems can occur:

- A single-cycled path might be defined as an MCP
- An MCP might have an incorrect cycle count
- An FP might be specified as an MCP

Miscellaneous

```
set_load -pin_load 0.2 [get_ports {decalf_eeprm_data[15]}]
set_driving_cell -lib_cell BUFX8  -library xl_c [get_ports {strback}]
set_max_fanout 2 [get_ports {decalf_pmtch_strback}]
set_max_transition 2 [current_design]
```

In this code, the following problems can occur:

- There might be an undefined input transition
- `set_driving_cell` might be specified on a clock port

Example Files

You can access the following example files under the `<*install_dir*>/share/cfm/ccd/examples` directory. Each example contains a `dofile.ccd`.

- `FP_EX`—Validates FPs within an SDC file.
- `RULES_EX1`—Flags overlapping clock tree issues within an SDC file.
- `RULES_EX2`—Flags issues with input and output delays that are specified versus clocks that are not virtual.

Accessing Online Help and Documentation

Launching Cadence Help

The online documentation system is called Cadence Help.

From the main GUI, click on the Help menu item and navigate to the HTML version of the document that you wish to view. This will bring up Cadence Help.

Some GUI windows also have a Help button that will launch Cadence Help.

Getting Help for Cadence Help

After launching Cadence Help, press F1 or choose Help - Contents to display the help page for Cadence Help.

Getting Help on Commands to Run Tools

You can display a list of options for any of the tools and utilities by typing the tool or utility name followed by the -help option as follows:

```
% tool_name -help
```

Example:

```
% ccd -help
```

Getting Help on Commands and Messages

Use the `MAN` command without any options to list all of the available commands. However, to view specific help information, use the following commands:

- `command_name`—To view command usage for a specific command, enter the `MAN` command followed by the command name. For example:

  ```
  man read design
  ```

- `-verbose`—To view expanded information about a specific command, enter the `MAN` command, followed by the command name, and the `-verbose` option. For example:

  ```
  man read design -verbose
  ```

- `message_name`—To view help for a particular rule check message, enter the `MAN` command followed by the message name. For example:

```
man f10
```

- `-message`—To view a list of all the rule check messages, use the `MAN` command with the `-message` option. For example:

  ```
  man -message
  ```

For more information on the `MAN` command, use the following command from within the tool:

```
%man man
```

Searching the Help Database for Specified Strings

The `SEARCH` command searches the Help database of commands and options for matches to strings you specify. Include the `-usage` option to display the command and its options.

Using the Help Menu

You can use the *Help* menu to get more information on commands, licenses, documentation, and Cadence support.

Accessing Help from Command Windows

A *Help* button is available in many command windows. Unlike the *Help* button on the main window, when you left-click the *Help* button in command windows, the Conformal software automatically executes *Help – Commands* and displays the information for the related command in the Command Help window.

Accessing User Documentation

Use the following procedure to view the user guides and reference manuals.

1. Click the *Help* pull-down menu located at the far right end of the menu bar.
2. Click *<Book Name> (pdf)* or *<Book Name> (html)*.

   The PDF reader launches and displays the PDF version of the book. Or, Cadence Help launches the HTML version. If you choose the HTML version, you will have access to all the other books in the documentation set through Cadence Help.

   Note: You must have a PDF reader to access the documentation. To download the current version of Adobe Acrobat Reader, visit the following web page:

   http://www.adobe.com/support/downloads/main.html

Accessing Product Information

Use the following procedure to display the Cadence company logo, the product version number and date, mailing address, phone and fax numbers, and web page and E-mail addresses.

1. Click the *Help* drop-down menu located at the far right end of the menu bar.
2. Click *About*.

Accessing License Information

From the *Help* drop-down menu in the main window, click *License* to view information regarding all the installed Conformal software licenses. The report appears in the Transcript window and includes information such as the current user, feature, and expiration date.

You can also use the `LICENSE` command to review the current license status. The current status of the license appears in the transcript output.

Related Documents

The following lists the documents related to Conformal Constraint Designer:

- *Conformal Constraint Designer User Guide*

  Describes how to use the Conformal Constraint Designer solution.

- *Conformal Constraint Designer Command Reference*

  Describes the commands for Conformal Constraint Designer.

- *Conformal Constraint Designer Database Access Object and Attribute Reference*

  Describes the database access objects and attributes that control

- *Conformal Constraint Designer Rule Check Reference*

  Describes the modeling messages, policy rule checks lint rule checks, the CDC rule checks, and the atomic checks.

- *ConformalHDL Rule Check Reference*

  Describes the HDL rule checks that apply to all Conformal tools.

2

Additional Resources

Related Documents

The following lists the documents related to Conformal Constraint Designer:

- *Conformal Constraint Designer User Guide*

  Describes how to use the Conformal Constraint Designer solution.

- *Conformal Constraint Designer Command Reference*

  Describes the commands for Conformal Constraint Designer.

- *Conformal Constraint Designer Database Access Object and Attribute Reference*

  Describes the database access objects and attributes that control

- *Conformal Constraint Designer Rule Check Reference*

  Describes the modeling messages, policy rule checks lint rule checks, the CDC rule checks, and the atomic checks.

- *ConformalHDL Rule Check Reference*

  Describes the HDL rule checks that apply to all Conformal tools.

Additional Learning Resources

Cadence offers the following training course on Conformal Constraint Designer:

Conformal Constraint Designer

3

Getting Started

Product and Installation Information

For product, release, and installation information, see the README file at any of the following locations:

- downloads.cadence.com, where you can review the README before you download the Conformal software.
- In the software installation, where it is also available when you are using or running the Conformal software.
- At the top level of your installation hierarchy.

Supported File Formats

The following table lists the file formats and versions that the Conformal software supports, and the related commands that parse these files.

VHDL	IEEE Std 1076-1993 (default) IEEE Std 1076-1987	`READ DESIGN -vhdl` `READ LIBRARY -vhdl`
Verilog	IEEE 1364-1995 (default) IEEE 1364-2001	`READ DESIGN -verilog` `READ LIBRARY -verilog`
SystemVerilog	IEEE 1800-2005	`READ DESIGN -systemverilog` `READ LIBRARY -systemverilog`
Liberty	2007.3	`READ DESIGN -liberty` `READ LIBRARY -liberty`
SDC	2.0	`READ SDC` `READ HIERARCHICAL SDC`

Start-Up Command Options

The following table shows the options to the `ccd` command for starting the Conformal Constraint Designer software.

`-L`	Launches Encounter® Conformal® Constraint Designer L
`-XL`	LaunchesEncounter® Conformal® Constraint Designer XL with advanced validation capabilities

`-MCC`	Launches Encounter® Conformal® Constraint Designer XL with multi-mode SDC checks and SDC comparison capabilities
`-VDS`	Launches Encounter® Conformal® Constraint Designer with SDC checks and SDC integration capabilities (available only with the Virtuoso Digital Signoff Timing Solution).

The `ccd` and `ccd-t` commands have the following additional options. This list is also available using the `ccd -help` command *before* you start your session.

`-Help`	Lists out all the available start-up options.
`-Gui | -NOGui`	Starts the session in GUI or non-GUI mode.
`-TclMode`	After the session starts, the tool enters Tcl mode. **Note:** This does not apply to the `ccd-t` start-up command.
`-NOColor`	Turn off color-coded messaging when in non-GUI mode.
`-Banner | -NOBanner`	Specifies whether to display the Conformal Constraint Designer banner during startup.
`-Info`	Display the product information and exit.
`-Version`	Displays the product version. Once you have started your session, you can also use the `VERSION` command to display the software version number. This is useful when starting a transcript log file to ensure that the file contains a reference to the version that created the results.
`-NOLIcwait`	If all licenses are checked out, exit immediately.
`-Dofile <filename>`	Runs the script *`<filename>`* after starting Conformal Constraint Designer. See "Dofile Command Files" on page 55 for more information.
`-LOGfile <filename>`	Sets up a log filed called *`<filename>`*.
`-RESTART_checkpoint <checkpoint_file_name> [-protect <password>]`	Restarts a session that was saved using the `CHECKPOINT` command.

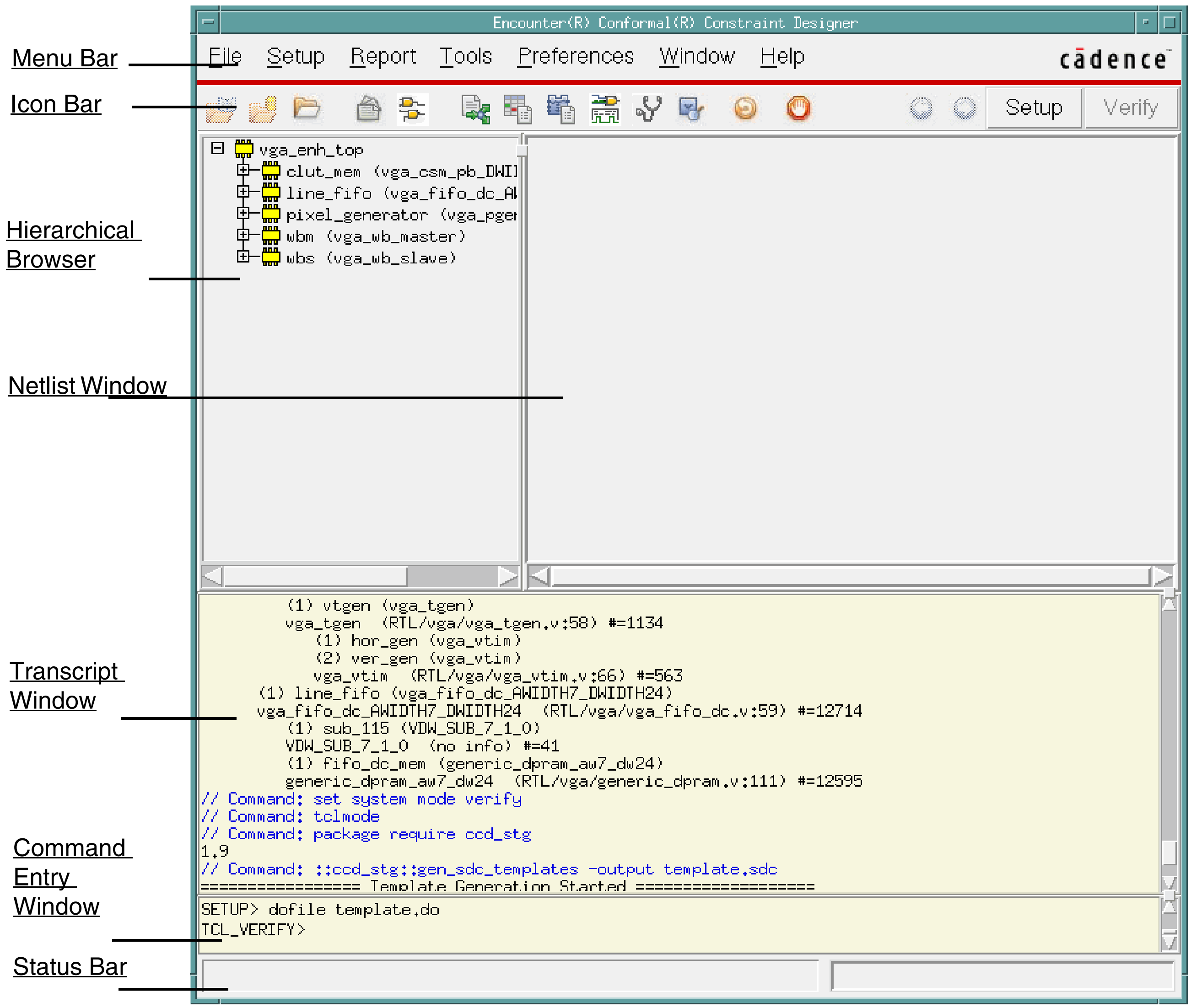
`-PARallel <filename>`	Start Conformal Constraint Designer in parallel execution mode.

4

Interacting with Conformal Constraint Designer

Conformal Constraint Designer Main Window

This section describes the basic features of the Conformal Constraint Designer main window (the window that appears when you launch Conformal Constraint Designer in GUI mode).



Selecting Multiple Items

Various Conformal Constraint Designer windows let you select multiple items using any of the following methods:

- Click and drag, highlighting each item as you drag the mouse.
- Hold down the `Shift` key and click on two items; this selects every item on the list between the two.
- Hold down the `Control` key and click on items that you want to select. With the `Control` key depressed, you can jump around the item list.

Copying Information from Infoboxes

You can copy information from the infoboxes into various GUI windows using the following key strokes:

- `Ctrl-q` copies the infobox contents into a static text window. You can have several infoboxes displayed at once.
- `Ctrl-m` copies the infobox contents into the transcript window where it is added to the the log file.

Drag and Drop

The Conformal Constraint Designer offers the drag and drop functionality to provide shortcut methods for performing particular tasks.

To perform drag and drop:

1. Select or highlight the item you want to drag and drop. To select an item, point and click on it.
2. Press and hold the *middle mouse button* while you drag the item to its destination.
3. Release the mouse button to drop the item in place.

Note: When you click with the middle button, the Conformal Constraint Designer displays the name of the selected object in an ivory text box. As you move the box to the Source window, the background of the text box changes to black if you have reached a window into which you can drop items.

Menu Bar

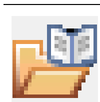
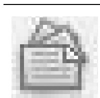
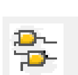
The menu bar represents categories of commands. Each of the headings supports a drop-down menu of related commands. Click on a heading to display the group of represented commands. The menu names are enabled or disabled (grayed) according to the current operating mode (Setup or Verify). With the drop-down menu visible, click an enabled command to execute it.

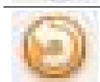
The drop-down menus support meta-key invocation for menu commands using mnemonics. The mnemonic for each command name is indicated by an underscore. For example, invoke the *File– Read Design* command by typing meta-f, then d. The meta key is usually the diamond key on Sun keyboards, or the Alt key on other keyboards.

Icon Bar

The main Conformal Constraint Designer GUI window's Icon Bar includes buttons that perform specific commands. Click an icon to execute the related command or access the related tool or window. If an icon is grayed out, it is not available in your current system mode. For example, if the system is in the Setup operating mode, the Static Property Manager icon is grayed out, since it is available in the Verify operating mode only.

The following table lists the various icons accompanied by a short description and the cross-reference to additional information.

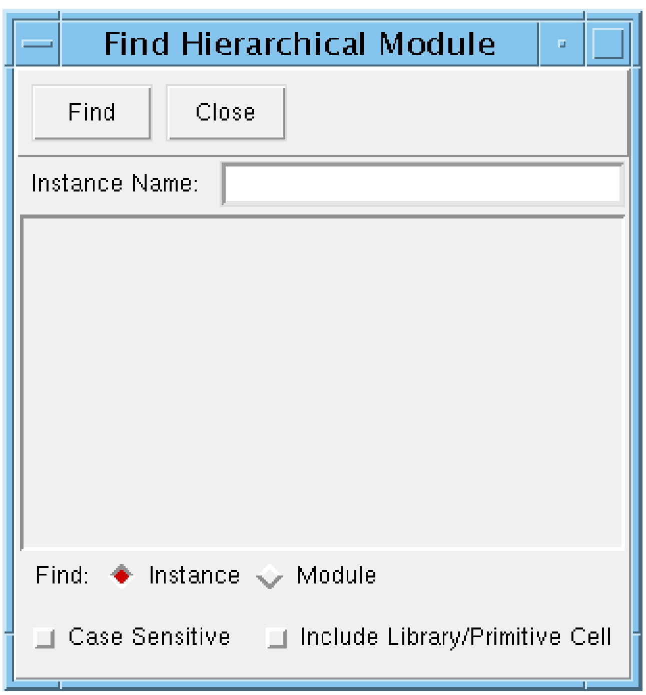
Icon	Icon Name	Description
	*Read Library*	Opens the Read Library window.
	*Read Design*	Opens the Read Design window.
	*Read SDC*	Opens the Read SDC window.
	*Source Code Manager*	Opens the Source Code Manager.
	*Flattened Schematics*	Opens the Flattened Schematics window.

Icon	Icon Name	Description
	*Rule Manager*	Opens the Rule Manager.
	*SDC Lint Manager*	Opens the SDC Lint Manager.
	*Modeling Rule Manager*	Opens the Modeling Rule Manager.
	*SDC Rule Manager*	Opens the SDC Rule Manager.
	*Validation Manager*	Opens the Validation Manager. See “Managing Exception Checks from the GUI” on page 174.
	*SDC Advisor*	Opens the SDC Advisor. See “SDC Command Generation” on page 117.
	*Find*	Opens the Find Hierarchical Module window.
	*Refresh Hierarchy*	Refreshes the main window display. Module expansions are minimized and the netlist section is cleared.
	*Stop*	Interrupts processing.
	*Previous Page*	Switches Netlist section to the left (previous) page of the currently displayed page. **Note:** This icon is enabled when a module contains more than 500 elements (including pins, nets, and instances).
	*Next Page*	Switches Netlist section to the right (next) page of the currently displayed page. **Note:** This icon is enabled when a module contains more than 500 elements (including pins, nets, and instances).
Setup	Setup Mode button	Changes the system mode to Setup.

Icon	Icon Name	Description
Verify	Verify Mode button	Changes the system mode to Verify.
cadence	*About*	Opens the Company/Product Information window.

Finding Hierarchical Modules

Click the *Find* icon or press `Ctrl-f` to open the Find Hierarchical Module form.



The following lists the fields and options for the Find Hierarchical Modules form.

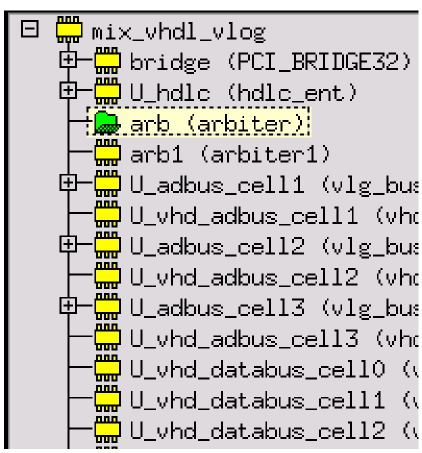
Instance Name	Specifies the instance or module name to search.
Object List	Lists all matching instance or module names. Double-clicking on the object name highlights the selected object in the Hierarchical Browser.

Find	Specifies either an *Instance* or *Module* object for the search.
Case Sensitivity	Turns on the case-sensitivity for the search.
Include Library/Primitive Cell	Extends the search.

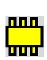
Hierarchical Browser

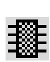
The Hierarchical Browser is located in the main Conformal Constraint Designer GUI window. It displays the hierarchical modules of the design. The root module is displayed along with its hierarchical contents. Click the square-enclosed (+) and (-) icons to expand or compress the hierarchical display. The instance name is displayed first, and module names are enclosed in parentheses ( ).

Note: You can drag and drop items from the hierarchy browser into the Source Code Manager.



The elements in the Hierarchy Browser are represented by the following icons.

 Module

 Blackbox

Executing Commands on Selected Modules

You can execute certain commands when using an individual module or instance’s pop-up menu in the hierarchical display. You cannot execute commands on library cells.

1. Click a module or instance to select it.
2. Right-click to display the pop-up menu.
3. Drag the cursor to choose a command.

The following tables list the executable commands.

Pop-Up Options for Root Modules

You can execute certain pop-up options from within the Hierarchy Browser. (Commands cannot be executed on selected library cells.) The following table of options relates to the root module.

Option	Description
Set SDC Design	Runs the `SET SDC DESIGN /` command.
Pin Constraints	Opens the Pin Constraints form.
Pin Equivalences	Opens the Pin Equivalences form.
Tied Signals	Opens the Tied Signals form.
Source Code	Opens the Source Code viewer for the module.
Schematics	Opens the schematic view of the root module.

Pop-up Options for Modules or Instances Other Than Root

Execute the following commands from within the Hierarchy Browser for a hierarchical module or instance that is not a root module.

Command	Description
Root Module	Executes the `SET ROOT MODULE` command in Setup system mode.
Add Black Box	Executes the `ADD BLACK BOX` command in Setup system mode for the selected module and all its instances.
Tied Signals	Opens the Tied Signals form.
Source Code	Opens the Source Code viewer and highlights the selected location.

Command	Description
Schematics	Opens the schematic view of the selected module.

Setting the Current SDC Design

Use the *Set SDC Design* pop-up command while in Setup mode to set the current SDC design.

To set the current SDC design to the selected module instance:

➤ Right-click on a module instance or the root design, and choose *Set SDC Design – Current* from the pop-up menu.

To set the current SDC design to the root design:

➤ Right-click on a module instance, and choose *Top* from the pop-up menu.

Note: When you set the current SDC design to a module instance, you might not see the other blocks in the hierarchical browser. To view the other blocks, set the current SDC design to the root design (using *Set SDC Design – Top*).

Adding Black Boxes

The *Add Black Box* pop-up menu option adds or deletes instances or modules as black boxes in the Hierarchy Browser display. The black box icon appears or disappears, accordingly.

To add a black box module:

When using the following procedure, a blackbox symbol appears adjacent to the module name and all the instances of that module.

1. In the Hierarchy Browser, click an instance or module to select it.
2. Right-click and choose *Add Black Box* from the pop-up menu.

To delete a black box module:

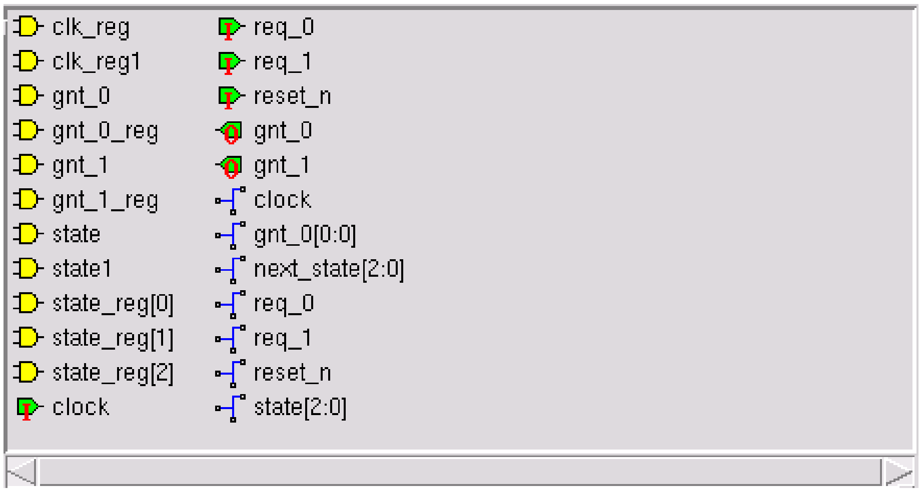
When using the following procedure, the blackbox symbol disappears from the position next to all the instances of the applicable module.

1. In the Hierarchy Browser, click a black box instance or module to select it.

2. Right-click and choose *Delete Black Box* from the pop-up menu.

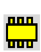
Netlist Window

The Netlist window is located in the main Conformal Constraint Designer GUI window. It displays the netlist of the module you selected in the Hierarchy Browser.



Netlist Window Icons

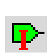
The elements in the Netlist window are represented by the following icons.

 Module

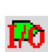
 Circuit Element

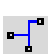
 Library Cell

 Assertion Constraint

 Input Pin

 Output Pin

 Input/output Pin

 Wire

Viewing Source Code or Gates for Selected Modules

Open the Flattened Schematics window or the Source Code Manager when you select an element in the Netlist window. This does not apply to selected library cells.

1. Click an element to select it.
2. Right-click to display the pop-up menu.
3. Drag the cursor to choose *Source Code* or *Flattened Schematics*.

Transcript Window

The Transcript window is located in the main Conformal Constraint Designer window. It displays information regarding the current session, including warnings and error messages. Additionally, when you enter a report command, the report information is displayed in the Transcript window. The text is color-coded for greater visual accessibility. For example, error messages appear in red text.

```
// Parsing file lib_nw.lib ...
// Warning: (IGN5.1) Liberty State Table is not supported and is ignored (occurrence:1)
// Warning: (HRC1.4) Module/entity is empty (black-boxed) (occurrence:4)
// Note: Read Liberty library successfully
// Command: read library lib_core.lib -liberty -append
// Warning: Resetting all SDC information and CCD parameters.
// Parsing file lib_core.lib ...
// Warning: (IGN5.1) Liberty State Table is not supported and is ignored (occurrence:1)
// Warning: (HRC1.4) Module/entity is empty (black-boxed) (occurrence:28)
// Note: Read Liberty library successfully
// Command: read design pci_block.v
// Parsing file pci_block.v ...
```

You can save the transcript to a file.

Clearing the Transcript Window

To clear the contents of the Transcript window, right-click in the Transcript window to open the pop-up menu and choose *Clear*.

Command Entry Window

The Command Entry window is located near the bottom of the main Conformal Constraint Designer GUI window. It provides a way to enter commands from the keyboard, as an alternative to using the menus and icons.

Clearing the Command Entry Window

Right-click in the Command Entry window to open the pop-up menu, and choose *Clear*.

Status Bar

The Status Bar is located at the bottom of the main Conformal Constraint Designer GUI window. It displays the status of certain processing commands, such as:

- Processing the design
- Opening the schematic viewer
- Proving properties

The progress meter at the right end of the status bar changes incrementally and a corresponding percentage number shows the level of completeness.

Exiting the GUI and Software

Use the following procedures to exit from the GUI mode and Conformal software, and save and restore GUI settings.

Exiting the GUI

To switch from GUI mode to the non-GUI mode, do the following:

1. Click the *File* drop-down menu.
2. Click *Exit GUI*.

Exiting the Software

By default, the Conformal software does not automatically save GUI settings for future sessions. To save your preferred settings, use the GUI Exit window and click the *Save GUI settings* check box. Included in the list of supported settings are:

- Window size and location (excluding schematics and source code windows)
- Fonts
- Schematic colors
- Waveform Display window widths for: main window, signal name, and value field

To exit from the session:

The Exit window prompts you to confirm the exit action, which is a precautionary measure, as all design, mapping, comparison, and diagnosis information is lost when you terminate the session.

1. Choose *File – Exit*.
2. If you want to save you preferred settings for future sessions, select *Save GUI settings*.
3. Do one of the following:
   - ❑ To confirm exit, click *Yes.*
   - ❑ To close the Exit window and return to the session, click *No*.

File Menu

The following forms are accessible from the *File* menu:

- *Read Library*—Specify library filenames you will include with a design.
- *Read Design*—Specify the filename the Conformal software reads in as the design.
- *Read SDC*—Select the constraint file that you want to apply to your design.
- *Read Critical Path*—Specify a timing report to generate false paths.
- *Save Dofile*—Save commands to a dofile to be used later as a batch file to repeat the Conformal Constraint Designer session.
- *Do Dofile*—Execute a batch file of commands, or a Dofile set of commands from a previous session.

- *Save Transcript*—Save a transcript to a file at any point during a session. It contains all of the information from the beginning of the session up to the point when you save the file.
- *Exit GUI*—Switch Conformal from the GUI mode to the non-GUI command line mode.
- *Exit*—Exit the Conformal software completely.

Setup Menu

The following menu options are accessible from the *Setup* menu:

- *Log File*—Create or append to an existing transcript file.
- *Alias*— Create, view, or remove an alias.
- *Search Path*—Create, modify, or delete directory search paths.
- *Environment*—Set global options for the design.
- *Pin Constraints*—Add and delete pin constraints to primary input pins.
- *Pin Equivalences*—Add and delete pin equivalences.
- *Tied Signals*—Add and delete tied signals to floating nets and pins.
- *Root Module*—Specify a new root module.
- *Renaming Rule*—Add, delete, and test renaming rules.
- *Notranslate Modules*—Add and delete design or library modules that will not be translated.
- *Initial State*—Specify an initialization sequence for a circuit through a VCD dump file or an initial sequence file, and add individual initial states.
- *Assertion Constraint*—Turn assertion library instances into proof assumptions or proof obligations.

Report Menu

See "Running Reports" on page 363

Tools Menu

The *Tools* drop-down menu gives you access to the integrated debugging tools:

- *Source Code Manager*
- *Flattened Schematics*
- *HDL Rule*

  Display the library and design rule checks that the Conformal Constraint Designer performs during parsing. If the Conformal Constraint Designer generates any messages, the corresponding rule check number is highlighted in red text in the HDL Rule Manager. Use the integrated debugging tools to determine the cause of violations.
- *Modeling Rule*

  In Setup mode, you can change the severity level of rule violations. In Verify mode, you can view checks performed during design elaboration. If the Conformal Constraint Designer generates any messages, the corresponding modeling rule is highlighted in red text in the Modeling Rule window. Use the integrated debugging tools to determine the cause of violations.
- *SDC Rule*
- *SDC Advisor*
- *SDC Generation*
- *SDC Integration*
- *Validation Manager*
- *SDC Compare*
- *Open Module Schematics*

  Displays the schematic representation of the selected module. This troubleshooting feature allows you to examine module structure and automatically trace and isolate module elements.
- *Close All Schematics*

  Closes all schematic viewer windows.
- *Clock Tree Schematics*

  Opens the Clock Tree window to examine a clock's structure in the Flattened Schematics window.
- *Rule Manager*
- *SDC Lint Manager*

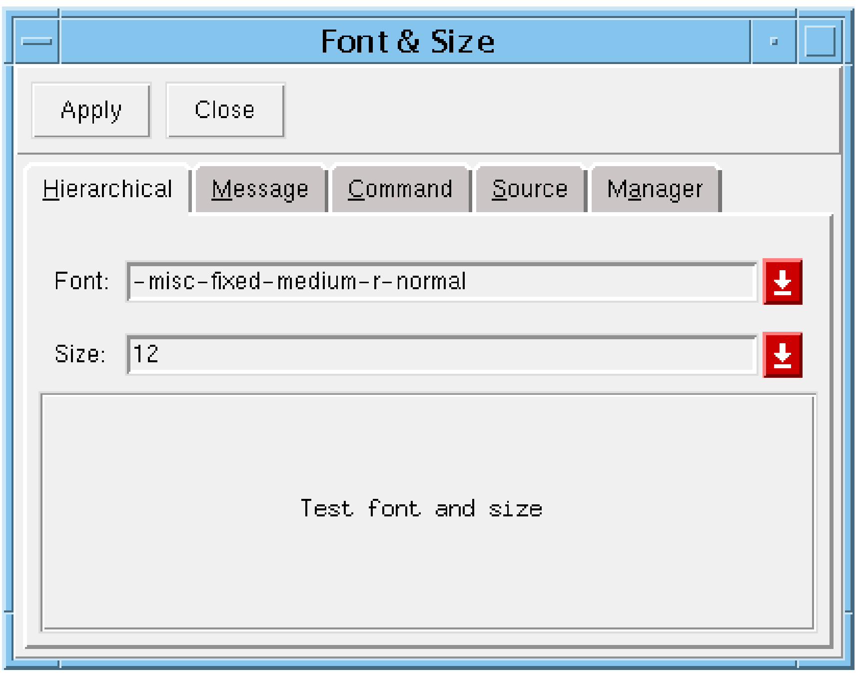
Preferences Menu

You can use the `Preferences` pull-down menu from the main window.

Font & Size

Click on the *Preferences* drop-down menu to access the Font & Size window. You can use the Font & Size form to change the font style and font size for various Conformal windows. This also displays an example of the selected font style and size.

➤ Choose *Preferences – Fonts*.



The Font & Size form has five tabs (pages) for the following:

- *Hierarchical* – Hierarchical Module window
- *Message* – Transcript window
- *Command* – Command Entry window
- *Source* – Source Code Manager
- *Manager* – Manager windows

Changing Font Style

To change the font style, click the *Font* down-arrow to display a list of font styles, select the font style, and click *Apply*.

To change the font size, click the *Size* down-arrow to display a list of font sizes, select the font size, and click *Apply*.

Browser On

Displays or hides the Browser in the main window.

Icon Bar

Displays or hides the Icon Bar in the main window.

Show Static Infobox

Enables or disables the information box that displays when moving your mouse pointer over the object. When this is on, the information box will remain after moving your pointer away from the object. When off, the information box will disappear when moving the mouse pointer from the object.

Simplified Schematic Viewing

You can control how many netlists are displayed in the Schematics Viewer. By default, the tool determines the number of netlists to display (which can cause redundant netlists and/or complicated netlist routing to be included).

To enable simplified schematics, use the Preferences - *Simplified Schematics Viewing Options* menu item from the main Conformal window.

This feature is available for beta testing, its functionality may change prior to final release. For more information this feature, refer to the Web Interface, which you can launch using the `SET WEB_INTERFACE` command

Initial Command Files

When you start the Conformal software, it searches for and executes initial command files (`.ccd.rc`). The software checks for the `CCD_RC` environment variable. If this variable is set, Conformal uses the file this variable refers to and does not search for other files.

If the `CCD_RC` variable is not set, the software continues the search as follows:

1. Installation directory
2. Home directory
3. Current working directory

Note: The software does not include `.ccd.rc` in the release.

If one or more of these files exist, the software runs them in the order noted above. This search order gives you flexibility in using the initial command file. You can set up initial command files for any or all of the following purposes:

- Global initial command file for all users
- Global initial command file for an individual user
- Initial command file for a test case

The file contents vary according to your needs; for example, they can include commands, aliases, and dofiles. You can use this file for any purpose at the system, user, and local levels.

Important

Do not use an initialization file to run a complete batch file. Use dofiles, as explained in the following section, for this purpose.

Dofile Command Files

The Conformal Constraint Designer command files (other than initial command files) are called dofiles. As you execute commands in GUI mode using the drop-down menus and windows, the Conformal software displays the text for the corresponding commands in the Transcript window, which is located in the lower portion of the main window.

Execute dofiles during startup or with the `DOFILE` command. When you create a dofile, follow these guidelines:

- Each new command must begin on a new line.
- Two or three slashes (`//` or `///`) precede comments.
- Dofiles can execute additional dofiles.

You can use the `DOFILE` command (or the `-dofile` command option at startup) to read in and execute a command file that includes any set of commands.

Using a Dofile at Startup

In GUI mode, the `-dofile` option is useful for running a set of commands that set up your environment and advance to a specific point in the verification session. The following example command substitutes your dofile name for `my_dofile`:

```
UNIX%  -dofile my_dofile
```

In non-GUI mode, you can use the `-dofile` option for running batched sets of commands. The following example command substitutes your dofile name for `my_dofile`:

```
UNIX%  -nogui -dofile my_dofile
```

Saving a Dofile

To save the commands entered during a current session that you can use later as a batch file to repeat the session, use the `SAVE DOFILE` command, or the Save Dofile form in GUI mode (*File – Do Dofile*).

When running a session from a dofile, this command does not save individual commands that might have been included in a separate dofile (that is, it saves the manually entered commands, which might include a `dofile <filename>` command).

Use the Save Dofile form to save commands to a dofile to be used later as a batch file to repeat the Conformal Constraint Designer session.

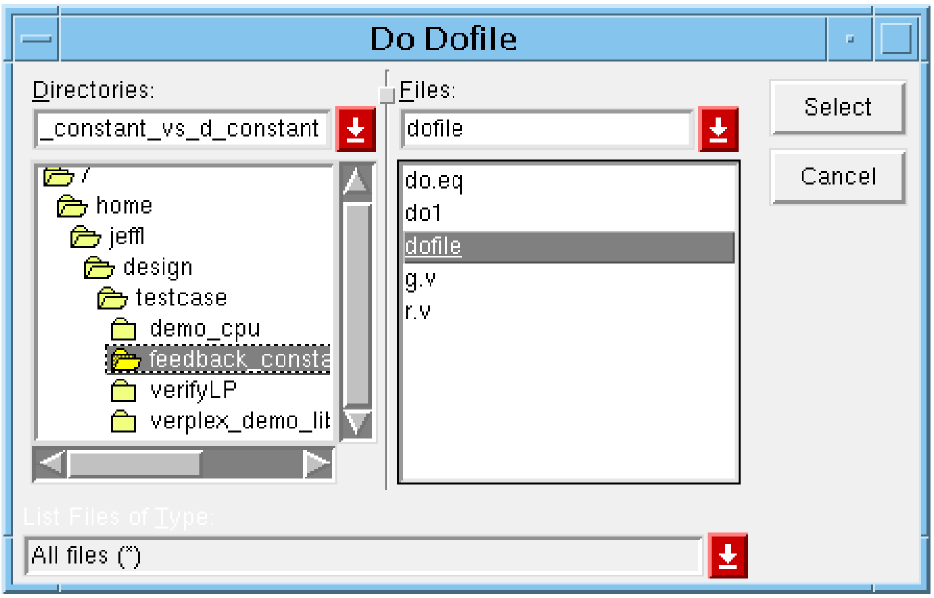
Save Dofile Fields and Options

Filename	Specifies the name of the dofile. You can enter the path of the dofile or click *Browse* and select a location from the Save Dofile browser window.
Open Mode	Overwrites or appends to the dofile. *Replace* overwrites the contents of an existing dofile, and *Append* appends to the contents of an existing dofile.

Executing Commands in a File

At any time during a session, execute commands in a batch mode using the `DOFILE` command or the Do Dofile form in GUI mode (*File – Do Dofile*). By default, the dofile aborts at any command that generates an error message.

Use the Do Dofile form to execute a batch file of commands, or run a set of commands from a previous session.



Do Dofile Fields and Options

Directories	Double-click the file folders to expand the directories and view the dofile names in the *Files* list.
Files	Shows the available files. Use the *List Files of Type* pull-down menu at the bottom of the form to filter file display. You choose *All files*, *Dofiles*, or *Command files*.

Interrupting a Dofile

Within a dofile, use the `BREAK` command to interrupt a dofile and return to the current system mode.

Resuming Running a Dofile

When a dofile executes the `BREAK` command, the Conformal software issues a warning and prompts you to use the `CONTINUE` command to resume running the dofile:

```
//Warning: Break dofile 'my_dofile' at line 32. Use 'continue' command to continue.
```

Specifying Error Handling

Use the `SET DOFILE ABORT` command in a dofile to specify how the Conformal software responds to errors it encounters:

- `set dofile abort on`

  Aborts the dofile and generate a message.

- `set dofile abort off`

  Continues with the dofile and generate a message.

- `set dofile abort exit`

  Exits the session.

Comments in Dofiles

The Conformal software provides two types of comments in a dofile:

1. Two slashes (`//`) comments out the rest of command. `//` must have space before it if you add it to the middle of the text.

   In this example, the following command lines are commented out:

   ```
   //read library ../library/lib_01.lib ../library/lib_02.lib \
       ../library/lib_03.lib ../library/lib_04.lib \
       ../library/lib_05.lib ../library/lib_06.lib \
       -liberty
   ```

   In this example, the read library command is run for `lib01.lib` through `lib_03.lib`, commenting out `lib04.lib` through `lib_06.lib`, and not specifying the `-liberty` option:

   ```
   read library ../library/lib_01.lib ../library/lib_02.lib \
       ../library/lib_03.lib // ../library/lib_04.lib \
       ../library/lib_03.lib
       ../library/lib_05.lib ../library/lib_06.lib \
       -liberty
   ```

2. Three slashes (`///`) comments out the rest of the line. `///` must have a space before it if you add it to the middle of the text.

   In this example, the first line only runs the read library command, commenting out `lib_01.lib` and `lib_02.lib`, and including `lib03.lib` through `lib_06.lib`, and specifying the `-liberty` option:

   ```
   read library ///../library/lib_01.lib ../library/lib_02.lib \
   ```

```
../library/lib_03.lib ../library/lib_04.lib \
../library/lib_05.lib ../library/lib_06.lib \
-liberty
```

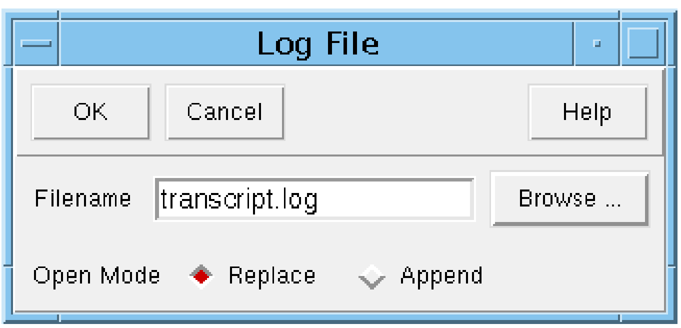
Transcript Messages

In both the GUI and non-GUI modes, you can choose to turn the *transcript output* on or off. This is especially useful for batch processing in the non-GUI mode. With the transcript output turned off, none of the regular transcript output is displayed to the screen. Rather, the Conformal Constraint Designer retains the transcript in a file. To save the transcript output in a log file, see Recording Transcript Log Files on page 58.

To turn the transcript output on or off, use the `SET SCREEN DISPLAY` command.

Creating a Transcript File

To create or append to an existing transcript file, use the Log File form in GUI mode (*Setup – Log File*).



Tip

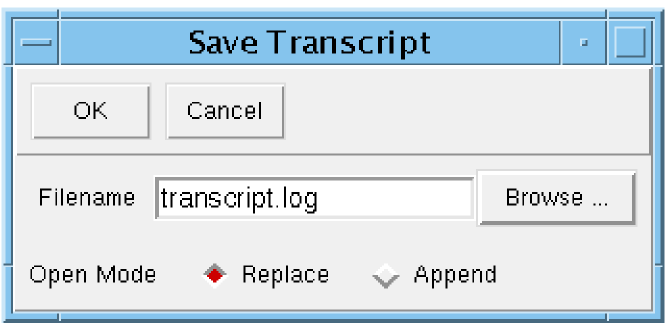
Recording in this file begins after you click *OK*; therefore, you might want to create a log file at the beginning of your session. However, if you begin a session and decide to save the transcript at a later point, see Saving a Transcript File on page 58 to capture a transcript of the beginning of the session.

Log File Form Fields and Options

Filename	Specifies a transcript name. Type a path, or click *Browse* to choose an existing file from the Log File browser window.
Open Mode	*Replace* replaces the existing contents with the new contents. This is the default. *Append* adds the contents to an existing file.

Saving a Transcript File

You can save a transcript to a file at any point during a session, use the Save Transcript form in GUI mode (*File – Save Transcript*). It contains all of the information from the beginning of the session up to the point when you save the file.



Save Transcript Form Fields and Options

Filename	Type the path of the transcript file, or click *Browse* to choose a location from the Save Transcript browser window.
Open Mode	*Replace* (the default) overwrites the contents of an existing file. *Append* appends the contents to an existing file.

Recording Transcript Log Files

You can start or stop a transcript log file at any time during a session using the `SET LOG FILE` command. Furthermore, you can save multiple log files during a session. However, only one log file is active at a time. If you create a new log file without stopping a previous log file, Conformal ends the previous log file and starts recording in the new file.

The `SET LOG FILE` command options allow you to overwrite (replace) or append existing files. *There is no default;* therefore, if you enter an existing filename without specifying the replace or append option, the Conformal software responds with an error message.

Aliases

To reduce typing, you can use an alias (single word) in a session. For example, if you frequently use the `REPORT ENVIRONMENT` command in a session, define an alias for that command with the `ADD ALIAS` command, or use the Alias form in GUI mode (*Setup – Alias*).

In the following example, the `ADD ALIAS` command adds `renv` as the alias for the `REPORT ENVIRONMENT` command:

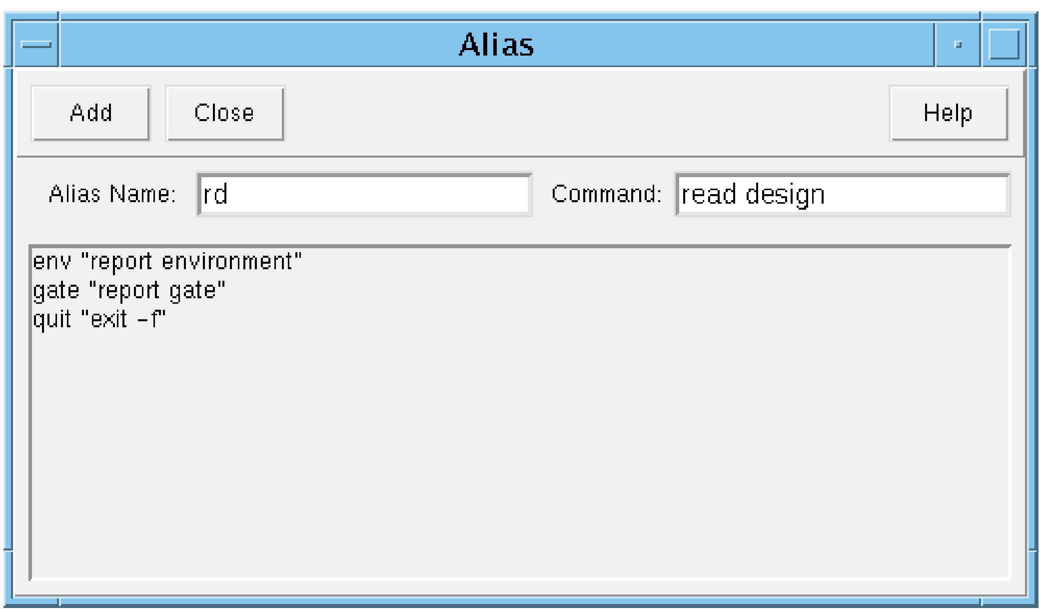
Example command:

```
add alias env report environment
```

If you re-use an existing alias name, the Conformal software accepts (overwrites) the former alias.

Alias Form

You can use the Alias form in GUI mode (*Setup – Alias*) to add, delete, or view alias names.



Important

If you type a command name incorrectly, the Conformal software accepts your entry, but returns an “Unknown command” error message when you attempt to use the alias. In this case, delete or overwrite the faulty alias with the correct command.

Alias Form Fields and Options

Alias Name	Specifies the alias name.
Command	Specifies the name of the command that will be represented by the alias.
Alias List Box	Lists the aliases. To delete an alias, right-click to open the pop-up menu and select *Delete Alias*.

Checkpoint and Restart Facility

The checkpoint and restart facility saves all the data from a session (CHECKPOINT command) as a *checkpoint* such that it can be restarted at a later time (<start_up_command> -restart_checkpoint <*checkpoint_file_name*> [-protect <*password*>]).

Note: The GUI mode will be disabled when you restart the checkpoint process.

Applicable commands	CHECKPOINT INFO CHECKPOINT <start_up_command> -restart_checkpoint <checkpoint_file_name> [-protect <password>]
Data preserved	When you save your session as a checkpoint, the tool preserves the: ■ Hierarchical and flattened databases ■ Environment settings ■ Constraints ■ Verification results ■ User-defined variables ■ User-defined procedures

Supported Platform	Linux

Limitations:

This feature has the following limitations:

- If you are creating a checkpoint file that you plan to restart using a different license server, add the restart license server to the `LM_LICENSE_FILE` variable before invoking Conformal and *before* creating the checkpoint file; otherwise, you will not be able to restart the checkpoint file with the new server. For example:

  ```
  setenv LM_LICENSE_FILE "$LM_LICENSE_FILE":5280@mylic01
  ```

- Do not enter the GUI mode if you plan to create a checkpoint file that you will want to run later in the GUI mode. If a checkpoint file is created after having entered GUI mode, when the checkpoint file is restarted, it will restart and run in non-GUI mode and the GUI mode is disabled. If a checkpoint file is created before entering the GUI mode, the checkpoint file can enter the GUI mode when it is restarted.

- Checkpoint and restarts works on only the following Linux platforms:
  32/64-bit Linux kernel versions 2.6.9-34, 2.6.9-42, 2.6.9-67, 2.6.9-78, 2.6.9-89, 2.6.10, 2.6.14, 2.6.16, 2.6.18, 2.6.25, 2.6.26 and 2.6.27

- You cannot specify the stack limit in a restarted tool process. You can, however, specify the stack limit when you save the checkpoint:

  ```
  CHECKPOINT -stack <multiplier>
  ```

  Default multiplier is 1 (in other words, 64MB).

Command Entry Modes

In Conformal, there are two command entry modes:

- Default Conformal command-entry mode (also called `vpxmode`)

  Enter this mode by invoking the tool using the `ccd` start-up command. While in this mode, you must use the Conformal command syntax (`read design` instead of `read_design`).

  Notes:

  - If you are in `tclmode` (accessed through the `ccd -tclmode` start-up command or through the `tclmode` command), you can switch use the `vpxmode` command to switch to the default Conformal command-entry mode.
  - If you used the `ccd-t` start-up command, you cannot switch to `vpxmode`.

- Tcl command entry mode (also called `tclmode`)

There are several ways to use enter the Tcl command entry mode.

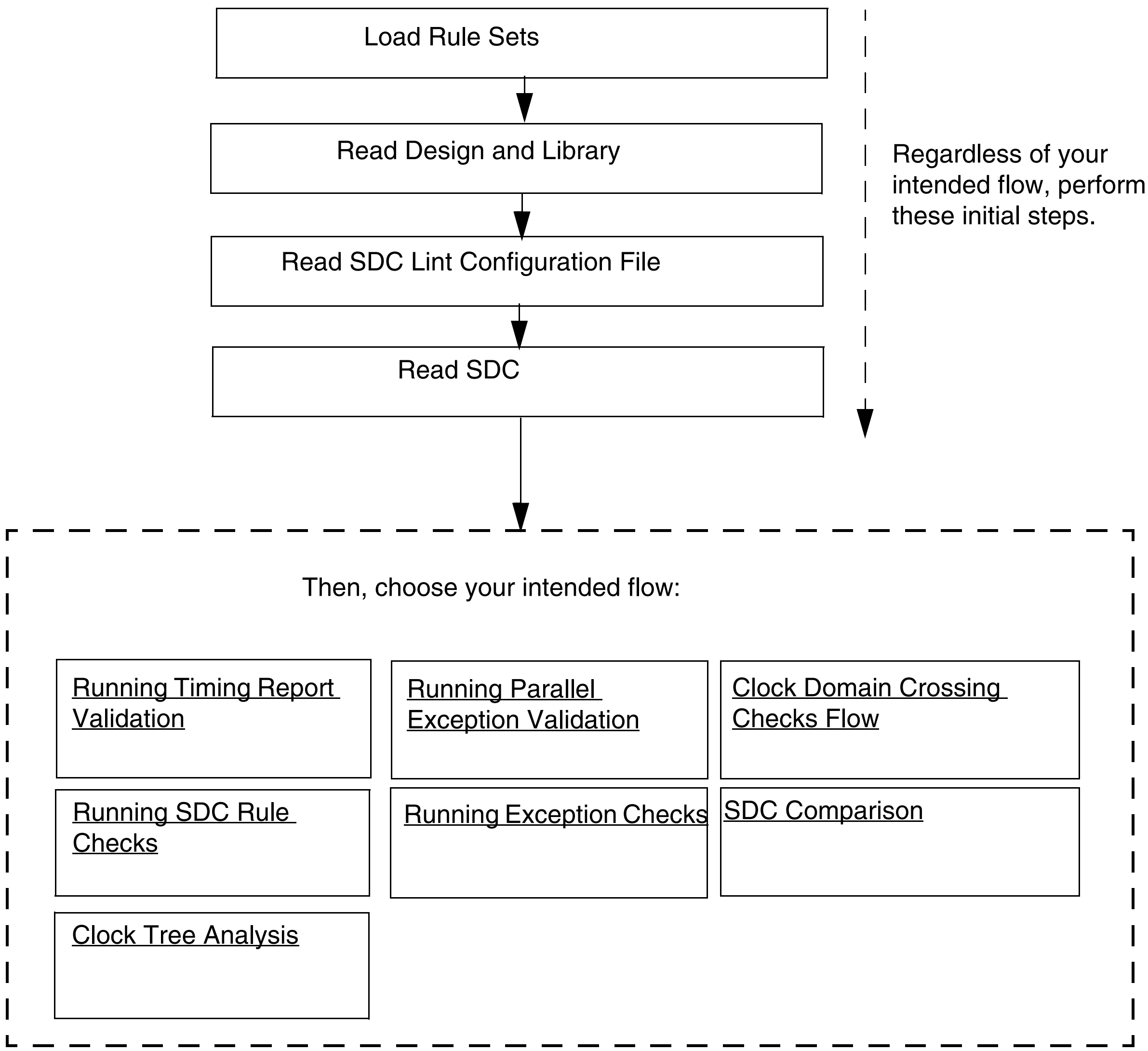
- ❑ Use the `ccd-t` start-up command. While in this mode, you cannot access the `vpxmode`.
- ❑ If you want to switch between `tclmode` and vpxmode, invoke the tool using the ccd `-tclmode` command Tcl command entry mode while in `vpxmode` through the `tclmode` command. You can switch back and forth between vpxmode and tclmode. While in this mode, you must use the Tcl command syntax (`read_design` instead of `read design`).

Note: While in Tcl mode, you must use the Tcl command syntax (`read_design` instead of `read design`).

5

Conformal Constraint Designer General Flow

Conformal Constraint Designer, General Flow Diagram



General Flow Steps

- Load Rule Sets on page 67
- Reading the Library on page 67
- Reading in Design Files on page 70
- Reading in SDC Lint Configuration Files
- Reading SDC on page 74

See also "Additional Tasks" on page 83.

For more information on the commands mentioned in this section, refer to the *Conformal Constraint Designer Command Reference*.

Load Rule Sets

HDL and modeling rules are disabled by default. To enable them, you must load a set of rules with severities before loading the design.

See "Rule Sets" on page 224.

Reading the Library

When design modules are defined in a library (such as Verilog simulation libraries) you must use the `READ LIBRARY` command *before* the `READ DESIGN` command. If there are duplicate modules, the Conformal software uses the first module found and ignores all others. However, you can use the `-lastmod` option to specify that the Conformal software use the last module and ignore earlier ones.

Conformal Constraint Designer requires using technology cells in the Liberty format for proper functionality.

Note: In the Conformal software, Verilog is the default format; therefore the Liberty format must be specified in the Read Library form. If you use Verilog libraries, Conformal Constraint Designer will error out and you will need to override this using `SET RULE HANDLING` command on that error and make it a warning. Be aware that some checks that depend on timing arcs will be incorrect.

You cannot use the `READ LIBRARY` command repeatedly. Therefore, if there are multiple library files, you can list all the library files in the `READ LIBRARY` command explicitly or with wildcards, as shown in the following example command. Use the backslash character (\) at the end of a line to indicate that the command you are entering continues on the next line.

```
read library file1.lib file2.lib file3.lib… -liberty
```

or

```
read library lib/*.lib -liberty
```

You can also use the Read Library form (*File – Read Library*) to specify library filenames you will include with a design.

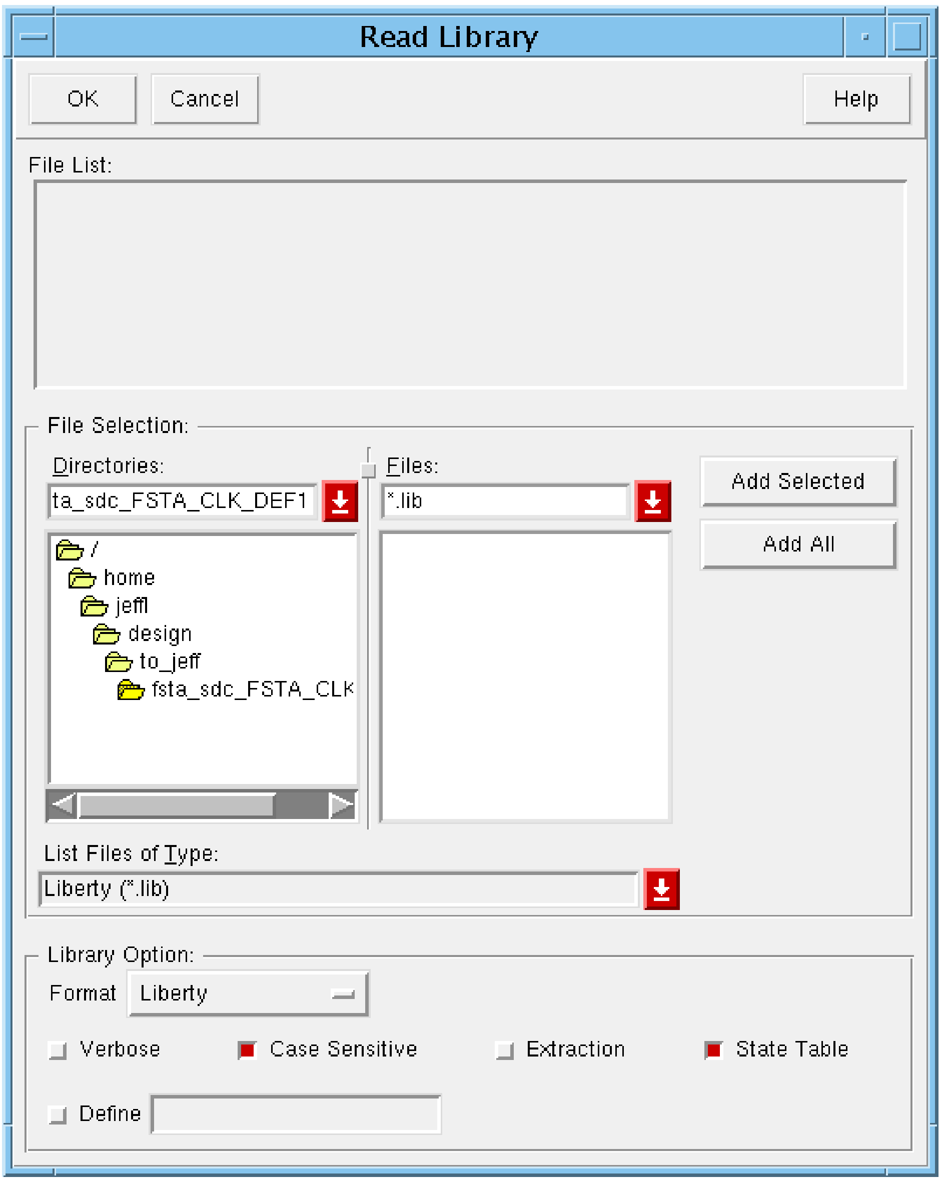


Table 5-1 Read Library Fields and Options

File List	Lists the library files that the Conformal Constraint Designer reads in for this session. As you build the list of files, the Conformal Constraint Designer adds them to this display. You can also delete files from this list by right-clicking and choosing *Delete* from the pop-up menu to delete the selected files. Or, right-click and choose *Delete All* to remove all the files from the File list.
File Selection	Specifies one or more library files. Double-click file folders in the *Directories* display to specify the location of the library files. From the *Files* list box, select the files you want to read and click *Add Selected* to add the selected files, or click *Add All* to add all the files in the *Files* list box
List Files of Type	Filters the file type display.
Format	Specifies the format of the library you intend to read. You can use the pull-down menu to choose *Verilog*, *Verilog2K*, *SystemVerilog*, *VHDL*, or *Liberty.*
Verbose	Displays the verbose messages for parsing and translating each library module.
Case Sensitive	Specifies that the Conformal Constraint Designer should handle the library as case-sensitive. **Note:** This option is not available for VHDL.
Extraction	Specifies that the Conformal Constraint Designer is to abstract transistor models into gate models.
State Table	Specifies that the library contains Synopsys Liberty state tables. Conformal can handle state tables that have single asynchronous inputs and no overlapping rule outputs. This option is only available when selecting *Liberty* from the *Format* pull-down menu.
Define	For Verilog formats, enter your Verilog `` `ifdef `` macro in this field.

Reading in Design Files

Use the READ DESIGN command to read in design files. This is accomplished in the Setup mode. The design formats currently supported are Verilog and VHDL. If you must overwrite the design, use the `-replace` option. If the Conformal software finds multiple modules with the same name, it uses the first module and ignores later modules with that name. However, you can use the `-lastmod` option with Verilog modules to specify that the Conformal software use the last module and ignore the earlier ones.

If your design contains mixed languages, use the `-noelaborate` option as shown in the following example:

```
read design sub.vhdl -vhdl -noelaborate
read design top.v -verilog
```

Cadence recommends the following methods to read multiple design files of the *same* language.

- Explicitly list all of the design files in the `READ DESIGN` command or use wildcards, as shown in the following syntax. Use the backslash character (\) at the end of a line to indicate that the command you are entering continues on the next line.

  ```
  REAd DEsign file1.v file2.v file3.v... \
  -verilog
  ```

  or

  ```
  REAd DEsign src/*.v -verilog
  ```

- Create a file that contains all the necessary design files. For example, a file called `foo.v` might contain the following:

  ```
  `include "file1.v"
  `include "file2.v"
  `include "file3.v"
  …
  ```

  Then, append the name of this newly created filename to the `READ DESIGN` command.

  ```
  REAd DEsign foo.v -verilog
  ```

You can also use the Read Design form (*File – Read Design*) to specify the design filenames.

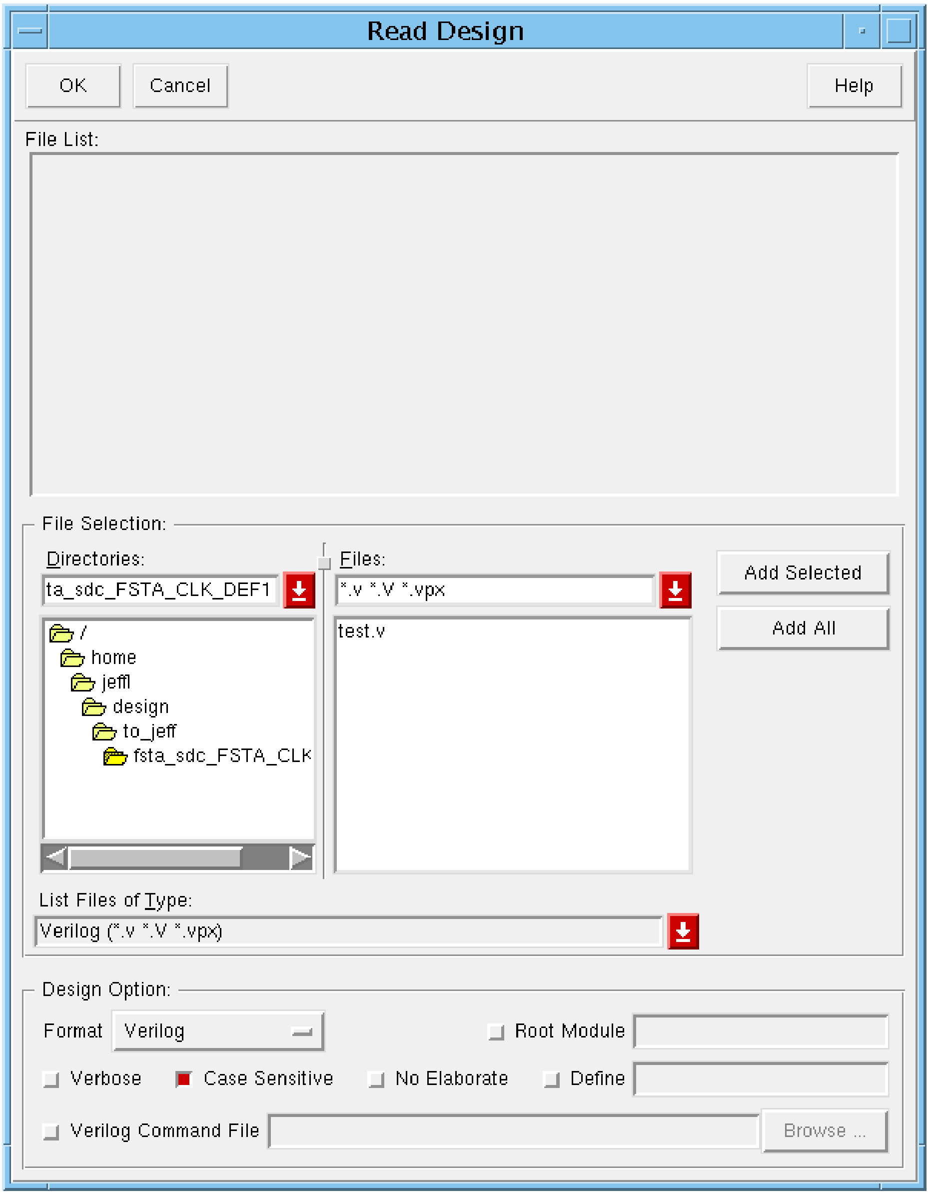


Table 5-2 Read Design Fields and Options

File List	Lists the design files the Conformal software reads in for this session. As you build the list of files, the Conformal software adds them to this display. You can also delete files from this list by right-clicking and choosing *Delete* from the pop-up menu to delete the selected files. Or, right-click and choose *Delete All* to remove all the files from the File list.
File Selection	Specifies one or more design files. Double-click file folders in the *Directories* display to specify the location of the library files. From the *Files* list box, select the files you want to read and click *Add Selected* to add the selected files, or click *Add All* to add all the files in the *Files* list box.
List Files of Type	Filters the file type display.
Format	Specifies the format of the file you intend to read. You can use the pull-down menu to choose a format. When selecting *VHDL*, the bottom portion of the form expands. See Specifying Design Options for VHDL Designs on page 104 for more information.
Root Module	If you intend to designate a root module other than the top module, this specifies the name of the intended top-root module. **Note:** If a single top level module exists, by default, the Conformal Constraint Designer uses it. However, if multiple top-level modules exist, it specifies one. Use this option to override that specification.
Verbose	Displays the verbose messages for parsing and translating each module in the design.
Case Sensitive	Specifies that the Conformal Constraint Designer should handle the design as case-sensitive. **Note:** This option is not available for VHDL.
No Elaborate	Specifies that you intend to read in multiple files of different languages.

Define	Specifies the text macro name.
Verilog Command File	If you are using Verilog command file lists, this specifies the name of the Verilog command file. You can type the name in this field, or click *Browse* to use the Verilog Command File window to select a file.

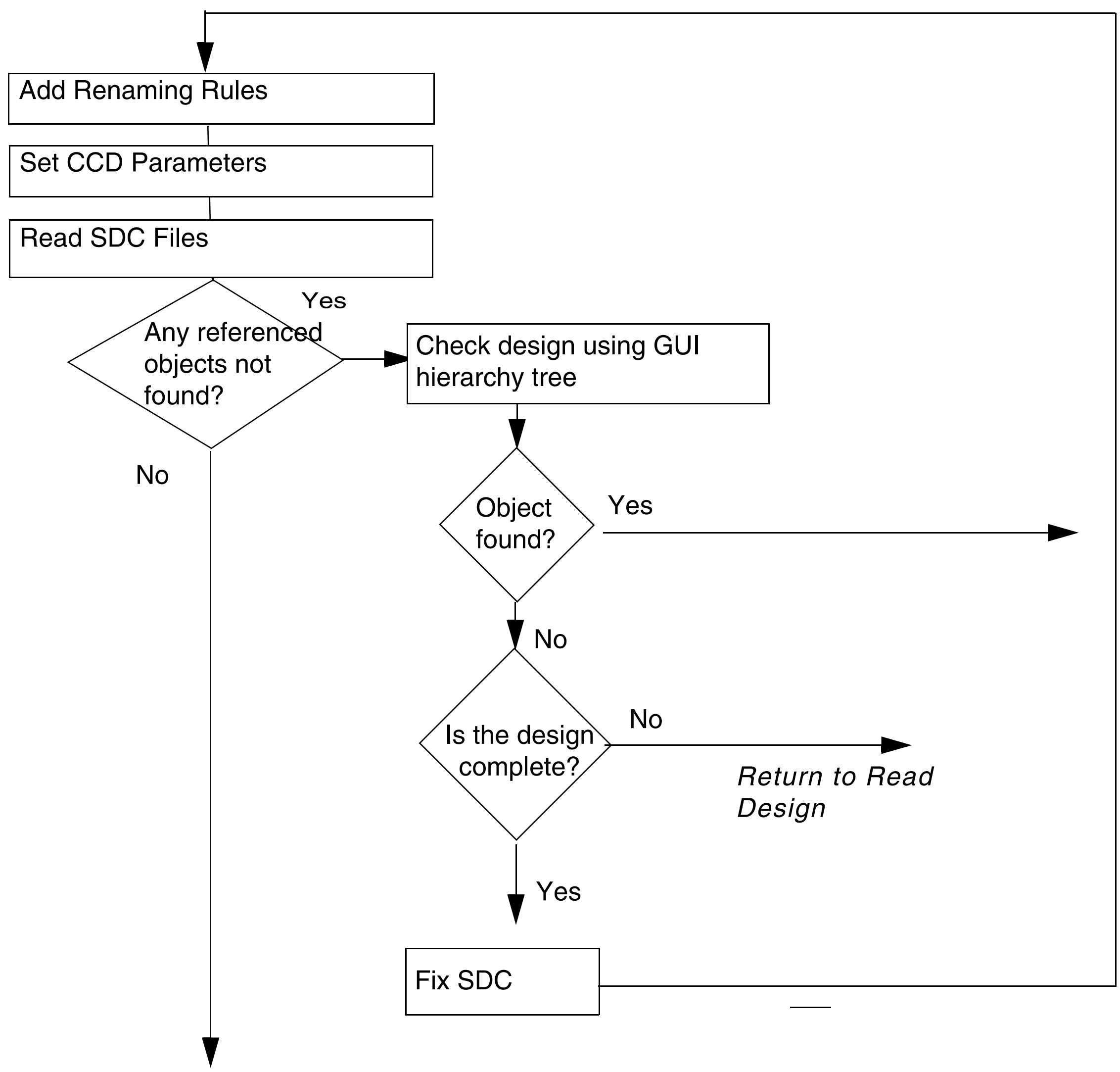
Reading in SDC Lint Configuration Files

SDC lint rule checks are built-in; they cannot be disabled. However, you can configure the SDC lint rule checks to meet your requirements using the `configure_lint_check` command (explained in "Configuring Lint Checks through configure_lint_check Command " on page 151) or through an SDC lint configuration file ("Configuring Checks Using an SDC Lint Configuration File" on page 152).

Note: If your design contains clock groups that are incorrectly defined, important constraint checks can be missed. Therefore, it is important that you check the clock groupings in your design (using the `REPORT CLOCK GROUP` command) before you perform policy checks. Using the results of this command, you can identify new clock groups and/or delete existing clock groups before committing them to your clock group settings (using the `COMMIT CLOCKS` command).

Reading SDC

This section describes the tasks involved in reading SDC.



Specify Renaming Rules when Reading in SDC

When the naming conventions in your design and constraint files are not the same, you can add renaming rules with the `ADD RENAMING RULE` command. For example, you have naming mismatches because your SDC files are at the gate-level and your design is at the RTL level. If there is an object name in the SDC file that the Conformal Constraint Designer cannot find in the design, it applies the renaming rules to that name and then tries to find the object using the resulting name.

In the case where pins inferred in the RTL will be named differently by different tools, use the `ADD RENAMING RULE` command to map the SDC name (e.g., as generated by the synthesis tool) to the Constraint Designer-generated pin names. For example, because the RTL Compiler will use the register pin names `clk` and `d`, whereas the Constraint Designer will use `CK` and `D`, you would run the following command:

```
add renaming rule d2D '/d$' '/D'
add renaming rule clk2CK '/clk$' '/CK'
```

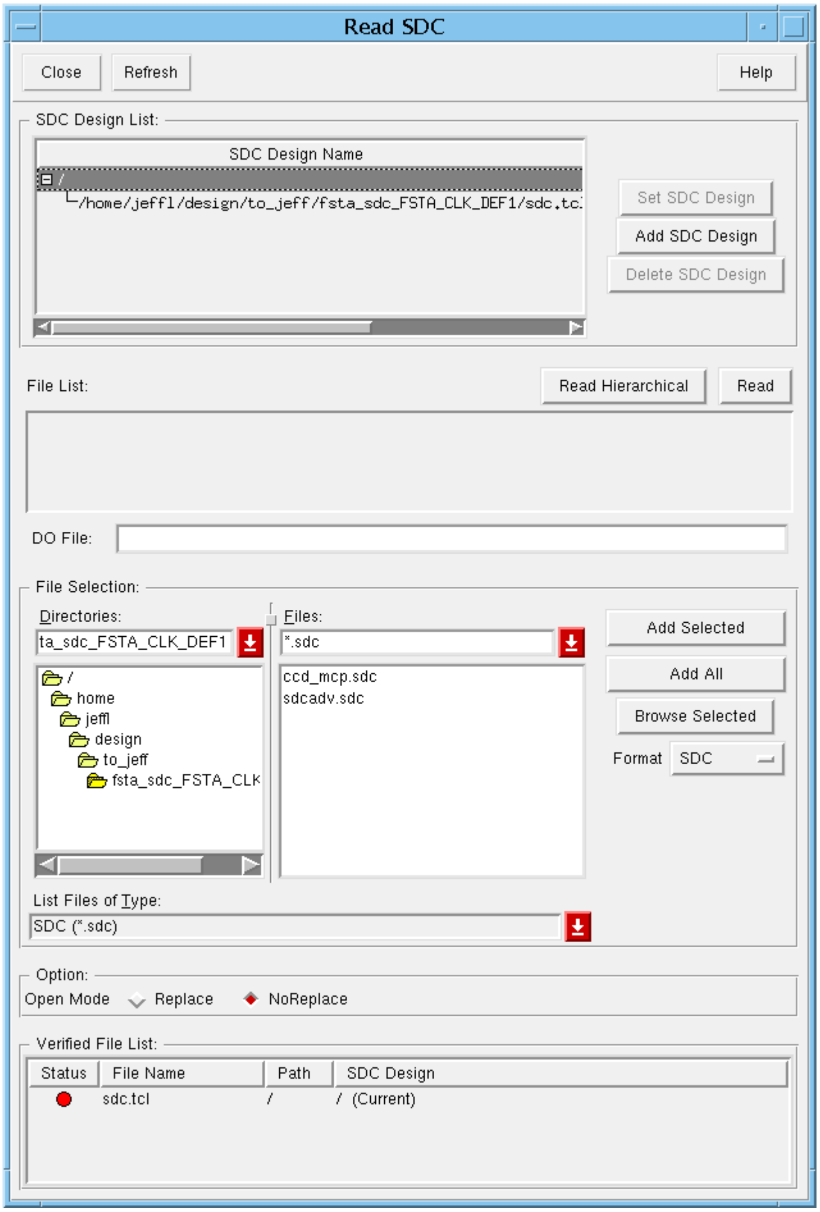
Setting CCD parameters (if necessary)

You can use the `SET CCD PARAMETER` command to set user-defined parameters that will affect later commands and rule checks.

Reading SDC Files

In this step, you read the SDC file that you want to verify. To read in your SDC files, use the `READ SDC` command or the Read SDC form (*File – Read SDC*).

Use the Read SDC form to select the constraint file that you want to apply to your design. You can also use the Read SDC form to open the SDC Source Code Manager for a particular file.



Tip

The SDC parser stops when it encounters commands in the constraint file that it does not recognize. You can use the sdc_add_unhandled Tcl command, before you read in your SDC file, to declare these unhandled commands.

Read SDC Fields and Options

SDC Design	Displays the currently defined SDC designs (top-level and blocks), specifies the current SDC design, and indicates whether you have switched to Verify mode for that design. You can also use this section to set the current SDC design, add SDC design definitions, or delete SDC design definitions
File List	Displays the list of constraint files specified in the *File Selection* section. Click *Read Hierarchical* to read in the SDC files for all specified SDC designs ('`/`' by default) and enter Verify mode in all SDC designs except for '`/`'. Click *Read* to read in the specified SDC files. This reads in files and performs syntax/usage rule validation. The validation results are reported in the *Verified File List* area.
File Selection	Use this section to choose one or more SDC files and an optional a dofile. Click *Add Selected* to add only the highlighted files, or click *Add All* to add all of the displayed files to the *File List* area. Click *Browse Selected* to open the SDC Source Code Manager on the selected file to display its contents. Click *Format* to select SDC or Dofile from the pull-down menu to filter the file list. To remove one or more files from the *File List*, see Deleting SDC Files from the File List on page 81.
List Files of Type	Filters the file display. *SDC* displays only the files with `.sdc` extensions, and *All files* displays all of the files.
Option	Use this section to control the *Open Mode* of your input file. *Replace* clears the results of the previously read SDC files and reads in a new file. *NoReplace* (the default) skips the file if any SDC file has already been processed.

Verified File List	Use this section to view a list of all the verified files and their status. This section also indicates the current SDC design. You can right-click in this section and use the pop-up menu to open a constraint file's Source Code Manager or SDC Rule Manager.

Viewing SDC Source Code

Use either procedure to view the SDC file's source code.

- To view the source code for a file that has not been verified:
  - **a.** Click on an SDC filename from the *Files* list box.
  - **b.** Click *Browse Selected*.
- To view the source code for a verified file:
  - **a.** Click on an SDC filename in the *Verified File List* section to highlight it.
  - **b.** Right-click and choose *Source Code* from the pop-up menu.

These procedures open the Source window in context-dependent mode.

Viewing SDC Rules

Use this procedure to view the SDC file's rule violations.

1. Click on an SDC filename in the *Verified File List* section to highlight it.
2. Right-click and choose *SDC Rule Manager* from the pop-up menu.

   The SDC Rule Manager appears. See "Managing Lint Checks Using the SDC Lint Manager" on page 154 for more information.

SDC Design Definitions

You can add/delete SDC design definitions to design instances.

Adding an SDC Design Definition

Use the `ADD SDC DESIGN` command to add SDC design definitions to design instances, or the following procedure using the Read SDC window:

Note: Make sure the current SDC design is “/” (top level).

1. Click the *Add SDC Design* button to open the Add SDC Design window.
2. Select a block.
3. Click *Add*.

   The Conformal software adds an SDC design definition for the selected block and updates the *SDC Design List* section of the Read SDC window.
4. Repeat steps 2-3 for all blocks.
5. Click *Close*.

Deleting an SDC Design Definition

Use the `DELETE SDC DESIGN` command to delete an SDC design definition, or the following procedure using the Read SDC window:

1. Select an object from the *SDC Design List.*

   Note: You cannot delete SDC design definitions for the current SDC design or the top-level design (“/”).
2. Click the *Delete SDC Design* button.

   The Conformal software deletes the selected SDC design definition, along with all SDC-related information, for the selected object.

Selecting SDC Files in the Hierarchical Flow

There are two procedures to read in SDC files while in the hierarchical flow: the automated procedure, and the manual procedure. After using either procedure, you can enter Verify mode (`set system mode verify`) and check the rules.

To run rule checks for full-chip, as well as hierarchical checks, run the command:

```
run rule check
```

To run hierarchical checks only, run the command:

```
run rule check *HIER*
```

Automated Procedure

The automated procedure uses the `READ HIERARCHICAL SDC` command, which in turn executes the necessary commands to read all the SDC files and get into Verify mode for each SDC design (block or top-level).

When doing this, it is not possible to stop inside any given SDC design to examine rule messages. Because this automated procedure performs all the steps in the correct order, Cadence does not recommend this if you only need to read all the SDC files and proceed straight to the hierarchical checks. You can also have a block-specific dofile executed before reading the SDC files for any SDC design. This can be useful to set any block-specific CCD parameters.

The automated procedure steps are:

1. Read the library and the design.

   Open the Read SDC window.

2. Specify all the blocks

   Click *Add SDC Design* and use the Add SDC Design window to add SDC design definitions for design instances.

3. For each SDC design, click on its line in the *SDC Design List* and proceed to select its SDC files and an optional dofile using the *File Selection* area, and clicking on *Add Selected* to feed the selected file(s) into the *File List* for the currently selected SDC design.

4. Press the Read Hierarchical button.

Manual Procedure

The manual procedure allows you to execute this flow step-by-step. Because this procedure is more complex, Cadence recommends this only if you need to examine and diagnose rule messages issued at the block level before checking the hierarchical rules. These are the two constraints to be satisfied by a manual procedure:

- The SDC files at the top-level SDC designs `'/'` (full-chip) and, optionally, `'||'` (top-level only, or glue-logic) must be read before going into Verify mode in the blocks.
- You must read all the block SDC files before you go to Verify mode at the top-level SDC design(s).

Manual procedure steps:

1. Read the library and the design.

Open the Read SDC window.

2. Add all SDC designs using the `ADD SDC DESIGN` command.

3. Read the SDC for SDC design '`/`'.

   See Reading SDC on page 74 for information on how to select SDC files.

4. If a top-level module (glue logic) SDC file is to be checked vs. the full-chip SDC file, set the current SDC design to '`||`' using the `set sdc design ||` command, and read the SDC file(s) for this SDC design. Return to SDC design '`/`' using the `set sdc design -force` command.

5. Set the current SDC design to a block.

Choose one of the following methods:

- From the Hierarchical Browser, right-click on a block and use the *Set SDC Design – Current* pop-up command.

- Use the `SET SDC DESIGN` command.

- From the Read SDC window, click on *Add SDC Design* and use the Add SDC Design window to add SDC design definitions for design instances.

1. Read in the SDC files for that block.

2. Enter Verify mode.

   ```
   set system mode verify
   ```

3. Analyze any reported messages.

4. Enter Setup mode.

   ```
   set system mode setup
   ```

5. Repeat steps 5 through 9 for all blocks.

6. If you ran step 4, set the current SDC design to '||', enter Verify mode, check any rule messages, and return to Setup mode.

7. Set the current SDC design to '/'.

Deleting SDC Files from the File List

Before you click *OK* to read in the specified SDC files, you can remove one or more filenames from the list with the following procedure:

1. Highlight one or more SDC files in the *File List*, by doing one of the following:

- To highlight a single SDC file, click a filename.
- To highlight multiple SDC files, do one of the following:
  - Click and drag, highlighting each file as you drag the mouse.
  - Hold down the `Shift` key and click on two files; this selects every file on the list between the two.
  - Hold down the `Control` key and click on files that you want to select. With the `Control` key depressed, you can jump around the file list.

2. Delete all or specified SDC files from the *File List* by doing one of the following:

- To delete the highlighted files, right-click and choose *Delete* from the pop-up menu.
- To delete all files, right-click and choose *Delete All*.

Checking the Design

If Conformal Constraint Designer reports that it cannot find referenced objects in your design, you need to check your design:

1. Check your design for objects with different naming conventions. If they exist, return to Specify Renaming Rules when Reading in SDC on page 74 and define renaming rules. Otherwise, proceed to the next step.
2. Check if your design is incomplete. If your design is incomplete, return to Reading in Design Files on page 70. Otherwise, proceed to Fixing the SDC on page 82.

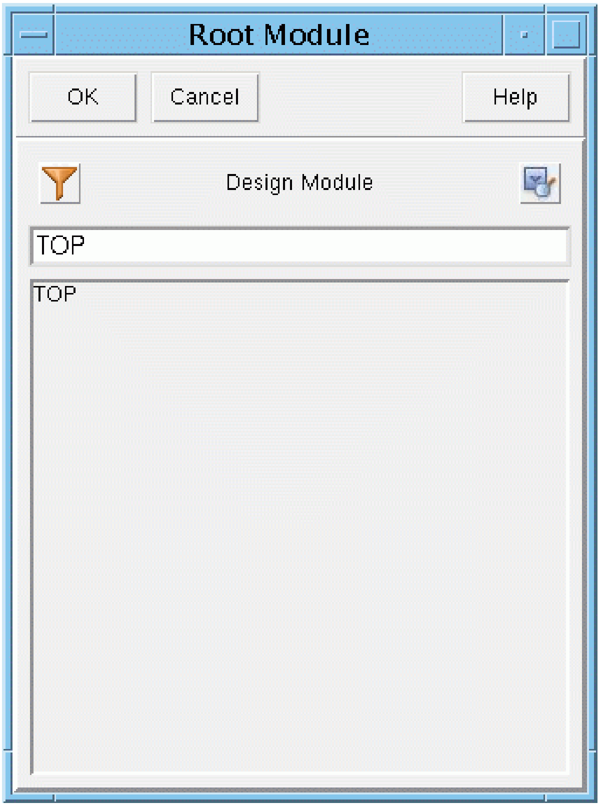
Fixing the SDC

If the Conformal Constraint Designer reports that it cannot find referenced objects in your design, but your naming conventions are correct and your design is complete, then you should fix your SDC file and restart the Reading SDC flow.

Additional Tasks

Changing the Root Module

Use the SET ROOT MODULE command, or the Root Module form (*Setup – Root Module*), to change the Conformal automatic root module assignment and to specify the name of the new root module in the design.



The current root module appears in the *Design Module* field. Below this field is a list of all the modules in the design that can be specified as a root module.

To specify a new root module, double-click a module name so that it appears in the *Design Module* field and click *OK*.

Tip

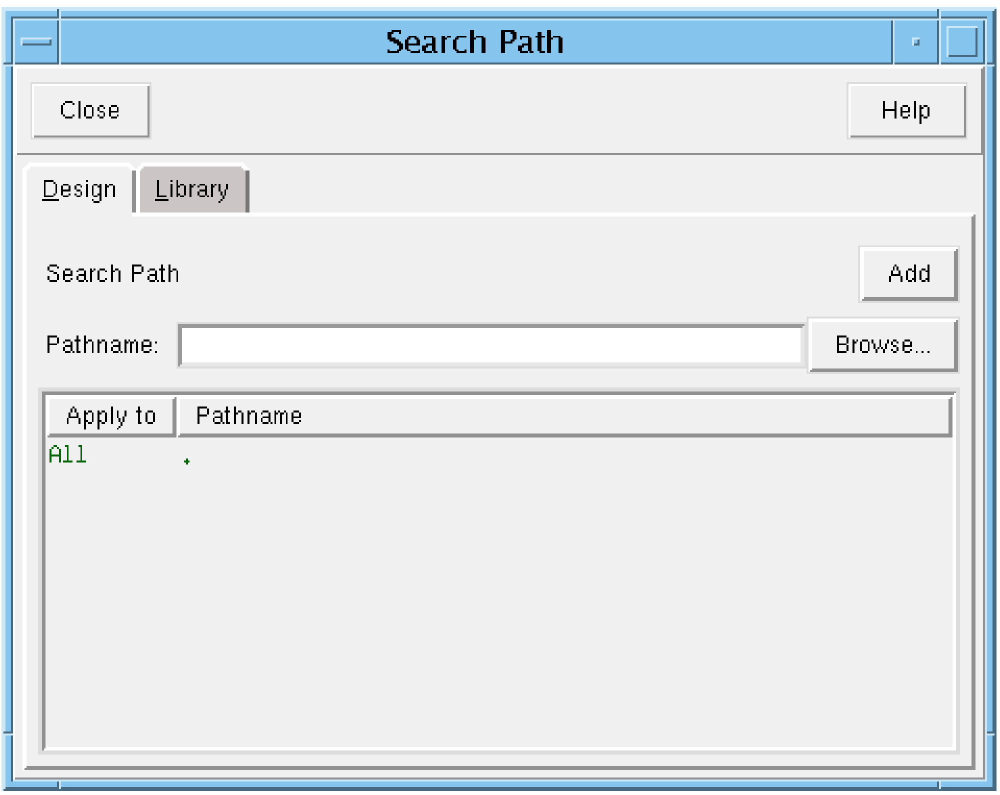
To sort the list alphabetically, right-click in the column display area and choose *Sort* from the pop-up menu.

Adding a Search Path

Use the `ADD SEARCH PATH` command, or the Search Path form (*Setup – Search Path*) to create, modify, or delete directory search paths. The Conformal Constraint Designer uses the search path to locate design and library files kept in directories other than the current working directory.

Note: If you do not add search paths, the Conformal Constraint Designer searches for filenames in the current directory.

To add a design search path, click the *Design* tab. To add a library search path, click the *Library* tab.



Search Path Form Fields and Options

Pathname	Specifies the search path. You can type the directory path or click *Browse* to open the Select A Directory window and locate the path you want to add.

Add	Adds the directory path to the list in the *Pathname* list box.
Pathname list box	Lists the directory search paths. To delete directory search paths, right-click on a path to bring up the pull-down menu and select either a *Delete Search Path* or *Delete All Search Paths*.

Interpreting Exit Status Codes

On exiting, the Conformal Constraint Designer returns a status code. A non-zero status code indicates a potential error (for example, failed properties and modeling rule violations). Conversely, a zero status code indicates that all proved properties passed, there were no counter-example warnings or modeling rule errors, and all commands executed successfully. To see the exit code, use the `-verbose` option with the `EXIT` command:

```
exit -force -verbose
```

Checking Clock Groups

Before you check the quality of your SDC and design files, ensure that your clock groups are correct. Your clock groups affect the rule checks that the Conformal Constraint Designer flags.

Use the `REPORT CLOCK GROUP` command to display the existing clock groups. Is the clock groups are incorrect, use the `ADD CLOCK GROUP` or `DELETE CLOCK GROUP` command to define your clock groups. For example:

```
delete clock group *
add clock group -name group1 -clock clk1 clk2
add clock group -name group2 -clock gclk3 clk4
```

If the clock groups are correct, use the `COMMIT CLOCK` command to commit clock groups. The SDC rule checks, exception validation, and generation will not be performed unless the clock groups have been commited.

Tip

The following lists SDC and CCD commands that can affect your clock groupings.

SDC constraints:

- `create_clock`: Each `create_clock` command implies a new clock group for the created clock.

- `create_generated_clocks`: The defined clock will be grouped with master clocks.
- `set_clock_groups`: This is a 1.7 SDC command; it defines the clock group settings, overriding other clock group settings (any `ADD/DELETE CLOCK GROUP` commands are disabled and have no effect).
- `set_false_path -from <clocks> -to <clocks>`: See `add clock groups -use_clk2clkfp` below.

CCD commands:

- `add clock group -single`

  Creates a single clock group for all clocks in the design.
- `add clock group -use_clk2clkfp`

  Works in conjunction with all of your clock-to-clock false path exceptions to determine your clock groups
- `delete clock group`

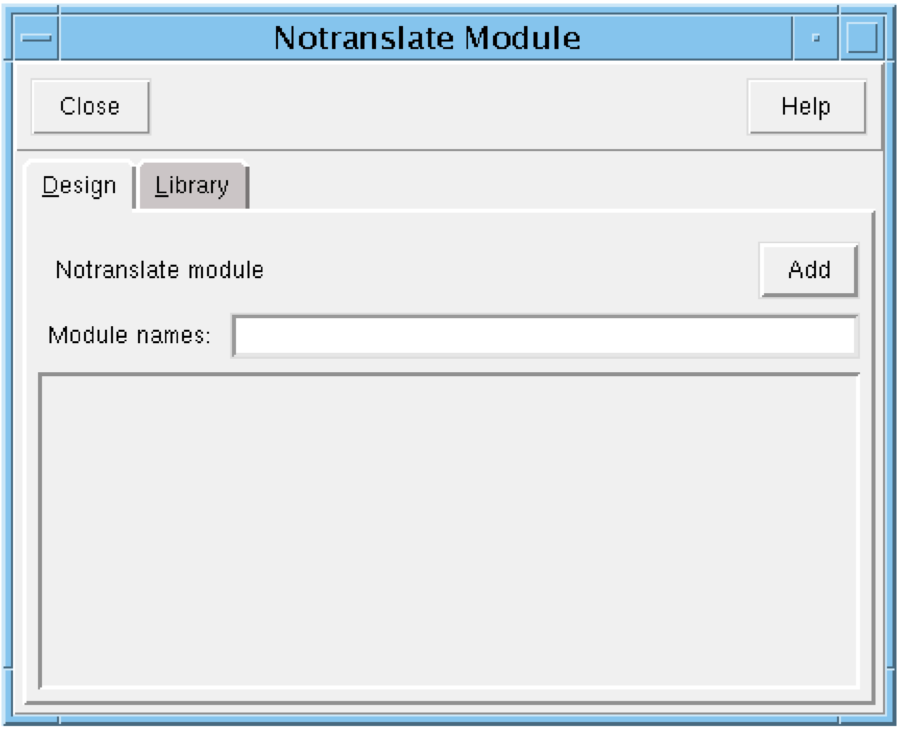
  Deletes clock groups; use this after `REPORT CLOCK GROUP`.

Adding Notranslate Modules

When you choose not to compile specific library or design modules, you must run the `ADD NOTRANSLATE MODULES` command. The specified modules (for example, non-synthesizable and memory modules) automatically become blackboxes.

The `ADD NOTRANSLATE MODULES` command is applied during initial parsing, so name matching applies only to original module names. For parameterized or VHDL generic modules whose names are determined and applied by Conformal after parsing and preprocessing, you must use the `ADD BLACK BOX` command.

Alternatively, you can use the Notranslate Module form (*Setup – Notranslate Module*) before reading designs or libraries to add and delete design or library modules that will not be translated.



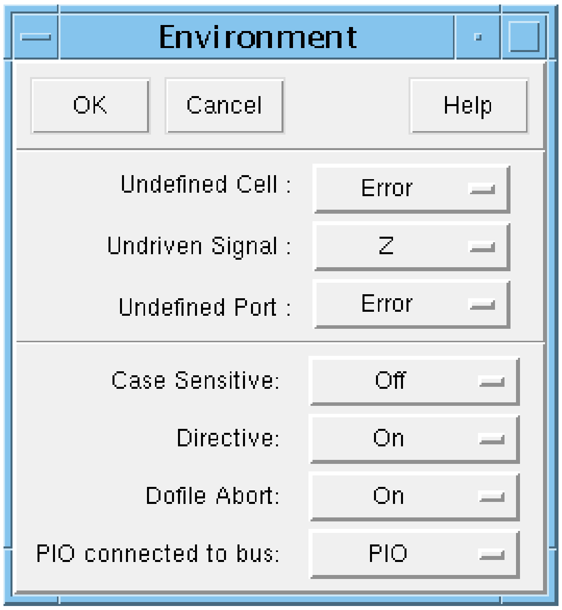
To delete one or all notranslate modules from designs and libraries, click a module name in the list box in the *Design* or *Library* page, and right click to open the pop-up menu and choose *Delete Notranslate Module* to delete a single notranslate module, and *Delete All Notranslate Module* to delete all notranslate modules.

Notranslate Module Form Fields and Options

Add	Add the notranslate modules, and adds the notranslate module names to the list box.
Module name	Specifies the name of the module that will not be translated.

Setting Global Options

Use the Environment form (*Setup – Environment*) to set global options for the design.



Environment Form Fields and Options

Undefined Cell	Specifies handling for any undefined cell the Conformal software encounters when reading designs and libraries. Click the pull-down menu to choose either *Error* or *Black Box*. Based on your selection, Conformal automatically reports undefined cells as errors, or it blackboxes them.
Undriven Signal	Specifies handling for any undriven signal the Conformal software might encounter when reading designs and libraries. Click the pull-down menu to choose *0*, *1*, *X*, or *Z*.
Undefined Port	Specifies handling for any undefined port the Conformal software might encounter when reading designs and libraries. Click the pull-down menu to choose either *Error* or *Ignore*.
Case Sensitive	*On* specifies that names you use are case-sensitive.
Directive	*On* enables the effects of Synopsys and Verplex synthesis directives when reading in a Verilog or VHDL file.

Dofile Abort — Specifies how the Conformal software responds when executing a dofile that generates an error message. Choose one of the following:

- *On*—The dofile terminates when an error message occurs.
- *Off*—The dofile continues even if an error message occurs.
- *Exit*—Conformal exits the session and returns to the system prompt if an error message occurs.

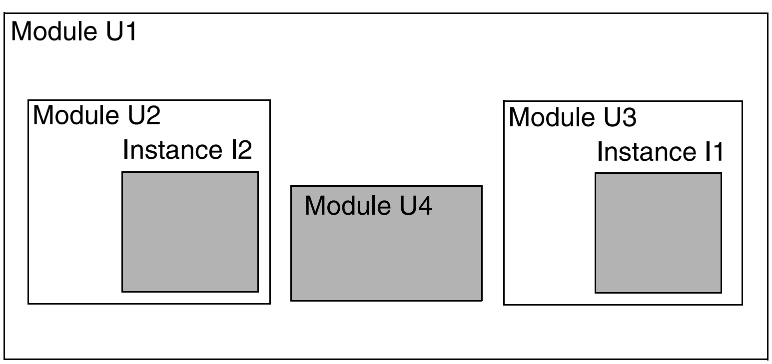
PIO connected to bus — Specifies how the Conformal Constraint Designer handles PIOs that are connected to buses. Choose one of the following:

- *PO*—The Conformal Constraint Designer treats them as primary outputs.
- *PIO*—The Conformal Constraint Designer treats them as primary inputs/outputs.

Handling Blackboxes

If there are modules or instances that you want to blackbox, use the `ADD BLACK BOX` command or the *Add Black Box* pop-up menu option from the Hierarchy Browser.

Note: Conformal Constraint Designer does not support wildcards with the `-instance` option.




```
SETUP> add black box /U1/U4 -module
SETUP> add black box /U1/U2/I2 /U1/U3/I1
```

You can add or delete instances or modules as black boxes in the Hierarchical Browser window display. The black box icon appears or disappears, accordingly.

Adding A Black Box Module

When you use the following procedure, a blackbox symbol appears adjacent to the module name and all the instances of that module.

1. In the Hierarchical Browser window, click an instance or module to select it.
2. Right-click and choose *Add Black Box* from the pop-up menu.

Deleting a Black Box Module

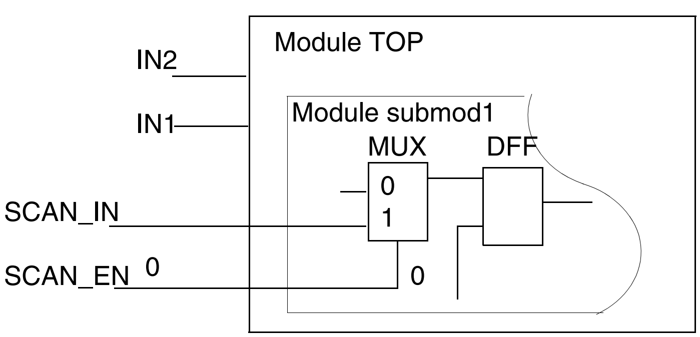
Use the DELETE BLACK BOX command, or use the following procedure in the Hierarchical Browser window:

1. Click a black box instance or module to select it.
2. Right-click to open the pop-up menu and choose *Delete Black Box*.

   The blackbox symbol disappears from the position next to all the instances of the applicable module.

Handling Pin Constraints

To add constraints to primary inputs, such as Logic-0 or Logic-1, use the ADD PIN CONSTRAINTS command.



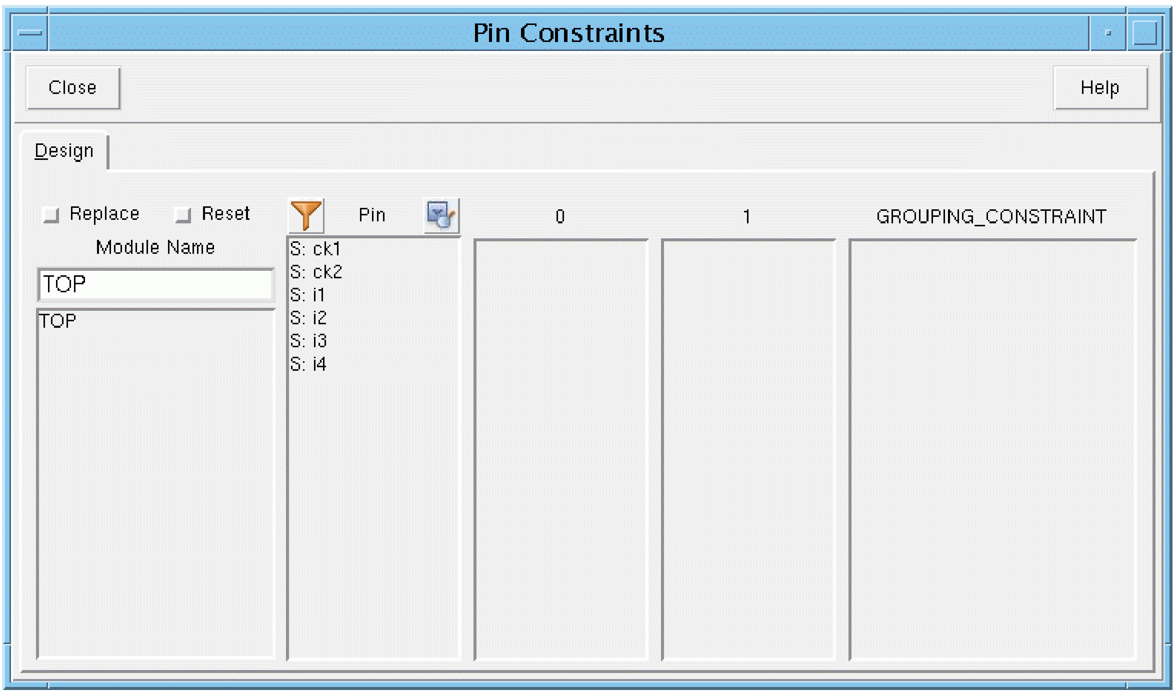

```
SETUP> add pin constraints 0 SCAN_EN
```

Note: The `ADD PIN CONSTRAINTS` command adds propagated constants to the design that possibly affect the results of the SDC rule checks. You should use `set_case_analysis`

SDC commands instead. The use of this command should therefore be restricted to Exception Validation and Generation, and if this is done, the results of the SDC rule checks must be considered as potentially inaccurate.

To delete pin constraints to primary input pins, use the DELETE PIN CONSTRAINTS command.

Alternatively, you can use the Pin Constraints form (*Setup – Pin Constraints*) to add and delete primary input pin constraints.



The Pin Constraints window includes five columns:

- *Module Name*—Lists the modules.
- *Pin*—Lists the primary inputs. Each primary input is either a system class primary input (*S:* name) or a user-defined class primary input (*U:* Name).
- *0*—Lists the primary inputs constrained to 0.
- *1*—Lists the primary inputs constrained to 1.
- *GROUPING_CONSTRAINT*—Lists *One Hot*, *One Cold*, *Zero One Hot*, and *Zero One Cold* pin groups with an identifying heading.

Tip

To sort the list alphabetically, right-click in the column you want to sort and choose *Sort* from the pop-up menu.

Selecting Primary Inputs

Use any of the following procedures to select primary inputs:

- Click a primary input to select it.
- Click and drag the cursor over a group of adjacent primary inputs to select them.
- Click the first primary input in a group, depress and hold the Shift key, and click the final primary input in a group to select the entire group.
- Depress the `Ctrl`-key and click a primary input to add it to the selected group.

Adding Pin Constraints to Primary Inputs

1. In the *Pin* column, select the primary input, or inputs, to which you want to add constraints.
2. Right-click and choose a constraint you want to add to the primary input.

Note: Adding pin constraints adds propagated constants to the design that possibly affect the results of the SDC rule checks. You should use `set_case_analysis` SDC commands instead. The use of this command should therefore be restricted to Exception Validation and Generation, and if this is done, the results of the SDC rule checks must be considered as potentially inaccurate.

Deleting Pin Constraints

To delete a single pin constraint:

1. Click a primary input in one of the columns to select it.
2. Right-click and choose *Delete Pin Constraint* from the pop-up menu.

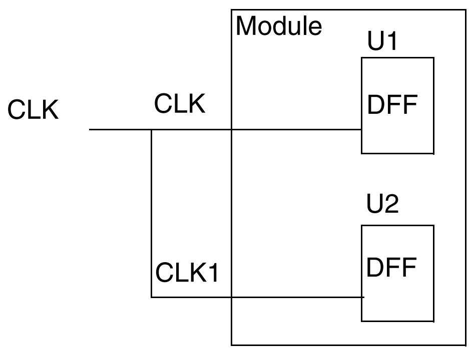
   The Conformal Constraint Designer removes the pin constraint. In the case of *GROUPING_CONSTRAINT*, the Conformal Constraint Designer deletes the entire group.

To delete all pin constraints from all columns:

1. Right-click in one of the columns.
2. Choose *Delete All Pin Constraints* from the pop-up menu.

Handling Pin Equivalences

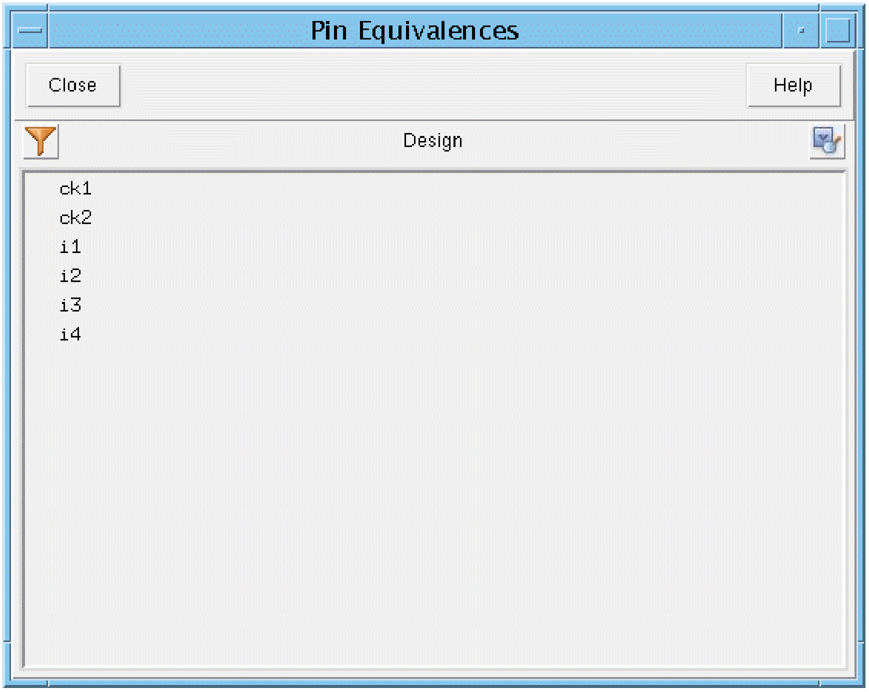
To create equivalences or inverted equivalences among primary inputs, use the `ADD PIN EQUIVALENCES` command.




```
SETUP> add pin equivalence CLK CLK1
```

To delete the added pin equivalences from the specified primary input pin that were placed on primary input pins, use the `DELETE PIN EQUIVALENCES` command.

Alternatively, you can use the Pin Equivalences form (*Setup – Pin Equivalences*) to add and delete pin equivalences



Tip

To sort the list alphabetically, right-click in the column you want to sort and choose *Sort* from the pop-up menu.

Adding a Pin Equivalence

1. Click a primary input to select it.
2. Right-click and choose *Set Target* from the pop-up menu.

   The font color of the selected primary input changes to red. This signifies the current target primary input.
3. Click to select the primary input that is equivalent to the target primary input.
4. Right-click to open the pop-up menu and choose *Add Pin Equivalence* or *Add Invert Pin Equivalence*.

   The Conformal Constraint Designer displays the added pin equivalences below the target primary input with a connecting line. The Conformal Constraint Designer denotes inverted pin equivalences with (-) following the primary input name.

Deleting Pin Equivalences

To delete a single pin equivalence:

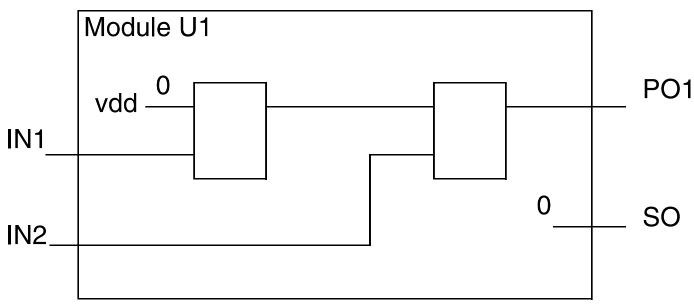
1. Click an equivalent primary input to select it.
2. Right-click and choose *Delete Pin Equivalence* from the pop-up menu.

To delete all pin equivalences:

1. Right-click in the primary input display area to open the pop-up menu.
2. Choose *Delete All Pin Equivalences*.

Handling Tied Signals

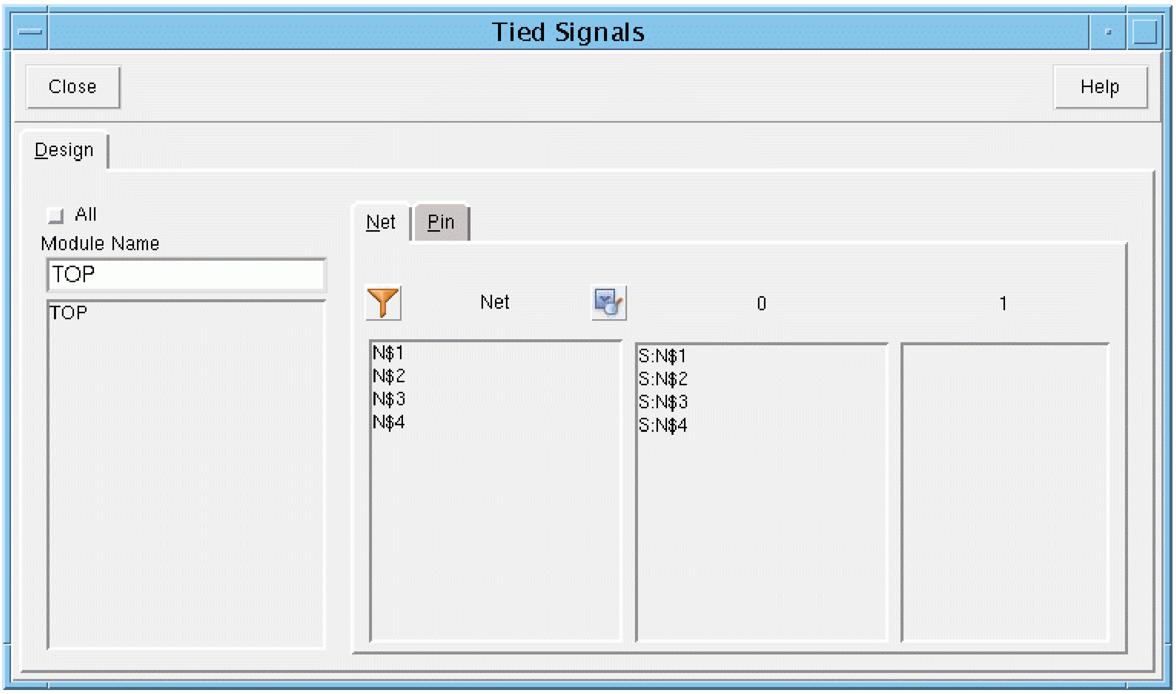
To tie any floating nets or pins to Logic-0 or Logic-1, use the ADD TIED SIGNALS command.




```
SETUP> add tied signals 0 vdd -pin -module U1
SETUP> add tied signals 0 SO -net -module U1
```

To delete specified tied signals, use the DELETE TIED SIGNALS command.

Alternatively, you can use the Tied Signals form (*Setup – Tied Signals*) to add and delete tied signals to floating nets and pins.



There are four columns in the design section of the window. Their headings are:

- *Module Name*—Lists the modules in the design.
- *Net* or *Pin*—Lists the floating nets and pins.
- *0*—Lists nets or pins with tied signals to 0.
- *1*—Lists nets or pins with tied signals to 1

Each tied signal belongs to one of two classes:

- System class tied signal (*S*: name) or
- User-defined class tied signal (*U*: name).

Tip

To sort the list alphabetically, right-click in the column display area and choose *Sort* from the pop-up menu.

Adding a Tied Signal to a Net or Pin

1. Double-click a module name to select it.

   The name appears in the *Module Name* field.

2. Click on the *Net* or *Pin* tab to show the list of either floating nets or pins:
3. In the *Net* or *Pin* column, click a net or pin to select it.
4. If the floating net or pin should be tied in all the modules, click the *All* check box.
5. Right-click to open the pop-up menu and choose *Add Tied Signal 0* or *Add Tied Signal 1*.

   The selected net or pin appears in either the *0* or *1* column, accordingly.

Deleting a Tied Signal from a Net or Pin

To delete a single tied signal from a net or pin:

1. Double-click a module name to select it.

   The name appears in the *Module Name* field.

2. Click the *Net* or *Pin* tab to show the appropriate list of tied signals.
3. Click a net or pin name under the *0* or *1* column to select it.
4. Right-click and choose *Delete Tied Signal* from the pop-up menu.

To delete all net or pin tied signals:

1. Right-click in the *0* or *1* column.
2. Choose one of the following from the pop-up menu:
   - ❑ *Delete All – User*—Deletes all tied signals for user-defined classes.
   - ❑ *Delete All – System*—Deletes all tied signals for system-defined classes.

Placing Design Constraints

After the Conformal Constraint Designer successfully reads the library and design files, you can place constraints on the design.

Important

Perform these steps *before* you read in your SDC file.

Constraints are used to:

- Exclude sections of a design from verification
- Disable unwanted functionality, such as test
- Specify initial conditions or states
- Specify special signal relationships
- Specify special behaviors, such as clocks

The Conformal Constraint Designer verifies automatic checks. A missing constraint often appears as one or more false properties. Generally, these are easily debugged since the Conformal Constraint Designer shows a counter-example that has behavior you do not expect. By adding the proper constraints, you avoid these false negatives.

Handling Assertion Constraints

Use the Assertion Constraint window to turn assertion library instances into proof assumptions (constraints).

➤ Choose *Setup – Assertion Constraints*.

Adding Assertion Constraints

Use the following procedure to define one or all assertion instances as constraints.

1. Click an *Instance Name* with *No* indicated in the *Is a constraint?* column.

   This selects (highlights) the instance name.

2. Right-click to open the pop-up menu, and choose *Add Assertion Constraint* or *Add All Assertion Constraints*.

   Yes appears in the *Is a constraint?* column to show that the Conformal Constraint Designer will treat the instance as a constraint.

3. Click *Close*.

Deleting Assertion Constraints

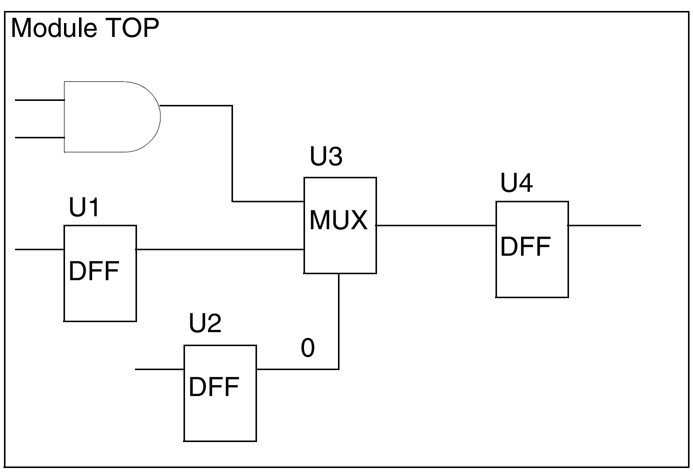
Use the following procedure to delete one or all assertion constraint instances.

1. Click an *Instance Name* with *Yes* indicated in the *Is a constraint?* column.
2. Right-click to open the pop-up menu, and choose *Delete Assertion Constraint* or *Delete All Assertion Constraints*.

   No replaces *Yes* in the *Is a constraint?* column to show that the Conformal Constraint Designer will *not* treat the instance as a constraint.
3. Click *Close*.

Handling Instance Constraints

To constrain any internal DFF or D-Latch output to Logic-0 or Logic-1, use the ADD INSTANCE CONSTRAINTS command. This command places constraints on a specified instance in the design by placing a state value on its output. This command takes a value of 0 or 1 only on the output of the instances.




```
SETUP> add instance constraints 0 /TOP/U2
```

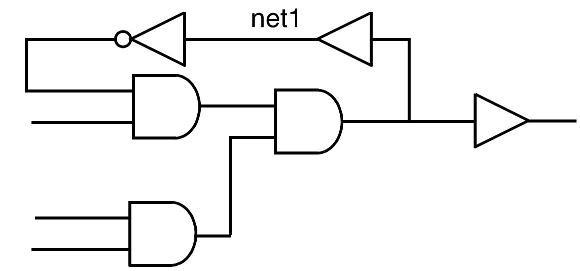
Note: The ADD INSTANCE CONSTRAINTS command adds propagated constants to the design that possibly affect the results of the SDC rule checks. You should use set_case_analysis SDC commands instead. The use of this command should therefore be restricted to Exception Validation and Generation, and if this is done, the results of the SDC rule checks must be considered as potentially inaccurate.

Use the DELETE INSTANCE CONSTRAINTS command to delete instance constraints that were added. Use the REPORT INSTANCE CONSTRAINTS command to display a list of all added instance constraints.

Specifying Primary Inputs

To specify primary inputs, use the ADD PRIMARY INPUT command. Use this command when you want to cut a large logic cone to help ease validation, or constrain a particular net. For example, for the following circuit:

Module u1

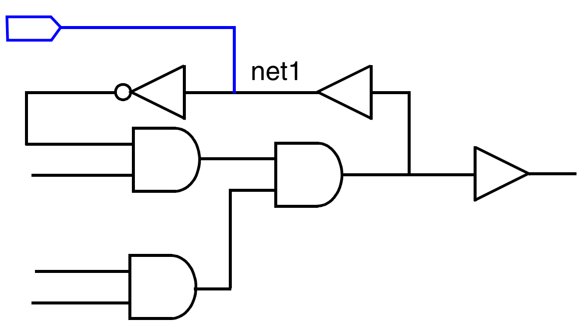


If you use the command:

```
add primary input /u1/net1
```

The Conformal software will interpret the circuit as follows:

Module u1



Specifying VHDL Library Mapping

You can specify how VHDL libraries are mapped using the `READ DESIGN` command's `-map`, `-mapfile`, or `-library` options.

The `-map` and `-library` options work the same in that they map logical library names to physical directories. You can use multiple `-map` commands to map multiple physical directories to one logical library. Use the `-mapfile` option for more specific library mapping, such as specifying that a list of files must be compiled into a specified library. If you read in a file without specifying its library mapping, that file is stored in a default library called `work`.

Note: You can map a file into more than one library. In this case, the file is stored in each library for which it is mapped.

Performing Library Mapping in VHDL

This section demonstrates how to use the `READ DESIGN` command to perform library mapping in VHDL.

For example, your current directory contains the following files:

Physical File/Directory	Contents
`top.vhd`	See Example 5-1.
`lib1/pkg1.vhd`	Package `package1`
`lib1/pkg1_body.vhd`	Package body of `package1`
`lib2/pkg2.vhd`	Package `package2`
`lib2/pkg2_body.vhd`	Package body of `package2`

Table 5-3 Desired Library Mapping

Logical Library Name	Physical File/Directory
`LIB1`	`lib1`
`LIB2`	`lib2`
`work`	`top.vhd` (implicit)

Example 5-1 Contents of top.vhd

```
-------- top.vhd begin --------
library LIB1;
use LIB1.package1.all;

library LIB2;
use LIB2.package2.all;

entity top ...;
```

```
architecture rtl of top ...;
-------- top.vhd end --------
```

To achieve the Desired Library Mapping outlined in Table 5-3, the `READ DESIGN` command should look like one of the following:

- `read design -vhdl top.vhd -map LIB1 lib1 -map LIB2 lib2`
- `read design -vhdl top.vhd -library LIB1 lib1 -library LIB2 lib2`
- ```
  read design -vhdl top.vhd \
       -mapfile LIB1 lib1/pkg1.vhd lib1/pkg1_body.vhd \
       -mapfile LIB2 lib2/pkg2.vhd lib2/pkg2_body.vhd
  ```

Note: The tool terminates the <file_list> for `-mapfile` when it encounters the next option or the end of the `READ DESIGN` command. For example, the following command does not generate the desired library mapping for this example. The tool terminates the file list at `top.vhd`; because of this, `top.vhd` is added to the `LIB2` library—not the `work` directory.

```
read design -vhdl \
     -mapfile LIB1 lib1/pkg1.vhd lib1/pkg1_body.vhd \
     -mapfile LIB2 lib2/pkg2.vhd lib2/pkg2_body.vhd \
      top.vhd
```

In the following example, `top.vhd` is correctly added to the `work` library because the `LIB2` file list terminates at `lib2/pkg2_body.vhd`.

```
read design -vhdl \
     -mapfile LIB1 lib1/pkg1.vhd lib1/pkg1_body.vhd \
     -mapfile LIB2 lib2/pkg2.vhd lib2/pkg2_body.vhd \
     top.vhd
```

Handling Unspecified Library Mappings in VHDL

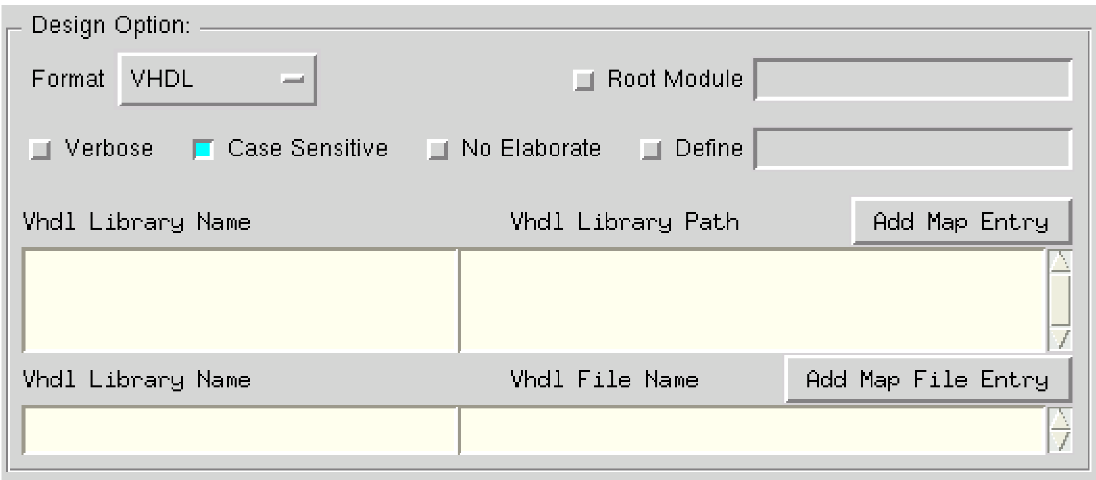
The tool handles `library.declaration` references as follows:

- If the library is defined and the declaration exists, the tool returns the declaration. Otherwise, the tool searches for the declaration in the `work` directory. If the tool finds the declaration, it returns the declaration.
- If the library is undefined, because of unspecified library mappings, the tool searches through the `work` library. If the tool finds the declaration in the `work` library, it returns the declaration with a note; otherwise, the tool returns an error message.
- If the tool finds a `work.declaration` reference while parsing a file that is stored in a logical library (for example, `lib1`), the tool searches through `lib1`, and then through the default `work` library for the declaration. Once the tool finds the declaration, it returns the
```

declaration. The tool notifies you when it returns a declaration from the default `work` library.

Specifying Design Options for VHDL Designs

If the design format is VHDL, the bottom portion of the form expands.



Add Map Entry	Opens the Add Vhdl Library Mapping window where you can select the library name and path of the specific VHDL libraries.
Add Map File Entry	Opens the Add Vhdl Library Mapfile window where you can specify exactly which files belong to a given library.
VHDL Library Name	Displays the VHDL library name. To delete, replace, or insert another VHDL library name, right-click in the display and choose *Delete*, *Insert*, or *Replace* from the pop-up window.
VHDL Library Path	Displays the VHDL path. To delete, replace, or insert another VHDL path, right-click in the display and choose *Delete*, *Insert*, or *Replace* from the pop-up window.

VHDL File Name	Displays the VHDL filename. To delete, replace, or insert another VHDL filename, right-click in the display and choose *Delete*, *Insert*, or *Replace* from the pop-up window.

Specifying Conformal Parameters

Use the SET CCD PARAMETER command to set user-defined parameters that will affect later commands and rule checks. Use this command before you use the READ SDC command. However, the SDC_CLOCK_GATING_CELLS parameter only requires that it is set before you enter Verify mode.

Note: For a list of user-defined parameters and setting information, see the SET CCD PARAMETER command documentation.

Using SDC_HIER_CLOCK_EQUIV

The SET CCD PARAMETER command's SDC_HIER_CLOCK_EQUIV parameter is used during the hierarchical flow to clarify any ambiguity that occurs when matching block clocks to their equivalent top-level clocks. The Conformal software tries to match block clocks to top-level clocks using the clock name, or information from the design and the SDC files. However, when the Conformal software cannot match clocks automatically, use the SDC_HIER_CLOCK_EQUIV parameter to define clock equivalences.

Tip

You can use the `report clock -hier` command to view a table of all the clock equivalences resulting from the current value assigned to SDC_HIER_CLOCK_EQUIV.

You can define this mapping using either explicit clock equivalences or renaming rules. For example, the following commands have the same effect. The first command uses explicit clock equivalences, and the second example uses a renaming rule:

```
set ccd parameter SDC_HIER_CL "CLK1 CLK1_virtual_*, CLK2 CLK2_virtual_*, CLK3 \
   CLK3_virtual_*, CLK4 CLK4_virtual_*"
```

```
set ccd par SDC_HIER_CLOCK_EQUIV "RENAME  (.+)_virtual_.*   @1"
```

Note: These renaming rules are independent from the rules defined using the ADD/DELETE/TEST RENAMING RULE commands, but they use the same syntax.

In the second example:

- Each block clock is temporarily renamed by removing the suffix that matches `_virtual_.*`
- In the replacement string, `@1` is evaluated to the part of the original name that matches whatever is between the parentheses in the pattern string.
- The asterisk is used as a regular expression operator, not as a wildcard, which is why a period is required before it, to match "any string."

Based on this, the Conformal software tries to find a top-level clock with the resulting name.

You can also define explicit equivalences and renaming rules within the same command:

```
set ccd parameter SDC_HIER_CLOCK_EQUIV "tclk1 bclk1, RENAME pattern1 replace1, \
   tclk2 b1clk2 b2clk2, RENAME pattern2 repl2"
```

Important

The order of the renaming rules is important, as they are applied sequentially, but it is not important if they appear before or after explicit equivalences.

The Conformal Constraint Designer uses the following order of precedence for matching a block-level clock with a top-level clock:

1. Apply clock equivalences defined explicitly by using the `SET CLOCK EQUIVALENCE` command.
2. Apply clock equivalences defined explicitly in `SDC_HIER_CLOCK_EQUIV`.
3. Match a block clock with the top-level clock that has the same name.
4. Match a block clock with the top-level clock that is defined on the same pin.
5. Match a block clock with the top-level clock whose clock tree propagates to the block clock's source pin if these two clocks have the same period and waveform.
6. Apply any renaming rules, defined in `SDC_HIER_CLOCK_EQUIV`, and retry step 3.
7. Attempt to find a match based on a comparison between the timing path start points related to the different clocks.

Note: When specifying the top-level equivalent clock of a block's clock explicitly, it is possible that a different block will have a clock of the same name with a different top-level equivalent. In this case, you can prefix the names with their respective block name, followed by two underscores. For example, to specify that `CLK` in block `b1` is equivalent to top-level `CLK1`, and `CLK` in block `b2` is equivalent to top-level `CLK2`, use `CLK1 b1__CLK`, `CLK2 b2__CLK`.

Specifying Renaming Rules

This section describes the procedures you can use to specify renaming rules. Renaming rules temporarily translate names so that the Conformal Constraint Designer can automatically map SDC names to design object names when they are not exactly the same.

You can add renaming rule commands to your dofile before you read in your SDC files, or you can create a renaming rule file to define your renaming rules. A renaming rule file contains ADD RENAMING RULE and DELETE RENAMING RULE commands specific to an SDC file. This is especially helpful when you use a single READ SDC command to read in multiple SDC files for which you want to apply particular renaming rules.

The following are examples on how to use a renaming file:

- Example 1: To assign a specific set of renaming rules to a particular SDC file:

  a. Create a renaming rule file. For example, create `rename2.tcl`.

  b. Add your renaming rule commands to the renaming rule file. For example:

  ```
  delete renaming rule r1
  #Deletes any existing renaming rule called r1
  add renaming rule r1 {"_reg_%d"} {"_reg[@1]"}
  #Adds renaming rule r1
  delete renaming rule r2
  #Deletes any existing renaming rule called r2
  add renaming rule r2 {"/N01"} {"/Q"}
  #Adds renaming rule r2
  ```

  c. When you issue the `READ SDC` command, read in the renaming rule file before you read in its corresponding SDC file. For example:.

  ```
  read sdc s1.sdc rename2.tcl s2.sdc
  ```

- Example 2: You have an existing renaming rule dofile and you want to use it in the Conformal Constraint Designer for SDC files.

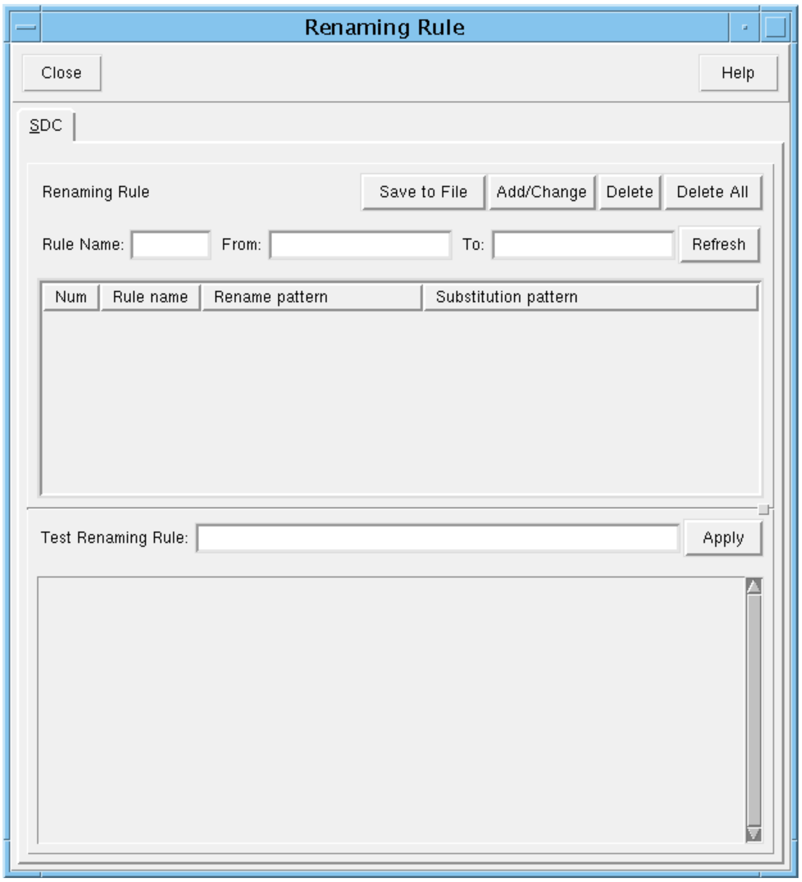
  a. Create a separate file that calls the existing renaming rule file and save it in your working directory. For example, create a file called `renaming.tcl` that contains only the following line:

  ```
  dofile existingrn.do
  ```

  b. Read in the renaming file before you read in the SDC files that require it.

  ```
  read sdc renaming.tcl s1.sdc
  ```

Use the Renaming Rule form (*Setup – Renaming Rule*) to add, delete, and test renaming rules. Adding or deleting renaming rules guides the SDC parser to rename the object name in the SDC file. You can test renaming rules for mapping performance based on name mapping.



Adding a Renaming Rule

Use the `ADD RENAMING RULE` command or use the following procedure in the Renaming Rule form to add renaming rules:

1. Type a unique rule name in the *Rule Name* field.
2. Type the renaming pattern in the *From* field.
3. Type the substitution pattern in the *To* field.
4. Click *Add/Change*.

   Conformal adds the renaming rule to the list.

Deleting Renaming Rules

You can use the DELETE RENAMING RULE command or, in the Renaming Rule form, click a renaming rule and do one of the following:

- To delete the selected rule, click the *Delete* button.
- To delete all rules, click the *Delete all* button.

Tip

You can use the pop-up menu to remove a single rule: After you click a rule, right-click and choose *Delete Renaming Rule*.

Writing Renaming Rules to a File

After you add renaming rules, use the following procedure to write them to a file.

1. Click the *Save to File* button.

   The Save Renaming Rules window appears.

2. Double-click the file folders in the *Directories* display.

   The folders expand and sub-directories appear.

3. Click a file in the *Files* display or type a name in the *Files* field.
4. Click *Select*.

Displaying Results of Renaming Rules

After you add renaming rules, you can use the TEST RENAMING RULE command or, in the Renaming Rule form, do the following to check their effectiveness.

1. Type the renaming pattern in the *Test Renaming Rule* field.
2. Click *Apply*.

   Conformal Constraint Designer displays a summary of the mapping results based on the added renaming rules.

Testing Renaming Rules

You can use the TEST RENAMING RULE command to test your renaming rules, indicating the object name for which you would like apply the renaming rules. For example:

1. Create two renaming rules using the following commands:

```
add renaming rule r1 "_reg_%d" "_reg[@1]"
add renaming rule r2 "/N01" "/Q"
```

2. Test these rules against the "`fsm_state_reg_2/N01`" object name using the following command:

   ```
   test renaming rule fsm_state_reg_2/N01
   ```

   The Conformal software displays messages similar to the following:

   ```
   Renaming rule: r1. Str: fsm_state_reg_2/N01
   Result: fsm_state_reg[2]/N01
   Renaming rule: r2. Str: fsm_state_reg[2]/N01
   Result: fsm_state_reg[2]/Q
   ```

Specifying Sets of Naming Rules

You can specify a set of naming rules of each read design or read library session. For example, if you ran the following command for VHDL as rule 1:

```
set naming rule "%L.%s" "%L[%d].%s" "%s" -variable
read design -vhdl  <all the vhdl design> -noelab
```

Then ran the following command for Verilog as rule 2:

```
set naming rule "%L.%s" "%L[%d].%s" "%s" -variable
read design -verilog <all the verilog design> -noelab
```

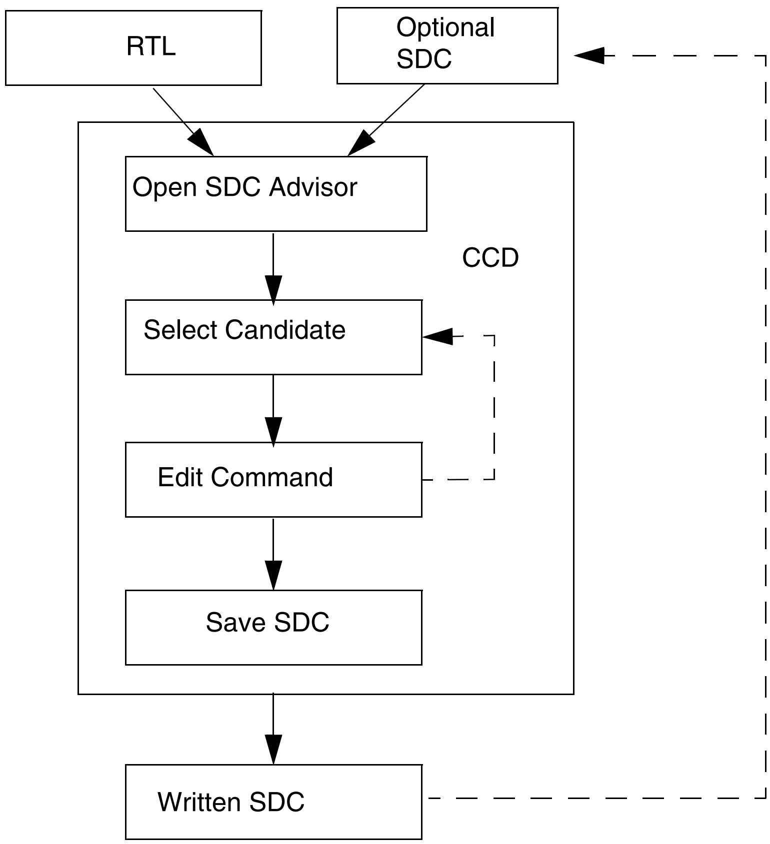
When running the commands, rule 1 can apply to the VHDL designs and rule 2 can apply to the Verilog designs.

6

SDC Command Generation

SDC Command Generation Flow

The following shows the typical flow for generating SDC commands:



Single Pass Flow

In a single pass flow, the Conformal software reads in a design and come up with the set of candidate source objects for you to define the constraints. For example, for clocks, you can choose the relevant clock source objects and open the SDC Command Editor to provide period, waveform, and other options.

Incremental Flow

In an incremental flow there could be a partial SDC (for example, user-defined clocks from a previous iteration). The Conformal Constraint Designer software uses the existing constraints and searches for any unconstrained objects to bring up the remaining candidates to generate new commands.

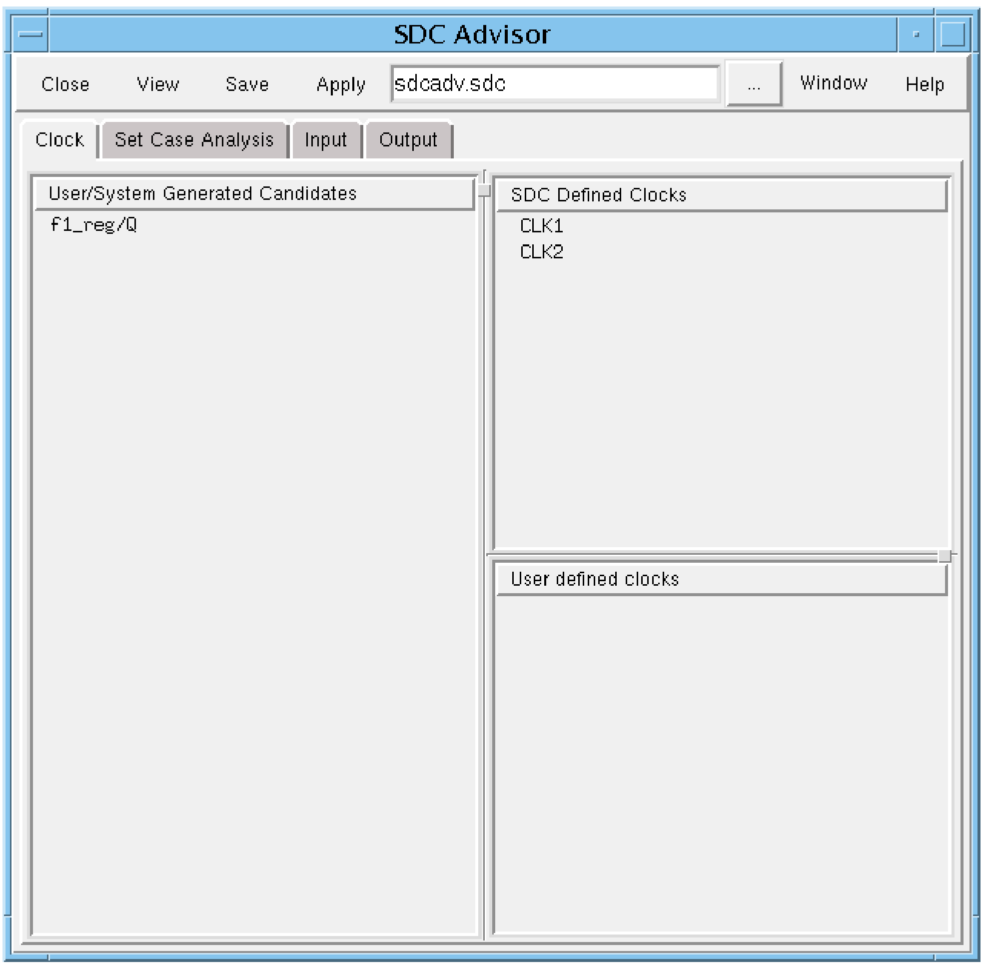
The advantage of an incremental flow is that the number of candidates can be greatly reduced by applying a new constraint. The disadvantage of the incremental flow is the effort it takes to write out the new constraints, read the augmented SDC in Setup mode, and then return to Verify mode.

SDC Advisor

Use the SDC Advisor to generate SDC commands based on the provided candidate objects. There are two ways to open the SDC Advisor from the Main window (in Verify mode):

Note: To enable this feature, you must have an XL license.

➤ Choose *Tools – SDC Advisor*.



For information on the typical flow for generating SDC commands, see SDC Command Generation Flow on page 112.

There are four tabs the display the following pages of the SDC Advisor:

- *Clock*

  See SDC Advisor Clock Page on page 116.

- *Set Case Analysis*

  See SDC Advisor Set Case Analysis Page on page 117.

- *Input*

  See SDC Advisor Input Page on page 118.

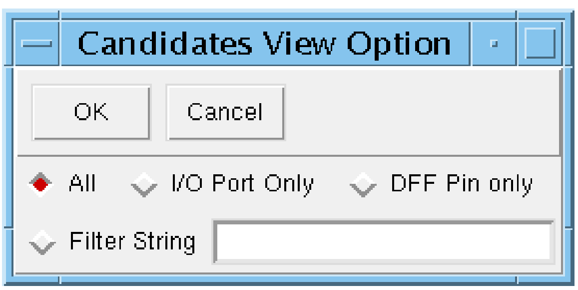
- *Output*

  See SDC Advisor Output Page on page 119.

Viewing Candidates List

Use the Candidates View Option form to filter the list of candidate pins in the *User/System Generated Candidates* display.

➤ In the SDC Advisor, click *View*.



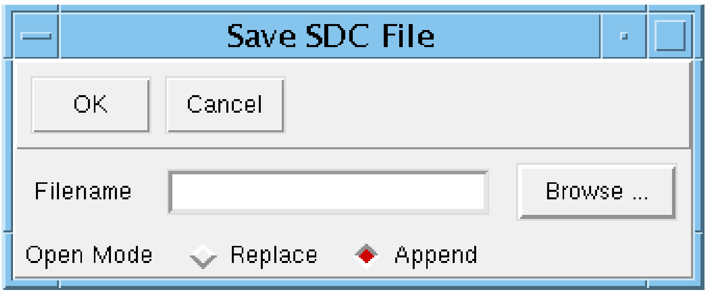
This form includes the following options:

All	Displays all candidates.
I/O Port Only	Displays only the I/O port candidates.
DFF Pin only	Displays only the DFF pin candidates.
Filter String	Filters the candidate display by name. Wildcards are accepted.

Saving SDC Files

Use the Save SDC File form to save the user-defined constraints to a file.

➤ In the SDC Advisor, click *Save*.



This form includes the following options:

Filename	Specifies the name of the SDC file. You can enter the path of the file or click *Browse* and select a location from the Save SDC File browser window.
Open Mode	Overwrites or appends to the SDC file. *Replace* overwrites the contents of an existing file, and *Append* appends to the contents of an existing file.

Applying User-Added Constraints

Use the Apply feature to write, read, and apply the constraints, and then refresh the list of candidates.

1. Type in an SDC filename, or select the '*...*' button to the right of the *Apply* field to open the SDC File window to select a saved SDC file.
2. In the SDC Advisor, click *Apply*.

Important

Not every command created in the SDC Advisor will be applied successfully in Verify mode. There are certain design objects that will not be available in Verify mode. In particular, pins that are not referenced in the original SDC file and that are not key points. If these objects are referenced in the new SDC commands, the only way to apply the new constraints is to go back to Setup mode and read in the new SDC, and then return to Verify mode.

SDC Advisor Clock Page

Use the SDC Advisor's *Clock* page to generate SDCs based on the provided candidate objects for clocks using the `CREATE_CLOCK` and `CREATE_GENERATED_CLOCK` clock generation commands.

The SDC Advisor's *Clock* page includes three display areas:

User/System Generated Candidates	Lists the candidate pins for defining new clocks (on the same pin or in some other point in the pin's fan-in) so that all registers in the design will be clocked. Click on the object to select it, and right-click to open the pop-up menu, where you can select the following: *Add Candidate*—Opens the Gate Browser. *Delete Candidate*—Deletes the candidate object from the list. *Edit Candidate*—Opens the Gate Browser. *Create Clock*—Opens the SDC Command Editor to select a command, object, or string and write it into the SDC file browser. *Create Generated Clock*—Opens the SDC Command Editor to select a command, object, or string and write it into the SDC file browser. *Schematics*—Opens the candidate in the Schematic viewer. *Source Code*—Opens the candidate in the Source Code viewer.
SDC Defined Clocks	Lists the clocks defined in the SDC files read in Setup mode. Click on the object to select it and right-click to open the pop-up menu, where you can select *SDC Command Browser* to show the details of the command and how it relates to the SDC rule violations.
User Defined Clocks	Lists the clock defined so far in the SDC Advisor.

SDC Advisor Set Case Analysis Page

Use the SDC Advisor's *Set Case Analysis* page to generate SDCs based on the provided candidate objects the `SET_CASE_ANALYSIS` command.

The SDC Advisor's *Set Case Analysis* page includes three display areas:

set_case_analysis Candidates	Lists the `set_case_analysis` candidates. Click on the object to select it, and right-click to open the pop-up menu, where you can select the following: *Add Candidate*—Opens the Gate Browser. *Delete Candidate*—Deletes the candidate object from the list. *Edit Candidate*—Opens the Gate Browser. *Set Case Analysis*—Opens the SDC Command Editor. *Schematics*—Opens the candidate in the Schematic viewer. *Source Code*—Opens the candidate in the Source Code viewer.
set_case_analysis commands from SDC file	Lists the definitions in the SDC files read in Setup mode. Click on the object to select it and right-click to open the pop-up menu, where you can select *SDC Command Browser* to show the details of the command and how it relates to the SDC rule violations.
User defined set_case_analysis commands	Lists the definitions so far in the SDC Advisor.

SDC Advisor Input Page

Use the SDC Advisor's *Input* page to generate SDCs based on the provided candidate objects the `SET_INPUT_DELAY` command.

The SDC Advisor's *Input* page includes three display areas:

set_input_delay Candidates	Lists the `set_input_delay` candidates. Click on the object to select it, and right-click to open the pop-up menu, where you can select the following: *Add Candidate*—Opens the Gate Browser. *Delete Candidate*—Deletes the candidate object from the list. *Edit Candidate*—Opens the Gate Browser. *Set Input Delay*—Opens the SDC Command Editor. *Schematics*—Opens the candidate in the Schematic viewer. *Source Code*—Opens the candidate in the Source Code viewer.
set_input_delay commands from SDC file	Lists the definitions in the SDC files read in Setup mode. Click on the object to select it and right-click to open the pop-up menu, where you can select *SDC Command Browser* to show the details of the command and how it relates to the SDC rule violations.
User defined set_input_delay commands	Lists the definitions so far in the SDC Advisor.

SDC Advisor Output Page

Use the SDC Advisor's *Output* page to generate SDCs based on the provided candidate objects the `SET_OUTPUT_DELAY` command.

The SDC Advisor's *Output* page includes three display areas:

set_output_delay Candidates	Lists the `set_output_delay` candidates. Click on the object to select it, and right-click to open the pop-up menu, where you can select the following: *Add Candidate*—Opens the Gate Browser. *Delete Candidate*—Deletes the candidate object from the list. *Edit Candidate*—Opens the Gate Browser. *Set Input Delay*—Opens the SDC Command Editor. *Schematics*—Opens the candidate in the Schematic viewer. *Source Code*—Opens the candidate in the Source Code viewer.
set_output_delay commands from SDC file	Lists the definitions in the SDC files read in Setup mode. Click on the object to select it and right-click to open the pop-up menu, where you can select *SDC Command Browser* to show the details of the command and how it relates to the SDC rule violations.
User defined set_output_delay commands	Lists the definitions so far in the SDC Advisor.

SDC Command Editor

Use the SDC Command Editor to select a command, object, or string and write it into the SDC file browser. Click *Apply* in the Editor to add the command to the *Command List*.

For the *Clock* page, you can open the SDC Command Editor as follows:

- Right-click on an object in the *System Generated Candidates* column and select *Create Clock*.

- Right-click on an object in the *System Generated Candidates* column and select *Create Generated Clock*.

For the *Set Case Analysis* page, right-click on an object in the *set_case_analysis Candidates* column and select *Set Case Analysis*.

For the *Input* page, right-click on an object in the *set_input_delay Candidates* column and select *Set Input Delay*.

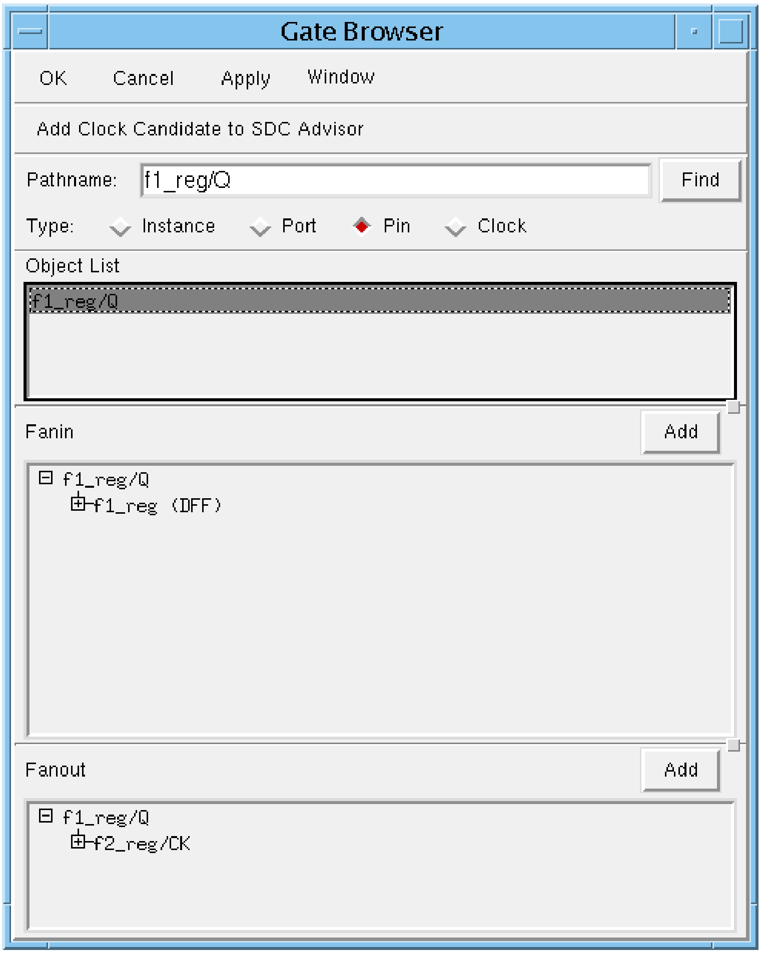
For the *Output* page, right-click on an object in the *set_output_delay Candidates* column and select *Set Output Delay*.

Gate Browser

Use the Gate Browser to add candidates, as well as open the Source Code and Schematic Viewer for Fan-Ins and Fan-Outs.

With the (Editable) Command Browser, you can write a command, object, or string into the SDC File Browser.

➤ To open the Gate Browser from the SDC Advisor, right-click on an object in the *Candidates* column and select *Add Candidate* to open the Gate Browser.



This form includes the following options and display areas:

Pathname	Specifies the name of the path for the candidate. You can type in the name or click *Find* to open the browser to select a pathname.
Type	Specifies the type of candidate to display. Choose *Instance*, *Port*, *Pin*, or *Clock*.

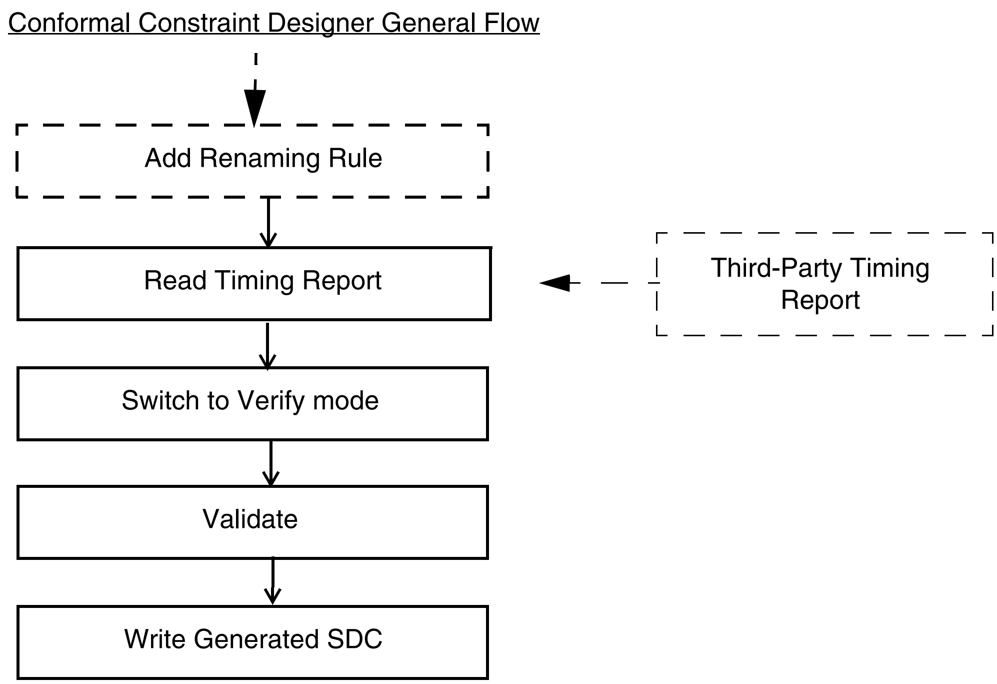
Object List	Click on the object to select it and right-click to open the pop-up menu, where you can select the following: *Delete*—Deletes the object from the list. *Delete All*—Deletes all objects from the list. *Fanin/Fanout*—Opens the fan-in or fanout of the object.
Fanin/Fanout	Click on the object to select it and right-click to open the pop-up menu, where you can select the following: *Delete*—Deletes the object from the list. *Schematics*—Opens the Module Schematic viewer. *Source Code*—Opens the Source Code viewer. *Fanin/Fanout*—Opens the fan-in or fanout of the object.

7

Running Timing Report Validation

Timing Report Validation Flow

This section describes how to run timing report validation. The Conformal software reads timing paths from the timing report, and then performs functional checks to determine whether any of them are false paths. This approach eliminates the need to use an external validation tool to validate generated false paths.



Steps in the Timing Report Validation Flow

- Add Renaming Rules (Optional) on page 125
- Reading the Timing Report on page 125

Add Renaming Rules (Optional)

(Optional) Define renaming rules with the `ADD RENAMING RULE` command in case you need to rename the timing report to map to the design.

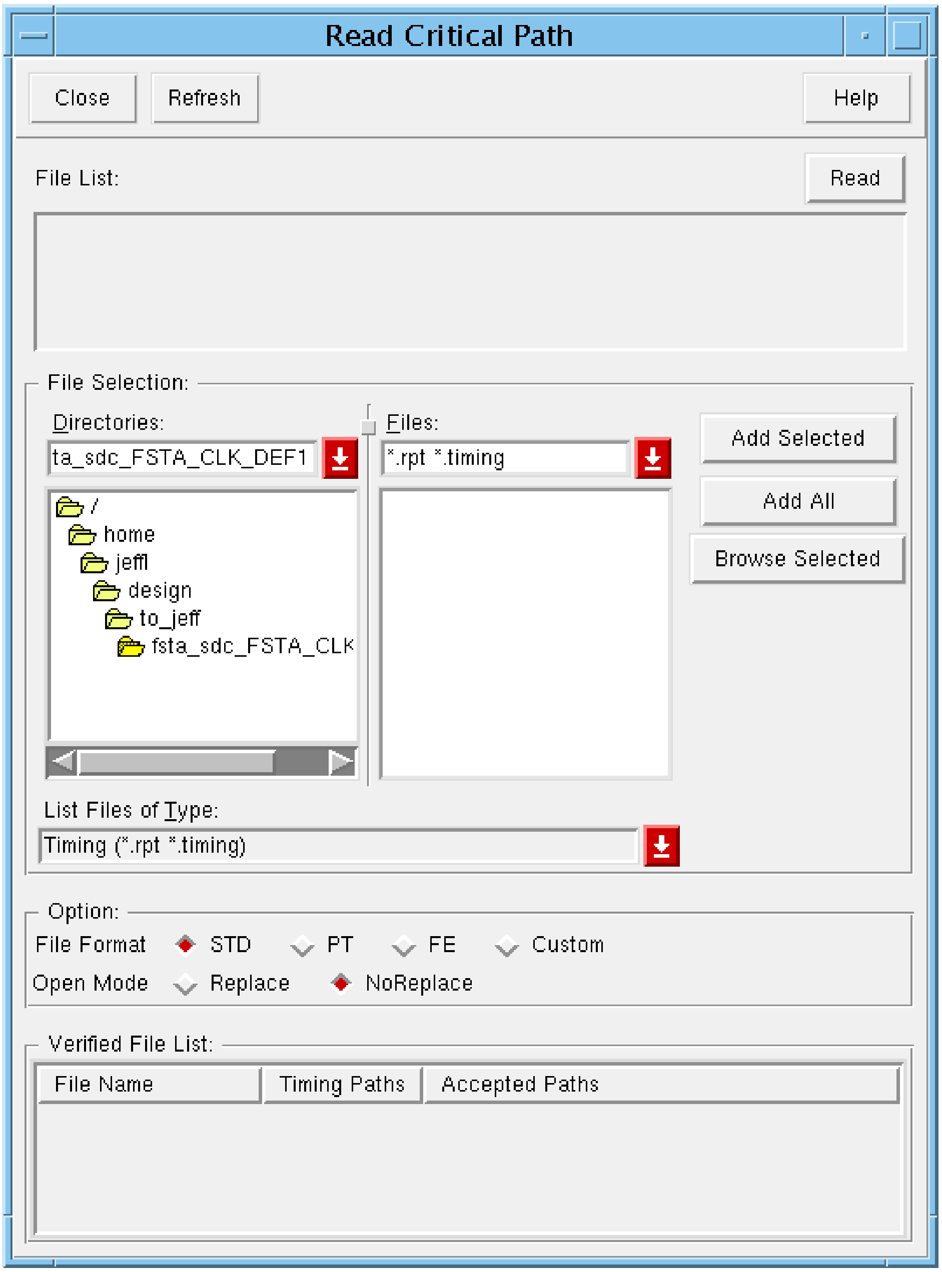
Reading the Timing Report

You can read in a generated timing from the SoC or RTL Compiler Cadence tools, or non-Cadence PrimeTime (third-party) tool:

For PrimeTime (a non-Cadence tool), you can generate a standard timing report using the `READ CRITICAL PATH` command's `-STD` option. This script is contained at:

`<install_dir>/share/cfm/ccd/tool_kit/ccd_pt_std.tcl`

Alternatively, you can use the Read Critical Path form (*File – Read Critical Path*) to read in timing reports to generate false paths.



Read Critical Path Fields and Options

File List	Displays the list of timing report files you specified in the *File Selection* section.
File Selection	Use this section to choose the timing report files. Double-click file folders in the *Directories* display to specify the location of the desired files. Click *Add Selected* to add the selected files, or click *Add All* to add all the files in the *Files* list box Click *Browse Selected* to open the SDC Source Code Manager on the selected file to display its contents.
List Files of Type	Filters the file display. By default, this displays only the files with `.rpt` and `.timing` extensions. *All files* displays all of the files.
Option	Use this section to specify the type of timing report. You can select one of the following formats: ■ *PT*—PrimeTime timing report. ■ *RC*—RTL Compiler timing report. ■ *DC*—Design Compiler timing report. ■ *FE*—First Encounter® timing report.
Verified File List	Use this section to view a list of all the verified files and their status. This section also indicates the number of timing paths and accepted paths within the timing report. You can right-click in this section and use the pop-up menu to open a timing report's Source Code Manager.

Additional Tasks Related to Timing Report Validation

Using a Standard Timing Report

Cadence recommends that you input a standard timing report format when reading critical paths. The standard timing report format contains a representation of each critical path, including start points, end points, and through points.

The release tree contains Tcl procs for RC, PT and FE to generate standard report to be read in (using the `read critical path -std` command). The `<install_dir>/share/cfm/ccd/tool_kit` directory includes the following Tcl files:

- `ccd_fe_std.tcl`

  Contains the Tcl proc for First (FE) format. To create a timing report within FE that the Conformal software can read, source the Tcl file in FE and use the following command in FE:

  ```
  ccd_critical_path_fe [-slack <value> -report <filename>]
  ```

- `ccd_pt_std.tcl`

  Contains the Tcl proc for PrimeTime (PT) format. To create a timing report within PT that that Conformal Constraint Designer can read, source the Tcl file in PT and use the following command in PT:

  ```
  verplex_report_timing <[list <pt options>]> [<filename>]
  ```

  Note: The square brackets for the `list` argument are required. For example:

  ```
  verplex_report_timing [list -max_paths 50 -slack_lesser_than 0]
  ```

- `ccd_rc_std.tcl`

  Contains the Tcl proc for RTL Compiler format. To create a timing report within the RTL compiler that the Conformal software can read, source the Tcl file in the RTL Compiler and use the following command in the RTL Compiler:

  ```
  ccd_critical_path_ep [-slack <integer>]
  ```

  where `-slack <integer>` is the value of slack below which endpoints are taken.

Using a Custom Timing Report

Tip

Although this section describes how to use in a custom timing report, Cadence does not recommend using this approach. Cadence recommends that you input a standard timing report format (see "Using a Standard Timing Report" on page 128).

To use a custom timing report, you can customize the `share/cfm/ccd/gui/tcl/read_timing_report.tcl` file in the install tree based on your timing report format. Customize the variables in this file before you use the `READ CRITICAL PATH` command.

The header for the `read_timing_report.tcl` file lists the minimum set of variables required to run critical path analysis:

```
# This file parses PT/FE timing report into a tcl array variable 'reportInfo'.
# Following variables are the minimum set of variables required for
# creating internal critical path analysis data structural (by cpa::deposit_path
# command).
#
# cpa::reportInfo(Startpoint)   : Name of start point
# cpa::reportInfo(Endpoint)     : Name of end point
# cpa::reportInfo(Path)         : Signals along the path separated by space.
#                                 NOTE: 1. Do not include signals on clock
#                                          network.
#                                       2. Valid signal names should be pin
#                                          names and net names
# cpa::reportInfo(PathStLineNo) : Line number points to beginning of path
#
# CCD calls following tcl commands to trigger parsing. The valid return values
# are defined at the beginning of each command
#  cpa::read_pt_report          : Read PT timing report
#  cpa::read_fe_report          : Read FE timing report
#  cpa::read_custom_report      : Read timing report with user defined format
#
```

Tip

If you want to ignore signals along a particular path, you can use the `skip_signal_pattern` Tcl variable.

Writing out False Path Exceptions

Use the `WRITE TRV SDC` command to write out the false path exceptions for the paths in the timing report that are functionally false.

Automatically Running the TRV Flow

The TRV flow can be run automatically can be run automatically using the following Cadence software tools:

- For SoC Encounter, use `deriveFalsePathCCD` command to generate a standard formatted timing file to pass to the CCD software:

  ```
  deriveFalsePathCCD -timingFile <filename>
  ```

  Note: To use this command, you must the entire path to the CCD installation before running SoC Encounter.

  Note: Running this command executes the entire flow automatically.

- For RTL Compiler, use the generated timing report when running the `write_do_ccd` command.

  Note: Running the RTL Compiler's `generate_constraints -trv` command executes the entire flow automatically.

8

Running Parallel Exception Validation

Note: This feature is only available with the XL license.

You can run parallel exception validation to provide better CPU efficiency for exception validation by parallel execution on multiple machines.

Note: Due to the overhead in parallel exception validation (for example, save, restore, and file I/O), parallel exception validation might not always result in performance improvements over validation on a single machine. Parallel exception validation yields better performance for designs with many exception statements containing large number paths to be validated.

The following scenarios show the different ways you can run parallel exception validation. In both scenarios, there can only be one parallel validation in the dofile because clients terminate after the first parallel validation command.

Scenario 1- One Host with Multiple Clients

In this scenario, the clients validate different exceptions in parallel and save the results to be collected by the host. When there are no more exceptions to be validated, the clients terminate.

The host collects the results of the exceptions. You can monitor the progress of parallel validation on the host machine. When the results for all exceptions are collected, the host continues with the remaining dofile.

Note: The host and each client must have a Conformal XL license.

1. In the dofile, add:

   ```
   set ccd option -parallel -manual -host <hostname>
   ```

   where `<hostname>` is the name of the host machine.

2. In the dofile, add or modify the following for the `VALIDATE` command:

   ```
   validate -parallel
   ```

3. Run the Conformal software on each host and client machine using the same dofile.

Scenario 2 - Multiple Clients

In this scenario, the clients validate different exceptions in parallel and save results to be collected later. With all the machines performing validation, this scenario provides optimal use of the licenses.

After all clients finish with exception validation, you can run the Conformal software on the host machine to collect the results. When the host collects all the results, it continues with the remaining dofile.

Note: Each client must have a Conformal XL license.

1. In the dofile, add:

   ```
   set ccd option -parallel -manual
   ```

2. In the dofile, add or modify the following for the `VALIDATE` command:

   ```
   validate -parallel
   ```

3. Run the Conformal software on each client machine using the same dofile.

4. After the clients finish the validation, in the dofile, modify the following for the `SET CCD OPTION` command:

   ```
   set ccd option -parallel -manual -host <hostname>
   ```

   where `<hostname>` is the name of the host machine.

5. Run the Conformal software on the host machine to collect the results.

Synchronization Files

The exception validation between different clients is synchronized using the following two files in the current directory:

- `.ccd_scheduled_file_<dofile_name>`—Contains checks that need to be scheduled for validation on some client.
- `.ccd_completed_file_<dofile_name>`—Contains the checks that have been completed.

The Conformal software writes the validation results for the completed checks to the `.ccd_save_dir_<dofile_name>` file. The host machine uses the `.ccd_scheduled_file_<dofile_name>`, `.ccd_complete_file_<dofile_name>`, and the files in the `.ccd_save_dir_<dofile_name>` to collect the results.

If these files and results already exist in the current directory from a previous parallel validation session, you have the following three options:

1. Start a fresh parallel validation session.

   Delete `.ccd_scheduled_file_<dofile_name>` and `.ccd_completed_file_<dofile_name>` and `.ccd_save_dir_<dofile_name>` before running the Conformal software. If there are any lock files, you must also delete them (`.ccd_scheduled_file_<dofile_name>.lock` and `.ccd_completed_file_<dofile_name>.lock`).

2. Continue from last session and run checks not completed.

   Delete `.ccd_scheduled_file_<dofile_name>` and `.ccd_completed_file_<dofile_name>` before running the Conformal software. If there are any lock files, you must also delete them (`.ccd_scheduled_file_<dofile_name>.lock` and `.ccd_completed_file_<dofile_name>.lock`).

3. Collect only the existing results without continuing with validation of the remaining checks.

   Delete `.ccd_scheduled_file_<dofile_name>` and `.ccd_completed_file_<dofile_name>`, and delete any lock files (`.ccd_scheduled_file_<dofile_name>.lock` and `.ccd_completed_file_<dofile_name>.lock`), and use the following commands on the host machine:

   ```
   set ccd option -parallel -recover_only
   set ccd option -parallel -manual -host <hostname>
   ```

With the following command:

```
validate -parallel
```

Interrupting a Client

If a client is interrupted (`Ctrl-c`), or does not complete the validation of the check that has been scheduled, it will exit, while other clients will continue with the remaining checks and the host will continue collecting results. However, under Scenario 1, the host will have no indication if the interrupted check is still being validated or not, and will wait indefinitely to collect the results for that check. You must interrupt the host with `Ctrl-c`.

Unique Host and Client Log Files

Because clients and hosts can use the same dofile, you might want to create unique log file for each host and client. Otherwise, the last host or client log file will overwrite all others. You can use the host name as part of the name of log file to create a unique log file for each host and client. For example:

```
set log file log.$HOST -replace
```

where `$HOST` is the environment variable the contains the name of the host or client machine.

Note: If there are multiple runs of the same dofile in the same machine (for example, a machine with multiple CPUs), using the machine is not enough to uniquify the log file. In this case, multiple dofiles must be created with unique log file names to prevent the overwriting in parallel exception validation.

9

Running SDC Rule Checks

Rule Classification

SDC Rule checks fall into two categories:

- Lint Rule Checks on page 138
- Policy Rules on page 138

Lint Rule Checks

You can configure these rules, but you cannot disable them.

Unlike RTL lint rules, you can run SDC lint rules against the SDC language and configure them to meet your methodology or flow requirements. RTL lint rules cannot be configured in this way.

SDC lint rules also ensure that the starting point for policy rules are violation free, thus making it easier to isolate and diagnose policy rule violations.

For list of lint rule checks, refer to the "SDC Lint Rule Checks" chapter of the *Conformal Constraint Designer Rule Check Reference.*

Policy Rules

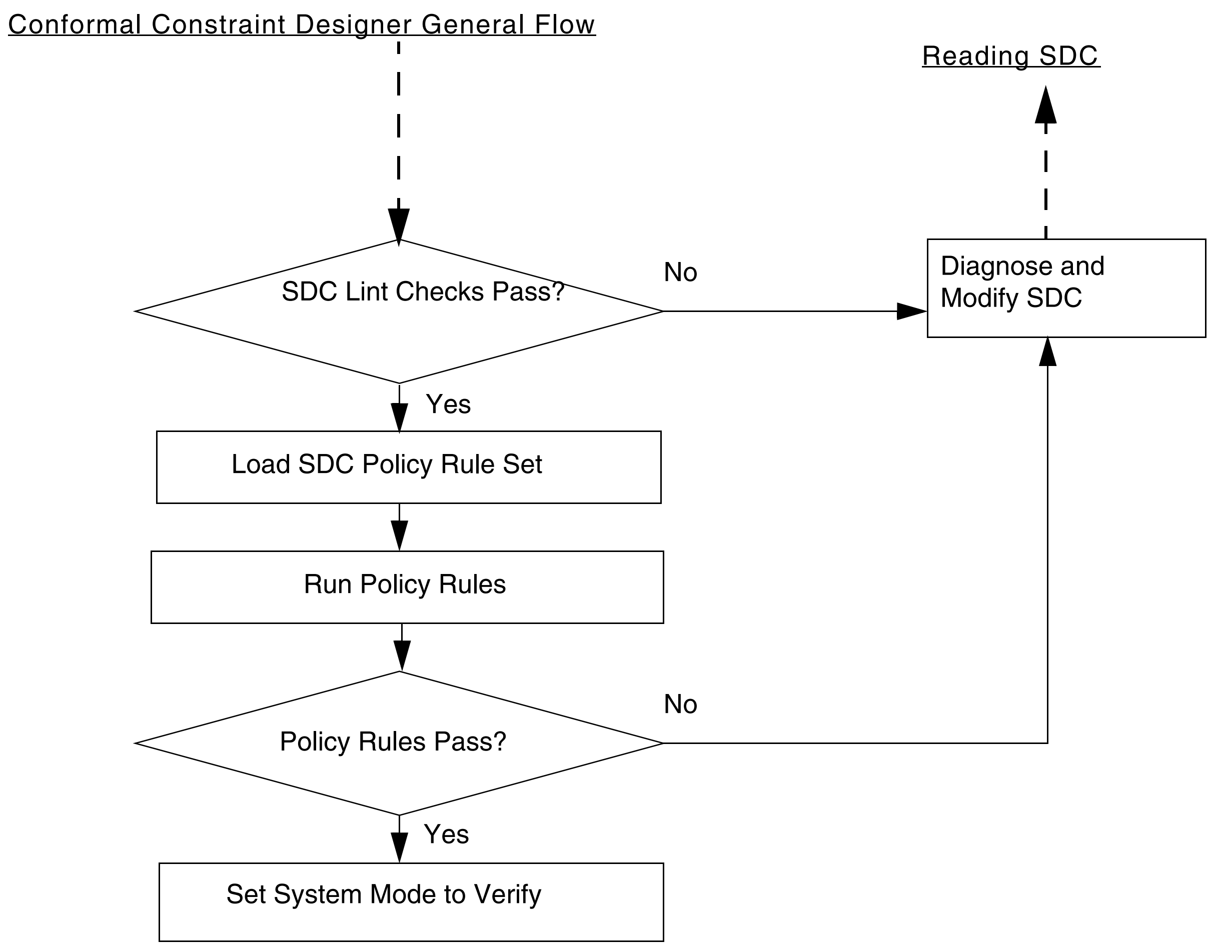
A *policy rule* is a check performed on a design, with or without SDC, that reports whether a given criteria is met. Check levels can vary, from basic structural analysis to complex timing graph analysis—or a combination of both.

You can customize SDC policy rules, write your own rules (for example, reports can be turned into policy rules), and perform Tcl commands on SDC policy rules.

SDC policy rules are disabled by default. You must enable them to perform checks. To enable SDC built-in policy rules, you must read in rule sets. See "Creating Rule Sets" on page 224.

For list of policy rule checks, refer to the "SDC Policy Rule Checks" chapter of the *Conformal Constraint Designer Rule Check Reference*.

Rule Check Flow



Steps in the Rule Check Flow

- Loading SDC Policy Rule Sets on page 140
- Diagnosing SDC Syntax and Semantic Rule Violations on page 141

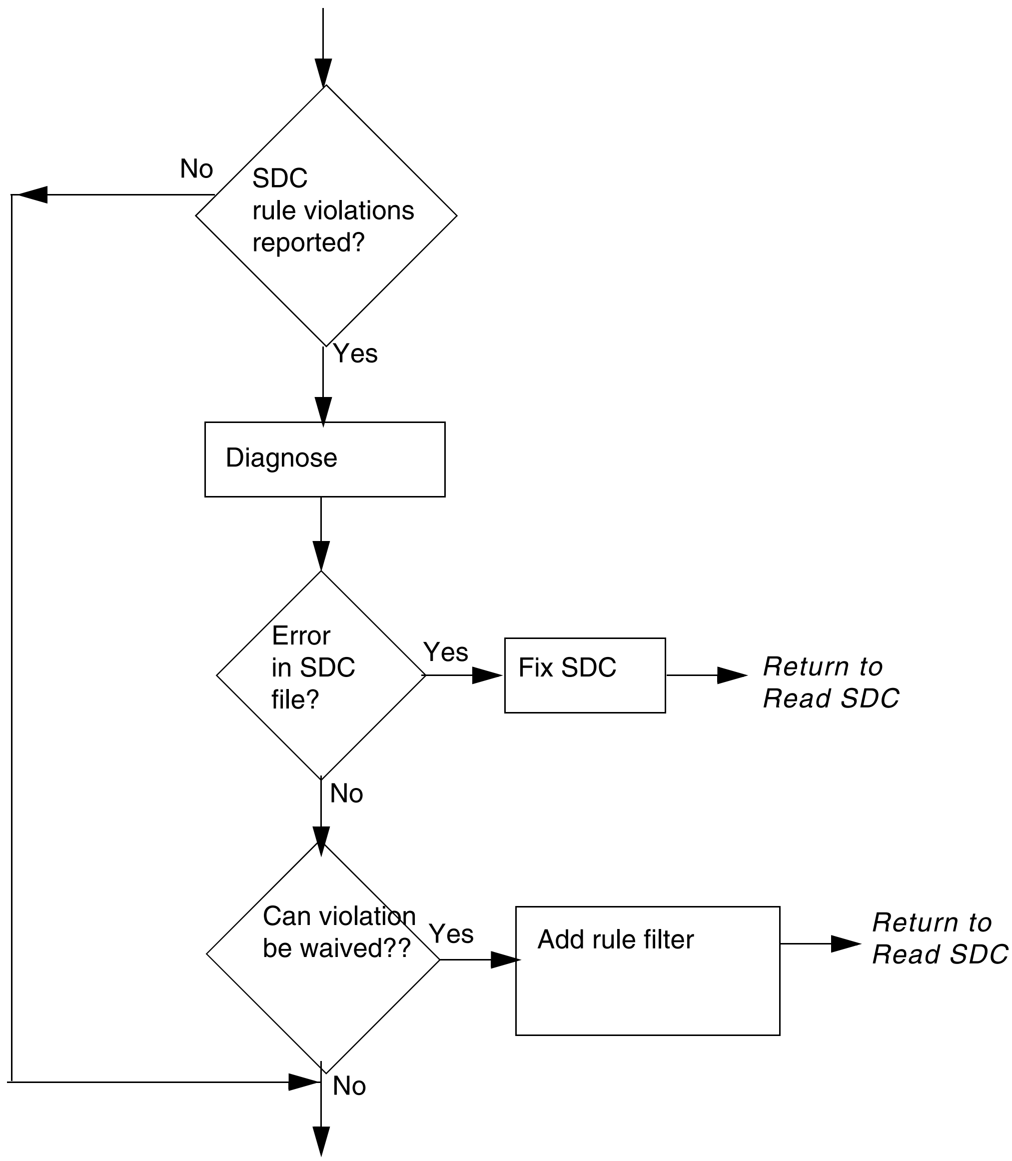
Loading SDC Policy Rule Sets

SDC lint rule checks are built-in checks; they are enabled by default. You can configure lint rule checks, but you cannot disable them.

SDC policy rules are disabled by default. You must enable them to perform checks. SDC policy rules are enabled through *rule sets*. See "Creating Rule Sets" for information on creating and enabling a rule set. If a rule check is added to a rule set, then it is checked. To disable a rule check, you must delete it from the rule set.

Diagnosing SDC Syntax and Semantic Rule Violations

This section describes the tasks involved in diagnosing syntax and semantic rule violations.



Tasks:

- <u>Reporting SDC Rule Violations</u> on page 142
- <u>Diagnosing SDC Rule Violations</u> on page 142

Reporting SDC Rule Violations

After you read in your files, the Conformal Constraint Designer performs extensive semantic checks and displays the rule violations.

To view this information, use the `REPORT RULE CHECK` command.

Diagnosing SDC Rule Violations

You can employ a combination of the integrated diagnosis tools to examine data. These tools include the Schematic Viewer, Waveform Viewer, and the Source Code Manager.

1. If the SDC rule violations are due to errors in the SDC file, fix the SDC file and return to Reading SDC on page 78. Otherwise, proceed to the following step.
2. If the SDC rule violations can be waived, use the `ADD RULE FILTER` command to select which occurrences can be ignored by commands such as `REPORT RULE CHECK`, then return to Reading SDC on page 78:

   For more information on how to disable rule checks, see "Enabling Rule Checks" on page 126.

Working with Lint Checks

Configuring Lint Checks through configure_lint_check Command

The current release implements an extensible design platform that offers the ability to configure SDC lint checks. SDC lint rule checks are built-in SDC quality rule checks. You can configure these rules, but you cannot disable them.

Use the `configure_lint_check` Tcl command to configure the criteria on which SDC lint checks are based.

Example

The `SDC_LINT_OPT7` lint check is flagged when a command does not specify a recommended option. The following illustrates how you can use the `configure_lint_check`'s usage parameter to configure a command's recommended options.

This example adds `-waveform` as a recommended option for the `create_clock` command.

1. View the current configuration for the `create_clock` command, use the `REPORT LINT CONFIGRATION` command:

   ```
   SETUP> report lint configuration create_clock
   configure_lint_check create_clock {

   command {
   ...
   -waveform {
   usage                  dontcare
   ...
   ```

2. Add the `-waveform` option to the `create_clock` command's recommended options:

   ```
   SETUP> tclmode
   TCL_SETUP> configure_lint_check create_clock { -waveform { usage
   recommended } }
   TCL_SETUP> vpxmode
   ```

3. Review the current lint configuration, it reports:

```
SETUP> report lint configuration create_clock
configure_lint_check create_clock {

command {
...
-waveform {
usage                 recommended
...
```

The `-waveform` option is now listed as `recommended`. When you read back in your SDC file, the following command in your SDC file will flag `SDC_LINT_OPT7`:

```
create_clock -name CLK -period 10
```

For a list of all SDC lint rule checks, refer to the "SDC Lint Rules" chapter of the *Conformal Constraint Designer Rule Check Reference*.

Configuring Checks Using an SDC Lint Configuration File

All SDC lint rule checks are built-in and run when you read in your SDC file. SDC lint rules cannot be disabled. However, you can configure the SDC lint rule checks to meet your requirements using the `configure_lint_check` command explained in "Configuring Lint Checks through configure_lint_check Command" on page 143 or through an SDC lint configuration file.

An SDC lint configuration file is a text file that consists of `configure_lint_check` commands, the syntax for which can be derived using the `REPORT LINT CONFIGURATION` command (see example below). You can then read in this configuration file using the `READ LINT CONFIGURATION` command.

To specify for which SDC version should the linting be configured, use the "`-version`" option of this command. The Tcl variable `::sdc::sdc_version_list` contains all the SDC version numbers supported by the tool. To apply the same configuration change file `updated_config.txt` to all supported versions, write:

```
tclmode
foreach v $::sdc::sdc_version_list {
  read_lint_configuration updated_config.txt -version $v
}
```

Example

The `SDC_LINT_OPT7` lint check is flagged when a command does not specify a recommended option.

This example adds `-waveform` as a recommended option for the `create_clock` command using an SDC lint configuration file.

1. View the current configuration for the `create_clock` command using the `REPORT LINT CONFIGRATION` command:

```
SETUP> report lint configuration create_clock
configure_lint_check create_clock {

 command {
  warn_if_overwritten   yes
  usage                 supported
  is_sdc_compliant      yes
  mandatory_options     { -name | source_objects }
  }
...
  -waveform {
  usage                 dontcare
  is_sdc_compliant      yes
  valid_types           { float integer }
  valid_range          { ($length >= 2) && (($length % 2) == 0) }
  }
...
```

2. You can use the output of this command to create an SDC lint configuration file.

```
SETUP> report lint configuration create_clock >
createclockconfig.txt
```

3. Edit the `createclockconfig.txt` file created in the previous step such that the usage parameter is set to `recommended`.

```
  -waveform {
  usage                 recommended
  is_sdc_compliant      yes
  valid_types           { float integer }
  valid_range          { ($length >= 2) && (($length % 2) == 0) }
  }
...
```

   Save the file.

4. Read in the SDC lint configuration file using the `READ LINT CONFIGURATION` command:

```
SETUP> read lint configuration createclockconfig.txt
// Command: read lint configuration createclockconfig.txt
// Note: Read lint configuration successfully: 1 attribute changed
```

5. Review the current lint configuration, it reports:

```
SETUP> report lint configuration create_clock
configure_lint_check create_clock {

command {
...
-waveform {
usage                 recommended
...
```

The `-waveform` option is now listed as `recommended`. When you read back in your SDC file, the following command in your SDC file will flag `SDC_LINT_OPT7`:

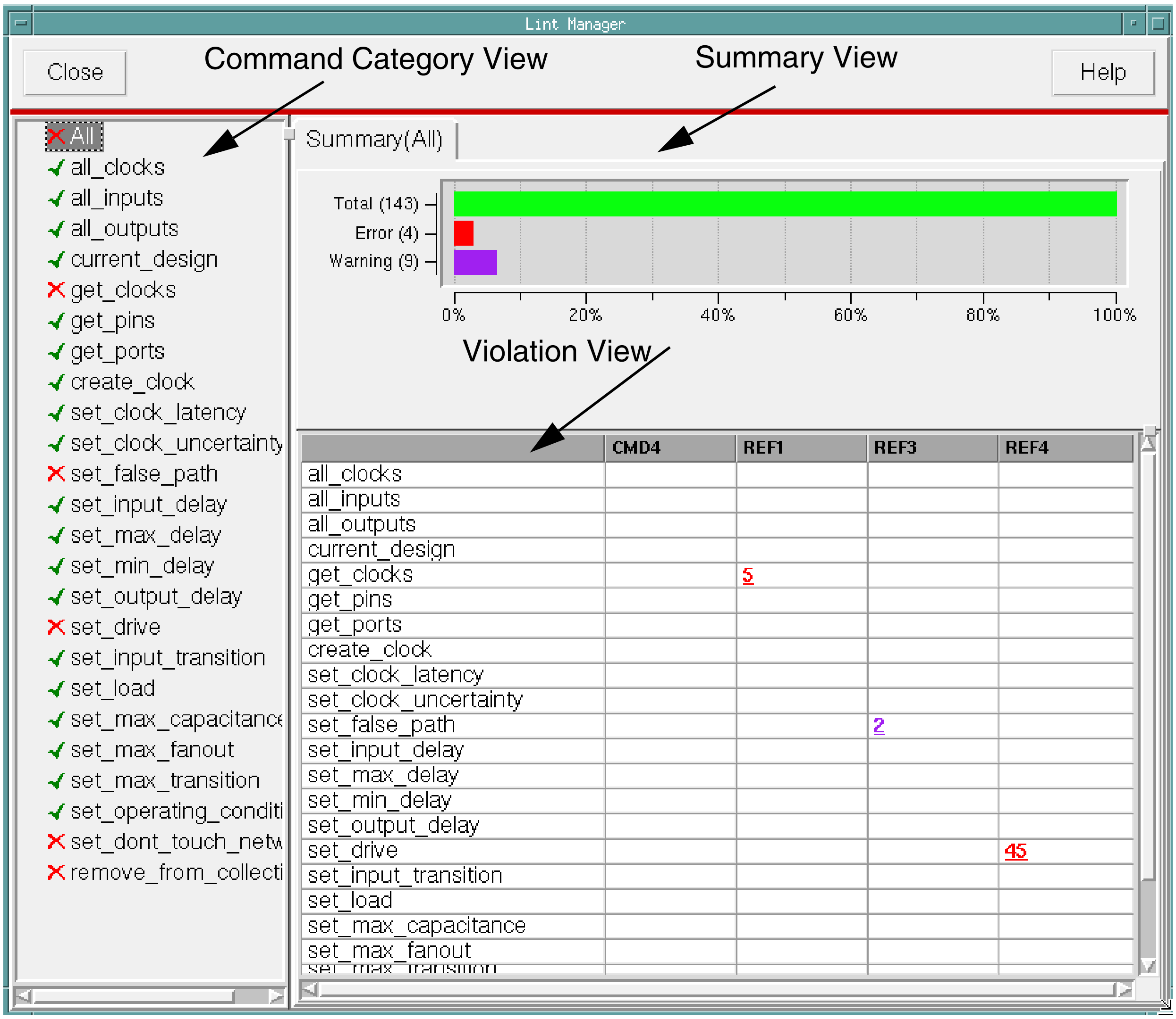
```
create_clock -name myclock -period 10 [get_ports clk]
```

Managing Lint Checks Using the SDC Lint Manager

The SDC Lint Manager displays the lint rules based on what is defined in the rule set (defined with the `ADD RULE SET` and `ADD RULE GROUP` commands).

Note: You can configure SDC lint rules, but you cannot disable them.

➤ Choose *Tools – SDC Lint Manager*



This window has three major sections:

- Command Category View —The left pane of the SDC Lint Manager displays all the SDC command categories in the design and indicates whether:

✕	The command category has lint violations.
✓	There are no lint violations for this command category.

- Summary View—Summarizes the total number of violations for a specific command category.

- Violation View—Categorizes the lint violation by rule check and by command category.

Reporting and Writing Out Current Settings

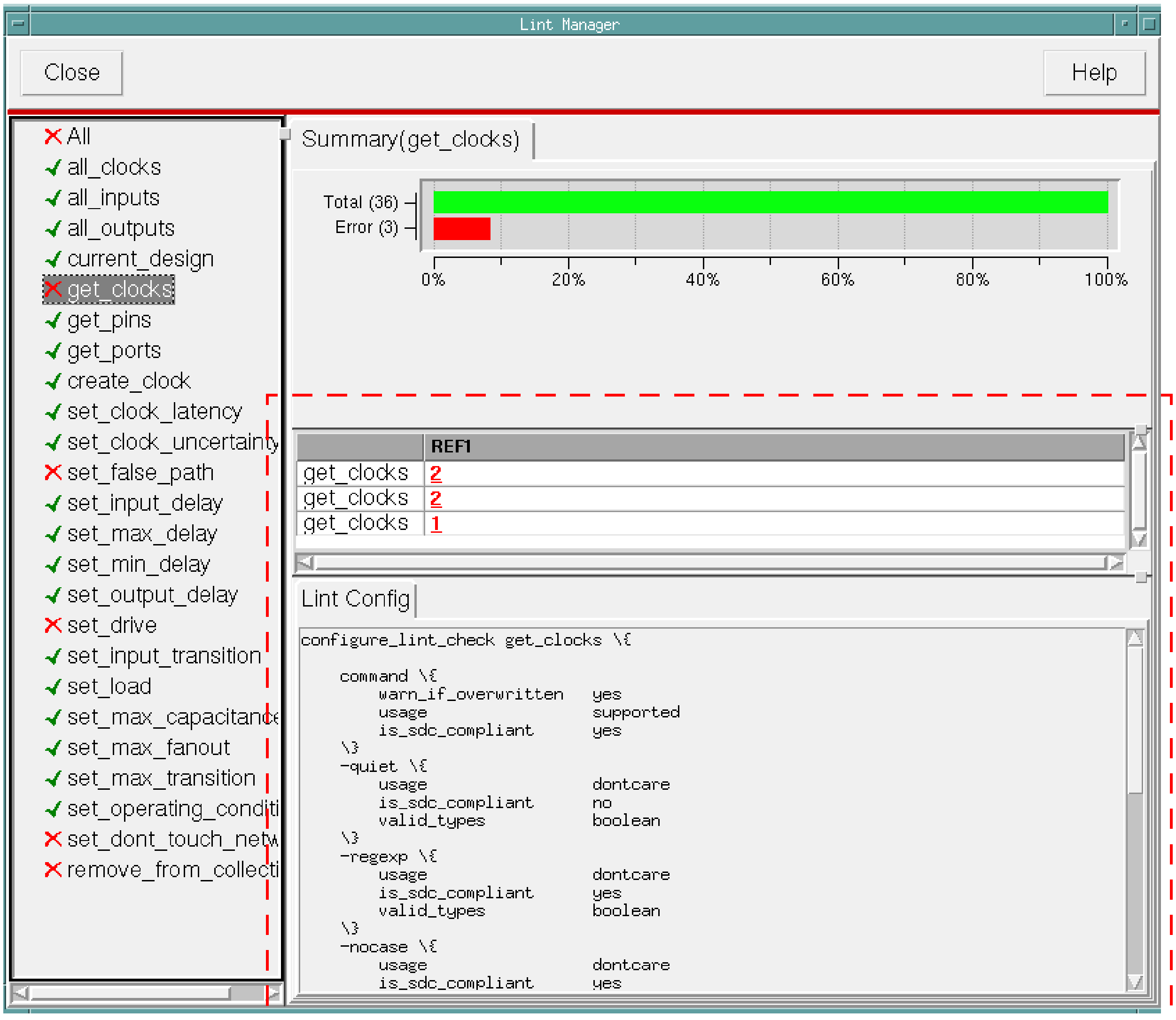
From the left pane, you can right click on a command

to access the following commands:

- `REPORT LINT CONFIGURATION`—Reports the current settings used by CCD to perform SDC lint checks. `WRITE LINT CONFIGURATION`—Writes out the current settings used by CCD to perform SDC lint checks
- `WRITE LINT CONFIGURATION`—Writes out the lint configuration used by CCD to perform SDC lint checks.
- `REPORT RULE CHECK`—Reports the rule occurrences encountered when running the specified rule instances.

Refer to the *Conformal Constraint Designer Reference* for more information on these commands.

The configuration displays in the lower right hand pane.

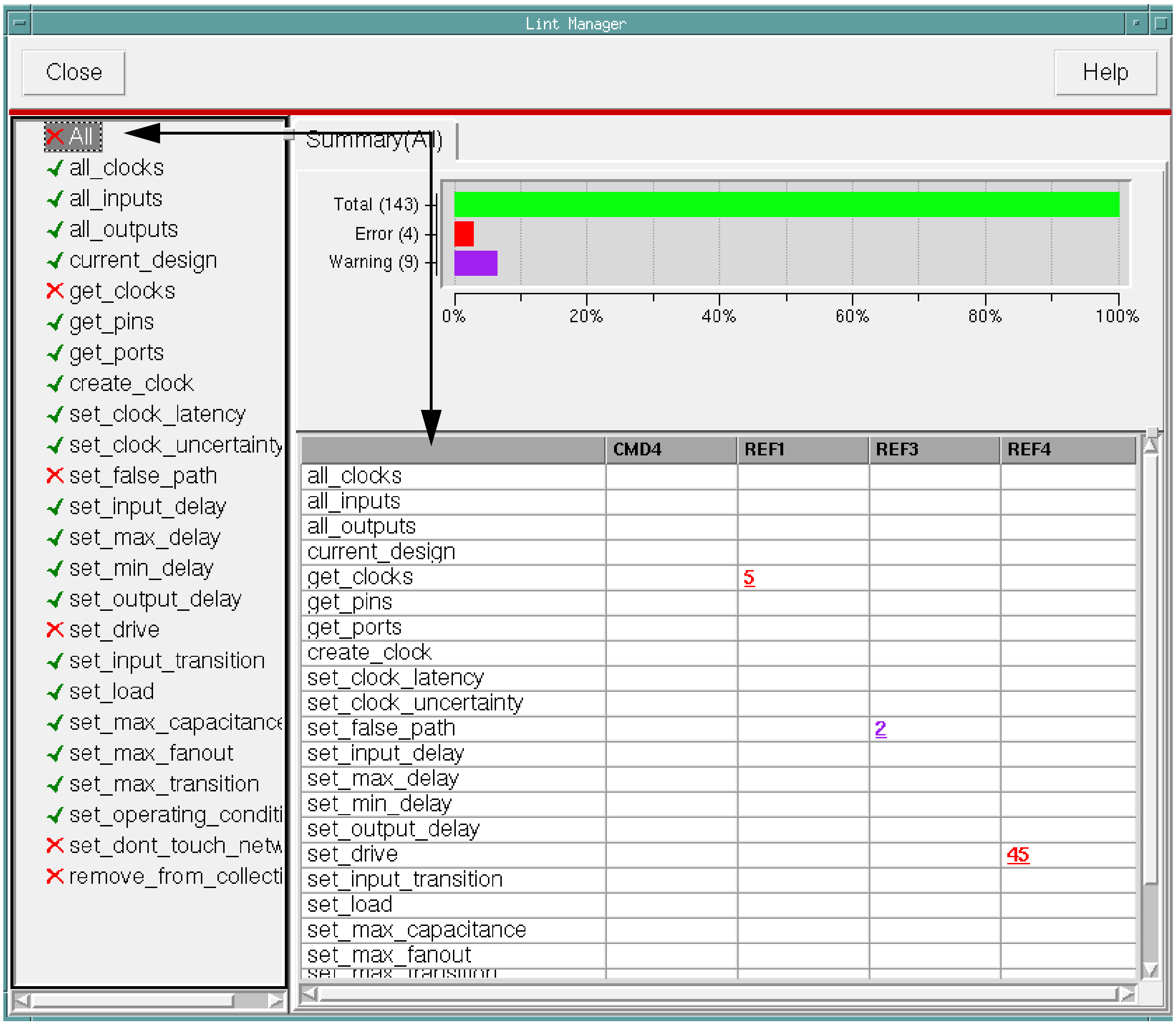


Reviewing SDC Lint Message Violations

The lower right hand pane categorizes lint violations by specific rule check and command category.

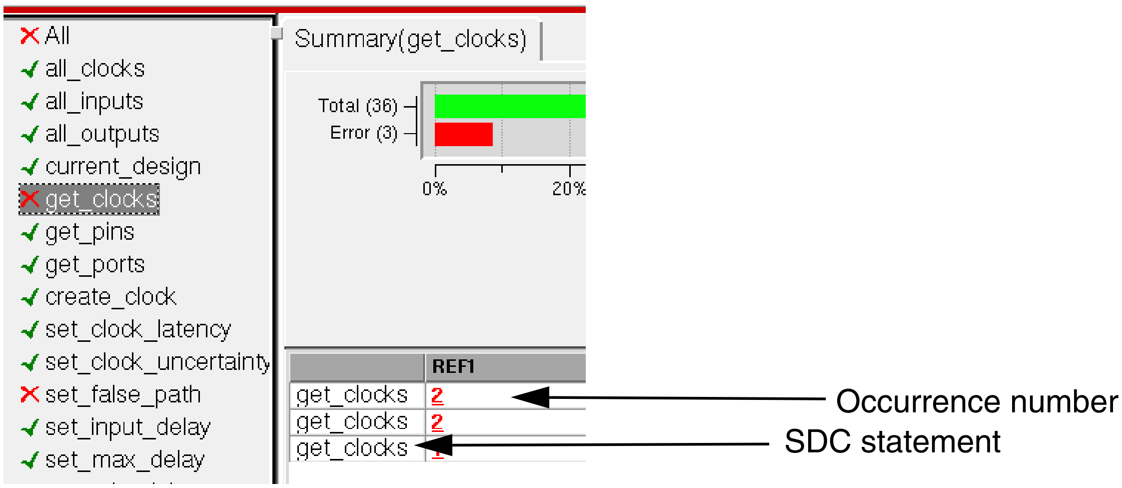
For example, if you select *All* in the Command section, the lower right hand pane displays the number of lint violations for a particular rule check and command category. The numbers are also color coded to indicate the type of lint message.

Note: If there are multiple violations for a particular rule check, the SDC Lint Manager displays only the strongest violation.



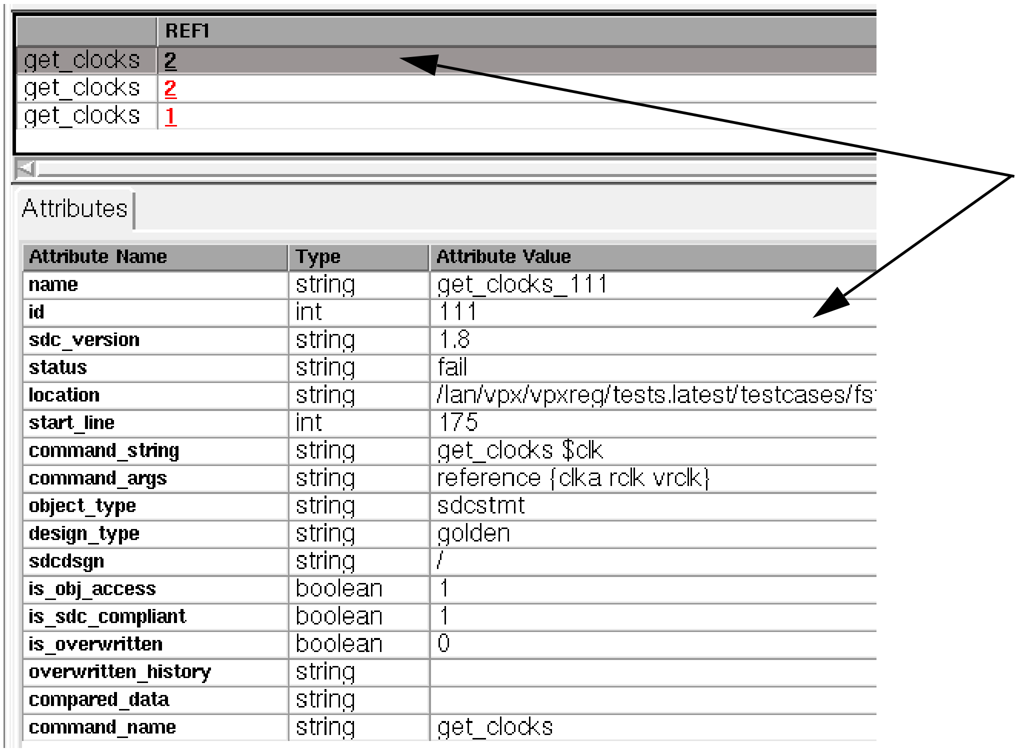
Reviewing a Specific Command Category

If you select a specific command category in the Command section, the lower right hand pane displays the number of lint messages per command occurrence.

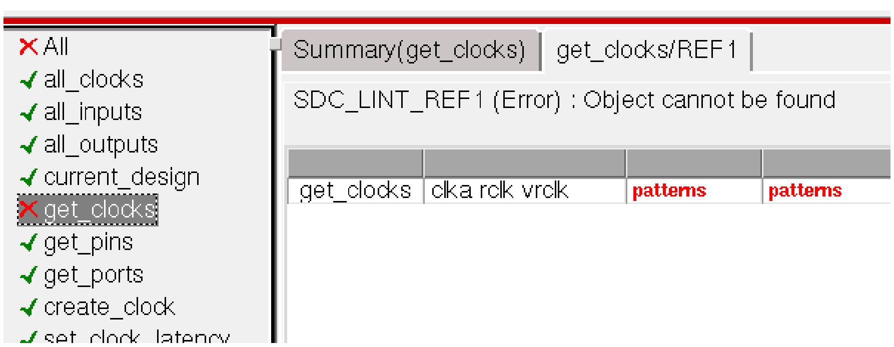


Displaying Attributes

To display the attributes for a command category, click on the command category from the lower-right hand pane:



You can also click on the occurrence number to display more information about the lint message (a page opens behind the Summary page):



Right click and select *Close Page* to close the page.

Changing the Severity of a Lint Check

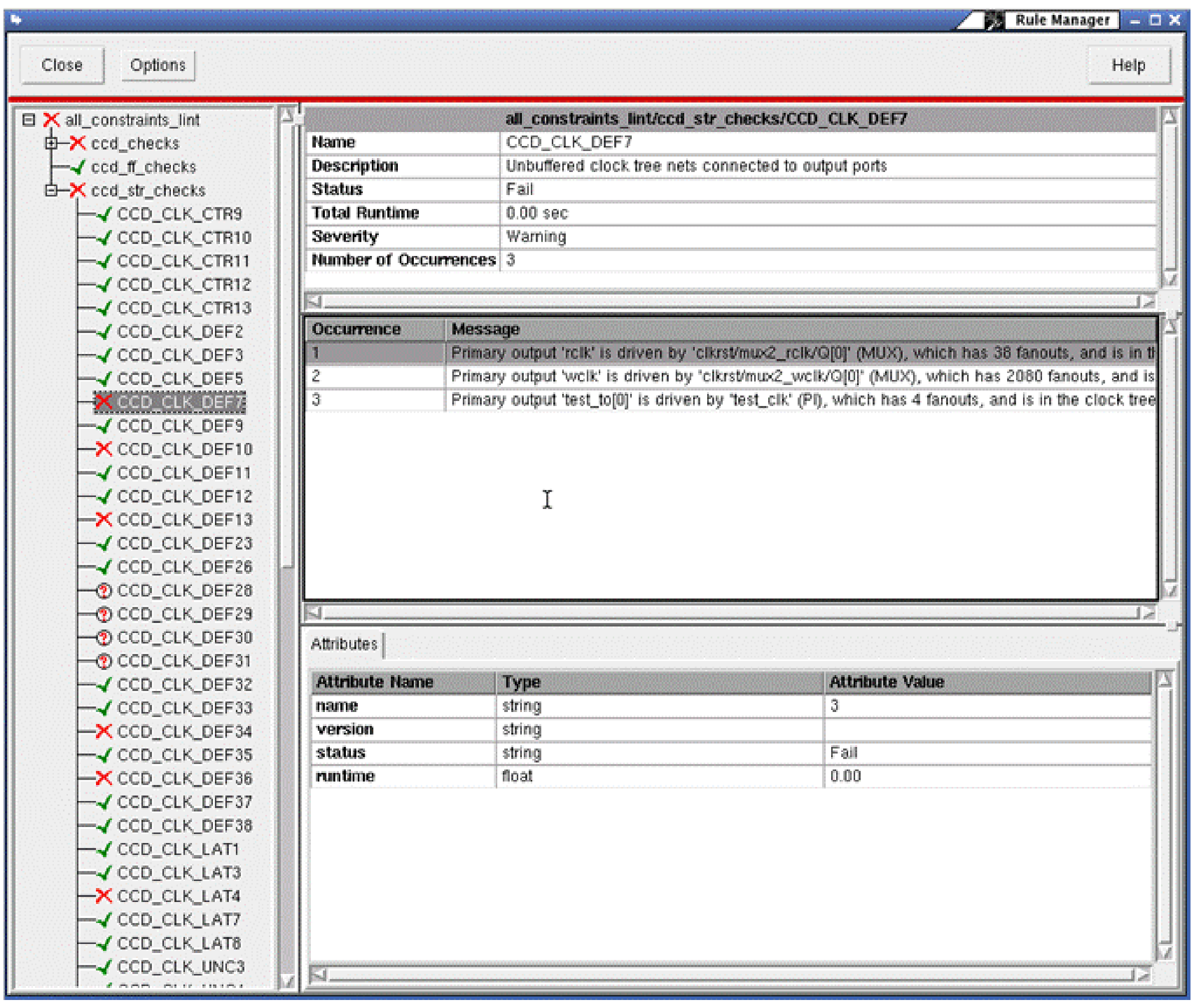
You can use the `set_attribute` Tcl command to change the severity of a lint or policy rule check. For example, the following command changes the severity of `SDC_LINT_OPT2` to Note:

```
set_attribute [ find -ruleinst SDC_LINT_OPT2 ] severity Note
```

You cannot, however, downgrade lint rules with a severity of Error. For example:

```
TCL_SETUP> set_attribute [find -ruleinst SDC_LINT_VAL1] severity warning
// Error: Severity level of SDC_LINT_VAL1 cannot be changed.
```

➤ Click the Rule Manager toolbar widget.



Running Multi-Mode Checks

Note: This feature requires both XL and MCC XL-Option licenses.

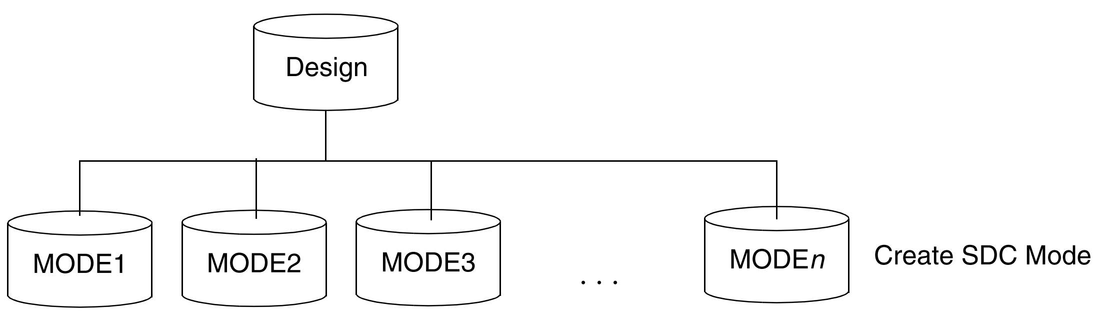
The multi-mode SDC capability for the Conformal Constraint Designer helps you to create and ensure clean and optimal SDC constraints for a multi-mode design. It helps safe-guard SDC constraint quality before multi-mode optimization for any design synthesis and implementation place-and-route tool. With multi-mode SDC, you can read in multiple modes SDC files, run constraint designer checks, and report any inconsistencies or missing constraints in different modes.

To create an SDC mode, or list of modes, use the `ADD SDC MODE` command. After creating an SDC mode with the `ADD SDC MODE` command, you can modify a mode, or a list of modes, with the `SET SDC MODE` command.

- Multi-Mode SDC Rule Check Example on page 154
- Multi-Mode vs Non Multi-Mode Flow on page 155
- Non-Multi-Mode Sample Dofile on page 156
- Multi-Mode Sample Dofile on page 156

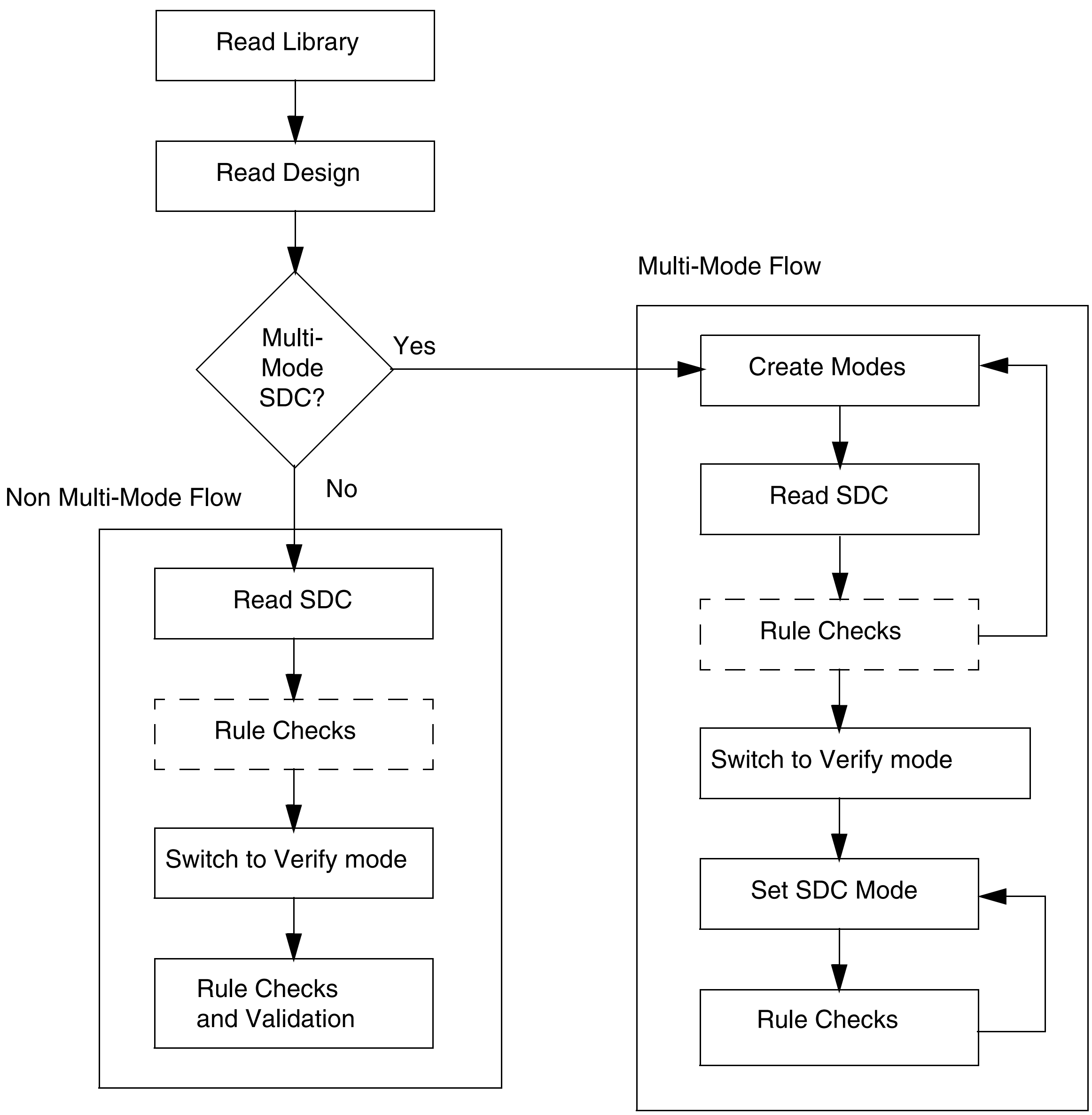
Multi-Mode SDC Rule Check Example

The following graphic applies to the example below:



- `Set_false_path` timing exception in `MODE1` but missing in `MODE2`.
- Different mode of constraint cannot be met. For example `MODE1` has `max_delay` of 5 and `MODE2` has `min_delay` of 6 on the same path.
- Regions are unconstraint due to incomplete mode

Multi-Mode vs Non Multi-Mode Flow



Non-Multi-Mode Sample Dofile

```
read library
read design
read sdc
report rule check
set system mode verify
add rule set -file <ruleset>
run rule check <ruleset>
report rule check <ruleset>
```

Multi-Mode Sample Dofile

```
read library
read design
add sdc mode M1 M2
read sdc -mode M1 <sdcfile>
read sdc -mode M2 <sdcfile>
report rule check
set system mode verify
add rule set -file <ruleset>
add rule instance <name> <rule_src_name>
run rule check <ruleset>
report rule check <ruleset>
```

10

Running Exception Checks

MCP Validation

Note: This feature requires an XL license.

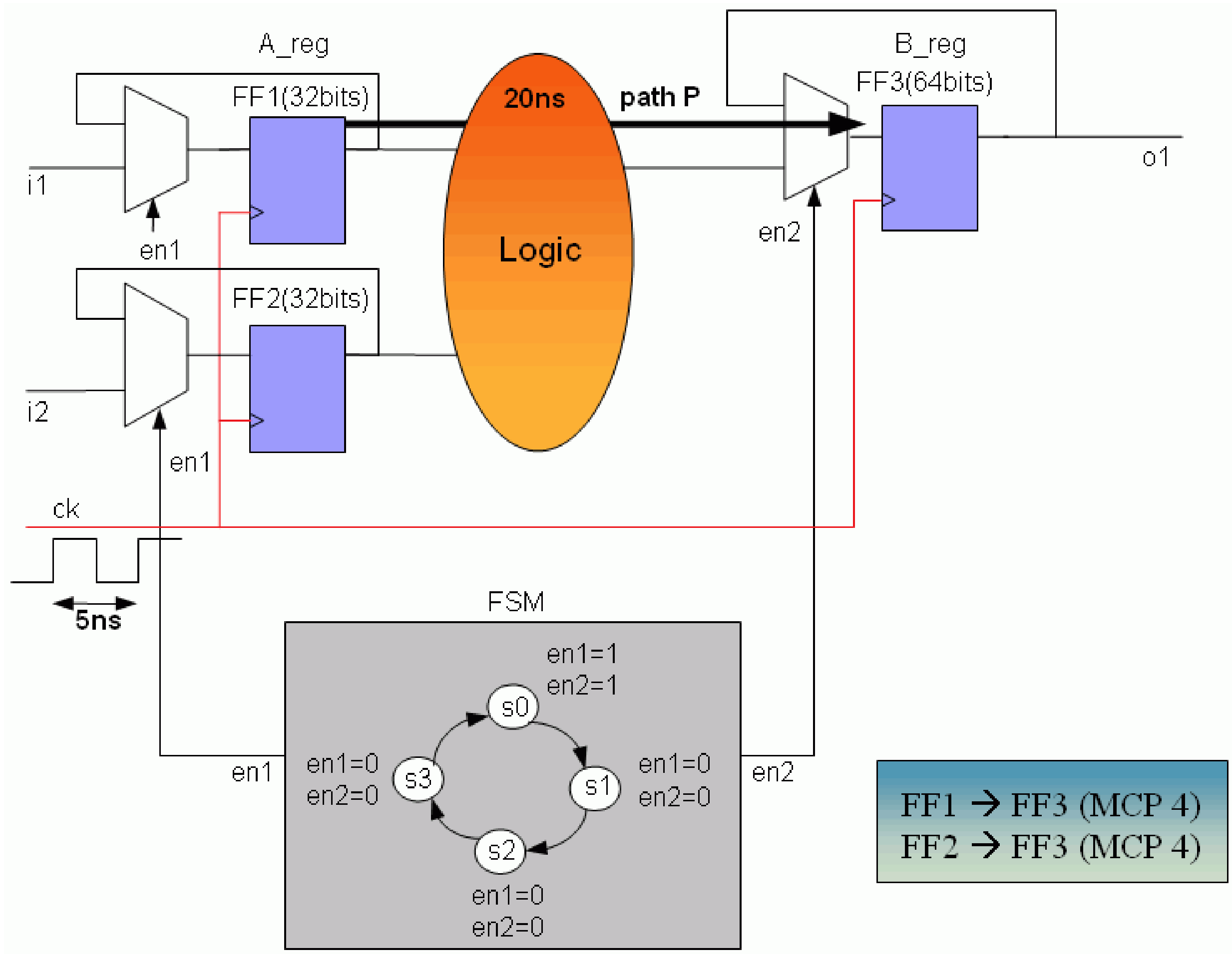
Multi-cycle paths (MCPs) are timing exceptions that start from a valid starting point and through combinational path, and terminate in a valid end point, where the source register changes and the delay through the path is more than a single cycle (N cycle) before it propagates to the destination register.

The Conformal Constraint Designer software's stability, trigger, and holding checks ensure that when the source data changes, the destination register will not change for the specified number of clock cycles (N cycles) given in the exception specified in the SDC file during timing analysis. For example:

```
set_multicycle_path <N> -from  A to B
```

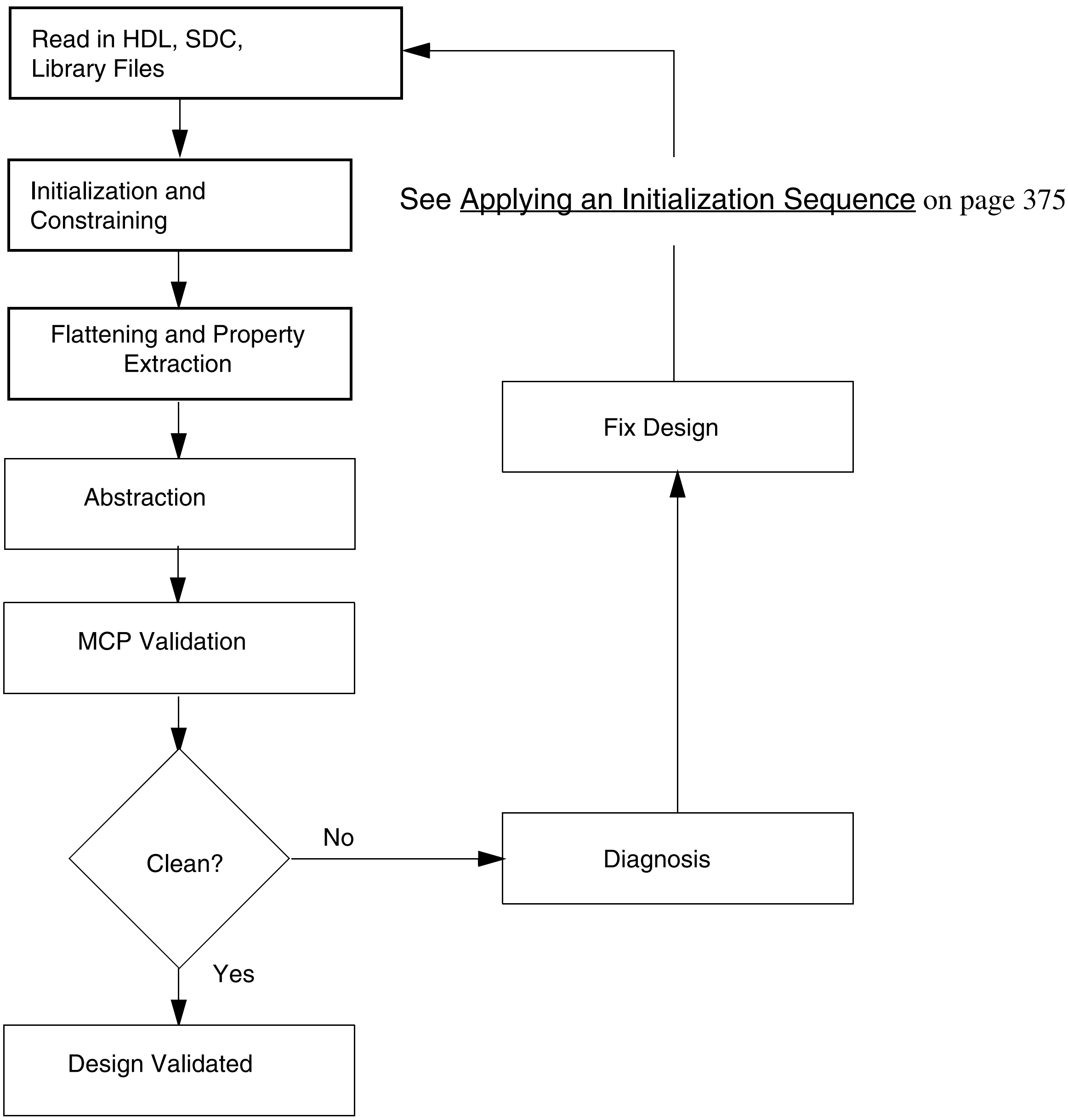
where `N` must be greater than 1 cycle.

The following figure shows an example of an ideal MCP design:



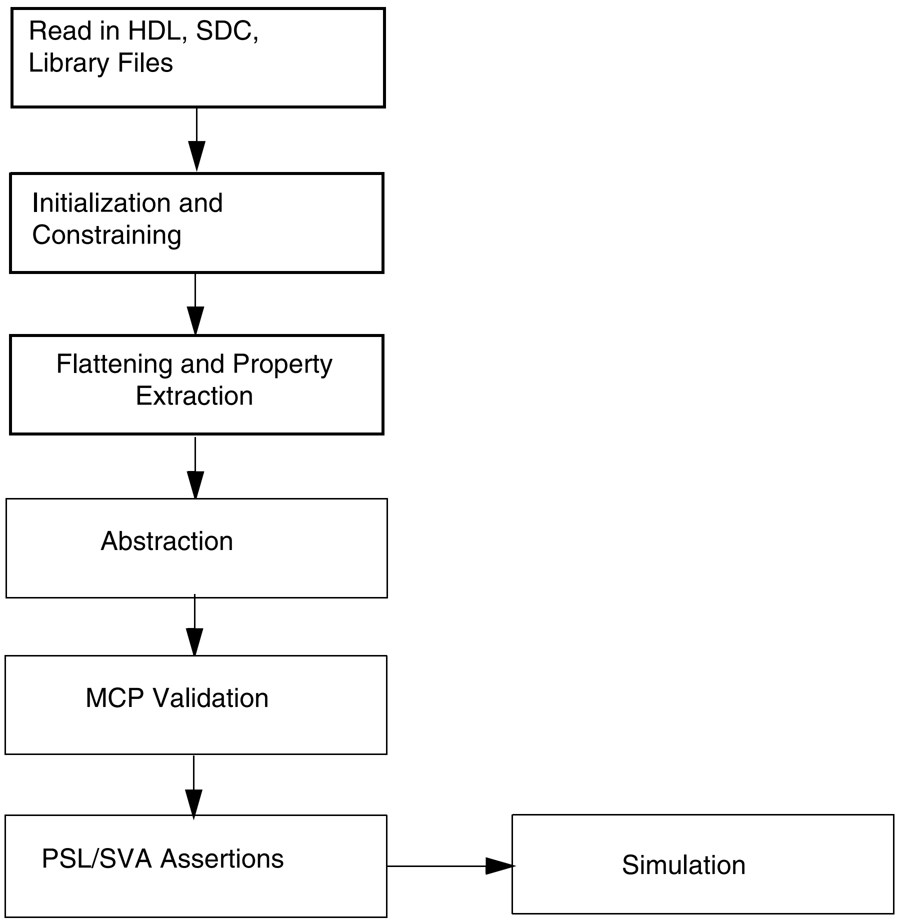
MCP Validation Process Flow

The following flow is for use within the Conformal Constraint Designer software:



MCP Validation Process Flow for Simulation

In the following flow, when the MCP validation result in explored depth, the Constraint Designer can output assertions for the timing exception property to be verified in simulation.



MCP Properties

You can instruct the Conformal Constraint Designer software to do only a certain property that might satisfy the MCP requirement for validation. For example, an MCP with a ‘from’ can only be satisfied by source holding property itself. Similarly, for the ‘to’ destination, holding might be enough in itself.

The following lists the MCP atomic properties (or sub checks) you can select when running the `SET CCD OPTION -MCP_CHECKS` command to validate for each MCP check:

- Source stability

  `SET CCD OPTION -MCP_CHECKS -SRC_STB`

  Path enabled implies source must have held in previous n-1 clock cycles.

  Note: Only source stability is a necessary condition for all MCP checks.

- Source availability

  `SET CCD OPTION -MCP_CHECKS -SRC_AVL`

  Path enabled implies source must have enabled at previous n-th clock cycle.

- Destination stability

  `SET CCD OPTION -MCP_CHECKS -DEST_STB`

  Path enabled implies path not enabled in the previous n-1 clock cycles.

- Source hold

  `SET CCD OPTION -MCP_CHECKS -SRC_HOLD`

  Source enabled implies source must hold in the next n-1 clock cycles.

- Destination hold

  `SET CCD OPTION -MCP_CHECKS -DEST_HOLD`

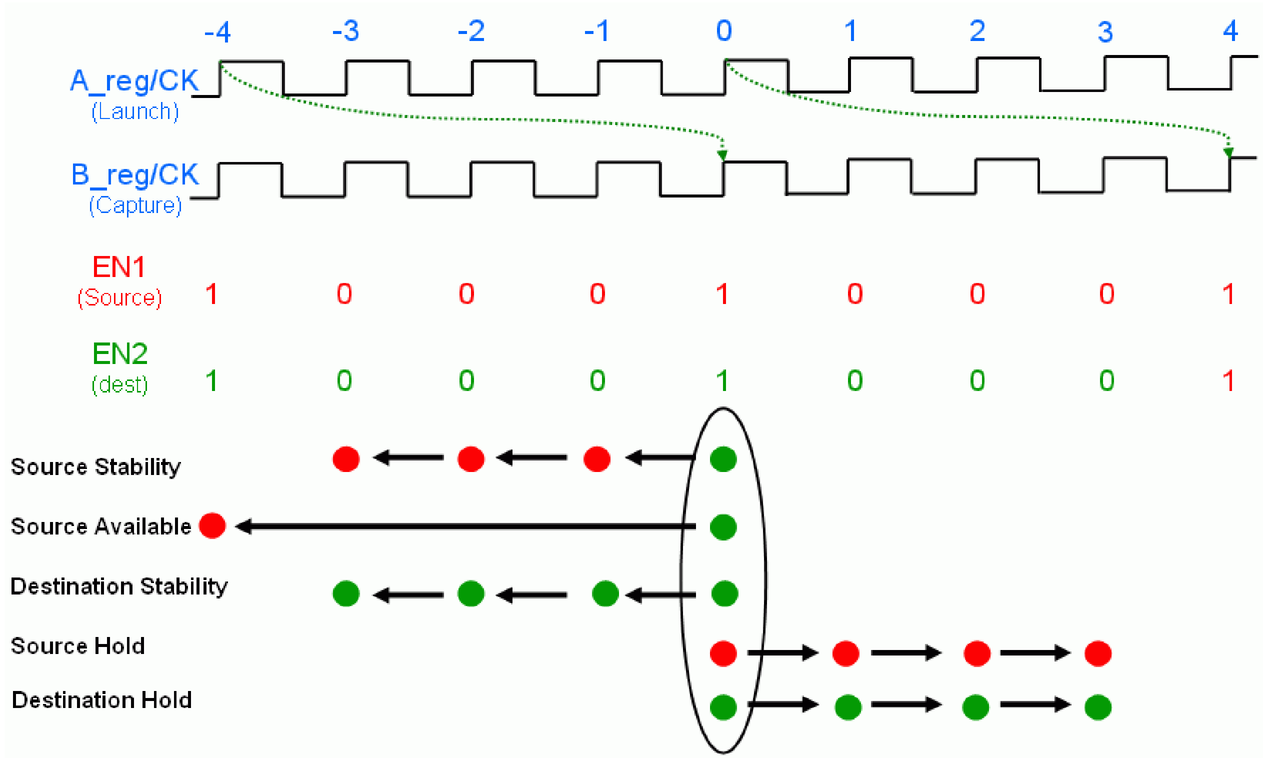
  Destination enabled implies destination must hold in the next n-1 clock cycles.

You can choose all MCP atomic properties to validate for each of the above MCP checks with the `SET CCD OPTION -MCP_CHECKS -ALL` command.

MCP checks are considered passed if all the sub checks selected to be validated all passed. The proof status assigned to a MCP check is the worse result of the all the sub checks. For example, if `-SRC_STB` check passed while `-SRC_AVL` check failed, then the result is a fail for the MCP check.

The following figure shows an example waveform for the following command:

```
set_multicycle_path -setup 4 -from A_reg -to B-reg
```



MCP Dofile Example

```
read design my_design.v
read sdc my_cons.sdc
read initial state init.seq -sequence.
add pin constraint 0 rst
set system mode verify
add sdc check mcp
validate
```

Managing Exception Checks from the GUI

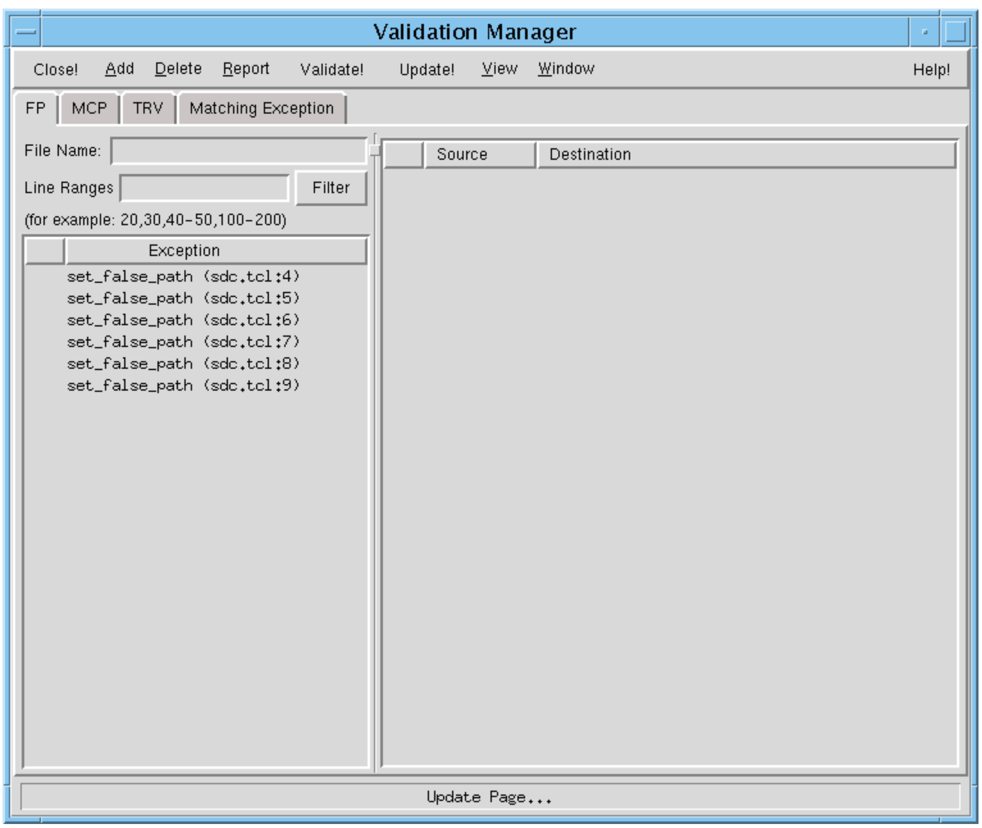
Use the Validation Manager to add, delete, report, and validate exception checks. Use the Sequential Exploration Manager to manually add constraints.

- Validation Manager on page 164
- Using the Sequential Exploration Manager on page 172
- on page 173

Validation Manager

Use the Validation Manager to add, delete, report, and validate exception checks, and to diagnose false exceptions.

➤ To open the Validation Manager, choose *Tools – Validation Manager* from the main window during Verify mode.



File Name specifies the file name, or portion of the file, to filter in the *Exception* list. Click *Filter* to apply the name display.

Line Ranges specifies the line number range to display in the *Exception* list. Click *Filter* to apply the range number display.

The left *Exceptions* panel displays exception statements for the specified source and destination types. The right *Source* and *Destination* panel shows the individual exceptions with the following columns:

- *Status*—Proof status icon (see Proof Status Icons on page 168 for a description of each)
- *Source*—Net or instance origin
- *Destination*—Net or instance end

You can access four pages in Validation Manager with the following tabs:

- *FP*—False-paths.
- *MCP*—Exceptions and multi-cycle-paths
- *TRV*—Exceptions and timing report validation
- *Matching Exception*—Matching exceptions

Menu Bar

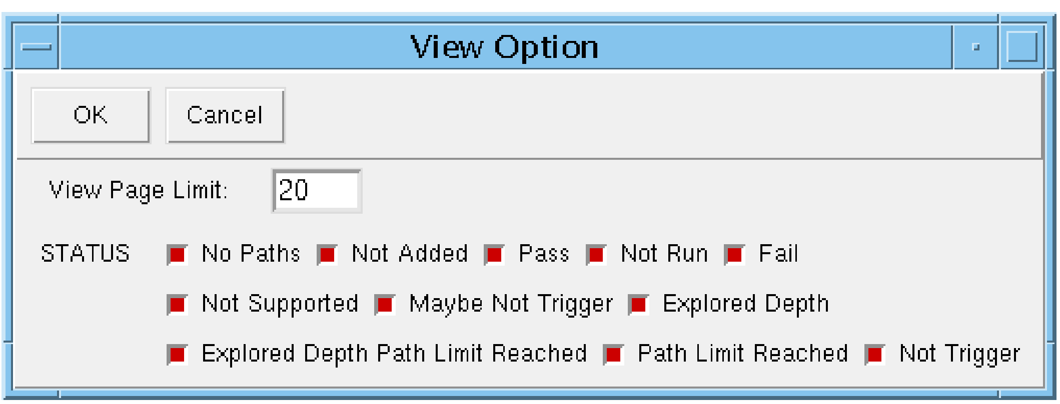
Menu	Available Options	Description
Close	None	Closes the Validation Manager.
Add	*All Checks*	Adds all checks for the active tab (FP or TRV).
Delete	*All Checks*	Deletes all checks for the active tab (FP or TRV).
Report	*All Checks*	Displays a verbose report for all checks for the active tab (FP or TRV). **Note:** Report information displays in the transcript section of the main Conformal Constraint Designer window.

Menu	Available Options	Description
Validate	None	Initiates the proof process. **Note:** Conformal Constraint Designer prints a summary of the results in the transcript section of the main Conformal Constraint Designer window.
Update	None	Refreshes the window.
View	*Option*	Opens the View Option window where you can specify page limit and display paths according to their proof status. See View Option Form Settings on page 167.
	Name Format	Specifies how the Conformal Constraint Designer displays names. ■ *Full*—Displays full name. ■ *Short*—Displays only `...<final_29_characters>`
Window		Lists every open GUI window and lets you move any of them to the front or display them in cascading view.

Setting Viewing Options

You can set the viewing options for the Validation Manager with the View Option form.

➤ In the Validation Manager, choose *View – Option*.



View Option Form Settings

View Page Limit	Specifies the number entries to display at one time in the Validation Manager window. If there are more entries than the specified view page limit, click *<number> more paths* to view additional specified entries in the window.
No Paths	Shows exception statements that do not expand to any exception path.
Not Added	Shows exception statements not added to be checked.
Pass	Shows exception statements that have been proved pass.
Not Run	Shows exception statements that have been added, but not validated.
Fail	Shows exception statement that have been proved fail.
Not Supported	Shows exception statements that contain exception paths that CCD cannot model.
Maybe Not Trigger	Shows MCP exception statements where CCD was unable to exhaustively prove that the specified MCP is triggerable or not triggerable.
Explored Depth	Shows exception statements that the software was unable to find a counter-example, such as a failed check, and unable to exhaustively prove pass.
Explored Depth Path Limit Reached	Stops validation for the current exception statement if it reaches the specified limit and moves on to the next exception statement.
Path Limit Reached	Shows the exception statement that reached the specified path expansion limit. See `SET CCD OPTION -sdc_path_limit`.
Not Trigger	Shows MCP exception statements that have been proved not triggerable.

Adding Exception Checks

Use the following procedure to add one or more exception checks to the “Prove” list. From the *Exception* list box in the Validation Manager:

1. Highlight an exception statement.
2. Right-click and choose *Add Check* from the pop-up menu.

   The Conformal Constraint Designer inserts a circled question mark (?) in the status column.

Deleting Exception Checks

Use the following procedure to remove an exception check from the *Prove* list. From the *Exception* list box in the Validation Manager:

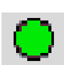
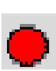
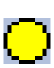
1. Highlight an exception statement.
2. Right-click and choose *Delete Check* from the pop-up menu.

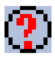
   The Conformal Constraint Designer removes the status icon from the list box.

Validating Timing Exceptions

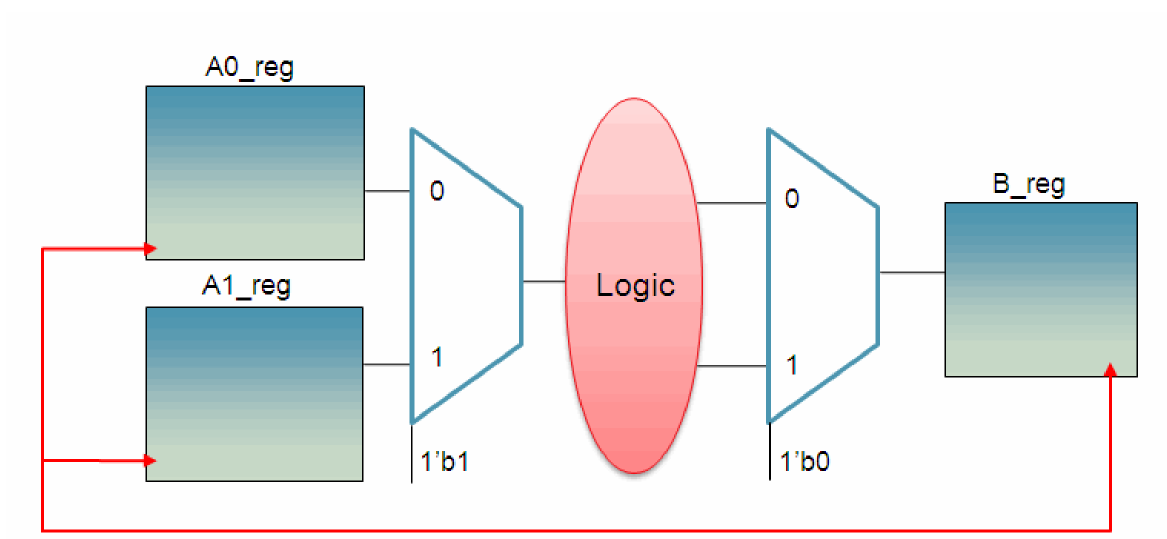
To validate the exception (after Adding Exception Checks), click on the Validate menu item. This initiates the proof process. Conformal Constraint Designer prints a summary of the results in the transcript section of the main Conformal Constraint Designer window and updates the Validation Manager with the exception check's proof status.

Table 10-1 Proof Status Icons

Icon	Status	Description
	Pass	A pass status indicates that the exception statement has been proven to be functionally correct.
	Fail	A fail status indicates that the exception statement has been proven to be functionally incorrect.
	No Paths	A no path status incidates that the path described by the exception statement does not exist or cannot expand to a path as described. This usually happens when the object exists, but the paths described by `-from`, `-to`, and `-through` do not form a path in the design.
	Explored Depth	Shows exception statements that the software was unable to find a counter-example, such as a failed check, and unable to exhaustively prove pass.

Icon	Status	Description
	Not Run	Shows exception statements that have been added, but not validated.
	Not Trigger	Shows MCP exception statements that have been proved not triggerable.
	Maybe Not Trigger	Shows MCP exception statements where CCD was unable to exhaustively prove that the specified MCP is triggerable or not triggerable.
	Not Supported	Shows exception statements that contain exception paths that CCD cannot model.
	Path Limit Reached	Shows the exception statement that reached the specified path expansion limit. See `SET CCD OPTION -sdc_path_limit`.
	Explored Depth Path Limit Reached	Shows MCP exception statements that have been proved not triggerable.

Example 10-1 FP Example for Pass, Fail, and No Path



For FP exceptions, the path:

- From `a0_reg` to `a1_reg` would return a No Path status, as there is no path between them.
- From `a1_reg` to `b_reg` would return Fail, because the path is a real path—not a false path.

- From `a0_reg` to `b_reg` would return Pass, because it is a false path (both selects can never be 0 concurrently).

Reporting Exception Checks

Use the following procedure to view a verbose report for one or more of the exception checks. From the *Exception* list box in the Validation Manager:

1. Highlight an exception statement.
2. Right-click and choose *Report Check* from the pop-up menu.

   The Conformal Constraint Designer reports exceptions or proven data in the transcript section of the main GUI window.

Diagnosing Exception Checks

Once you have validated your exception checks, you can use the following procedure to diagnose an individual exception and to view a counter-example.

From the *Source* and *Destination* list box in the Validation Manager:

1. Click on a path to select it.
2. Right-click and choose *Diagnose* from the pop-up menu.

   The Conformal Constraint Designer reports the results and the counter-example in the transcript section of the main GUI window, and opens the Waveform Viewer and the Flattened Schematic window.

Reporting Validated SDCs

Use the following procedure to view a verbose report for an exception checks. From the *Exception* or *Source* and *Destination* list box in the Validation Manager:

1. Highlight an exception statement or path.
2. Right-click and choose *Report Validated SDC* from the pop-up menu.

   The Conformal Constraint Designer reports the results in the transcript section of the main GUI window.

Debugging with the Multi-Timeframe Schematics

Use the following procedure to view the path of the diagnosed exception using the Schematic Viewer. From the *Source* and *Destination* list box in the Validation Manager:

1. Click a path to select it.
2. Right-click and choose *Schematics*.

   The Conformal Constraint Designer opens the specified schematic.

Viewing Design Source Code

Use the following procedure to view source code for the design file. From the *Source* and *Destination* list box in the Validation Manager:

1. Click a path to select it.
2. Right-click, choose *Design Source Code*, and then choose one of the following from the pop-up menu:
   - *Source*
   - *Destination*

   The Conformal Constraint Designer opens the Source window and automatically scrolls to the appropriate line of code, which it highlights in aqua blue.

Viewing SDC Source Code

Use the following procedure to view context-dependent source code for the SDC file. From the *Exception* or the *Source* and *Destination* list box in the Validation Manager:

1. Click an exception statement or path to select it.
2. Right-click and choose *SDC Source Code* from the pop-up menu.

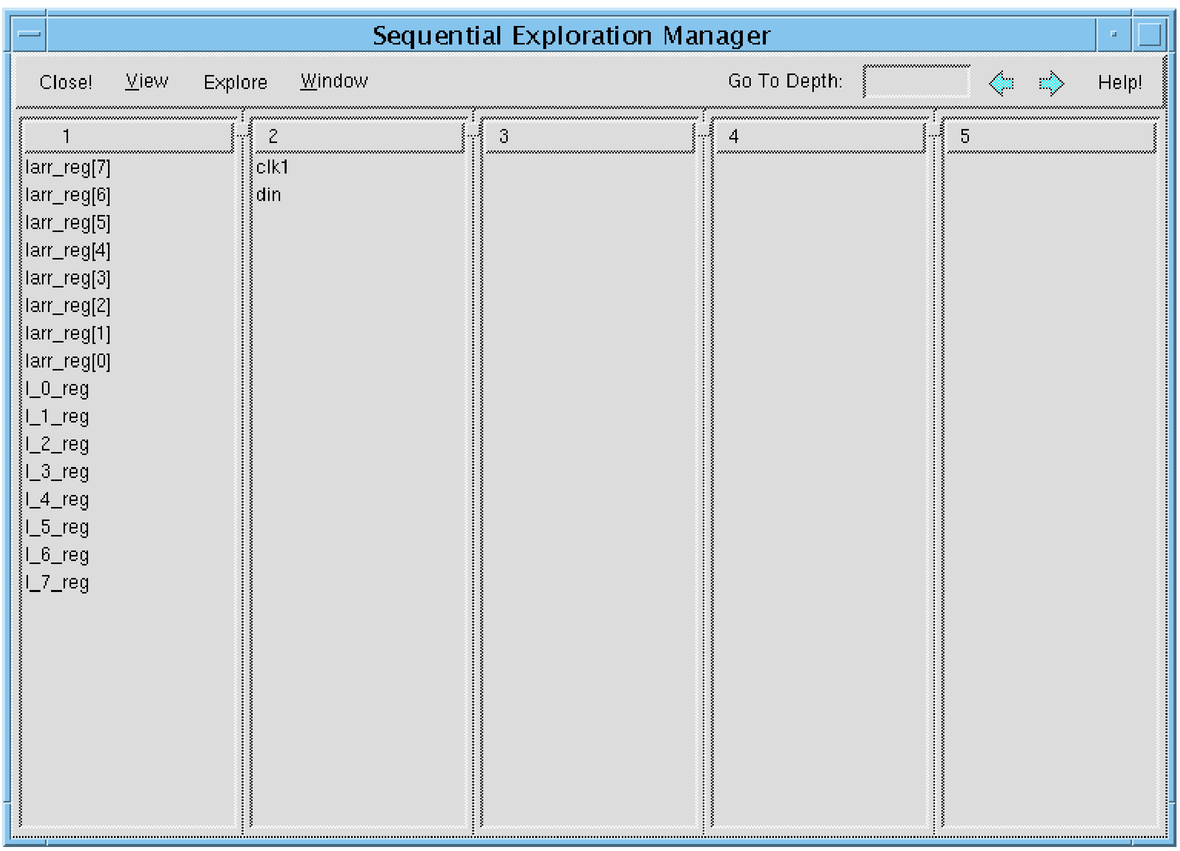
   The Conformal Constraint Designer opens the Source window and automatically scrolls to the appropriate line of code, which it highlights in aqua blue.

Using the Sequential Exploration Manager

You can add manual constraints with the Sequential Exploration Manager by setting a particular instance of the support cone to 1 or 0, or by performing manual abstraction by temporarily cutting the support cone.

To open the Sequential Exploration Manager, do the following:

1. In Verify mode, choose *Tools – Validation Manager* from the main window.
2. From the *Exception* or *Source* and *Destination* list box, right click on an entry in the FP or TRV panel to open the pop-up menu.
3. Choose *Sequential Explorer*.



The constraints and abstractions are done temporarily for “what-if” diagnosis. The Conformal Constraint Designer performs validation using the user-specified constraints and abstraction and reports the results. However, the constraints and abstractions are temporary and they do not change the original FP and MCP validation results.

Saving and Restoring Validation Sessions

You can use the `SAVE SDC CHECK` and `RESTORE SDC CHECK` commands to save and restore an exception validation session for later analysis without revalidating the exceptions. This can help increase performance time in the batch mode validation, where upon completion, the result of the exception validation is saved to a file. You can later restore the previously saved session to further analyze and diagnose the exception validation.

Note: All settings for the restore, such as the design files, library files, SDC constraints files, as well as the settings for ccd parameters and options, must be the same as the ones used for the save.

- To restore a saved validation session, the same version of the Conformal Constraint Designer software must be used as the one that was used to save the exception validation session.

11

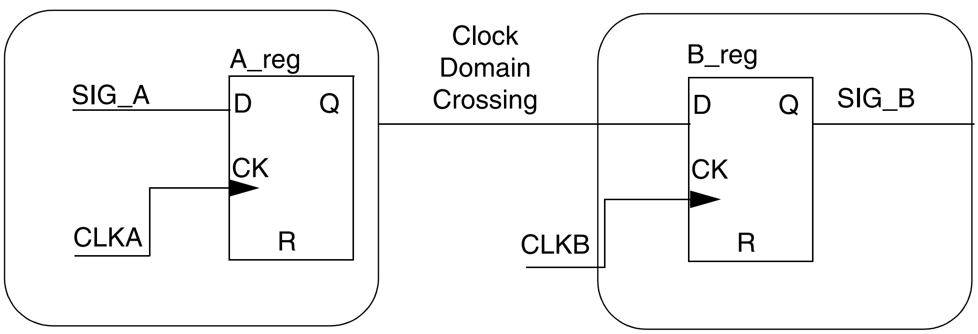
Running Clock Domain Crossing Checks

This chapter describes:

Clock Domain Crossing Overview

Conformal Constraint Designer provides Clock Domain Crossing (CDC) checks. A *clock domain* is defined as that part of the design driven by either a single clock or clocks that have constant phase relationships. A clock and its inverted clock or its derived divide-by-two clocks are considered a clock domain (synchronous). Conversely, clocks that do not have a known phase or frequency relationship between them are known as *asynchronous clocks*. A *clock domain crossing* (CDC) occurs when a signal crosses between two asynchronous clocks. (as illustrated in the following figure, where `CLKA` and `CLKB` are asychronous clocks).

Figure 11-1 Clock Domain Crossing



Because the relationship between the two clocks is non-deterministic, transferring data between the sender and receiver clocks can cause problems. The following section describes the common issues that occur when there is a clock domain crossing.

Clock Domain Crossing Issues

This section describes the common issues that occur when there is a clock domain crossing.

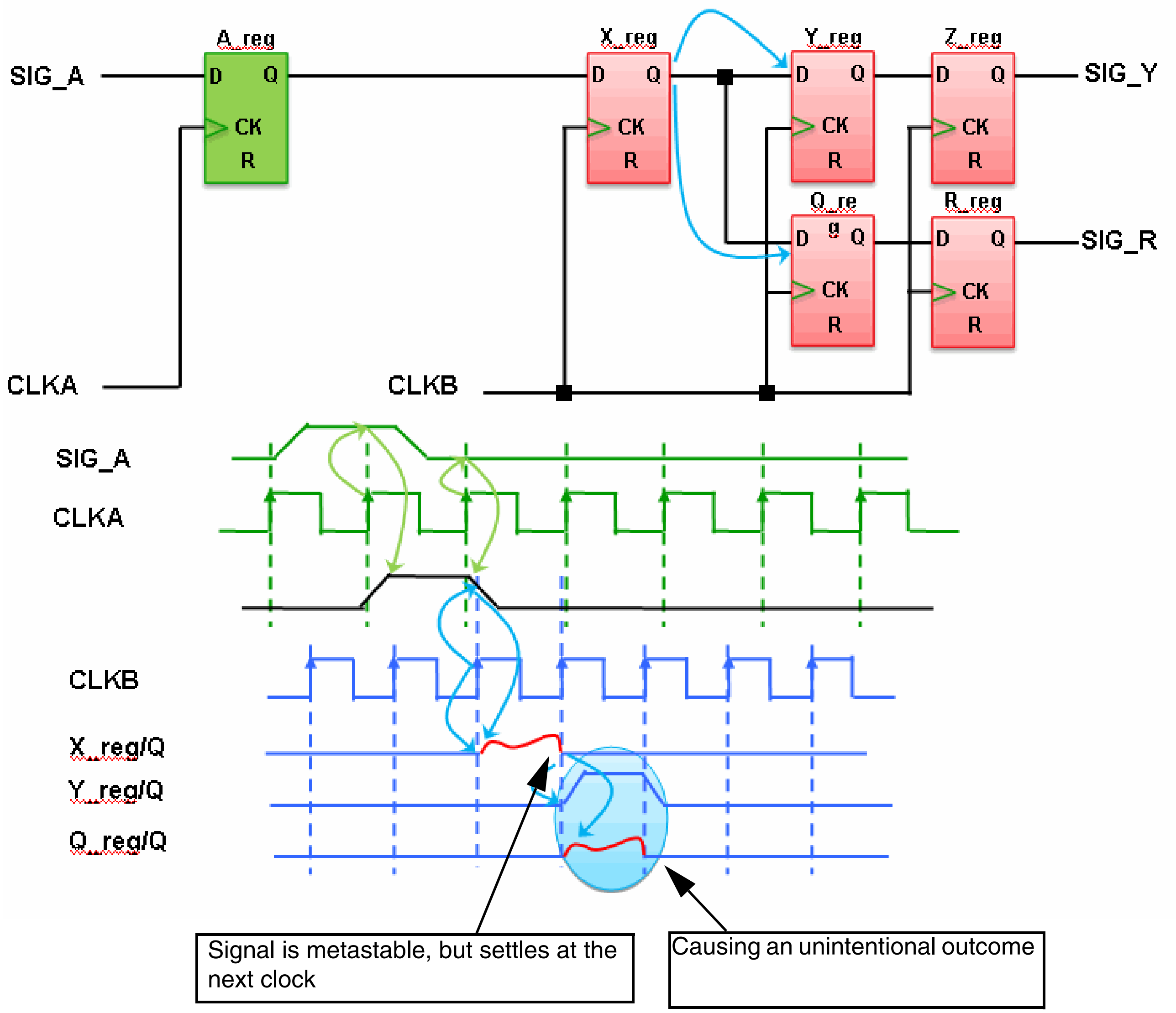
When there are setup and hold-time violations in any flip-flop, the signal goes into a non-deterministic state (or, *metastable* state). Eventually, the signal settles into a known state (`1` or `0`). This process is known as *metastability.*

You cannot prevent metastability in asynchronous designs, but you can prevent signals that are in a metastable state from propagating forward using *synchronizers*. See “Synchronizers and Schemes” on page 180.

Synchronizers can only help prevent metastable signals from propagating forward. However, there are other problems in asynchronous designs that need to be addressed:

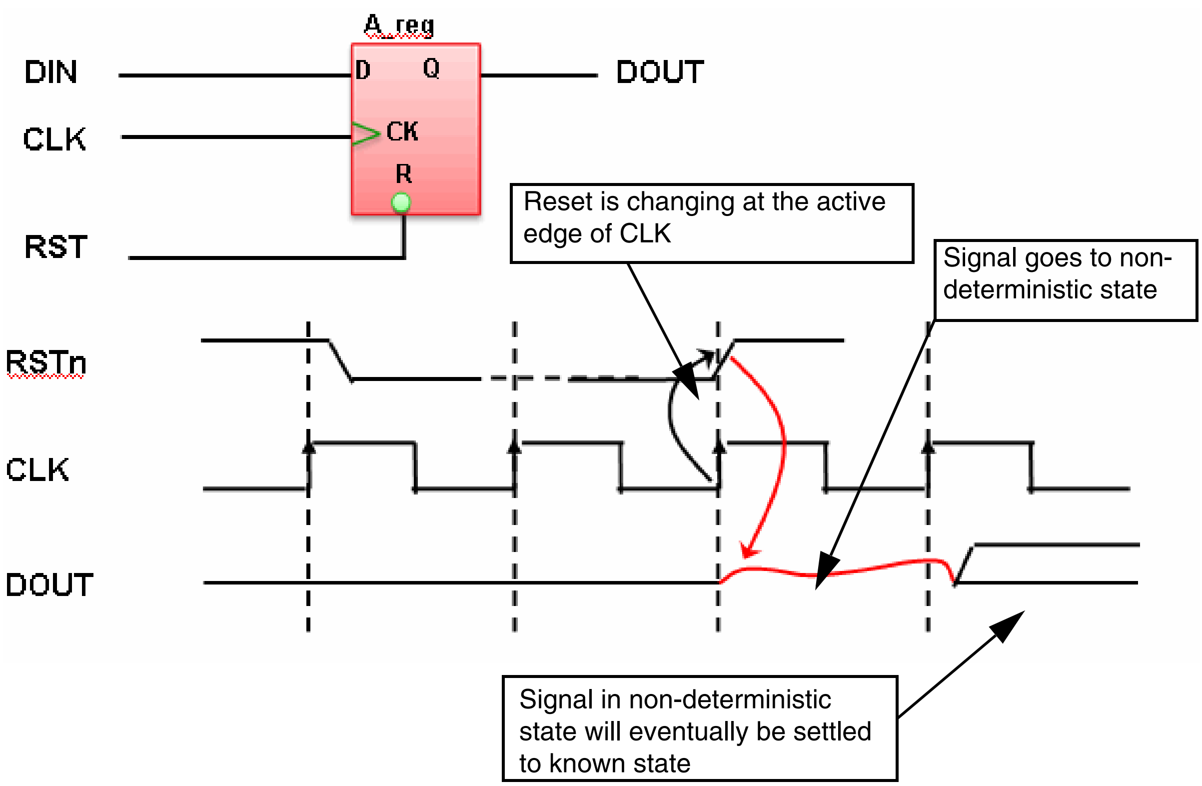
- *Propagation of unintentional values to receiver.*

Figure 11-2 Unintentional Value Propagation



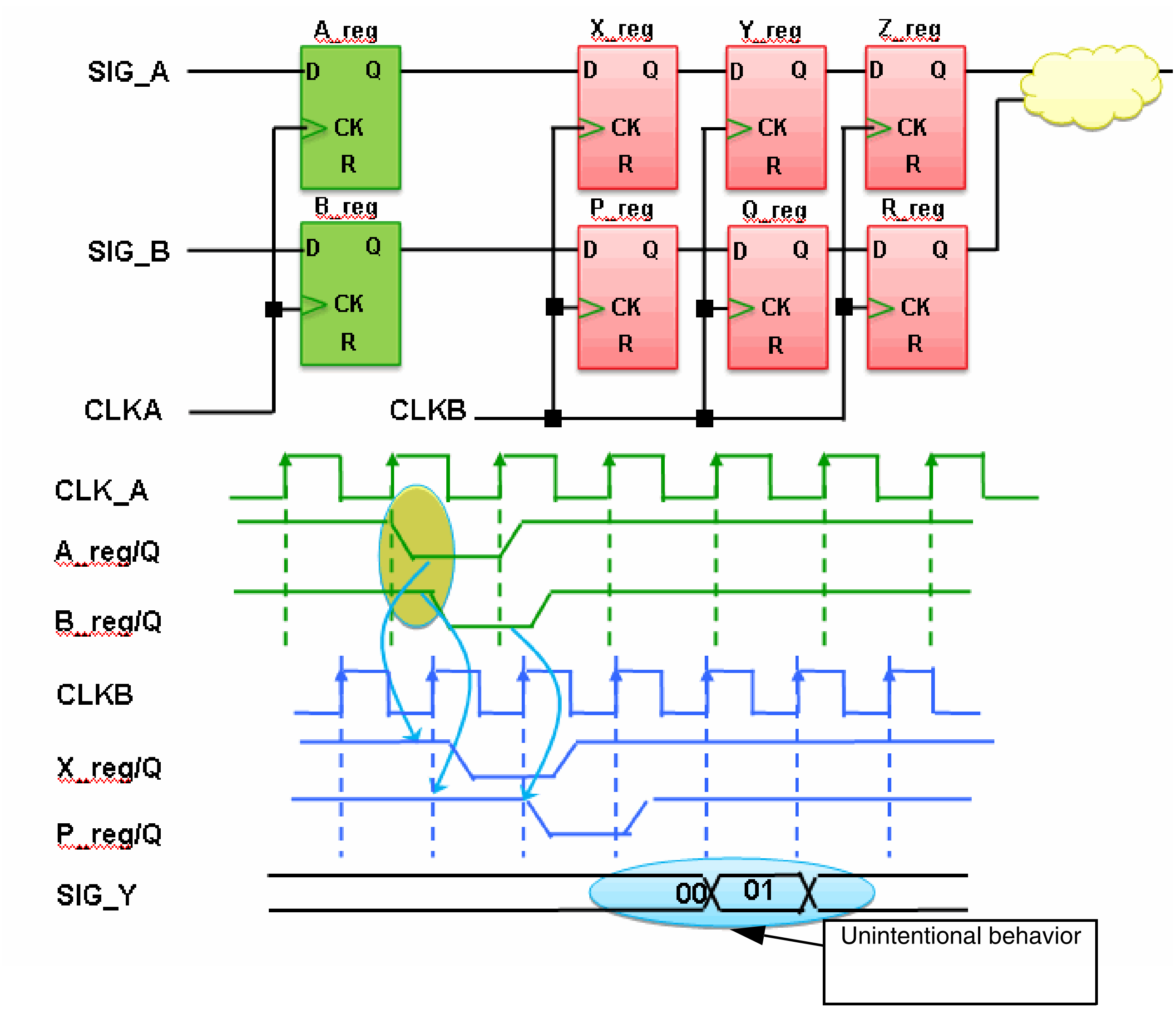
- *Set/reset synchronization*—Set/reset is asynchronous to most—if not all— the clocks in the design, thus making it a control signal to all the sequential elements in the design. When set/reset is asynchronous to all the clocks in the design, timing requirements for asynchronous paths can be violated at any time, unless there are appropriate design structures in place.

Figure 11-3 Set/Reset



- *Convergence* in the receiving domain.

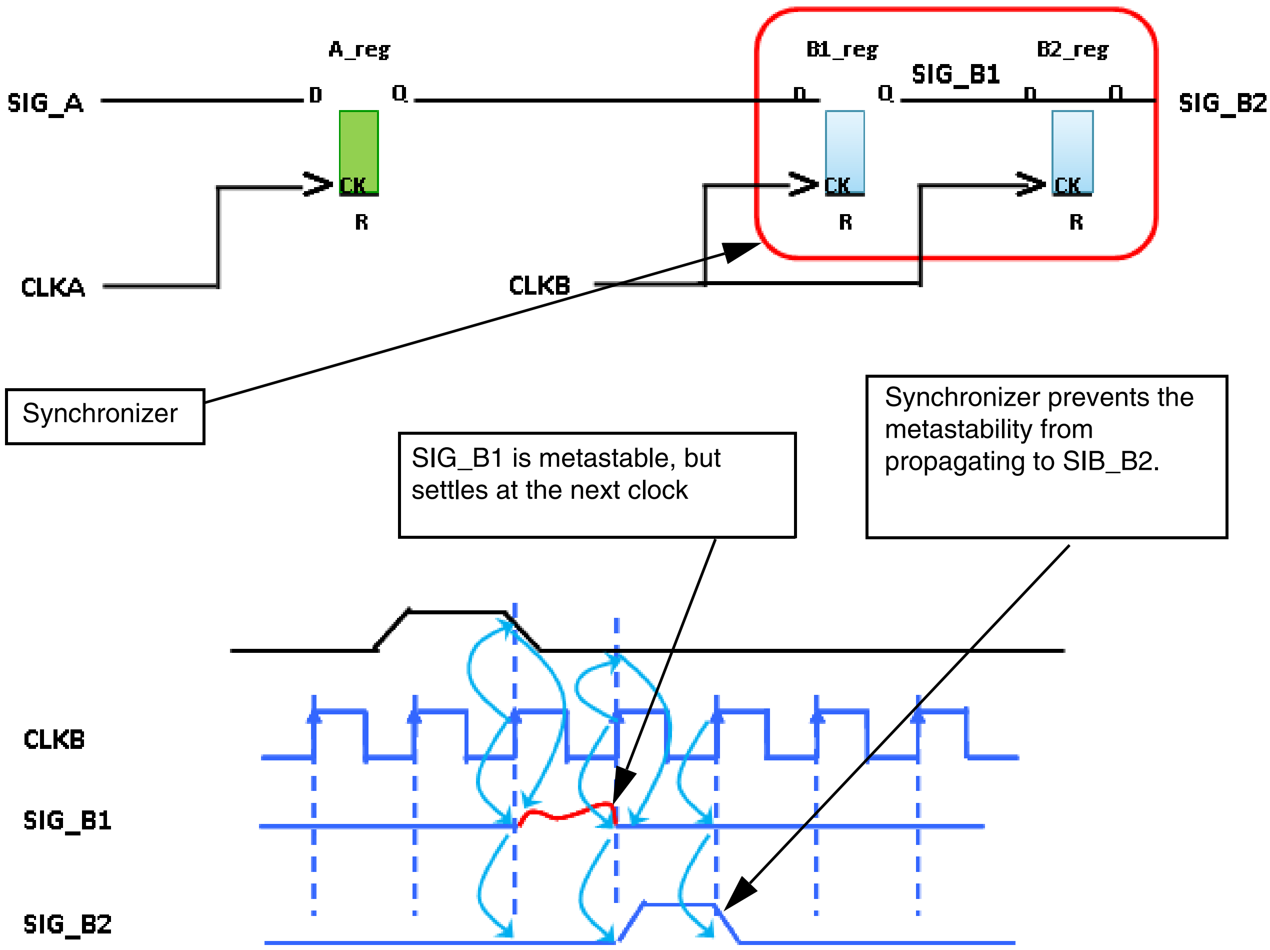
Figure 11-4 Convergence



To verify that the data is transferred consistently and reliably across clock domains, Conformal Constraint Designer supports CDC checks (See "Types of CDC Checks" on page 186).

Synchronizers and Schemes

In asynchronous designs, mestability cannot be prevented, however you can use synchronizers to prevent signals that are in a metastable state from propagating forward.



The following figures illustrate the most commonly used synchronization schemes for handling CDC signals.

- "Flip-Flop Synchronization Scheme" on page 181
- "MUX Synchronization Scheme" on page 182
- "FIFO Synchronization Scheme" on page 183
- "Set/Reset Synchronization Scheme" on page 184
- "Set/Reset to Multiple Domains" on page 185

Figure 11-5 Flip-Flop Synchronization Scheme

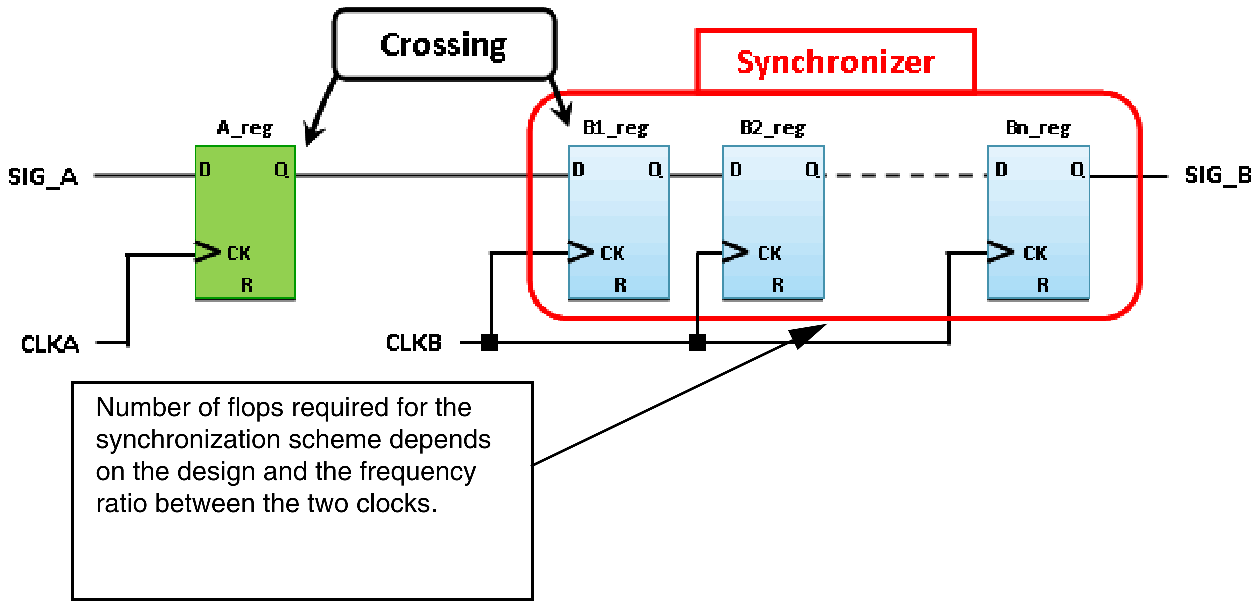


Figure 11-6 MUX Synchronization Scheme

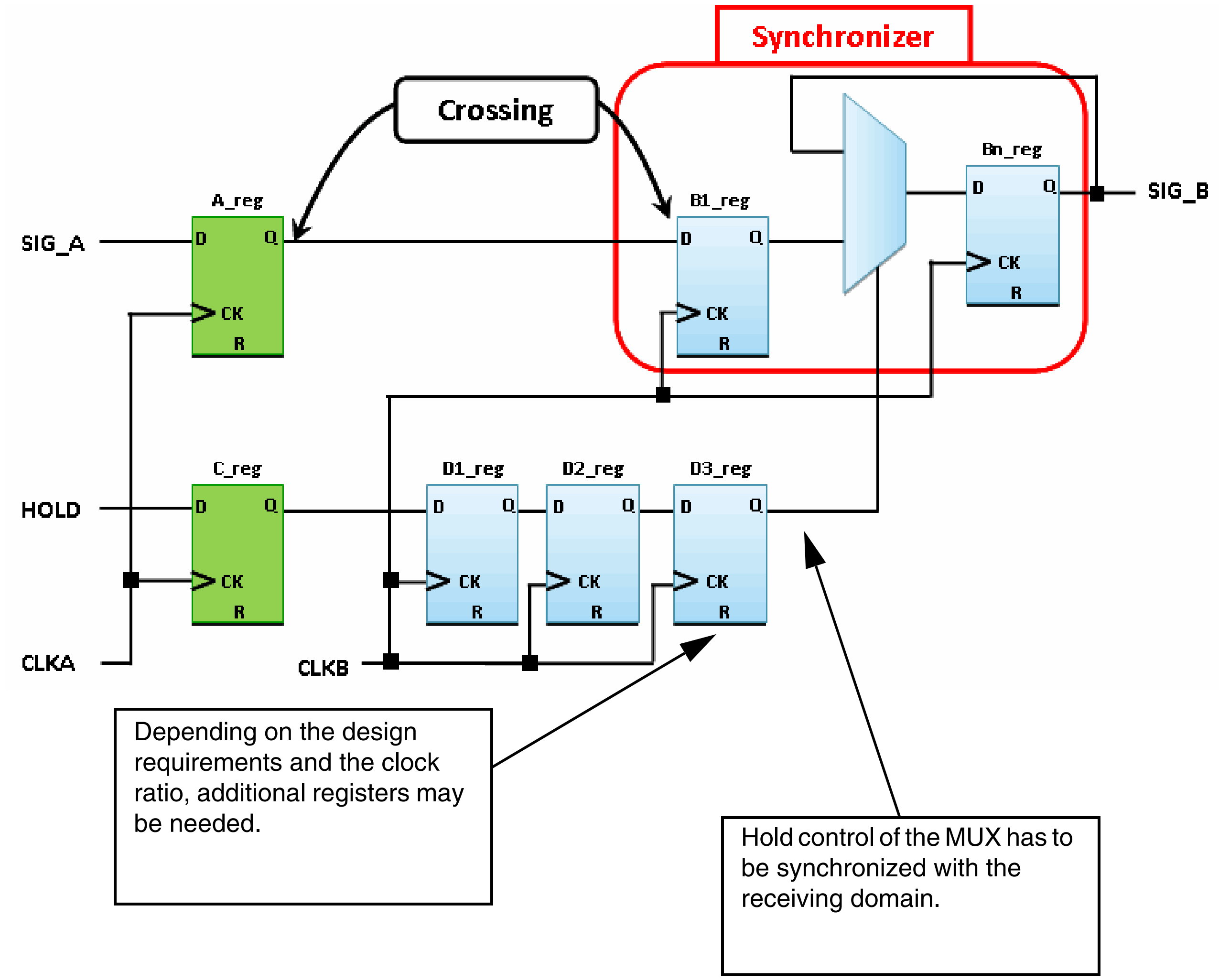


Figure 11-7 FIFO Synchronization Scheme

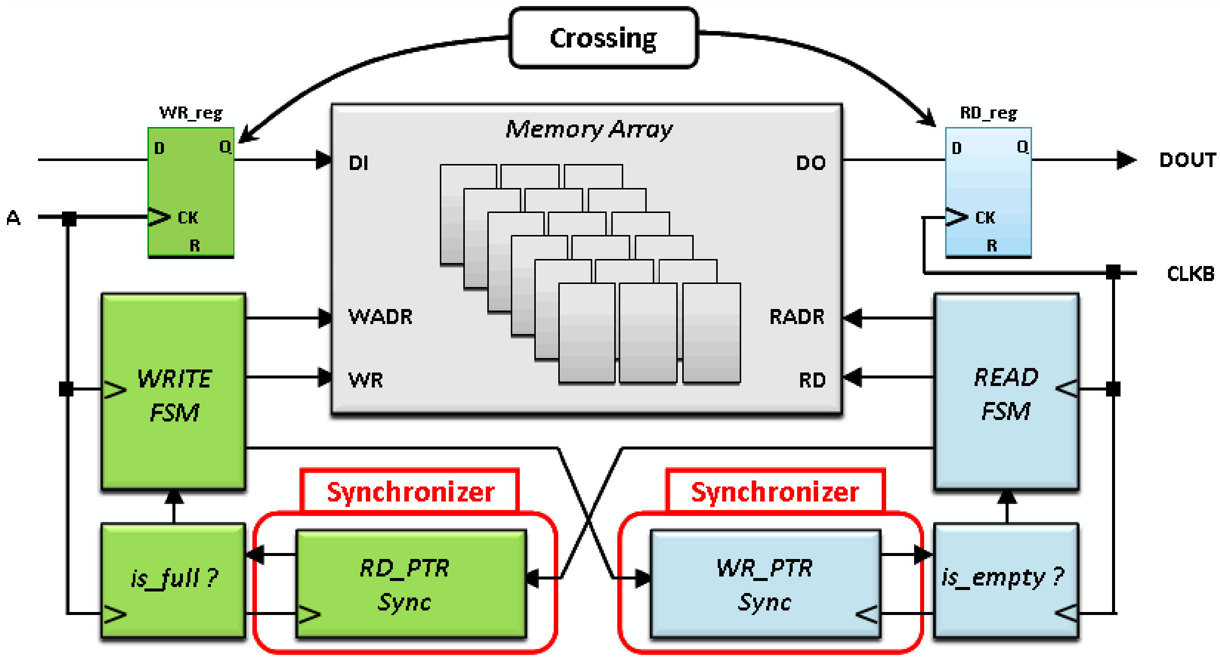


Figure 11-8 Set/Reset Synchronization Scheme

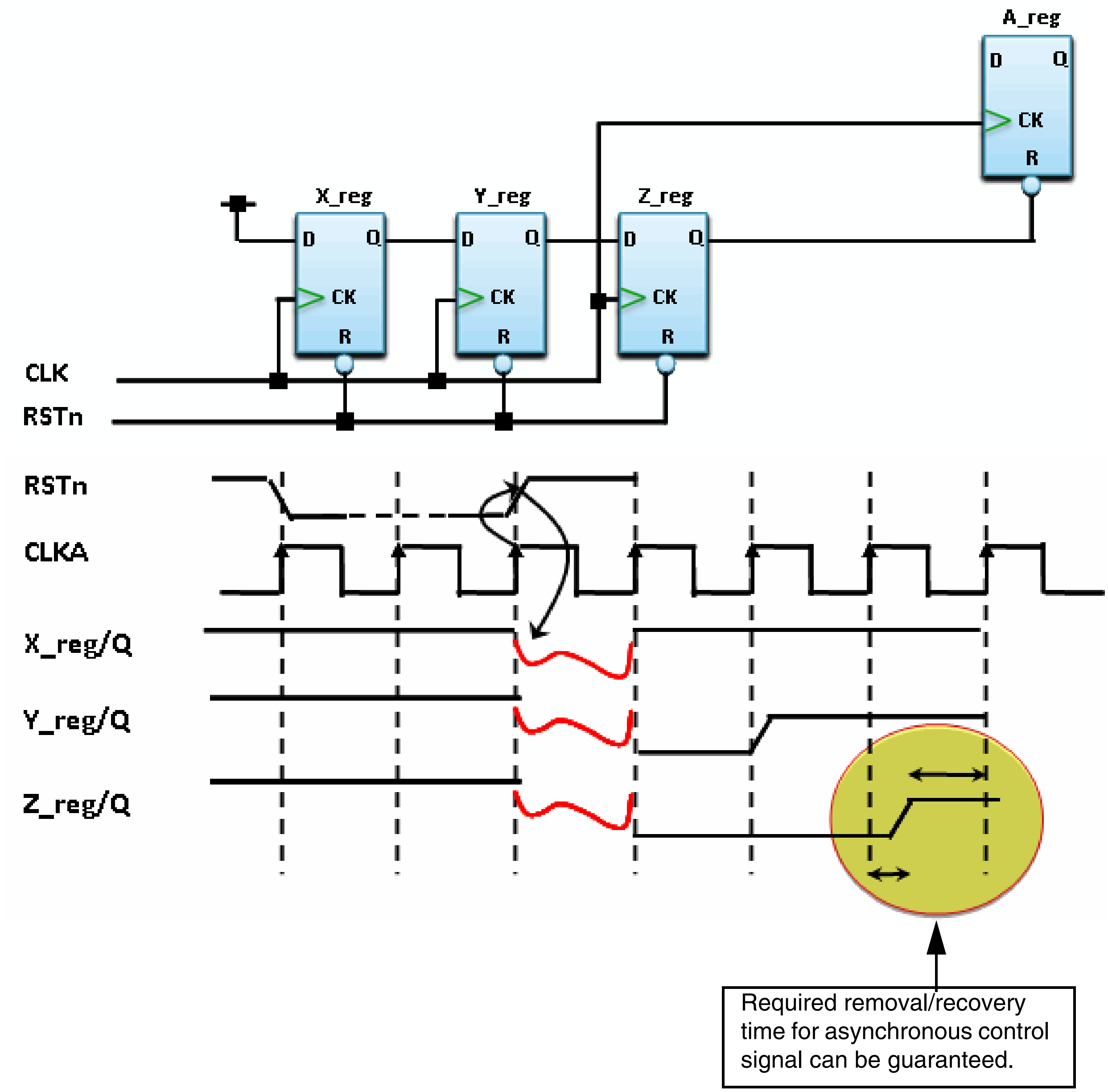
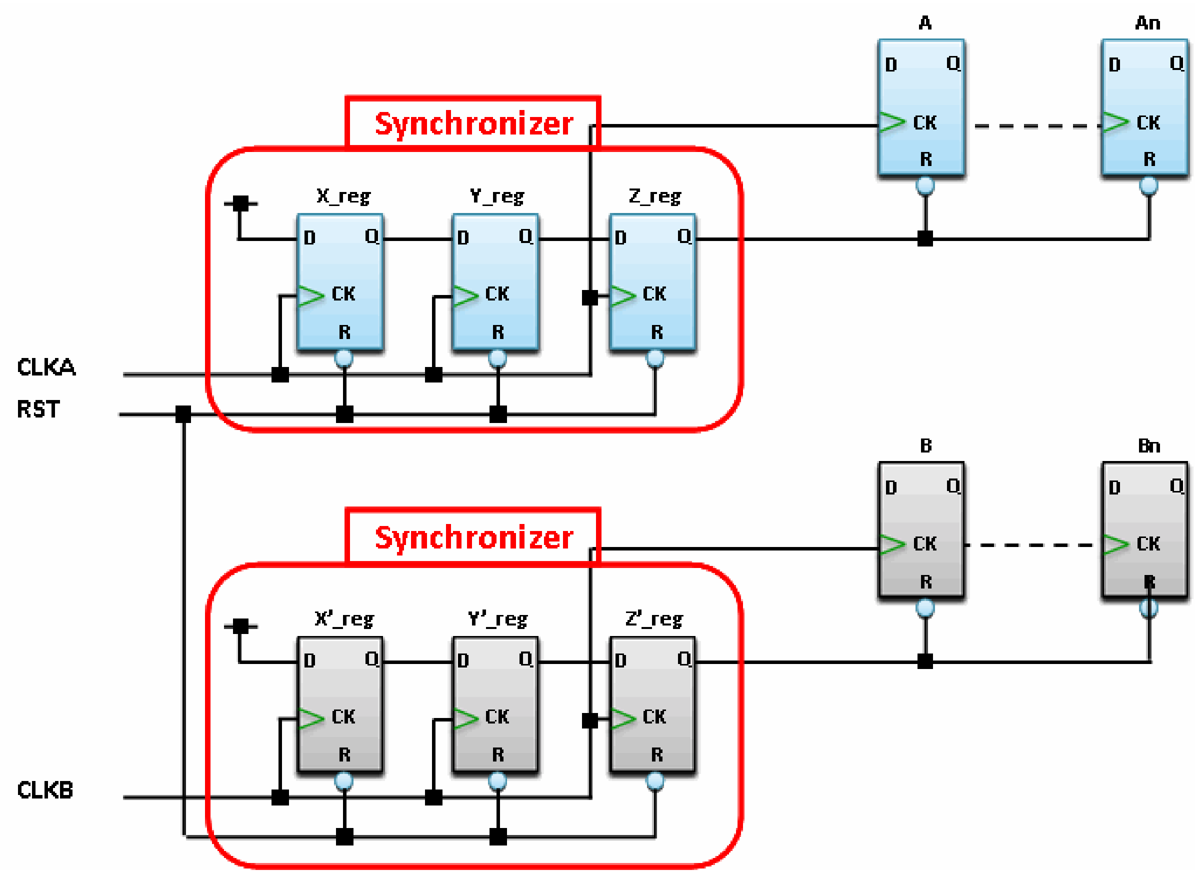


Figure 11-9 Set/Reset to Multiple Domains



Types of CDC Checks

CDC checks fall into two categories:

- *Structural* Clock Domain Crossing checks (CDC) verify that the design has the correct structure. These checks validate an RTL or gate design's clocking schemes, and are structural in nature (that is, they are not static timing checks).

  Conformal Constraint Designer extracts the clock domains of the circuit and performs Structural CDC checks. A violation occurs when information or data from one clock domain depends on the information or data from another clock domain without proper synchronization. Conformal Constraint Designer supports the following synchronizers, which promote proper synchronization, to prevent meta-stability problems:

  - D flip-flop synchronizer
  - Multiplexer synchronizer
  - Module synchronizer

  Before executing Structural CDC checks, define clocks that accurately reflect design intent. Conformal Constraint Designer supports considerable user control over defining what constitutes a clock domain. Conformal Constraint Designer uses the definitions to determine the clock domains for each state element.

- *Functional* CDC checks complement and build on structural CDC checks; these checks functionally validate the behavior of structurally-validated CDC paths between source and destination clocks.

  Functional CDC checks are formal proofs and are used to verify that source clock domain data is held long enough to be captured correctly by the receiving clock domain. Functional checks also verify single bit changes for vectors that cross the domains.

CDC Rule Set

CDC rule checks are grouped into a predefined rule set called `cdc_def_rs`. To add the CDC default rule set, use the "`add rule set -file ccd_default_cdc_ruleset.tcl`" command.

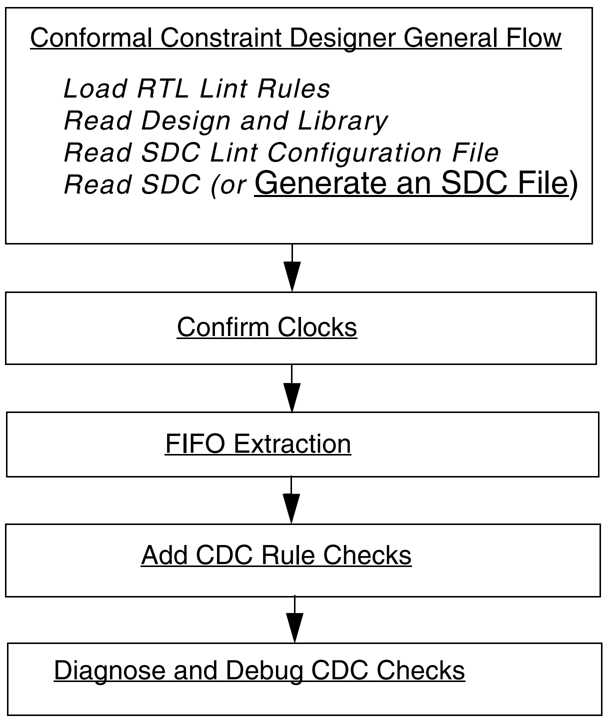
The CDC rule set contains the following rule groups (these checks include both structural and functional modes):

- `cdc_checks`—Structurally and functionally verifies that each crossing between the source and destination clocks is correctly synchronized.
- `conv_checks`—Structurally and functionally verifies that the CDCs are not converging at the destination domain.
- `set_rst_checks`—Structurally verifies that the set/reset drivers are synchronized to the destination domain.
- `sr_sync_crossing_check`—Structurally and functionally verifies that each set/reset crossing between two registers is correctly synchronized

Tip

To see what rules are included in a predefined rule set, you can use the `REPORT RULE SET` command.

Clock Domain Crossing Checks Flow



CDC Flow Steps

The following sections illustrate the steps in the CDC flow using command and GUI examples.

- Generate an SDC File on page 189
- Confirm Clocks on page 190
- FIFO Extraction on page 192
- Add CDC Rule Checks on page 199
- Configure a CDC Rule Check on page 199 (optional)
- Run CDC Structural Checks on page 199
- Run CDC Functional Checks on page 200
- Diagnose and Debug CDC Checks on page 201

Important

Perform CDC checks only *after* both Lint Rule Checks and Policy Rules have been performed on the SDC constraints. CDC checks are performed based on the extraction of clock domains from the SDC constraint files; if your SDC constraints are inaccurate or incomplete, you run the risk of insufficiently checking your designs' CDC crossings.

Generate an SDC File

In the clock domain crossing flow, you must read in an SDC file to establish the clock groups. However, you might not have an SDC file readily available. Conformal CD has an SDC Generation capability that will generate a CDC-flow ready SDC file that: generates clock definitions, input delays, output delays, and set case analysis statements (commented out).

To generate an SDC file from the command line:

1. Read in the design.
2. Set the system mode to Verify.
3. Enter Tcl mode.
4. Use the `gen_sdc_template` Tcl utility to generate the SDC file.
5. Review the contents of the generated SDC file to verify correctness. Specifically, modify the clock waveforms for clock domain crossing functional checks.

For example:

```
read design test1.v
set system mode verify
tclmode
package require ccd_stg
::ccd_stg::gen_sdc_template  -cdc -output out.sdc
```

In the next tool iteration you can use the generated SDC to run the CDC checks. For example:

```
read design test1.v
read sdc out.sdc
set system mode verify
commit clock
add rule set -file ccd_default_cdc_ruleset.tcl
run rule check cdc_def_rs/*
```

Confirm Clocks

Note: Before you check the quality of your SDC and design files, ensure that your clock groups and clock associations are correct as clocks are derived from the SDC file.

Confirming Clock Groups

From the command line:

```
setenv SDC /user1/design/constraints/
read sdc  $SDC/top.sdc  -replace
```

The following illustrates the sample output from these commands:

```
Statistics for commands executed by read SDC:
--------------------------------------------
get_clocks                    - Successful:   5   Failed:    0   Total:    5
get_ports                     - Successful:   7   Failed:    2   Total:    9
create_clock                  - Successful:   4   Failed:    0   Total:    4
set_input_delay               - Successful:   2   Failed:    2   Total:    4
set_output_delay              - Successful:   1   Failed:    0   Total:    1
```

The `COMMIT CLOCK` command approves existing clock groups. If the SDC file does not contain any clock grouping instructions, then Conformal CD uses the default grouping.

From the command line:

```
set system mode verify
report clock groups
commit clock
```

The following illustrates the sample output from these commands:

```
==============================================================================
=                               Clock Groups                                 =
==============================================================================
  name: CK1 (system)
  number of clocks: 1
  Clocks: CK1
------------------------------------------------------------------------------
  name: CK2 (system)
  number of clocks: 1
  Clocks: CK2
==============================================================================
```

```
TCL_VERIFY> commit_clock

Following clock groups are created:
----------------------------------
CK1                                  (system) :    1 clock
CK2                                  (system) :    1 clock
```

In Conformal Constraint Designer, once the SDC constraints are read in and all necessary clocks are appropriately defined, clocks are automatically propagated. By default, all clocks are propagated across combinational gates. In order to customize clock propagation or prevent clocks from propagating through, you can use the SDC command `set_clock_sense` to configure how you want clocks propagated.

You should use `REPORT CLOCK` and `REPORT CLOCK GROUPS` to evaluate the clocks and clocks group assignments in the design. Once you evaluate the assignments, use the `COMMIT CLOCK` command before you perform CDC checks.

For more information on checking clock groups, refer to "Checking Clock Groups" on page 90.

[Go back to the Clock Domain Crossing Checks Flow.]

Clock Associations

Before CDC checks are performed, each object in the design needs to be associated with a clock domain. The association is done when the SDC constraints are read in:

- Registers get the association from `create_clock` commands
- Input/output ports get the association from the `set_input/output_delay` commands
- RAM/ROM blocks get the association from the `.lib` models

Unclocked objects belong to the unclocked domain. The CDC path is still checked but the result will be saved in the *clocked->unclocked* or *unclocked->clocked* instance.

FIFO Extraction

First In First Out (FIFO) synchronizers are typically used for domain crossovers, passing data from one clock domain to another asynchronous clock domain to help buffer the data in a RAM or a register file.

Extracting FIFO information can help reduce the number of false errors generated by the tool for designs that contain FIFO instantiations.

Extracting FIFO Information

1. Extract FIFO Information

   ❑ To extract default (or automatically inferred) FIFOs:

   ```
   add fifo instance -default
   ```

   This command runs FIFO extraction for every two-dimensional memory component in all modules. In the `REPORT FIFO INSTANCE` reports, these FIFOs are labeled “System” FIFOs.

   ❑ (Optional) To extract user-defined FIFOs:

   ```
   add fifo instance ...
   ```

   In the `REPORT FIFO INSTANCE` reports, these FIFOs are labeled “User” FIFOs. Note that user-defined FIFOs (regardless of whether they have passed all atomic checks) are considered valid FIFO crossings for CDC structural checking.

Reporting FIFO Status

Use the `REPORT FIFO INSTANCE` command to identify the status of the FIFO instances that are in the current session. You can also use this command to determine whether a FIFO instance is automatically inferred (System) or user manually added (User), and lists any checks performed on the FIFO instance.

- `REPORT FIFO INSTANCE -PASS` (or just `REPORT FIFO INSTANCE`) reports all valid FIFO crossings.

  This includes default FIFOs with a status of pass, and all user-defined FIFOs with a status of pass.

- `REPORT FIFO INSTANCE -FAIL` reports all default FIFOs with a status of fail.

You can then use this information to determine the atomic checks that cause the particular FIFO instance to fail. See "Specify/update Parameters of FIFO Instances." on page 193 to update the FIFO instance based on this information. Note: Once you edit/ modify a FIFO, it then becomes a user-defined FIFO.

- `REPORT FIFO INSTANCE -cdc` reports all FIFOs whose information will be used for CDC structural checking.

See "Sample Report" on page 194.

Specify/update Parameters of FIFO Instances.

Use the `set_attribute` command to specify FIFO instance paramters/attributes. The "FIFO Objects" section of the *Conformal Constraint Designer Attribute Reference* describes these attributes.

Validate and Commit FIFO Instances

1. After specifying/updating the parameters of the FIFOs, use the `CHECK FIFO INSTANCE` command to check if the updated components are valid for the given FIFO criteria.

   Tip: You can use the `WRITE FIFO INSTANCE` command to write out all passing FIFOs (default and user-defined FIFOs) to a file for future use.

2. The `COMMIT FIFO` command approves existing FIFO instances. If you do not issue this command, the FIFO information will not be used for CDC structural checks.

   Note: For CDC structural checking, all user-defined FIFOs (regardless of whether they have a status of pass or fail) are considered valid FIFO crossings. To see all the FIFOs whose information will be used for CDC structural checking, use the `REPORT FIFO INSTANCE -cdc` command.

Sample Report

For example, given the following sample dofile:

```
add_fifo_instance -default
add_fifo_instance user1
set_attribute [find -fifo user1] memory [find -instance {fifo5_1/ram_ff_reg*}]
add_fifo_instance user2
set_attribute [find -fifo user2] memory [find -instance {fifo5_2/ram_ff_reg*}]
check_fifo_instance
report_fifo_instance
report_fifo_instance -cdc
commit_fifo
```

You would get the following results:

```
// Command: add_fifo_instance -default
================================================================================
=                                 FIFO Report                                  =
================================================================================
Status: Pass
Number of FIFOs: 1
----------------------------------------------------------------------------
Fifo1                                   (system)
--------------------------------------------------------------------------------
Status: Fail
Number of FIFOs: 0
================================================================================
// Command: add_fifo_instance user1
// Command: set_attribute [find -fifo user1] memory [find -instance {fifo5_1/
// ram_ff_reg*}]
// Command: add_fifo_instance user2
// Command: set_attribute [find -fifo user2] memory [find -instance {fifo5_2/
//ram_ff_reg*}]
// Command: check_fifo_instance

================================================================================
=                                 FIFO Report                                  =
================================================================================
Status: Pass
Number of FIFOs: 2
----------------------------------------------------------------------------
Fifo1                                   (system)
user1                                   (user)

--------------------------------------------------------------------------------
Status: Fail
Number of FIFOs: 1
----------------------------------------------------------------------------
user2                                   (user)
================================================================================
// Command: report_fifo_instance

================================================================================
=                                 FIFO Report                                  =
================================================================================
Status: Pass
Number of FIFOs: 2
----------------------------------------------------------------------------
Fifo1                                   (system)
```

```
user1                                     (user)
================================================================================
// Command: report_fifo_instance -cdc
================================================================================
=                                  FIFO Report                                 =
================================================================================
Status: Pass
Number of FIFOs: 2
----------------------------------------------------------------------------
Fifo1                                     (system)
user1                                     (user)
--------------------------------------------------------------------------------
Status: Fail
Number of FIFOs: 1
----------------------------------------------------------------------------
user2                                     (user)
================================================================================
// Command: commit_fifo

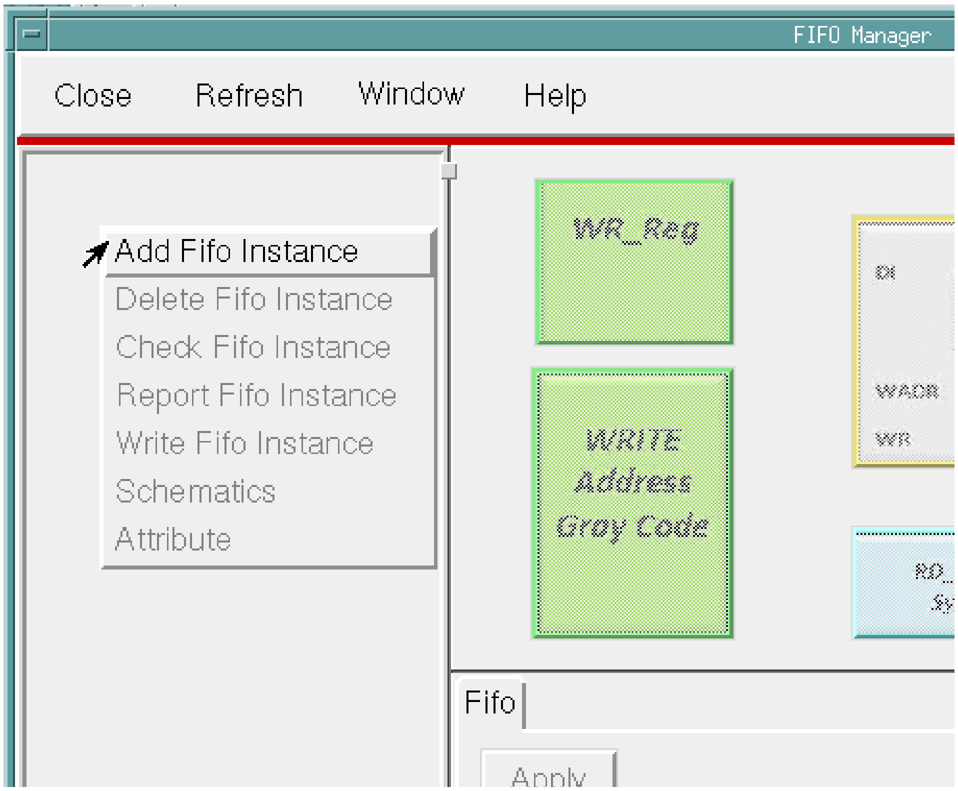
================================================================================
=                                  FIFO Report                                 =
================================================================================
Status: Pass
Number of FIFOs: 2
----------------------------------------------------------------------------
Fifo1                                     (system)
user1                                     (user)
--------------------------------------------------------------------------------
Status: Fail
Number of FIFOs: 1
----------------------------------------------------------------------------
user2                                     (user)
================================================================================
```

Using the FIFO Manager

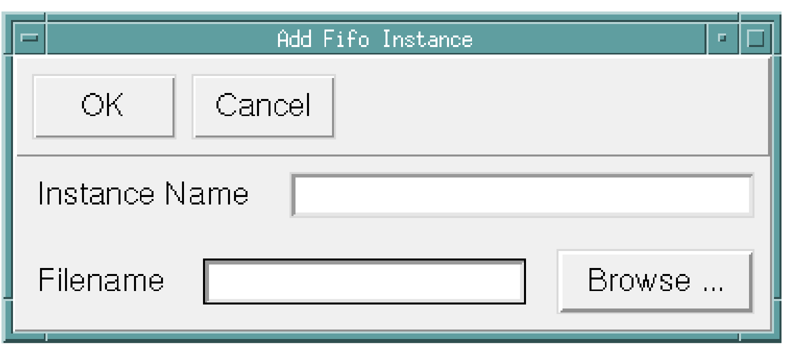
If you are in the GUI mode, you can enable FIFO extraction from the FIFO manager:

1. Choose *Tools - FIFO Manager* from the main Conformal CD window.

2. Right-click within the window and choose *Add Fifo Instance*.

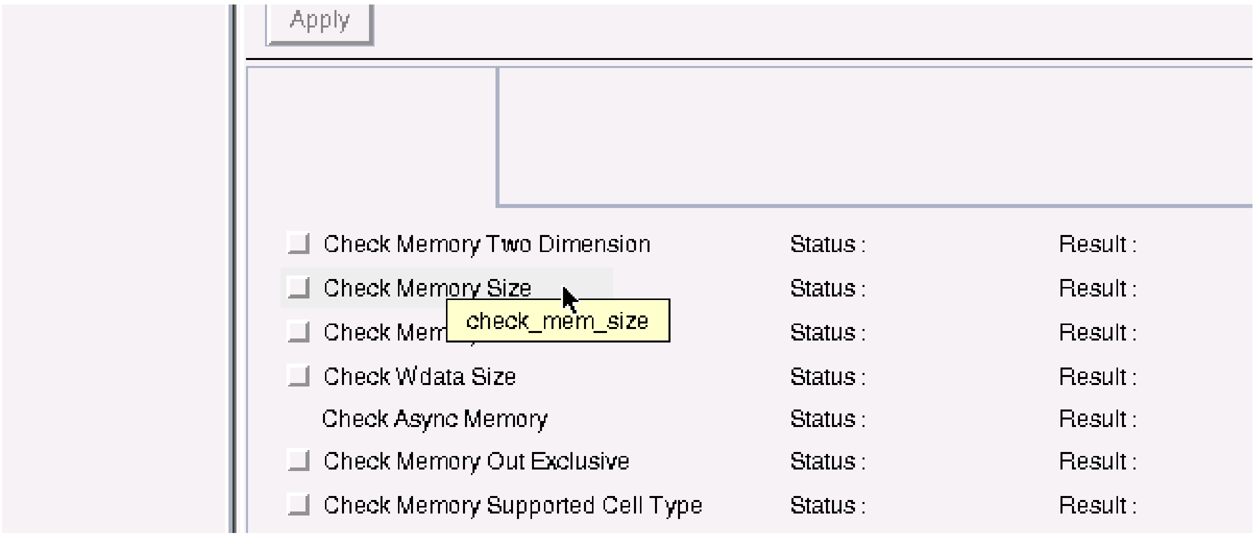


3. If you are manually defining a FIFO instance, in the dialog that displays, enter the instance name or specify a file to source that contains the FIFO definitions.

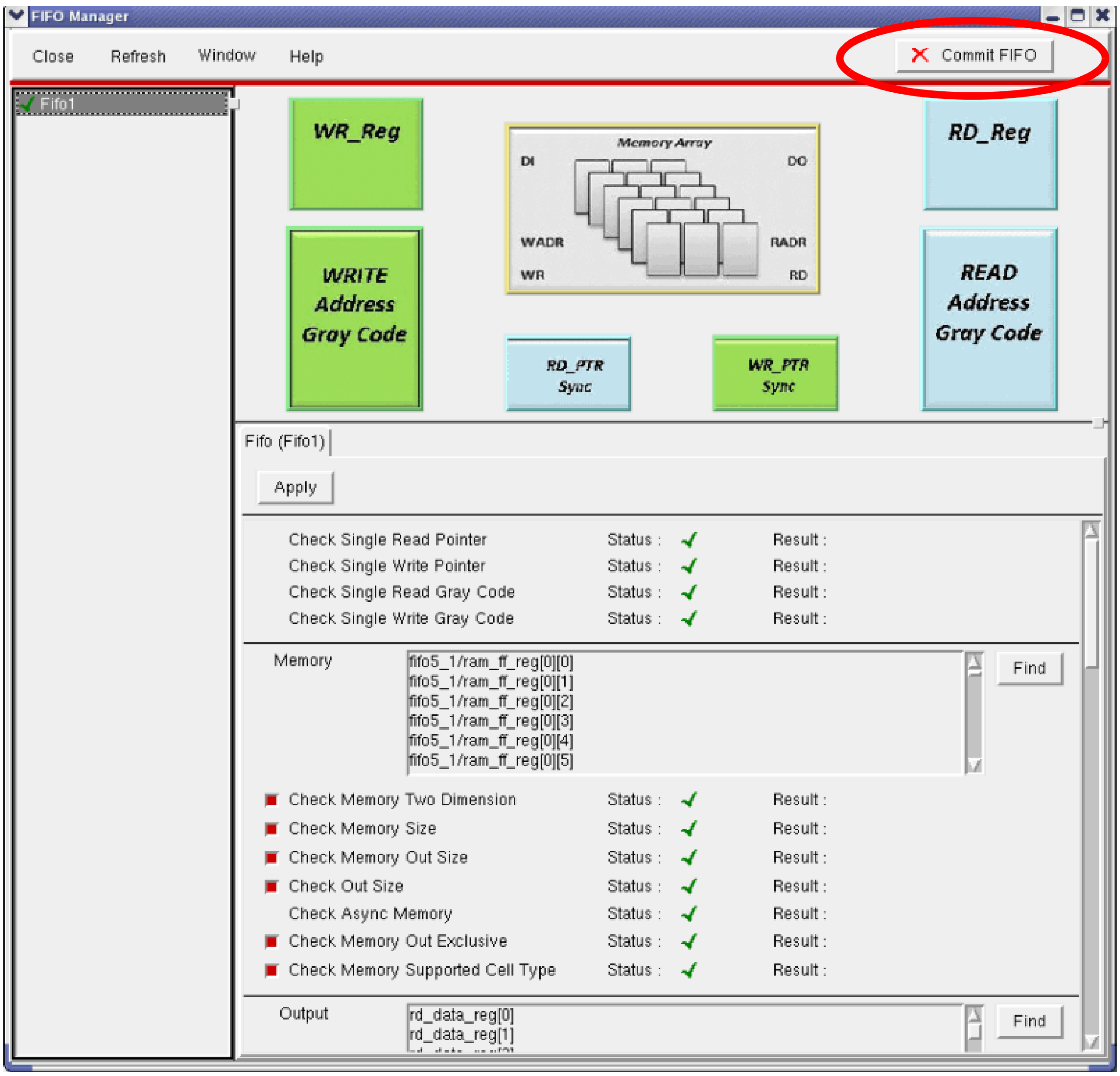


4. From there, you can specify/update the checks performed for the FIFO, and then use the *Check Fifo Instance* right-click menu option to check if the updated components are valid for the given FIFO criteria.

   Note: You can hover over a check to view the FIFO attribute name, which is settable through the `set_attribute` command.



5. Click on Commit to approve existing FIFO instances. If you do not issue this command, the FIFO information will not be used for CDC structural checks.



[Go back to the Clock Domain Crossing Checks Flow.]

Add CDC Rule Checks

To run CDC rule checks, you must first add the predefined rule set that includes all fo the CDC rule checks to your session using the following command:

`add_rule_set -file ccd_default_cdc_ruleset.ntcl`

or

`add rule set -file ccd_default_cdc_ruleset.tcl`

Configure a CDC Rule Check

Each CDC rule check has a set of rule attributes, which can be configured (or edited) to meet methodology requirements. You can configure rule attributes in two ways:

- At the **rule-source level**—Use this method if you want to edit the attribute and have the changes affect all the rule instances of the rule source. You must make these changes ***before*** you add a rule set or a rule instance.

  To configure a rule source, use the following format:

  ```
  set_attribute <rulesrc><option_name><option_value>
  ```

- At the **rule-instance level**—Configuring at this level affects only the rule instance. Configure rule attributes at the rule instance level ***after*** you add a rule set or instantiate a rule source.

  To do this, use the following format:

  ```
  set_attribute <ruleinst><option_name><option_value>
  ```

  For example:

  ```
  set_attribute [find -ruleinst cdc_def_rs/*/*] \
      source_clock [find -sdcobj <clk1>]
  ```

For a list of the attributes for each rule check, refer to the *Conformal Constraint Designer Rule Check Reference*.

Run CDC Structural Checks

Before you can run CDC functional checks, you must run the CDC structural checks and fix any failed CDC paths.

Important

You cannot run CDC functional checks if any of the structural checks have failed.

To run the CDC structural checks and identify the failed checks:

```
run rule check cdc_def_rs
report rule check cdc_def_rs -verbose -status fail
```

Run CDC Functional Checks

After all of the structural checks have passed, enable functional checks using the `set_attribute` command and re-run the `cdc_def_rs` rule set:

```
tclmode
set_attribute [find -ruleinst cdc_def_rs/*/*] analysis_mode "functional"
vpxmode
run rule check cdc_def_rs
report rule check cdc_def_rs -verbose -status fail
```

Related Topics

- For information on the CDC rule set and its applicable rule groups and rule instances, refer to the "Clock Domain Crossing Rule Checks" chapter of the *Conformal Constraint Designer Rule Reference*.
- For information on the Conformal attributes, refer to the *Conformal Constraint Designer Attribute Reference*.
- For information on rule sets (using predefined rule sets or creating rule sets), refer to "Rule Check Flow" on page 147.
- For information on viewing the contents of a rule set or the status of rule checks from the GUI, see "Rule Manager" on page 228.
- For information on the command entry modes (vpxmode and Tcl mode), see "Command Entry Modes" on page 64.

[Go back to the Clock Domain Crossing Checks Flow.]

Diagnose and Debug CDC Checks

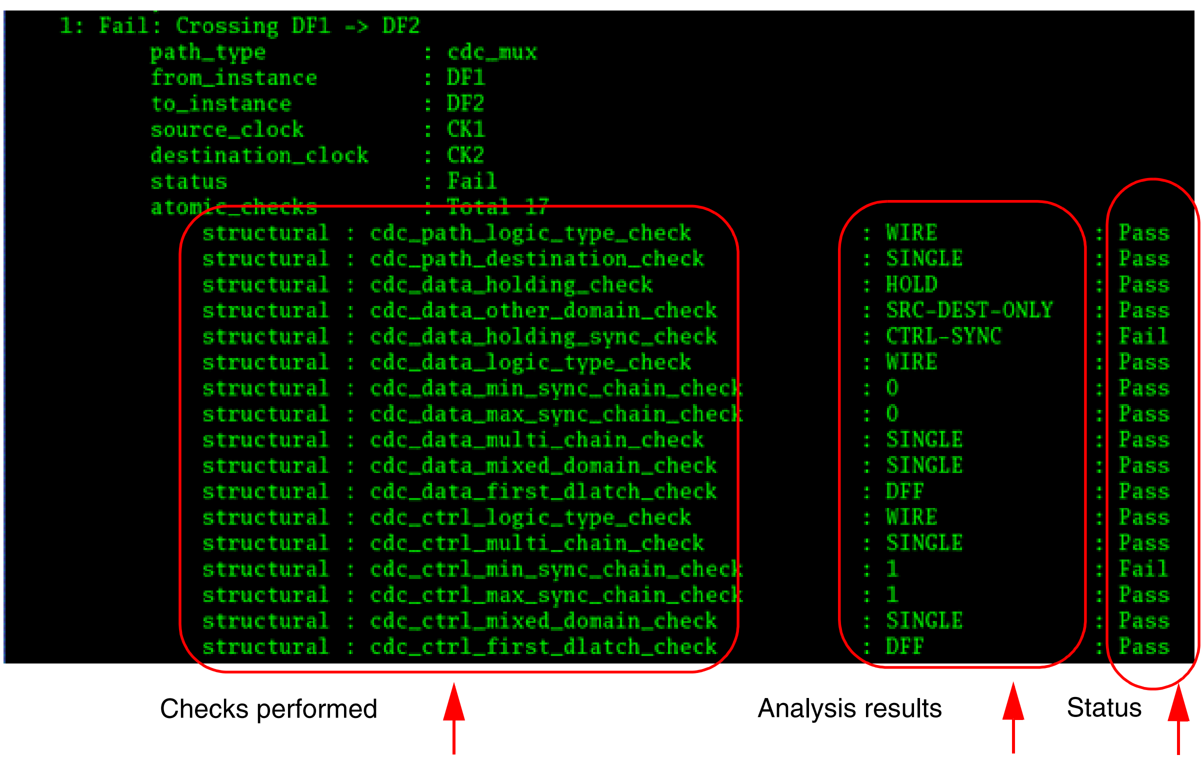
- Reporting on a Rule Instance on page 201
- Diagnose CDC Violations on page 205
- Customizing Clock Domain Crossing Analysis on page 209

Reporting on a Rule Instance

Use the following command to report on a particular rule instance:

```
report rule check cdc_def_rs/cdc_CLKA->CLKB //Reports on a rule instance
```

The tool reports the status of the crossings for that rule instance (as shown in the following figure):



The report displays the following for the failed crossing:

- `path_type`
- `from_instance`

- `to_instance` (for `conv_*` checks, the tool reports the `convergence_end_point`)
- `source_clock`
- `destination_clock`
- `status`—Pass or Fail
- `atomic_checks`—This section lists the total number of atomic checks performed for this crossing, and the details for the atomic checks:
  - Lists the type of atomic check (structural or functional)
  - Atomic check name (for example, `cdc_path_logic_type_check`)
  - Current value—Lists the current value (in the design) of the attribute that is proven by this atomic check.

    For example, `cdc_ctrl_min_sync_chain_check` checks proves the `dff_sync_scheme` attribute, which sets the minimum number of flops required in the sync chain. The report in this example lists that the check failed and the current value is 1 (the synch chain has only 1 flop). This means that there are not enough flops in the design.

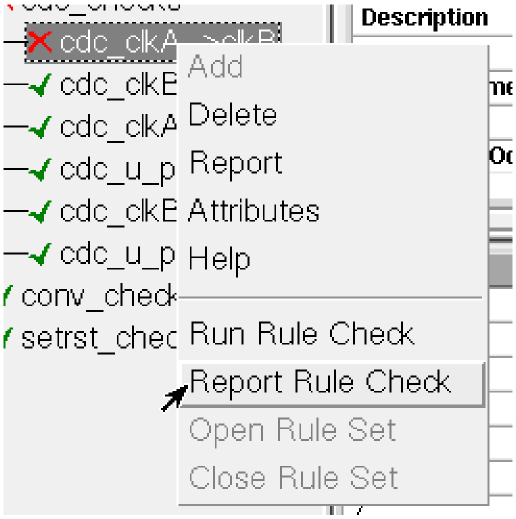
    To specify the minimum number of DFFs required in the sync chain:

    set_attribute <ruleinst> dff_sync_scheme min #

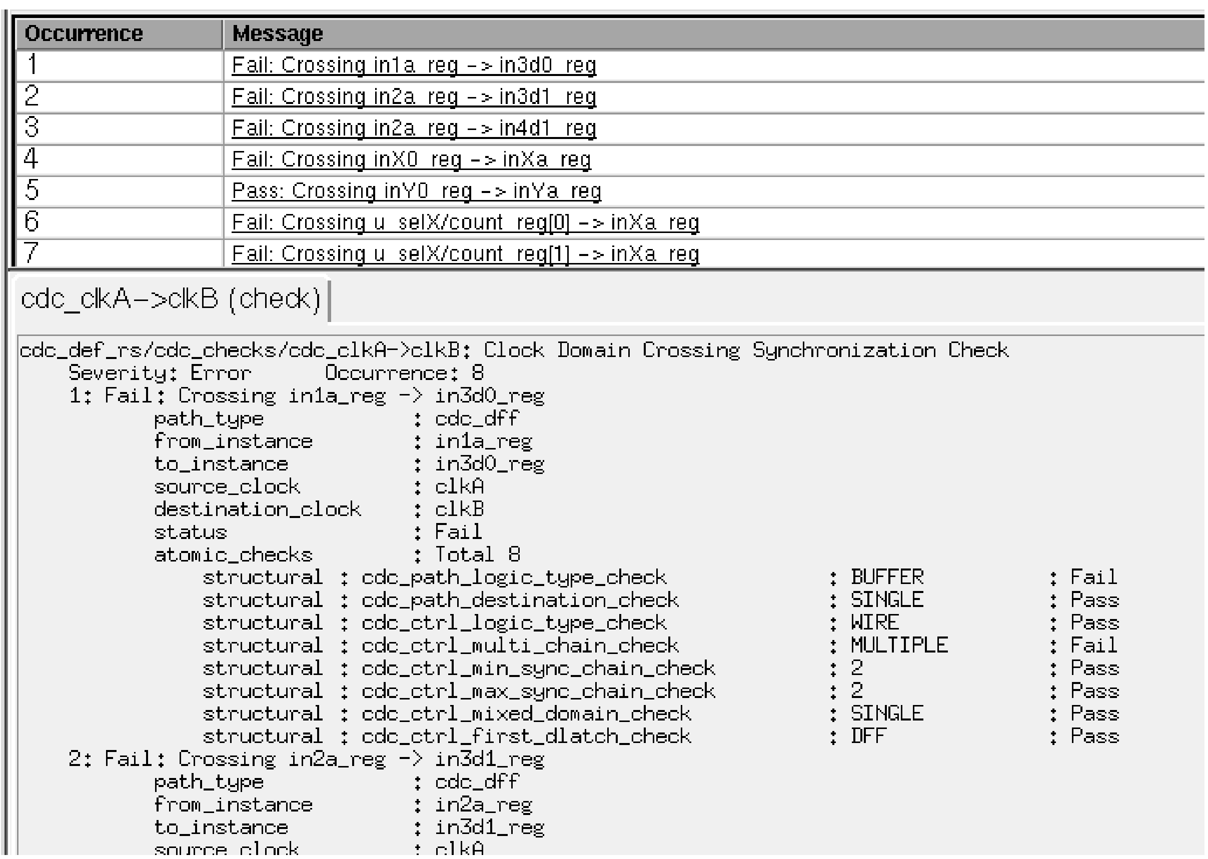
    Where # is 2 or more.
  - Pass/Fail

  For more information on the atomic checks, refer to "Atomic Checks" in the *Conformal Constraint Designer Rule Check Reference*.

You can use the Rule Manager also to report on a rule instance:



The results display in the lower right hand corner of the Rule Manager:



Tip

To report all paths that failed a crossing, use the `report rule check -status fail` command.

[Go back to the Clock Domain Crossing Checks Flow.]

Viewing the Default Values of Rule Attributes

To view the default value for an attribute, you can use the MAN command on its applicable rule check (default values are listed first) or use the `REPORT RULE SOURCE` command.

For example, to view the defaults for all the attributes of `cdc_setreset_sync_rule` (Note: The rule set called `ccd_default_cdc_ruleset.tcl` must be added to the current session before you can report on any of its attributes.):

```
==============================================================================
=                     Rule source 'cdc_setreset_sync_rule'                   =
==============================================================================
|   Name: cdc_setreset_sync_rule
|   Description: Set and Reset Synchronizer checks
|   Version: 2.1
|   Category: SDC_CDC
|   Severity: Error
|   Required State: commit_clock (after committing clocks)
|   Help File: help/cdc_setreset_sync_rule.help
|   Include Files: cfm_common_rule_util.tcl cfm_verify_shared_util.tcl
ccd_cdc_shared_util.tcl
|   Rule Source Parameters:
|       1) analysis_mode: Tcl_Obj*
|           Description: Specify the analysis mode
|           Default Value: "structural"
|           Valid Values:  structural functional
|       2) destination_clock: Tcl_Obj*
|           Description: Target clock for set & reset check
|       3) consider_clock_phase: Tcl_Obj*
|          Description: Checking if synchronization registers have same clock phase
|           Default Value: "yes"
|           Valid Values:  yes no
...
```

Source and Destination Clocks (Clocked versus Unclocked)

Note that `clocked` and `unclocked` are keywords that can be used to designate source and destination clocks (these keywords cannot be combined with SDC objects). The source and destination clocks are specified through the `set_attribute` command. For example:

```
set_attribute <rinst> source_clock clocked
set_attribute <rinst> destination_clock unclocked
```

Tip

In Conformal Constraint Designer, you use the command `set_attribute` to configure the various attributes on rule instances. The advantage of this mechanism is that you can update attributes more efficiently and independently, according to your design needs. For example, you can specify multiple sync modules:

```
set_attribute user_sync_modules [find -design user_sync_module*]
```

The command above will find all designs (modules) with names starting with `user_sync_module`.

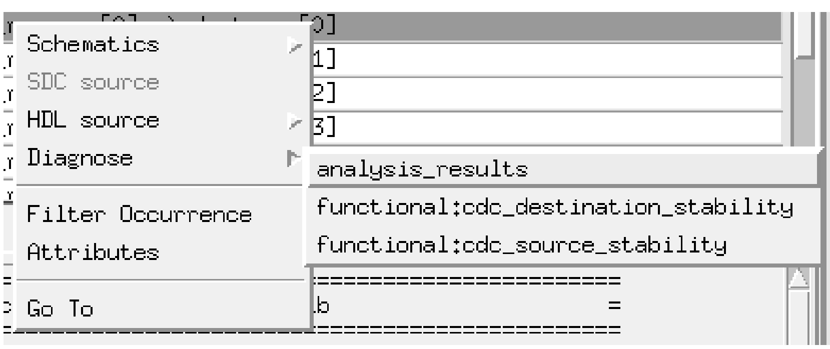
[Go back to the Clock Domain Crossing Checks Flow.]

Diagnose CDC Violations

To diagnose a particular rule check, use the following command:

```
diagnose rule check cdc_def_rs/cdc_CLKA->CLKB //Diagnoses a rule instance
```

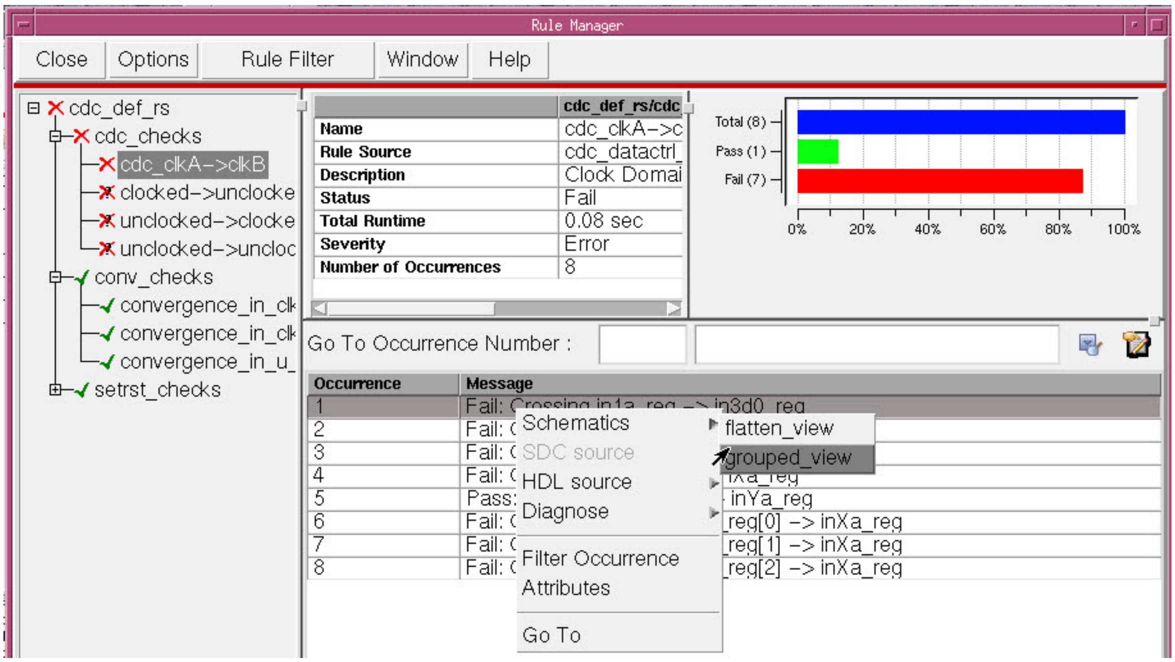
From the Rule Manager, you can also right-click on an occurrence of a rule instance to view the various diagnostic options:



[Go back to the Clock Domain Crossing Checks Flow.]

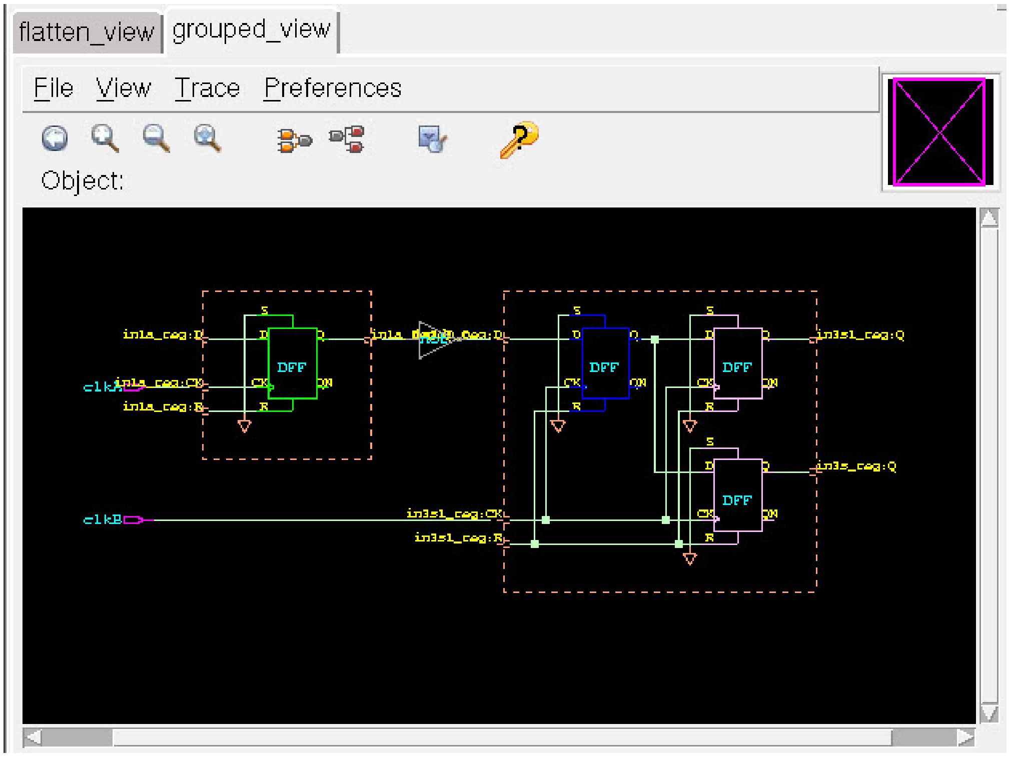
Viewing CDC Violations in the Schematics Window

From the Rule Manager, you can also right-click on an occurrence of a rule instance debug using the schematics window.



- flatten_view
- grouped_view—Groups together objects of the same type.

The following is a sample of a `grouped_view` schematics window.



Color Legends for Schematics Window

Figure 11-10 cdc_datactrl_sync_rule

Color	Description
yellow	Flip-flops clocked by source clock
cyan	Flip-flops clocked by the destination clock
khaki	Configuration register
plum	Sync chain
green	Source flip-flop of CDC path
blue	Destination flop of CDC path
gray	All other logic gates not covered by the colors above

Figure 11-11 cdc_conv_check_rule

Color	Description
yellow	Flip-flops clocked by source clock
cyan	Flip-flops clocked by the destination clock
khaki	Configuration register
blue	Convergence point
cyan	Convergence end point
gray	All other logic gates not covered by the colors above

Figure 11-12 cdc_setreset_sync_rule

Color	Description
yellow	Flip-flops clocked by source clock
cyan	Flip-flops clocked by the destination clock
khaki	Configuration register
blue	Set/reset driver
cyan	Set/reset end point
gray	All other logic gates not covered by the colors above

Customizing Clock Domain Crossing Analysis

This section describes how you can customize clock domain crossing analysis in Conformal Constraint Designer.

[Go back to the Clock Domain Crossing Checks Flow.]

Filtering Rule Instances

In Conformal Constraint Designer, you can filter rule instances using the `set_attribute filter_paths` command (use before running rule checks) or the `ADD RULE FILTER` command (used after running rule checks).

Customizing CDC Rule Instance Reports

Conformal Constraint Designer provides numerous rule check options for clock domain crossing analysis. Sometimes, this can mean wading through long rule instance reports (generated using the `REPORT RULE CHECK`) as all of these options are displayed by default.

You can exclude options from the report header using the `suppress_options_in_header` attribute. This can help narrow down a long report, especially when options contain a long list.

For example, to exclude the paths specified by the `filter_paths` and `configuration_regs` attributes from the `REPORT RULE CHECK` header:

```
add_rule_set -file ccd_default_cdc_ruleset.tcl

set rule_instance [find -ruleinst cdc_def_rs/cdc_checks/cdc* ]

set_attribute $rule_instance suppress_options_in_header \
[ list configuration_regs filter_paths]
```

Excluding Modules from Rule Instances

You can constrain the rule instance of a CDC structural check such that it is not applied to the CDC paths of the specified modules. To do this, use the `exclude_module` attribute. For example:

```
set_attribute $rule_instance exclude_module \
[ find -design mod_a ]
```

Excluding Atomic Checks from Rule Instances

You can exclude/disable atomic checks for a specific rule instance (can only be done for rule instances of `cdc_datactrl_sync_rule` and not `cdc_conv_check_rule` or `cdc_setreset_sync_rule`). To do this, use the `exclude_atomic_checks` attribute. For example:

```
set_attribute $rule_instance exclude_atomic_checks \
  [list cdc_path_logic_type_check \
   cdc_path_destination_check ]
```

12

Managing Rules

Rule Sets

HDL rules and SDC lint rules are enabled by default. To enable other rules, they must be included in a rule set and loaded using the `ADD RULE SET` command. You can use one of predefined rule sets listed below or build your own rule set (using the `ADD RULE SET`, `ADD RULE GROUP`, and `ADD RULE INSTANCE` commands).

Using Predefined Rule Sets

The following predefined rule sets are available for you to use (in Tcl mode, use the `*.ntcl` extension):

`cfm_default_modeling_ruleset.vpx`—Includes all of the modeling rules.

`ccd_default_cdc_ruleset.tcl`—Includes all clock domain crossing (CDC) policy rules.

`ccd_default_sdc_ruleset.tcl`—Includes all the SDC checks.

`ccd_report_clock_tree_ruleset.tcl`—Includes all the clock tree policy rules.

If you are in Tcl mode, you must use the `*.ntcl` extension. For example:

```
TCL_VERIFY> add_rule_set -file ccd_default_cdc_ruleset.ntcl
```

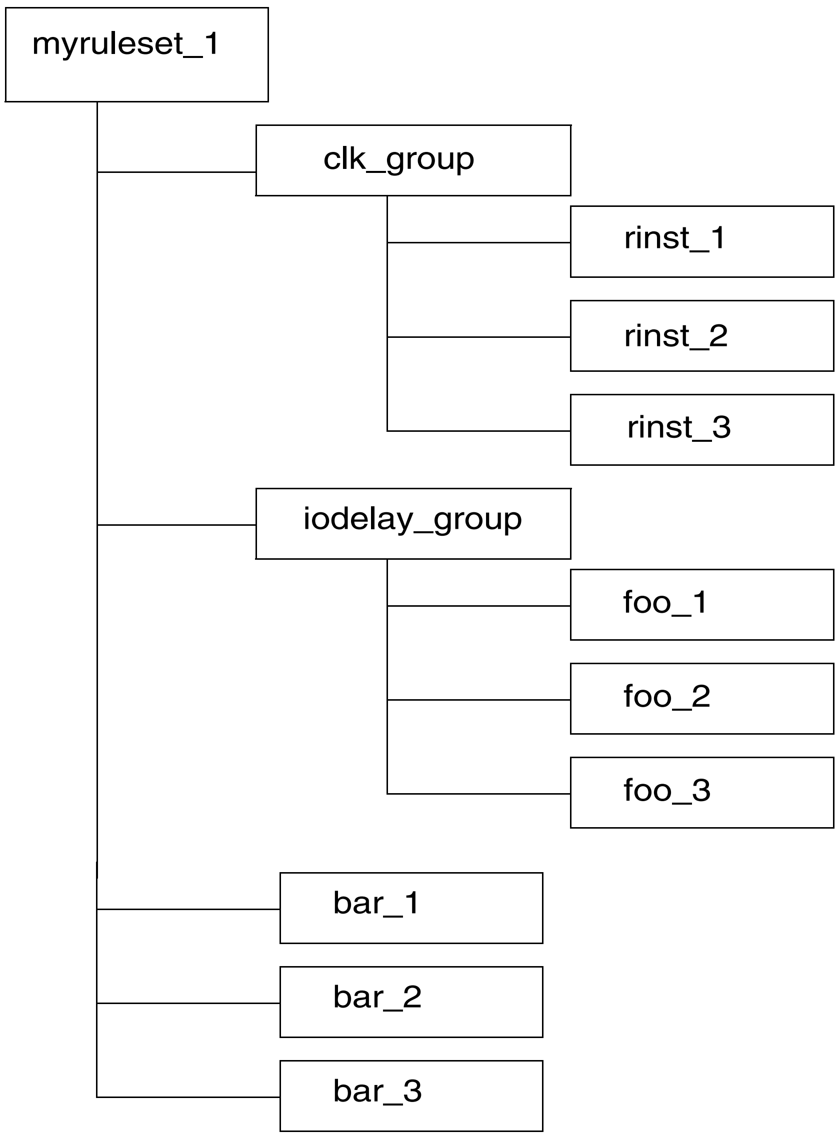
Creating Rule Sets

Rule sets can be created using the following sets of commands:

`ADD RULE SET`	Adds one or more rule sets to the rule tree. A rule set is a collection of rule groups or rule instances. **Note:** The tool will define the last added set to be the current rule set. Any subsequent rule group or instance added will be included under that rule set.
`ADD RULE GROUP`	Adds one or more rule groups to the rule tree. A rule group is a collection of rule instances. **Note:** The tool will define the last added group to be the current rule group. Any subsequent rule instance added will be included under that rule group.

`ADD RULE INSTANCE`	Adds rule instances to a rule set or group. As with object-oriented programming, where programmers instantiate a class to create an object, you first must instantiate a rule source to create a rule instance, which is the basic rule object to perform any check or query.
`ADD RULE SOURCE`	Adds rule sources to the current session. Although rule sources are automatically loaded when they are referenced in the creation of rule instances, you can load a rule source to inspect its attributes. When running this command, the software looks for Tcl file '`<name>.tcl`' in the search path for rules. If the rule source file is not in the search path or is not named '`<name>.tcl`,' you can give its direct file path by using the `-file` option. **Note:** There must be exactly one rule source per file.

Example 12-1 Creating a Rule Set



You would use the following commands to implement the rule set described in the previous figure:

```
add rule set myruleset_1

//Adds clk_group and iodelay_group to the myruleset_1 rule set

add rule group clk_group iodelay_group

//Specifies clk_group as the current rule group
```

```
set rule group clk_group

//Adds 3 rule instances to the current rule group
//Each rule instance specifies an SDC build-in policy check to enable

add rule instance rinst_1 ccd_clk_def1
add rule instance rinst_2 ccd_clk_def2
add rule instance rinst_3 ccd_clk_def35

//Specifies io_delaygroup as the current rule group
set rule group iodelay_group

//Adds 3 rule instances to the current rule group
//Each rule instance specifies an SDC build-in policy check to enable

add rule instance foo_1 ccd_io_idl1
add rule instance foo_2 ccd_io_odl1
add rule instance foo_3 ccd_io_odl6

//Specifies myruleset_1 as the current rule set
set rule set myruleset_1

//Adds 3 rule instances to the current rule set
//Each rule instance specifies an SDC build-in policy check to enable
add rule instance bar_1 ccd_exc_flp1
add rule instance bar_2 ccd_exc_mcp1
add rule instance bar_3 ccd_exc_olp1
```

Rule Manager

Conformal Constraint Designer provides an interactive GUI that lets you select a rule occurrence and view the schematics, HDL source (source/destination), and analysis results associated with that violation. You can also add rule groups, instances, and rule objects from this GUI.

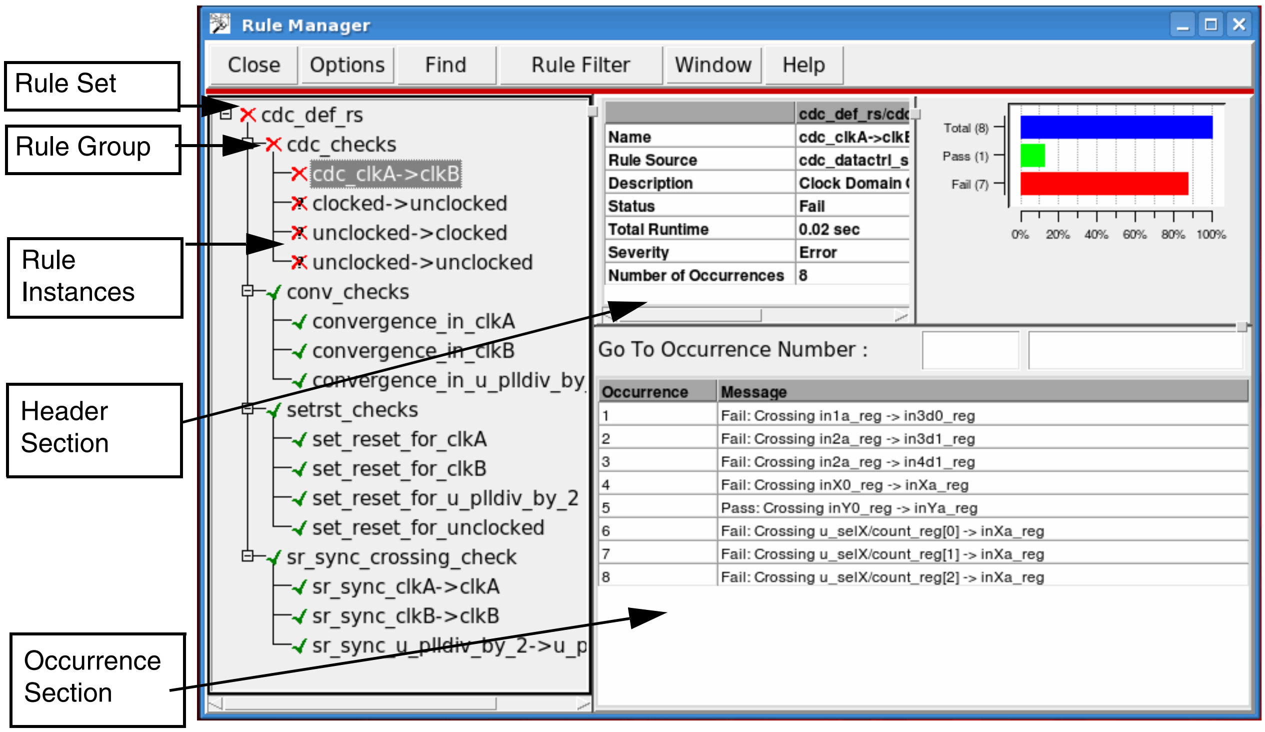
To launch the Rule Manager from the icon bar of the main window:

	*Rule Manager*	Opens the Rule Manager.

Or, use the *Tools - Rule Manager - <Rule Set>* command from the Menu bar. Where *<Rule Set>* is a predefined rule set (described in "Using Predefined Rule Sets" on page 212) or, a rule set that you created (see "Creating Rule Sets" on page 212).

When launched, the Rule Manager displays the rule set and breaks it down by rule group, and then by rule instances. When at the instance level, you can navigate through the instance's occurrences and identify the issues in the design that caused the violation.

Figure 12-1 Rule Manager



The Rule Manager displays violations in RED Xs, passed instances as green checks, and inconclusive instances as question marks enclosed in a red circle.

Setting Rule Options

Use the *Options* button of the rule manager to configure how you would like to display information in the rule manager.

View Header	Specifies what you want displayed in the header section. By default, both the *Summary* table and *Histogram* are displayed.

View Occurrence	Specifies what you want displayed in the occurrence section of the rule manager. *All*—Displays all rule occurrences. *Not filtered*—Displays the unfiltered rule occurrences. Any occurrences filtered out by `ADD RULE FILTER` or the *Filter* menu are not displayed. *Filter*—Displays rule occurrences that have been filtered by `ADD RULE FILTER` or the *Filter* menu.
Undock Note book	By default, when you diagnose an occurrence, the diagnosis information appears within the rule manager (underneath the occurrence section). When this option is enabled, the diagnosis information appears in a tabbed-style window (also called a notebook).
Undock Note page	By default, when you diagnose an occurrence, the diagnosis information appears within the rule manager (underneath the occurrence section). When this option is enabled, the diagnosis information appears in a separate window.

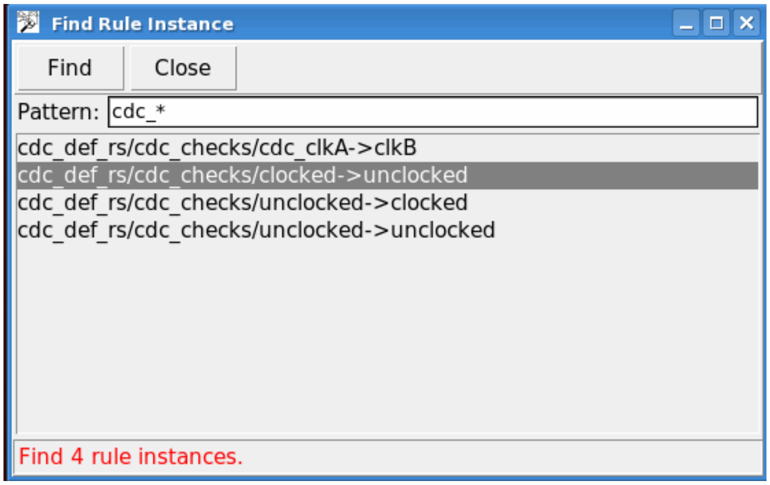
Searching for Rule Instances

To search for rule instances:

1. From the Rule Manager select the rule set from which you want to search. For example, select *cdc_def_rs*.
2. Click on the *Find* button.

   The *Find Rule Instance* dialog displays.

3. In the Pattern field, type in the pattern (wildcards accepted) you would like to search for.



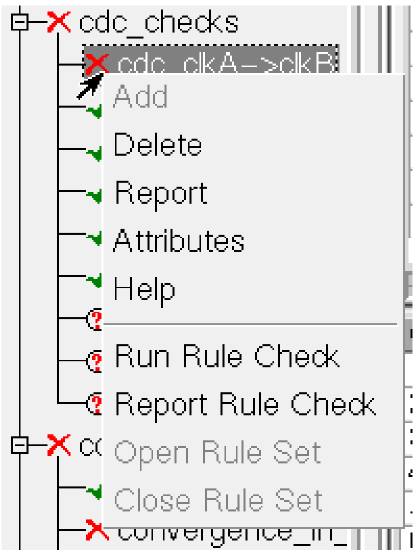
Adding Rule Filters

Use the *Rule Filter* button to add a rule filter. Rule filters are used to select which occurrences should be ignored by commands such as `REPORT RULE CHECK`.

For more information on rule filters, go to the MAN page of the `ADD RULE FILTER` command.

Working With Rule Instances

You can then right-click on a rule instance to delete, report on, or view attributes for the rule instance.

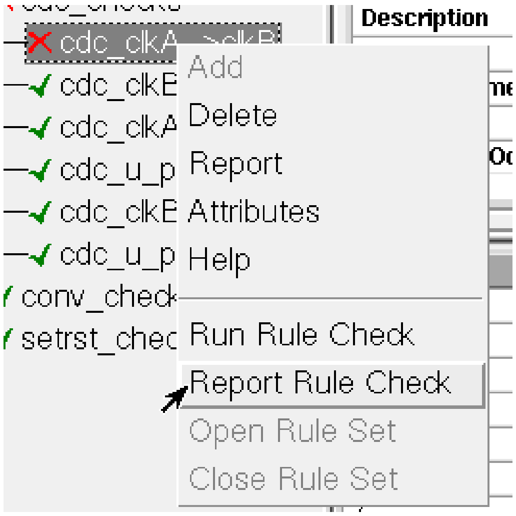


When a rule instance is highlighted in the Rule Manager, you can get more information about the rule instance:

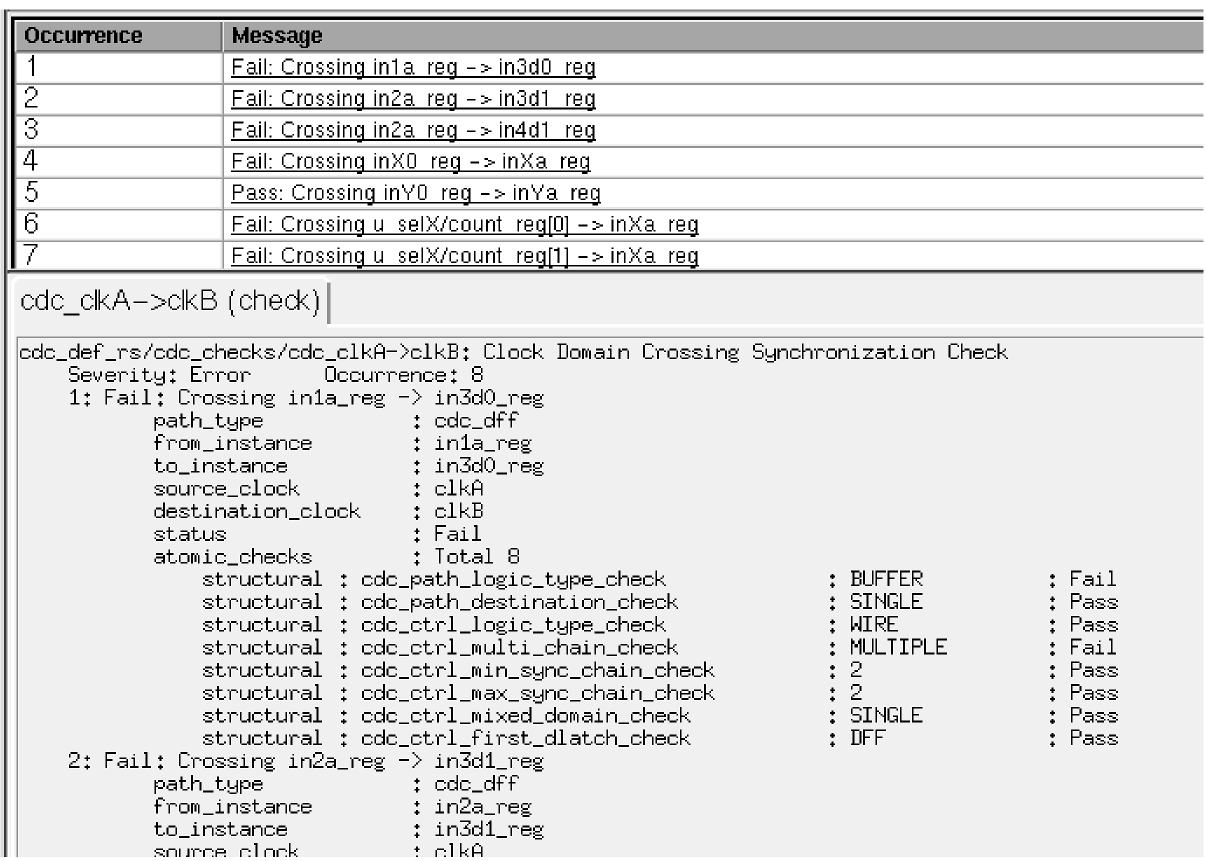
	cdc_def_rs/cdc_checks/cdc_clkA->clkB
Name	cdc_clkA->clkB
Rule Source	cdc_datactrl_sync_rule
Description	Clock Domain Crossing Synchronization Check
Status	Fail
Total Runtime	0.03 sec
Severity	Error
Number of Occurrences	8

If the rule instance has failures, from the Rule Manager, you can also right-click on an occurrence of a rule instance to view the various diagnostic options.

To generate a report on a rule instance, right click on the instance and select *Report Rule Check*.



The results display in the lower right hand corner of the Rule Manager:



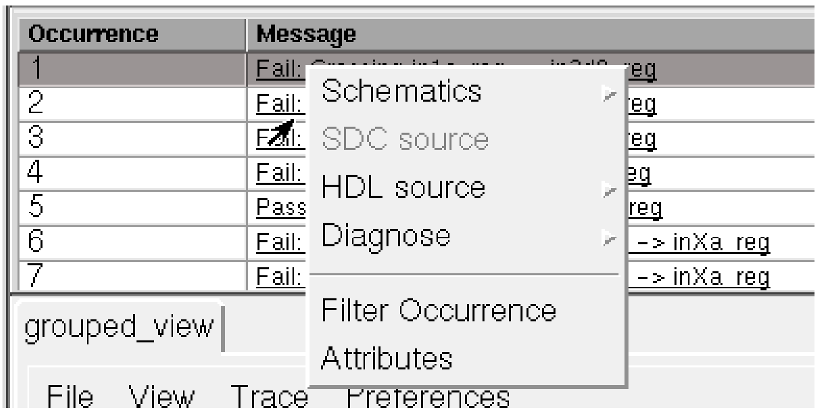
Occurrence	Message
1	Fail: Crossing in1a_reg -> in3d0_reg
2	Fail: Crossing in2a_reg -> in3d1_reg
3	Fail: Crossing in2a_reg -> in4d1_reg
4	Fail: Crossing inX0_reg -> inXa_reg
5	Pass: Crossing inY0_reg -> inYa_reg
6	Fail: Crossing u_selX/count_reg[0] -> inXa_reg
7	Fail: Crossing u_selX/count_reg[1] -> inXa_reg

cdc_clkA->clkB (check)

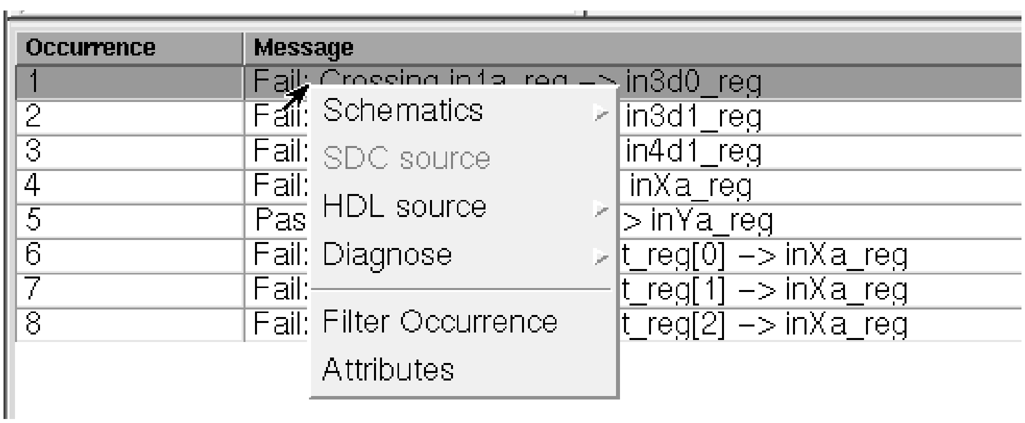
```
cdc_def_rs/cdc_checks/cdc_clkA->clkB: Clock Domain Crossing Synchronization Check
    Severity: Error      Occurrence: 8
    1: Fail: Crossing in1a_reg -> in3d0_reg
           path_type              : cdc_dff
           from_instance          : in1a_reg
           to_instance            : in3d0_reg
           source_clock           : clkA
           destination_clock      : clkB
           status                 : Fail
           atomic_checks          : Total 8
               structural  : cdc_path_logic_type_check          : BUFFER          : Fail
               structural  : cdc_path_destination_check         : SINGLE          : Pass
               structural  : cdc_ctrl_logic_type_check          : WIRE            : Pass
               structural  : cdc_ctrl_multi_chain_check         : MULTIPLE        : Fail
               structural  : cdc_ctrl_min_sync_chain_check      : 2               : Pass
               structural  : cdc_ctrl_max_sync_chain_check      : 2               : Pass
               structural  : cdc_ctrl_mixed_domain_check        : SINGLE          : Pass
               structural  : cdc_ctrl_first_dlatch_check        : DFF             : Pass
    2: Fail: Crossing in2a_reg -> in3d1_reg
           path_type              : cdc_dff
           from_instance          : in2a_reg
           to_instance            : in3d1_reg
           source_clock           : clkA
```

Diagnosing Violations

From the Rule Manager, you can right-click on an occurrence of a rule instance to view the various diagnostic options:



Managing HDL Rules



HDL rules consist of a group of rules that should be observed during design analysis, elaboration, and RTL construction. For example, the checker notifies you of the presence of UDPs, directives, and hierarchical coding; and alerts you to code that might lead to RTL and gate-level simulation mismatches. Thus, when these rules are violated, it is an indication of either a potential design error, or a possible mismatch between RTL and gate-level simulations for logically equivalent circuits.

HDL rule checks are enabled by default. Refer to the *Conformal HDL Rule Check Reference* for a list of all the supported HDL rule checks. You can also view all the HDL rule check messages and their details using `'man <rule>'` at the command line.

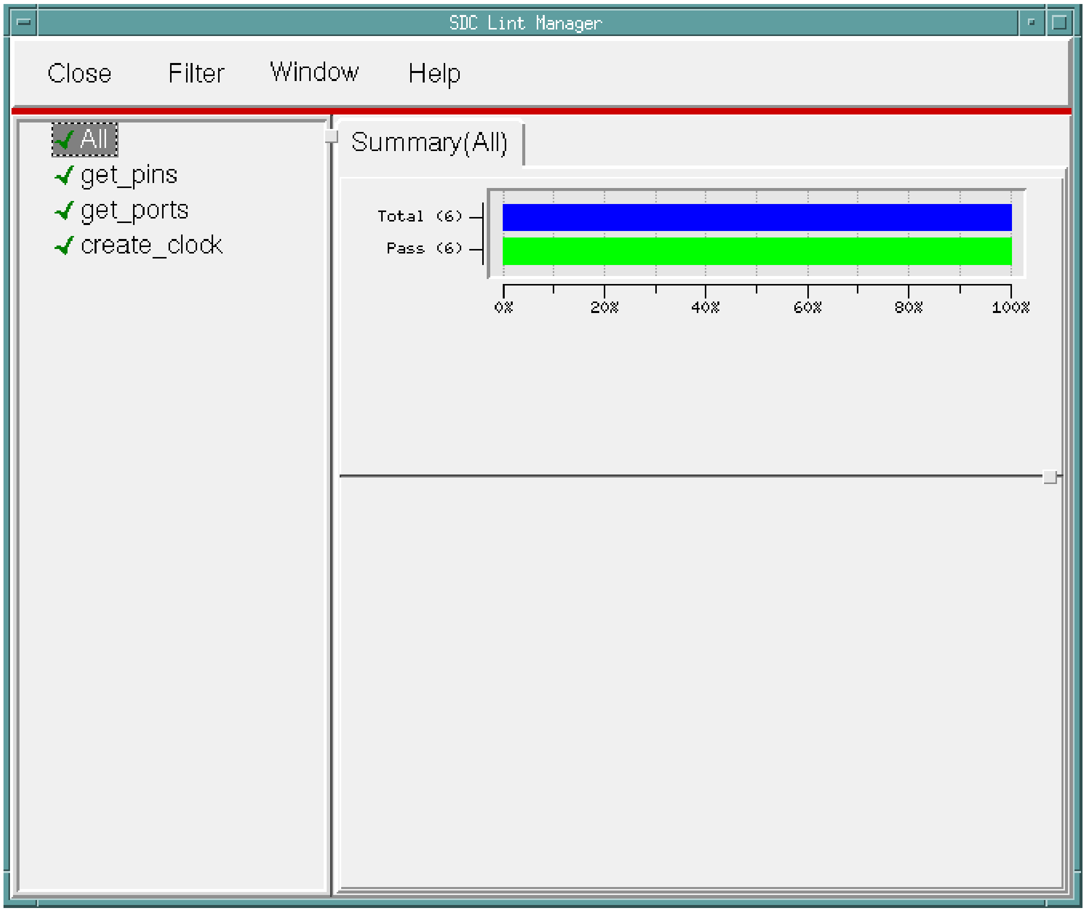
Using the Rule Manager, you can manage the HDL rule checks that are done when reading in libraries and designs:

➤ Choose *Tools – Rule Manager — hdl_default_checks*

Table 12-1 HDL Rule Fields and Options

Option	Click the *View* pull-down menu and choose *Rule with messages only* (the default), *All* to display a complete list of rules and the messages (violations) for each page, or *Hidden Rules* to display the hidden rules and the messages. Click the *Page Size* option t
Summary	For each page, this displays the total number of rules for the specified category, and the total number of rule violation occurrences (messages).
Undo	Unmarks the previous waive occurrence. This will only undo the last waiver mark. To undo previous waiver marks, select and select *Unwaive Occurrence*.
Filter	Opens the Filter Rule form where you can add or delete rule filters. For more information, see Filtering Rules on page 226.

SDC Lint Manager



Modeling messages indicate any modeling errors encountered during the analysis and modeling of the design. These errors are caused when the design is not initialized properly or when the design has problems that may lead to a mismatch between the logic behavior and the electrical behavior. Modeling Rule Checks are active as the system mode changes from Setup to Verify.

Modeling messages indicate any modeling warnings encountered during the analysis and modeling of the design.

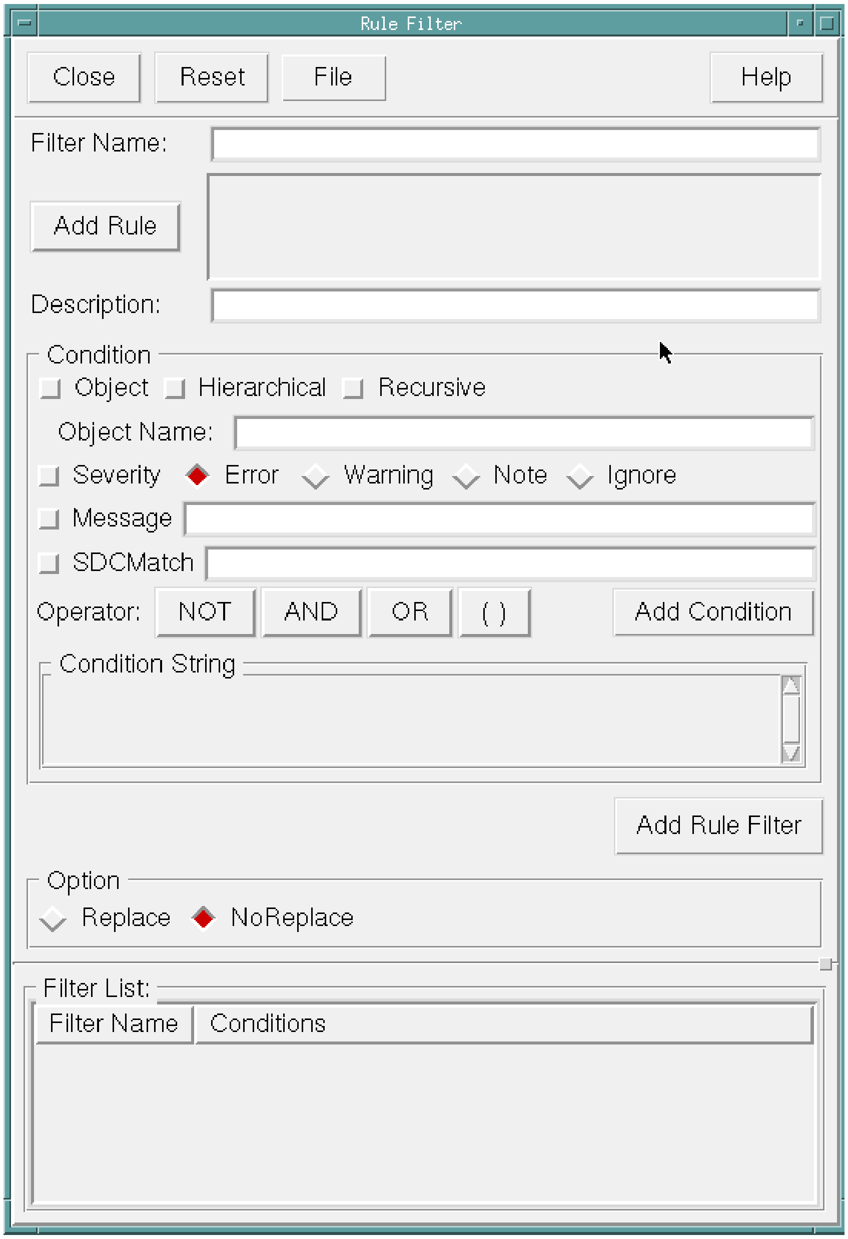
Table 12-2 Modeling Rule Manager Fields and Options

Option	Click the *View* pull-down menu and choose *Rule with messages only* (the default), *All* to display a complete list of rules and the messages (violations) for each page, or *Hidden Rules* to display the hidden rules and the messages. Click the *Page Size* option t
Summary	For each page, this displays the total number of rules for the specified category, and the total number of rule violation occurrences (messages).
Undo	Unmarks the previous waive occurrence. This will only undo the last waiver mark. To undo previous waiver marks, select and select *Unwaive Occurrence*.
File - Write Rule Check	Opens the Write Rule Check form where you.
Filter	Opens the Filter Rule form where you can add or delete rule filters. For more information, see Filtering Rules on page 226.

Filtering Rules

You can add or delete rule filters with the Rule Filter form.

➤ From the Rule Manager, click *Filter*.



Adding Rule Filters

To add a rule filter:

1. Specify the name of the filter in the *Filter Name* field.

   Note: If you do not enter a name in this field, a unique name is automatically generated.

2. Enter the Conditions.

   - ❑ *Rule* — Filters out all occurrences of specified rule(s). Type the name of the rule or use the pull-down menu.
   - ❑ *Object* — Matches rule occurrences related to objects that match the specified pattern.
     - *Hierarchical*—matches any instance in the design hierarchy against the specified pattern (analogous to the SDC command '`get_pins -hier <pinname>`').
     - *Recursive*—matches any object under an instance that matches the specified pattern (analogous to the Unix command '`grep -r ... <dirname>`').
   - ❑ *Severity* — Filters out all occurrences of the selected severity level(s).
   - ❑ *Message* — Matches rule occurrences whose verbose message matches the specified pattern.
   - ❑ *SDC Match* — Matches rule occurrences related to SDC statements matching the specified pattern. The string representing the SDC statement is the one displayed in the SDC command browser, not the one from the SDC file.

     Note: This condition is only available for SDC rules.
   - ❑ *Operator* — Specifies that the operator.

3. Enter the Options.

   - ❑ *Replace* — Specifies that the filter name can be that of an existing filter, which is then modified.
   - ❑ *NoReplace* — Specifies that the filter name cannot be that of an existing filter.

4. Click the *Add Rule Filter* button.

Deleting Rule Filters

To delete a rule filter, select a filter in the *Filter List*, and right-click and choose *Delete* or *Delete All* from the pop-up menu.

Managing Severity Levels

The severity levels are listed below from the most serious to the least serious:

- Error—The software might not allow you to begin verification until you resolve the error.
- Warning—The software allows you to begin verification; however, it warns you of potential errors in the design.
- Note—The software allows you to begin verification; however, it flags potential errors in the design.
- Ignore—The software does not report this severity by default.

Additional tasks:

- Changing the Severity of Lint or Policy Checks on page 228
- Reporting All Rule Messages (Regardless of Severity Level) on page 229

Changing the Severity of Lint or Policy Checks

You can change the level of severity for rule violations with the `SET RULE HANDLING` command. For example, to show the initial default severity level for `HRC7`, you would run the following command (in Setup mode):

```
report rule source hrc7
```

The output shows the rule name, default severity, and description:

```
=============================================================================
=                            Rule source 'HRC7'                             =
=============================================================================
|  Name: HRC7
|  Description: Module specified by the 'add notranslate modules' command cannot
|  be found
|  Category: LIBRARY_DESIGN
|  Severity: Warning
=============================================================================
```

To show the current severity of `HRC7`, which in this example has not been changed from its default severity level, you would run the following command:

```
report rule check *hrc7
```

```
hdl_default_checks/hierarchy_checks/HRC7: Module specified by the 'add notranslate
modules' command cannot be found
Severity: Warning      Not-Run
```

To change the severity level to an error, you would run the following command (in Setup mode):

```
set rule handling *HRC7 -severity -error
```

To show the new severity level for `HRC7`, you would run the following command:

```
report rule check HRC7 -setting
```

```
hdl_default_checks/hierarchy_checks/HRC7: Module specified by the 'add notranslate
modules' command cannot be found
Severity: Error      Not-Run
rs/HRC7: Module specified by the 'add notranslate modules' command cannot be found
```

Note: However, if after changing `HRC7` rule's severity to an error, you run the `report rule source hrc7` command, you will still get the (default) severity of `Warning`.

You can also use the `set_attribute` Tcl command to change the severity of a lint or policy rule check. For example, the following command changes the severity of `SDC_LINT_OPT2` to Note:

```
set_attribute [ find -ruleinst SDC_LINT_OPT2 ] severity Note
```

You cannot, however, downgrade lint rules with a severity of Error. For example:

```
TCL_SETUP> set_attribute [find -ruleinst SDC_LINT_VAL1] severity warning
// Error: Severity level of SDC_LINT_VAL1 cannot be changed.
```

Reporting All Rule Messages (Regardless of Severity Level)

By default, Conformal Constraint Designer only reports failed rule checks with a severity of Error.

To report all rule checks that have failed, use the following command:

```
SETUP> report rule check -ver -status fail
...
hdl_default_checks/ignored_checks/IGN3.2: Duplicate modules/entities are detected.
Subsequent modules/entities are ignored
Severity: Warning Fail Occurrence: 1
Type: Library Severity: Warning Fail Occurrence: 1
1: Fail: In line 234, file 'lib2.lib' (IO): Module name 'IO' is duplicated and later
one is ignored
In line 234, file 'lib1.lib' (IO): Module name 'IO' is duplicated and later one is
ignored
```

Note: This does not report HDL rule checks. To report failed HDL rule checks with a severity of Warning, use the following command:

```
report rule check hdl_default_checks -status fail
```

Working with Rule Checks

Enabling a Rule Check

Similar to RTL rule checks, SDC policy rule checks are disabled by default. To enable a rule check, it must be added to a rule set. To disable a rule check, you must delete it from the rule set. See "Creating Rule Sets" on page 127.

Running Incremental Rule Checks

Use the `WRITE RULE CHECK` and `READ RULE CHECK` commands to run incremental checks. The first time you run a session, write the rule violations into a rule file using the `write rule check <filename>` command. For subsequent runs, use the `read rule check -exclude <filename>` command to exclude the violations already flagged.

Reporting Rule Messages without Error Severity

By default, Conformal Constraint Designer only reports failed rule checks with a severity of Error.

To report all rule checks that have failed, use the following command:

```
SETUP> report rule check -ver -status fail
...
hdl_default_checks/ignored_checks/IGN3.2: Duplicate modules/entities are detected.
Subsequent modules/entities are ignored
Severity: Warning Fail Occurrence: 1
Type: Library Severity: Warning Fail Occurrence: 1
1: Fail: In line 234, file 'lib2.lib' (IO): Module name 'IO' is duplicated and later
one is ignored
In line 234, file 'lib1.lib' (IO): Module name 'IO' is duplicated and later one is
ignored
```

Note: This does not report HDL rule checks. To report failed HDL rule checks with a severity of Warning, use the following command:

```
report rule check hdl_default_checks -status fail
```

Possible Causes for a "Not-Run" status?

The following describes possible reasons for a rule check to return a Not-Run status:

- Incorrect design state

  Some rule checks require that you be in a particular design state. For example, if you execute `RUN RULE CHECK` on a hierarchical check, before you load block-level constraints, the tool returns a Not-Run status.

- No clock propagation information

  Some rule checks require clock propagation information (which is acquired when you switch from Setup to Verify mode). If you run a rule check that requires clock propagation information (such as `CCD_CLK_DEF1`) while the tool is in Setup mode, the tool returns a Not-Run status.

To determine the required design state for a rule check, you can use the `REPORT RULE SOURCE` command. For example:

```
SETUP> report rule source CCD_CLK_DEF1
================================================================================
= Rule source 'CCD_CLK_DEF1' =
================================================================================
| Name: CCD_CLK_DEF1
| Description: Unconstrained clock: A clock pin does not belong to any clock tree
(missing
create_clock or create_generated_clock command)
| Category: SDC_STRUCTURAL
| Severity: Warning
| Required State: flattening (after flattening)
================================================================================
```

Using the continue_on_error Attribute

In Conformal Constraint designer, rule objects have an attribute called `continue_on_error`; this attribute specifies whether failed rule checks with a severity level of Error will stop the certain commands from proceeding. In other words, the `READ LIBRARY`, `READ DESIGN`, and `READ SDC` commands will stop when they encounter a violation from a rule instance whose severity level is Error and whose `continue_on_error` attribute is set to 1. This includes rule checks whose severity was changed to Error after instantiation (described in "Changing the Severity of Lint or Policy Checks" on page 228).

In the current release, `SDC_LINT_*` rule checks and the following rule checks have `continue_on_error` set to 1 by default, which means that `READ SDC` continues even when it encounters failed rule checks whose severity level is Error. The following rules, which are also checked during parsing, also have `continue_on_error` set to 1 by default:

- `CCD_CLK_DEF4`

- CCD_CLK_DEF8
- CCD_CLK_DEF12
- CCD_CLK_DEF14
- CCD_CLK_DEF16
- CCD_CLK_DEF17
- CCD_CLK_DEF18
- CCD_CLK_DEF22
- CCD_CLK_DEF27
- CCD_CLK_LAT2
- CCD_CLK_LAT5
- CCD_CLK_LAT9
- CCD_CLK_LAT10
- CCD_CLK_UNC2
- CCD_CLK_UNC5
- CCD_CLK_CTR2
- CCD_CLK_CTR3
- CCD_CLK_CTR5:
- CCD_CLK_CTR7
- CCD_CLK_CTR8
- CCD_CLK_CTR14
- CCD_CLK_CTR15
- CCD_IO_IDL2
- CCD_IO_IDL7
- CCD_IO_IDL9
- CCD_IO_IDL10
- CCD_IO_IDL11
- CCD_IO_IDL12

- CCD_IO_IDL16
- CCD_IO_ITR4
- CCD_IO_ITR7
- CCD_IO_ITR8
- CCD_IO_ITR9
- CCD_IO_ODL2
- CCD_IO_ODL7
- CCD_IO_ODL9
- CCD_IO_ODL10
- CCD_IO_ODL11
- CCD_IO_ODL12
- CCD_IO_ODL14
- CCD_IO_ODL16
- CCD_IO_OLD2
- CCD_IO_OLD5
- CCD_EXC_FLP2
- CCD_EXC_FLP3
- CCD_EXC_MCP2
- CCD_EXC_MCP3
- CCD_SDC_STR7
- CCD_MISC_HFN10
- CCD_MISC_HFN13
- CCD_MISC_NAM1
- CCD_MISC_NAM2
- CCD_MISC_MSC2
- CCD_MISC_MSC3
- CCD_MISC_MSC7

- `CCD_MISC_MSC12`
- `CCD_MISC_MSC13`

Notes:

- The `continue_on_error` attribute does not affect the `RUN RULE CHECK` command.
- You can only change the value of `continue_on_error` for HDL rule checks whose default severity *is not* Error. This attribute is unsettable for all other types of rule checks.

13

SDC Integration

Note: SDC Integration requires an XL license.

With design complexity increasing, for designs that are partitioned into multiple blocks, the design constraints must also be partitioned accordingly. Full-chip constraints can become a challenge when maintaining correctness and consistency in the design constraints between block boundaries and between block constraints and the top-level. Also a challenge is integrating block-level design constraints for various IP cores developed at the block level into top-level, full-chip design constraints for global timing analysis and synthesis. This is usually done manually or with the aid of ad-hoc scripts.

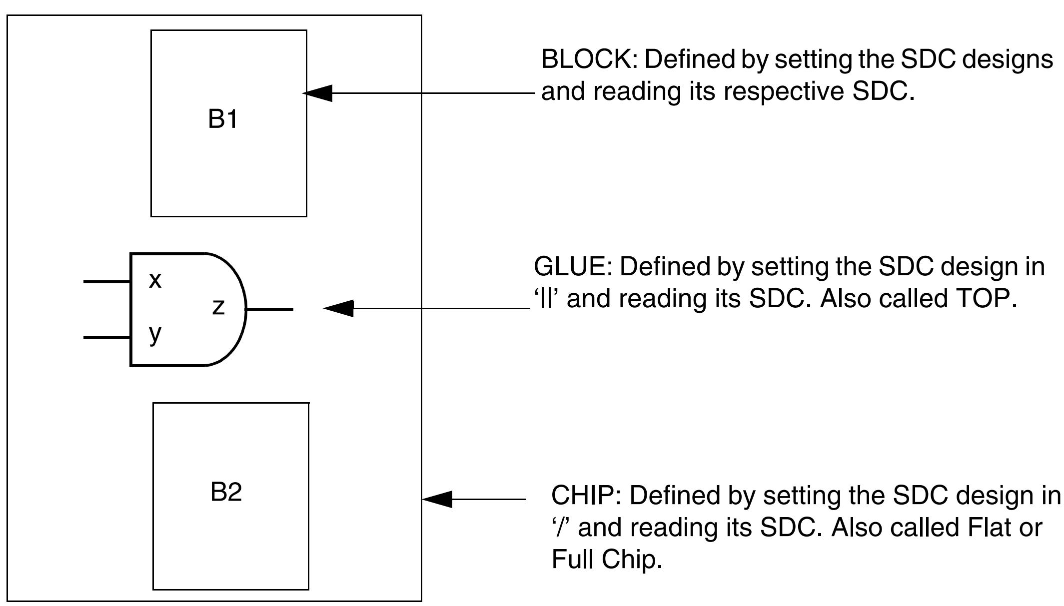
This chapter describes the Conformal Constraint Designer flow for integration of SDC files used to constrain blocks—and optionally, the top-level module alone—into a full chip constraint file. To support an iterative flow and to allow manual user intervention, a partial chip constraint file can also be supplied with contents that are considered *golden*.

Background and Terminology

Conformal Constraint Designer provides the capability to read and store separate sets of SDC information into entities called *SDC designs*, where:

- Each block can be defined as an SDC design, whose name is that of the block instance. A set of constraints read into such an SDC design is called *Block SDC*.
- Constraints for the top-level module only (without covering the blocks) are stored in an SDC design called '||'. We refer to these constraints as *Glue SDC*. Elsewhere, they are sometimes referred to as *Top SDC*.
- Constraints for the full chip (top module+blocks) are stored in an SDC design called '/'. This is the default SDC design after the design is read into the tool. We refer to these constraints as *Chip SDC*. Elsewhere, they are sometimes called *Flat SDC*, *Full-chip* SDC, and so on.

The following shows an illustrated example:



The term *top-level* is ambiguous because both Glue SDC and Chip SDC are applied at a top-level design.

In the hierarchical checks flow, Conformal Constraint Designer checks consistency of Block SDC or Glue SDC against Chip SDC.

Note: In the hierarchical checks, the messages refer generically to *block vs. top*, where *block* means either Block SDC or Glue SDC, and *top* means Chip SDC.

The process of SDC integration involves the analysis of the provided Block or Glue SDC checking against any existing Chip SDC, and decision making that can lead to Block or Glue SDC constraints being ignored or promoted. After an integrated SDC file is created, hierarchical rules should be checked and any messages diagnosed. When all the new constraints are found to be correct, the written SDC file can be appended to the original partial Chip SDC.

If some constraints were not properly integrated, the integration rules configuration should be adjusted and the integration step should be executed again. As a result, the final integration rule configuration represents a tailored integration methodology that can be reused for repeating the whole process consistently, as opposed to user-driven manual changes to the SDC file.

Single Pass SDC Integration Flow

This is a minimal sequence of commands (dofile) attempting SDC integration in a single pass.

```
read library ...
read design ...
add rule set myruleset
add integration_rule instance -default
read hierarchical sdc -sdc_design block1 blk1.sdc \
                      -sdc_design block2 blk2.sdc \
                      -sdc_design ||     glue.sdc \
                      -sdc_design /      partial_chip.sdc
set clock equivalence tclk1 -sdc_design block1 clk1 \
                            -sdc_design block2 vclk1
...
set system mode verify
integrate chip.sdc
report rule check ...
```

Incremental SDC Integration

In some cases, attempting to integrate all the constraints in a single is not practical. You will likely start with some initial, partial SDC file for the chip level, and then add different kinds of constraints to it in separate steps, checking each step's results before going on to the next step. The logical sequence is:

1. `set_case_analysis`—to set the constants that define the current mode.
2. clocks—rely on `set_case_analysis` for proper propagation.
3. I/O delays—need clocks to be specified
4. timing exceptions—need clocks and I/O delays if using `-from` or `-to` clocks.
5. other constraints

To implement this incremental SDC integration, the file created by each step will become the partial chip-level SDC, an input for the next step. Any given Chip constraints (read into SDC design '/') will be considered Golden and preserved unmodified during SDC integration.

Integration Rule Configuration Using Rule Instances

Each type of constraint is integrated by an integration rule, for example `CCD_SDC_INT1` (for `set_case_analysis`). These rules have configuration parameters, such as variables with predefined names and types. Through instances, you can define different cases for the constraints being integrated and specify how they should be handled.

To configure a rule, define rule instances with a name, a priority, and a list of settings to configuration parameters, which can be conditions or actions. Each rule instance must have settings for one or more conditions and action settings. When an SDC constraint satisfies all the conditions of a rule instance, the rule acts on that constraint based on the instance's action settings. Typical action settings are:

- Give a warning or not. If necessary, you can specify an instance-specific message when defining the rule instance.
- Promote the constraint.
- Other rule-specific options regarding how to promote a constraint.
- Do not promote the constraint
- Abort SDC integration (for example, if a contradictory `set_case_analysis` commands is detected).

If two instances conflict (if one says promote and the second says nopromote), rule instance priorities apply. If there is a conflict between two instances with the same priority, the constraint is not promoted.

By default, because there are no instances defined, the `INTEGRATE` command will not promote any constraint. You must define the integration rule instance to configure the process. The software provides a default set of rule instances in the form of a dofile that you can either copy, modify, and use independently, or source directly using the `ADD INTEGRATION_RULE INSTANCE -default` command.

There are two main integration approaches that influence many decisions on how certain attributes are promoted and how certain conflicts need to be handled: top-down and bottom-up. The default rule configuration file reflects these two approaches, under the control of the `integ_method` Tcl variable.

Top-Down Approach

- The full Glue constraints are GIVEN, as well as the Block constraints.
- The block constraints are promoted and stitched to the Glue constraints and written out as a Chip-level SDC.
- If the software finds any conflicts or any missing constraints in the Glue, it will give a warning, but it will not automatically change or add any of the top-level constraints provided as Glue, and it will not add any new top-level constraints.
- This methodology is activated by setting `integ_method` to `topdown`.

Bottom-Up Approach

- Glue constraints are optional.
- The block constraints are promoted and stitched to any available Glue constraints and written out as a Chip level SDC.
- If the software finds any missing constraints in the Glue, that can be derived from block constraints (for example, a top-level clock driving a block clock), it will add the missing top-level constraint to Chip.
- If it finds a conflict between Glue and blocks, it might give a warning or automatically resolve the conflict if the situation is clear.
- This methodology is activated by setting `integ_method` to `bottomup`.

Mapping Block Clocks to Top-Level Clocks

The goal of clock integration is to have a top-level equivalent clock for every glue or block clock. All other constraints related to block clocks will be promoted referring to their top-level equivalent. For example, if block clock `bCLK` becomes equivalent to top-level `tCLK`, then a block SDC command ‘`set_input_delay -clock bCLK ...`’ can become ‘`set_input_delay -clock tCLK...`’ in the written chip SDC.

If an initial partial chip SDC is provided, where some, or all, top-level clocks are defined, use the `SET CLOCK EQUIVALENCE` command to specify the top-level equivalent clock for all the glue/block clocks that are related to the provided top-level clocks.

After each integration stage (for example, after integrating clocks from glue SDC and before integrating clocks from block SDC), there is an automatic attempt to map block clocks to top-level clocks, based on the CCD parameter `SDC_HIER_CLOCK_EQUIV`, the clock names, and the clock tree connectivity.

If a block clock does not have a top-level equivalent, then some of the relevant rule instances of rule `CCD_CLK_INT1` can promote it to a top-level clock. If this does not happen, all related constraints (for example, `set_input_delay` with respect to that clock) cannot be promoted.

Finally, a dofile containing all the clock equivalences is written. Its name is the name of the written SDC file, with the addition of the suffix ‘`.clkeq`’. After reading the integrated file in Setup mode, use this dofile to obtain the proper mapping of block clocks to top-level clocks, which is needed to execute the hierarchical rules correctly.

The `REPORT CLOCK -hier` command shows the state of the clock mapping after the top-level clocks have been added to the Chip SDC and read back into CCD. The SDC Integration Window has a *Clock Eq* tab that displays this mapping and allows you to modify it. For more information, see Clock Equiv Page on page 246.

Promoting Block and Glue Constraints to Chip Constraints

Certain Block and Glue constraints can be promoted automatically. Others might have conflicts, while some have to be discarded.

For each type of constraint, and for each SDC design selected for SDC integration, CCD iterates all the deposited constraints and performs an analysis of the constraint, by applying to it the relevant rule instances.

Warnings are given using the SDC Rule mechanism. The message reported includes the name of the integration rule generating it, as well as the name of the rule instance that triggered it.

Note: The file written by the INTEGRATE command contains only constraints that were promoted from glue or block SDC to chip level. Original constraints applied at chip level in Setup mode are not written into this file.

Integration Stages (Optional)

For each integration rule (that is, for each type of constraint), the software scans all the instances and finds all those with the setting:

```
tcl_var = "stage:<stagename>"
```

It then collects the stage names specified and makes a list of stages, ordered by their first appearance in the instance list.

The integration process is repeated once for each stage, setting the Tcl variable "stage" to the current stage name before applying all the rule instances. This will affect rule instances whose condition depends on that variable.

Note: After deciding to promote or not promote a constraint is made in any stage, the constraint will NOT be considered again during later stages.

Promoting Generated Clocks

Generated clocks cannot be promoted before their master clock is mapped to a clock at chip level (when promoted or mapped manually by user commands). The clock integration procedure is repeated until no more clocks are promoted (or "nopromote"d) to allow for the master-generated-generated chains to be dealt with successfully without complicating the rule instances.

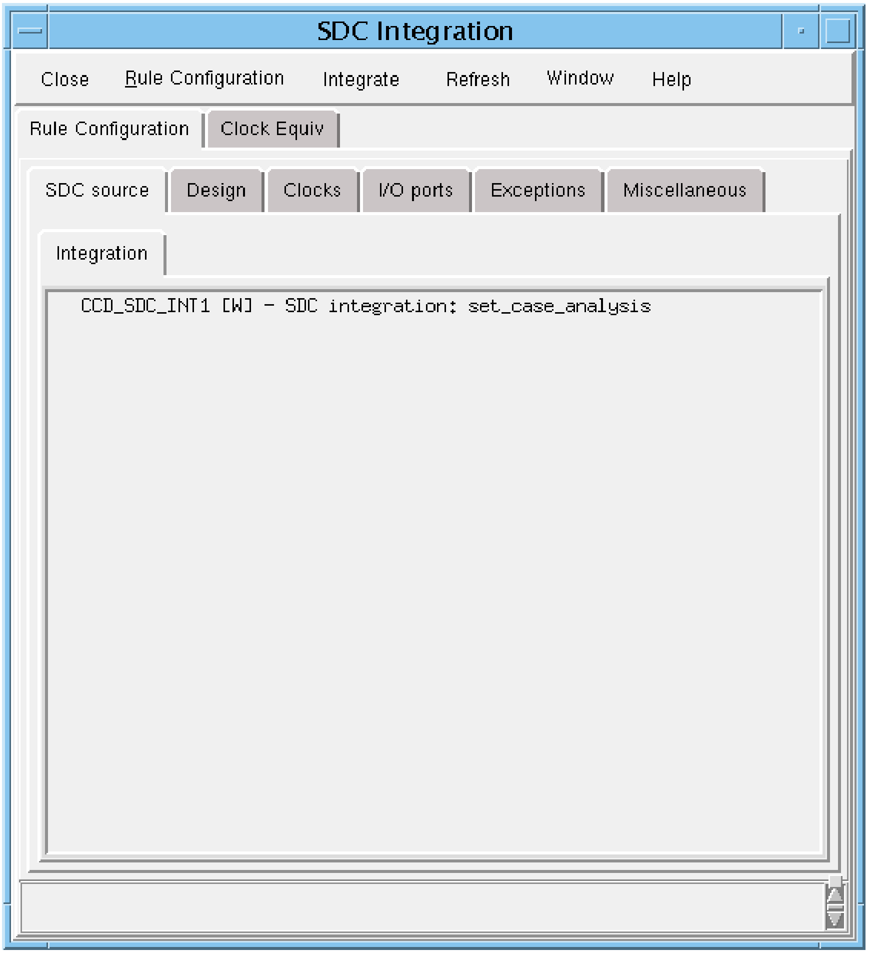
SDC Integration Window

Use the SDC Integration window to configure SDC integration rules and clock equivalences.

➤ Choose *Tools – SDC Integration.*

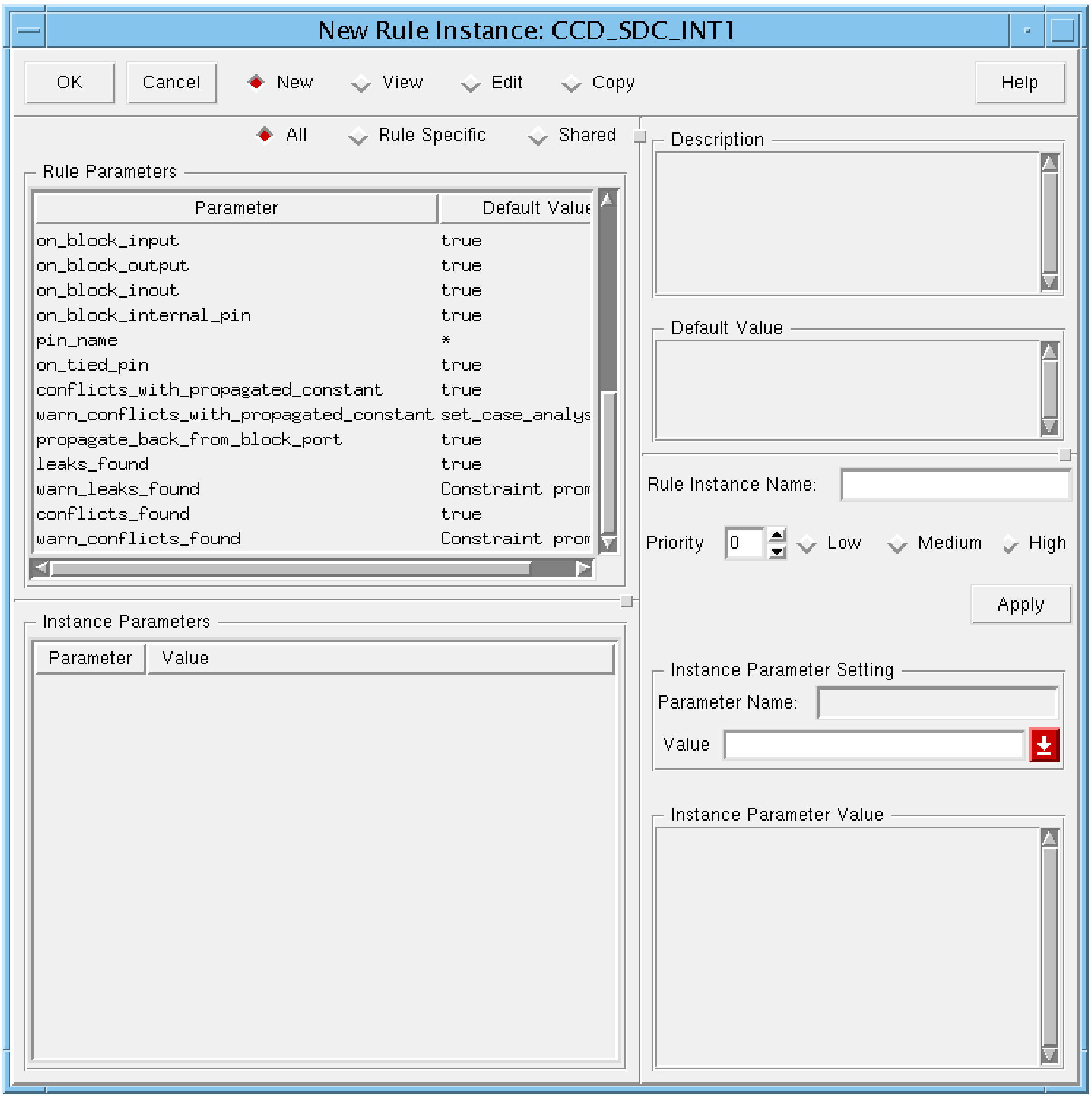
From the SDC Integration window, you can select two top-level tabs: *Rule Configuration* and *Clock Equiv*.

Rule Configuration Page



From the Rule Configuration page, you can configure the SDC Integration rules, which are organized into the same pages as in the SDC Rule Manager.

In the *Integration rules* display area, click on an integration rule to select it, right-click to open the pop-up menu, and select *New Instance* to open the New Rule Instance form.



This form is the same for viewing, editing, and copying. The form name will change to *View Rule Instance*, *Edit Rule Instance*, and *Copy Rule Instance*, respectively.

Some rule parameters are shared by all rules, while some are only relevant for specific rules. By default, the form displays all the rule parameters available for the rule being configured. You can filter the display by selecting *Rule Specific* or *Shared*.

Modifying Rule Instances

To add a rule parameter setting to the instance being edited, click on an entry in the Rule Parameters list to select it, right-click to open the pop-up menu, and select *Add Parameter*. This adds the parameter to the *Instance Parameters* list.

The following describes the fields and options for modifying rule instances:

- *Rule Instance Name* — Specifies the name of the instance to be added.
- *Priority* — Specifies a numerical priority for the instance. If a conflict between two instances is found, the one with higher priority prevails and a warning is issued. To avoid having too many messages, avoid using priorities when possible, by specifying instances with mutually exclusive conditions. However, sometimes this is not practical. For example, to override a general rule instance for a constraint on a particular object, it is easier to add an instance with a higher priority than to modify the general rule instance.

  You can type in the value, use the arrow keys, or select one of the following:

  - *Low*—value 0.
  - *Medium*—value 5.
  - *High*—value 10

Setting Values to Rule Parameters

The following describes the fields and options for changing rule parameter values:

- *Parameter Name* — Displays the name of the parameter that is selected in the *Instance Parameters* list.
- *Value* — Displays the current value setting for the parameter. Use the pull-down menu to select an alternative. For some parameters, the value field can be edited directly.
- *Apply* — Applies the changes for the instance parameter settings.

To delete the parameter from the *Instance Parameters* list, click on the parameter to select it, right-click to open the pop-up menu, and select *Remove Parameter.*

Editing Rule Instances

You can edit a rule instance by clicking the *Edit* radio button in the Rule Instance form (as long as you are not creating a new rule instance), or you can access the Edit Rule Instance form from the SDC Integration window as follows:

1. Click the square-enclosed (+) icon to show the rule instances.
2. Right-click on the rule instance to be edited to open the pop-up menu.
3. Select *Edit*.

Copying New Rule Parameters

You can copy a rule instance by clicking the *Copy* radio button in the Rule Instance form (as long as you are not creating a new rule instance), or you can access the Copy Rule Instance form from the SDC Integration window as follows:

1. Click the square-enclosed (+) icon to show the rule instances.
2. Right-click on the rule instance to be copied to open the pop-up menu.
3. Select *Copy*.

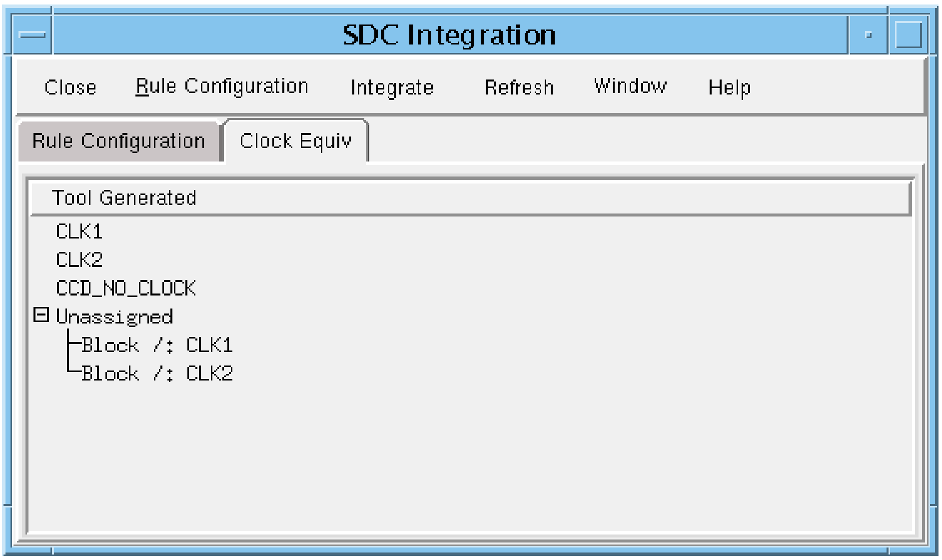
In the Copy Rule Instance form, you must change the name in the *Rule Instance Name* field. Click *Apply*.

Related Rule Occurrences

Displays the occurrences related to the selected rule instance. From here you have the same capability as that of the SDC Rule Manager. Conversely, in the SDC Rule Manager, rule occurrences generated by a rule instance have an option in their pop-up menu to view that rule instance.

Clock Equiv Page

Select the *Clock Equiv* tab to view the clocks defined at the chip-level (SDC Design '/') together with the clocks defined in the sub-designs, for which they are considered top-level equivalent.



The mapping of clocks is defined using the `CCD_HIER_CLOCK_EQUIV` CCD parameter or the `SET CLOCK EQUIVALENCE` command.

Setting Target Clock Equivalence

Click on a chip-level clock to select it, right-click to open the pop-up menu, and select *Set Target Clock Equivalence*.

Resetting Target Clock Equivalence

Click on a chip-level clock to select it, right-click to open the pop-up menu, and select *Reset Target Clock Equivalence*.

Setting Clock Equivalence

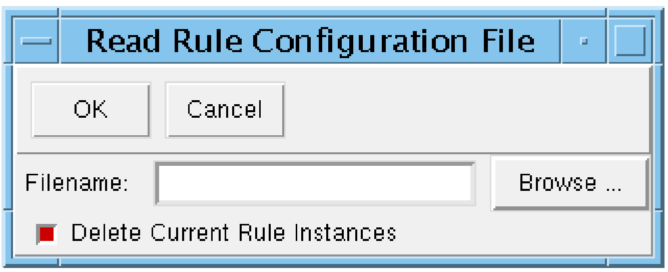
You can set the top-level equivalent clock for any block clock. This overrides the equivalences set by the `SDC_HIER_CLOCK_EQUIV` parameter for the specified block clocks.

To do this, click on a block clock to select it (for example, one under *Unassigned*), right-click to open the pop-up menu, and select *Set Clock Equivalence*.

Read Rule Configuration Menu

From the SDC Integration window, click on the *Rule Configuration* menu to select from the following options:

- Read Default—Runs the default rule configuration dofile.
- *Read*—Opens the Read Rule Configuration File form.

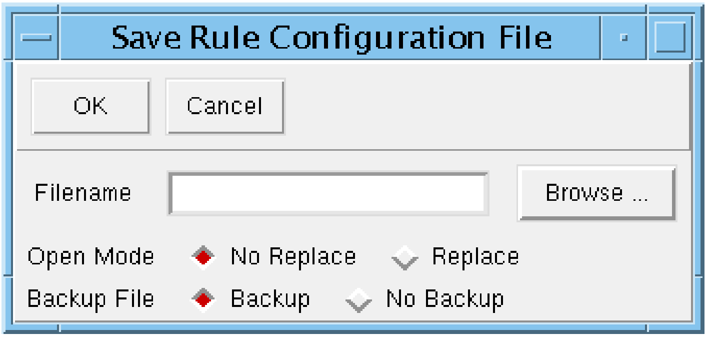


The following describes the Read Rule Configuration File fields and options:

Filename	Specifies the name of the rule configuration file. You can enter the name of the file, or click *Browse* and select a file from the Select Rule Configuration File window.
Delete Current Rule Instances	Deletes existing rule instances before the reading the rule configuration file.

- *Delete Existing Rule Instances*—Deletes all the existing rule instances in the window.

- *Save*—Opens the Save Rule Configuration File form.

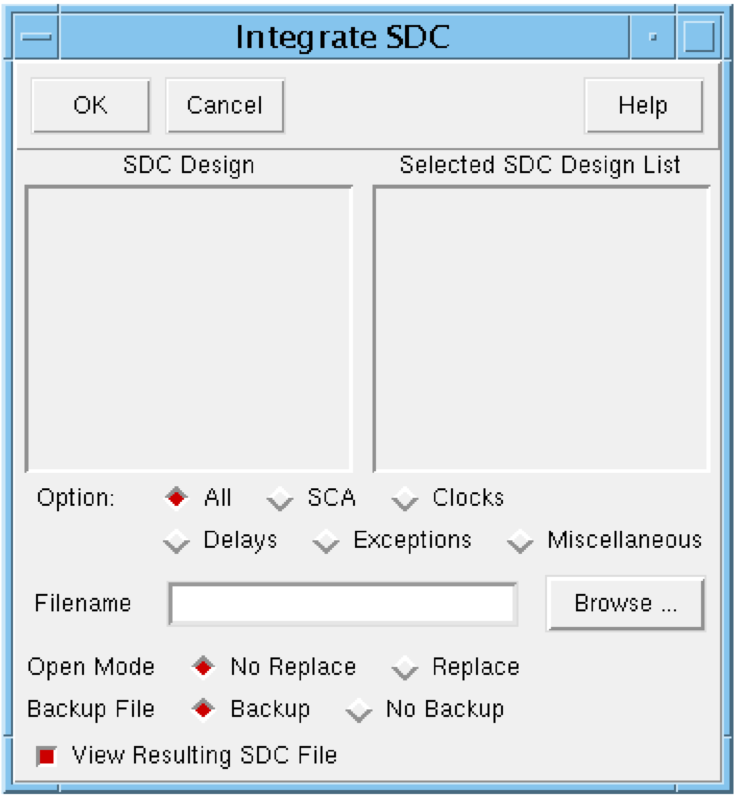


The following describes the Save Rule Configuration File fields and options:

Filename	Specifies the name of the configuration file. You can enter the name of the file, or click *Browse* and select a file from the Select Rule Configuration File window.
Open Mode	*NoReplace* (the default) does not overwrite an existing configuration file with the same name. *Replace* overwrites an existing configuration file with the same name.
Backup File	*Backup* (the default) backs up the existing configuration file by adding a tilde (~) to its name before overwriting it. *No Backup* does not back up the existing configuration file before overwriting it.

Integrate SDC

From the SDC Integration window, click on *Integrate* to open the Integrate SDC form to perform SDC integration using the current rule configuration.



All	Performs integration of all the SDC constraints.
SCA	Performs integration of `set_case_analysis`.
Clocks	Performs integration of clocks.
Delays	Performs integration of input and output delays.
Exceptions	Performs integration of timing exceptions.
Miscellaneous	Performs integration of other constraints.
Filename	Specifies the name of the integrated SDC file. You can enter the name of the file, or click *Browse* to use the Integrate SDC Output File window to select a file.

Open Mode	*NoReplace* (the default) does not overwrite an existing SDC file with the same name. *Replace* overwrites an existing SDC file with the same name.
Backup File	*Backup* (the default) backs up the existing SDC file by adding a tilde (~) to its name before overwriting it. *No Backup* does not back up the existing SDC file before overwriting it.
View Resulting SDC File	Opens the Source Code viewer and displays the SDC file result.

Timing Exception Leak and Conflict Checks

When considering the promotion of a timing exception from block or glue SDC to chip level, you should test whether it would "leak" or "conflict" with other existing block/glue constraints.

- *leak to block*—a promoted exception matches a chip-level path, that is partially or totally within a block where there is no matching exception.
- *leak to glue*—a promoted exception matches a chip-level path, that is not totally within any block, where there is no matching exception in the provided Glue SDC (the SDC file read for SDC design 'II').
- *conflict with block*—a promoted exception matches a chip-level path, that is partially or totally within a block where it is matched by a conflicting exception (i.e., one of different type or value).
- *conflict with glue*—a promoted exception matches a chip-level path, that is not totally within any block, that is matched by a conflicting exception provided in Glue SDC (the SDC file read for SDC design 'II').

To access the checks, the instances of SDC integration rule `CCD_EXC_INT1` can use the `forall_path` and `exists_path` integration rule configuration parameters. You can use these parameters multiple times in the same instance. Their values are strings that specify a condition (a Boolean expression).

When an original timing exception is *promoted* by a rule instance, a promotion candidate exception is created and is subjected to the evaluation of the path conditions specified.

A `forall_path` condition is true if all chip-level timing paths matching the candidate exception satisfy the expression. The Constraint Designer examines every promotion candidate by looking at all its matching paths to determine if the condition is true. Furthermore, if there is no matching path for the candidate exception, any `forall_path` condition is true, regardless of its expression.

An `exists_path` condition is true if and only if there exists at least one matching path satisfying the condition.

The `forall_path` and `exists_path` conditions are specified by assigning an expression to these parameters.

An expression consists of operators, constant numbers, block names, and functions, as shown in the following example:

```
add integration_rule instance CCD_EXC_INT1 sample_instance \
  from_block \
  promote \
    forall_path = "!exc_miss_in(glue)" \
```

```
    description = "This instance promotes exceptions from all the blocks, and
checks that the promoted exception does not leak to paths in glue logic"
```

The following sections describe the syntax of the expressions.

Operators

listed in descending order of precedence. Those with the same precedence are listed together, and associate in left-to-right order.

<table>
<tr><td>Parentheses</td><td><code>()</code></td><td>expression grouping</td></tr>
<tr><td>Unary Operators</td><td><code>!</code></td><td>logical not</td></tr>
<tr><td></td><td><code>-</code></td><td>negative value</td></tr>
<tr><td>Binary Operators</td><td><code>*</code></td><td>multiply</td></tr>
<tr><td></td><td><code>/</code></td><td>divide</td></tr>
<tr><td></td><td><code>+</code></td><td>add</td></tr>
<tr><td></td><td><code>-</code></td><td>subtract</td></tr>
<tr><td></td><td><code>></code></td><td>greater than</td></tr>
<tr><td></td><td><code>>=</code></td><td>greater than or equal</td></tr>
<tr><td></td><td><code><=</code></td><td>smaller than</td></tr>
<tr><td></td><td><code><</code></td><td>smaller than or equal</td></tr>
<tr><td></td><td><code>==</code></td><td>equal</td></tr>
<tr><td></td><td><code>!=</code></td><td>not equal</td></tr>
<tr><td></td><td><code>&&</code></td><td>logical AND</td></tr>
<tr><td></td><td><code>||</code></td><td>logical OR</td></tr>
<tr><td></td><td><code>-></code></td><td>implication ((X->Y) == (!X || Y))</td></tr>
<tr><td>Ternary Operator</td><td><code>?:</code></td><td>conditional expression. Cond?A:B equals A if Cond is true, otherwise it equals B.</td></tr>
</table>

Constant Numbers

A constant real number (decimal point is optional for whole numbers)

Block Names

A name of a block in the design. A block is a sub-instance declared as an SDC design (by using the `ADD SDC DESIGN <instance>` command). Wildcards are not supported when specifying a block name. However, there are seven keywords to match more than one block name, glue, or partial chip:

`chip`	partial chip SDC (on chip SDC design '/')
`glue`	glue SDC (on glue SDC design '//')
`any`	partial chip, glue or any block
`any_other`	partial chip, glue or any block but not the block where the candidate comes from
`any_block`	any block SDC
`any_other_block`	any block SDC but not the block where the candidate comes from
`source_sdc_design`	represents the SDC design from which the candidate exception is taken

Functions

The following functions are supported for `forall_path` and `exists_path` expressions. They are categorized into four groups: path shape, interexception event, candidate type, and path specification.

Path Shape

`path_num_boundary_crossing`	
	An integer number, indicates how many times a matching path crosses block boundaries. Going into a block counts 1 and leaving counts another 1.
`path_is_all_glue`	A Boolean, indicates if a path lies on glue only.
`path_is_all_block`	A Boolean, indicates if a path lies completely within a block.
`path_traverse(B)`	A Boolean function of an SDC design B, indicates if a path traverses design B. The seven keywords are applicable to B

Inter-Exception Event

`exc_miss_in(B)`	A Boolean function of a design B, indicates if the given candidate matches a path which does not match any exception in the SDC of design B. The seven keywords are applicable to B.
`exc_conflict_in(B)`	A Boolean function of a design B, indicates if the given candidate matches a path where it conflicts with the exception with highest priority in the SDC of design B. The seven keywords are applicable to B. **Note:** Because they have the highest priority, `set_false_path` candidates cannot satisfy this function within the same SDC where they originate.
`conflict_priority(B)`	An integer function of a design B, indicates the priority of the conflicting exception (from SDC design B) when any conflicts happen. If the given candidate conflicts with more than one exception, the highest priority is indicated. The seven keywords are applicable to B.

Candidate Type

`from_design(B)`	A Boolean function of a design B, indicates if the given candidate comes from design B. The seven keywords are applicable to B.
`is_set_fp`	A Boolean, indicates if the given candidate is a `set_false_path`.
`is_set_mcp`	A Boolean, indicates if the given candidate is a `set_multicycle_path`.
`is_smd`	A Boolean, indicates if the given candidate is a `set_max/min_delay`.
`value`	a real number. If the given candidate is a `set_multicycle_path`, value is the multiplier. If it is a `set_max/min_delay`, value is the delay.
`is_apply_setup_rise`	A Boolean, indicates if the given candidate is applied to setup/rise.

`is_apply_setup_fall`	A Boolean, indicates if the given candidate is applied to setup/fall.
`is_apply_hold_rise`	A Boolean, indicates if the given candidate is applied to hold/rise.
`is_apply_hold_fall`	A Boolean, indicates if the given candidate is applied to hold/fall.
`priority`	An integer number, indicates the priority of the given candidate.

Path Specification

`clock_in_from`	A Boolean, indicates if the path specification of the given candidate has from-list and there exists a clock in the list.
`port_in_from`	A Boolean, indicates if the path specification of the given candidate has from-list and there exists a top-level port in the list.
`block_pin_in_from`	A Boolean, indicates if the path specification of the given candidate has from-list and there exists a block boundary pin in the list.
`obj_in_from(B)`	A Boolean function of design B, indicates if the path specification of the given candidate has from-list including a timing object in SDC design B. The seven keywords are applicable to B.
`clock_in_through`	A Boolean, indicates if the path specification of the given candidate has through-list and there exists a clock in the list.
`port_in_through`	A Boolean, indicates if the path specification of the given candidate has through-list and there exists a top-level port in the list.
`block_pin_in_through`	A Boolean, indicates if the path specification of the given candidate has through-list and there exists a block boundary pin in the list.
`obj_in_through(B)`	A Boolean function of design B, indicates if the path specification of the given candidate has through-list including a timing object in SDC design B. The seven keywords are applicable to B.

`clock_in_to`	A Boolean, indicates if the path specification of the given candidate has to-list and there exists a clock in the list.
`port_in_to`	A Boolean, indicates if the path specification of the given candidate has to-list and there exists a top-level port in the list.
`block_pin_in_to`	A Boolean, indicates if the path specification of the given candidate has to-list and there exists a block boundary pin in the list.
`obj_in_to(B)`	A Boolean function of design B, indicates if the path specification of the given candidate has to-list including a timing object in SDC design B. The seven keywords are applicable to B.

Examples:

- The following condition is true if, and only if, the candidate exception does not leak to any SDC design for any path it matches.

  ```
  forall_path = "!exc_miss_in(any)"
  ```

- The following condition is true if, and only if, the candidate exception comes from SDC design `blk` and it does not leak into any other blocks.

  ```
  forall_path = "design_from(blk)&&!exc_miss_in(any_other_block)"
  ```

- The following condition is true if, and only if, the candidate exception is `set_false_path` and it matches some path where the matching exception with the highest priority in SDC design `blk` is conflicting with the promotion candidate.

  ```
  exists_path = "is_set_fp && exc_conflict_in(blk)"
  ```

- The following condition is true if, and only if, the candidate exception is `set_multicycle_path` with multiplier less than or equal to 2, and it matches some path where the matching exception with the highest priority in SDC design "`/`" is conflicting with the promotion candidate.

  ```
  exists_path = "is_set_mcp && value<=2 && exc_conflict_in(chip)"
  ```

Rule Configuration Parameters

The parameters listed below are used to create rule instances for configurable rules. In this release, only SDC Integration rules can be configured by the definition of rule instances.

- CCD_SDC_INT1 Configuration Parameters on page 261
- CCD_CLK_INT1 Configuration Parameters on page 263
- CCD_IO_INT1 Configuration Parameters on page 266
- CCD_EXC_INT1 Configuration Parameters on page 267
- CCD_MISC_INT1 Configuration Parameters on page 269

Understanding the Rule Parameter Characteristics

In the following tables, each parameter is followed by some characteristics and a description. The characteristics have the following meaning:

- `unique`

  The parameter can be assigned only once in any given rule instance.

- `multiple`

  The parameter can be assigned many times in the same rule instance.

- `free text`

  The parameter can be assigned any string, and not just one out of a list of available preset values.

- `boolean`

  The parameter can only be assigned 'true' (the default value) or 'false,' which is the same as `-NOT` before the parameter name.

- `pattern`

  The value is a wildcard pattern. Use '^pat' for negating a pattern, or '{pat1,pat2,pat3}' for alternative patterns.

- `candidate`

  The parameter is applied to the candidate SDC constraint that is the promoted version of an original constraint.

Note: The default value for a parameter is the first, or only, value listed under Preset Values. If no preset value is specified in the tables below, then the default value is the empty string. If a rule instance does not specify a parameter, then that parameter is completely irrelevant (a "don't care") for that rule instance. Only if the instance specifies just the parameter name (without assigning any value), then the default value will be used.

Exception: For `boolean` *action* parameters, the default and only possible value is `true`. Specifying the `false` value for such parameters is the same as not mentioning the parameter in the rule instance.

Table 13-1 Shared Condition Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
`from_block`	`unique, free text, pattern` The constraint comes from the SDC of a block whose name matches the specified pattern.	`-hier *`
`from_glue`	`unique, boolean` Constraint is from top-module SDC (i.e., read into SDC design ||. This is the same as '`from_top`'.	true (default), false
`from_top`	`unique, boolean` Constraint is from top-module SDC (i.e., read into SDC design ||. This is the same as '`from_glue`'.	true (default), false
`tcl_var`	`multiple, free text` The value of this parameter has the format variable:value. This parameter is a condition that evaluates to 'true' if the specified Tcl variable holds the specified value.	

Parameter Name	Characteristics and Description	Preset Values
match_instance	`multiple, free text` This condition is true if the preceding instance with the specified name matched the same original constraint. **Note:** That earlier instance might evaluate a different promotion candidate than the one created by the current rule instance, unless care is taken to make sure both rule instances use the same promotion options.	
nomatch_instance	`multiple, free text` This condition is true if the preceding instance with the specified name did not match the same original constraint. **Note:** That earlier instance might evaluate a different promotion candidate than the one created by the current rule instance, unless care is taken to make sure both rule instances use the same promotion options.	
on_block_input	`unique, boolean` Constraint is set on an input port of a block.	true (default), false
on_block_output	`unique, boolean` Constraint is set on an output port of a block.	true (default), false
on_block_inout	`unique, boolean` Constraint is set on an inout port of a block.	true (default), false
on_block_internal_pin	`unique, boolean` Constraint is defined on an internal pin of a block.	true (default), false
exists_in_chip_sdc	`unique, applies to promoted candidate, boolean` The promoted constraint already exists at the chip-level, with the same value.	true (default), false

Parameter Name	Characteristics and Description	Preset Values
`conflicts_with_chip_sdc`		
	`unique, applies to promoted candidate, boolean`	true (default), false
	The promoted constraint already appears at the chip-level, with a conflicting value.	
`has_clock_without_toplevel_equiv`		
	`unique, boolean`	
	An argument of the SDC command contains a clock that does not have a top level equivalent clock defined	

Table 13-2 Shared Action Configuration Parameters

Parameter Name	Characteristics and Description	Preset Value
`promote`	`unique, boolean`	
	Specify that the candidate constraint should be kept—that is, applied to the chip level.	
`nopromote`	`unique, boolean`	
	Specify that the candidate constraint should not be kept—that is, it should not be applied to the chip level.	
`abort`	`unique, boolean`	
	Abort SDC Integration.	
`message`	`unique, free text`	
	User-defined message.	
`description`	`unique, free text`	
	This action does nothing. It can be used to document a rule instance, and also as the single ‘non-condition’ of a rule instance that would otherwise have only condition parameters.	

Parameter Name	Characteristics and Description	Preset Value
warn_conflicts_with_chip_sdc		
	unique Issues a warning for a constraint that appears at the chip level but with a conflicting value.	This constraint conflicts with one already at chip level
warn_clock_without_toplevel_equiv		
	unique, free text Issue a warning for an SDC command that has a clock which has no top level equivalent clock defined.	The SDC command has a clock (%clock_without_toplevel_equiv%), which has no top level equivalent clock defined.

CCD_SDC_INT1 Configuration Parameters

Table 13-3 CCD_SDC_INT1 Condition Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
pin_name	unique, free text, pattern set_case_analysis is defined on a pin whose name matches the specified pattern.	*
on_tied_pin	unique, boolean set_case_analysis set on a pin that has a tied value.	true (default), false
conflicts_with_propagated_constant		
	unique, boolean set_case_analysis set on a pin that has a contradicting tied value.	true (default), false

Parameter Name	Characteristics and Description	Preset Values
`leaks_found`	`unique, applies to promoted candidate, boolean` promoted `set_case_analysis` affects glue logic or block inputs that did not have `set_case_analysis` before.	true (default), false
`conflicts_found`	`unique, applies to promoted candidate, boolean` promoted `set_case_analysis` is set on a pin with conflicting tied value, or it propagates to block inputs that have conflicting `set_case_analysis` value in the block SDC.	true (default), false

Table 13-4 CCD_SDC_INT1 Action Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
`warn_conflicts_with_propagated_constant`		
	`unique, free text` Issue a warning for a `set_case_analysis` set on a pin that has a contradicting tied value.	`set_case_analysis` value conflicts with propagated constant
`propagate_back_from_block_port`		
	`unique, boolean` When promoting a `set_case_analysis` on a block port, propagate it back traversing gates with a single effective fanin.	true (default), false
`warn_leaks_found`	`unique, free text` Issue a warning for a `set_case_analysis` that, if promoted, would leak to unconstrained logic (see leaks_found parameter).	Constraint promoted from `%org_pin_name%` (`value: %org_value%`) to `%new_pin_name%` (`value: %new_value%`) is ignored because it leaks.

Parameter Name	Characteristics and Description	Preset Values
warn_conflicts_found		
	unique, free text Issue a warning for a set_case_analysis that, if promoted, would conflict with an existing set_case_analysis in chip SDC or in block SDC (see conflicts_found parameter).	Constraint promoted from %org_pin_name% (value: %org_value%) to %new_pin_name% (value: %new_value%) is ignored because it conflicts.

CCD_CLK_INT1 Configuration Parameters

Table 13-5 CCD_CLK_INT1 Condition Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
is_virtual	unique, boolean The clock is created by the create_clock command without specifying any design objects.	true (default), false
is_generated	unique, boolean The clock is created by the create_generated_clock command.	true (default), false
is_real_nongenerated		
	unique, boolean The clock is created by the create_clock command, specifying design object(s).	true (default), false
clock_name	unique, free text, pattern The original clock name matches the specified pattern.	*
pin_name	unique, free text, pattern The clock is defined on a port/pin whose name matches the specified pattern.	*

Parameter Name	Characteristics and Description	Preset Values
on_single_object	unique, boolean The clock is defined on a single object.	true (default), false
on_port_or_seqelm_output	unique, boolean The clock is defined on primary port, or on an output pin of a sequential cell.	true (default), false
only_on_tied_pin	unique, boolean The clock is defined only on pin(s) with tied value in full-chip.	true (default), false
has_top_level_equivalent	unique, boolean The clock already has a top-level equivalent clock defined.	true (default), false
is_equivalent_to_CCD_NO_CLOCK	unique, boolean The top-level equivalent of this clock is CCD_NO_CLOCK.	true (default), false
leaks_found	unique, applies to promoted candidate, boolean Promoted clock candidate affects glue logic or block inputs that were not reached by a clock before.	true (default), false
conflicts_found	unique, applies to promoted candidate, boolean Promoted clock candidate is set on a pin with conflicting clock, or it propagates to block inputs that have conflicting clock in the block SDC.	true (default), false
promoted_clock_source_is_port	unique, applies to promoted candidate, boolean Promoted clock candidate source object is chip-level port.	

Parameter Name	Characteristics and Description	Preset Values
gclk_master_has_top_level_equivalent		
	unique, boolean The clock is generated, and its root master clock has a top-level equivalent clock defined (that is not CCD_NO_CLOCK). **Note:** If the master clock is also generated, then the root master is the first clock in the whole generation chain.	true (default), false
clock_exists_at_chip_level		
	unique, free text The specified clock exists in chip SDC. It could be a clock in the original SDC file read into SDC design '/', or a promoted clock.	'%org_clock_name%'

Table 13-6 CCD_CLK_INT1 Action Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
propagate_back_from_block_port		
	unique, boolean When promoting a clock defined on a block port, propagate it back traversing gates with a single effective fanin.	true (default), false
write_template	unique, boolean If propagation of a clock backwards from a block port fails due to multiple fanins, or if it reaches a sequential cell output, write templates for all the possible clock definitions as a comment following the promoted clock in the SDC file.	false (default), true

Parameter Name	Characteristics and Description	Preset Values
set_clock_name	unique, free text Specify the new clock name for the promoted clock. Note: this name might be prefixed by the block name if there is a name collision.	%org_clock_name%
set_clock_equivalence	unique, free text Specify an existing chip-level clock as the top-level equivalent clock.	%org_clock_name%
warn_leaks_found	unique, free text Issue a warning for a clock that, if promoted, would leak to additional logic (see leaks_found parameter).	The clock %org_clock_name% (from %sdc_design%), would be promoted from %old_pin_name% to %new_pin_name%, but it is ignored because it leaks.
warn_conflicts_found	unique, free text Issue a warning for a clock candidate that, if promoted, would conflict with an existing clock in chip SDC or in block SDC (see conflicts_found parameter).	The clock %org_clock_name% (from %sdc_design%), would be promoted from %old_pin_name% to %new_pin_name%, but it is ignored because it conflicts.

CCD_IO_INT1 Configuration Parameters

Table 13-7 CCD_IO_INT1 Condition Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
command	unique This parameter makes the rule instance specific for one SDC command: set_input_delay or set_output_delay.	set_input_delay, set_output_delay

Parameter Name	Characteristics and Description	Preset Values
pin_name	unique, free text, pattern The delay is defined on a port/pin whose name matches the specified pattern.	*
can_propagate_from_block_port_to_toplevel_port	unique, boolean The delay is defined on a block port connected to a single top-level port by gates with a single effective fanin.	true (default), false

Table 13-8 CCD_IO_INT1 Action Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
propagate_from_block_port_to_toplevel_port	unique, boolean When promoting a delay on a block port, propagate it to a single port traversing gates with a single effective fanin. This action parameter should be used in conjunction with the condition `can_propagate_from_block_port_to_toplevel_port`.	true (default), false

CCD_EXC_INT1 Configuration Parameters

Table 13-9 CCD_EXC_INT1 Condition Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
from_obj	unique, free text, pattern The timing exception has a `-from` list including an object whose name matches the specified pattern.	*

Parameter Name	Characteristics and Description	Preset Values
through_obj	`multiple, free text, pattern` The timing exception has a `-through` list including an object whose name matches the specified pattern.	*
to_obj	`unique, free text, pattern` The timing exception has a `-to` list including an object whose name matches the specified pattern.	*
has_rise_fall_from_to	`unique, boolean` The timing exception has one of the options `-rise_from`, `-rise_to`, `-fall_from` or `-fall_to`.	true (default), false
from_clock	`unique, boolean` The timing exception has a clock object in the `-from` list.	true (default), false
to_clock	`unique, boolean` The timing exception has a clock object in the `-to` list.	true (default), false
is_false_path	`unique, boolean` The timing exception is a `set_false_path`.	true (default), false
is_multicycle_path	`unique, boolean` The timing exception is a `set_multicycle_path`.	true (default), false
is_max_min_delay	`unique, boolean` The timing exception is a `set_max_delay` or a `set_min_delay`.	true (default), false
promoted_exc_has_matching_path	`unique, boolean` The candidate promoted timing exception matches some chip-level timing path	true (default), false

Parameter Name	Characteristics and Description	Preset Values
forall_path	multiple, free text The value of this parameter is an expression. This parameter is a condition that evaluates to ’true’ if all the top-level paths that match the promoted candidate exception satisfy the expression	”
exists_path	multiple, free text The value of this parameter is an expression. This parameter is a condition that evaluates to ’true’ if there is a top-level path that matches the promoted candidate exception, that satisfies the expression.	”

Table 13-10 CCD_EXC_INT1 Action Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
report_path_condition		
	unique, free text Reports the result of each specified forall_path/exists_path condition parameter, stating if it is true, false, or not evaluated, and showing a sample path whenever it is available.	Condition: %forall_exists% Result: %result%

CCD_MISC_INT1 Configuration Parameters

Table 13-11 CCD_MISC_INT1 Condition Configuration Parameters

Parameter Name	Characteristics and Description	Preset Values
command	unique, free text, pattern The SDC command name matches the specified pattern.	*

Parameter Name	Characteristics and Description	Preset Values
orig_sdc_match	multiple, free text, pattern The original SDC command description (as shown in the SDC Command Browser) matches the specified pattern.	*
prom_sdc_match	multiple, applies to promoted candidate, free text, pattern The promoted SDC command matches the specified pattern.	*
has_illegal_object_type	unique, applies to promoted candidate, boolean An argument of the promoted SDC command contains an object whose type is illegal for that argument.	

Table 13-12 CCD_MISC_INT1 Action Configuration Parameters

14

SDC Comparison

Overview

Note: This feature requires both XL and MCC XL-Option licenses.

With the SDC Comparison feature, you can compare two SDC designs and report any difference between them. The results can also show matching SDC statements between the two SDC designs. This feature introduces the following terminology:

- Golden and Revised SDCs: Two sets of SDC designs that are subject for comparison
- Matching SDC Statement(s): The SDC statement in the Golden (or Revised) SDC design that matches with the Revised (or Golden) SDC design, and vise-versa

Designers write SDC for RTL, then the implementation team takes the RTL and SDC designs through implementation tool. Each implementation tool outputs SDC designs with gate-level netlist. SDC Comparison is performed between the input and output SDC designs with gate-level netlist.

Typically, there are different versions of SDC constraints while the design evolves. You can add or delete constraints at different stages before finalizing the design and timing. SDC Comparison can show the difference and impact between the two SDC designs.

Causes for Fail Status

The following describes possible reasons for a fail status:

- Missing constraint

  There is no matching SDC statement in the target found for specified SDC statement. For example, the target Revised SDC is missing the matching SDC statement in the specified Golden SDC.

- Missing corner/value

  The specified SDC statement in the source found matching SDC statement(s) in the target. However, the matching SDC statement(s) in the target do not cover all the corners in the source.

  For example, the following scenario would result in a fail status for the `set_false_path` statement in the Golden SDC because some corners (for example, `-fall`) covered in the Golden SDC are not found in the Revised SDC.

  - Golden SDC: `set_false_path -from x -to y`
  - Revised SDC: `set_false_path -from x -to y -rise`

- Difference in value

  The specified exception statement matches another statement in the target, but the value specified in the two statements are different.

  For example, the following scenario would result in a difference in value:

  - Golden SDC: `set_multicycle_path 2`
  - Revised SDC: `set_multicycle_path 3`

- Inactive

  SDC comparison is not performed for the specified SDC statement because it is overwritten by another statement.

Causes for Not Compared Status

The following describes possible reasons for a not compared status:

- Undefined master clock

  The specified SDC statement has a reference to a generated clock whose master clock is not defined.

- Unmapped clock reference

  The specified SDC statement has a clock reference that is not mapped. Check SDC comparison fails in `create_clock` and `create_generated_clock`.

- Not a timing path

  The specified SDC exception statement does not have any timing path.

- No match timing path

  SDC comparison can not performed for the specified exception statement due to SDC comparison fails in `create_clock` or `create_generated_clock`, `set_case_analysis` or `set_disable_timing`.

- Not run

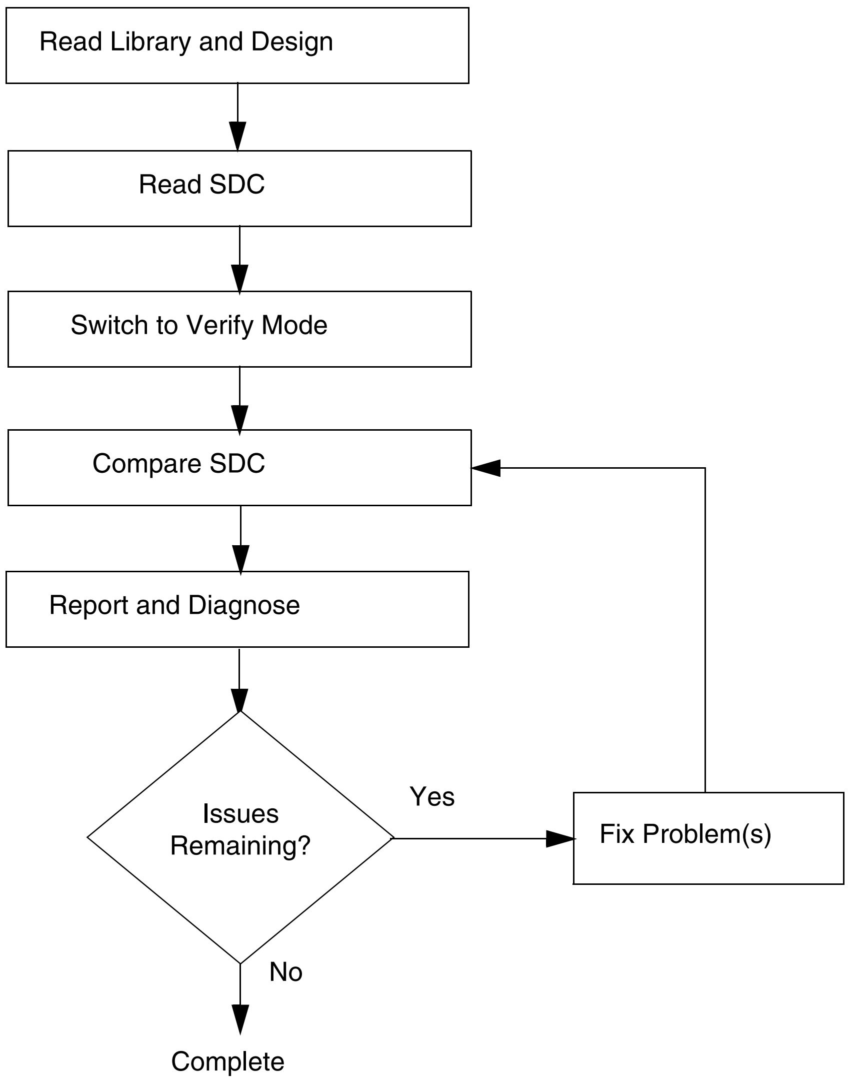
  SDC comparison was not performed.

- Not supported

  SDC comparison does not support the specified SDC statement (for example, some SDC 1.7 commands).

SDC Comparison Flow

The following shows a typical SDC Comparison flow:



The following is an example of a dofile that performs SDC comparison:

```
read library
read design
read sdc -golden <golden.sdc>
read sdc -revised <revised.sdc>
set system mode verify
```

```
compare sdc
report compared sdc
```

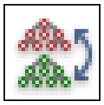
SDC Compare Manager

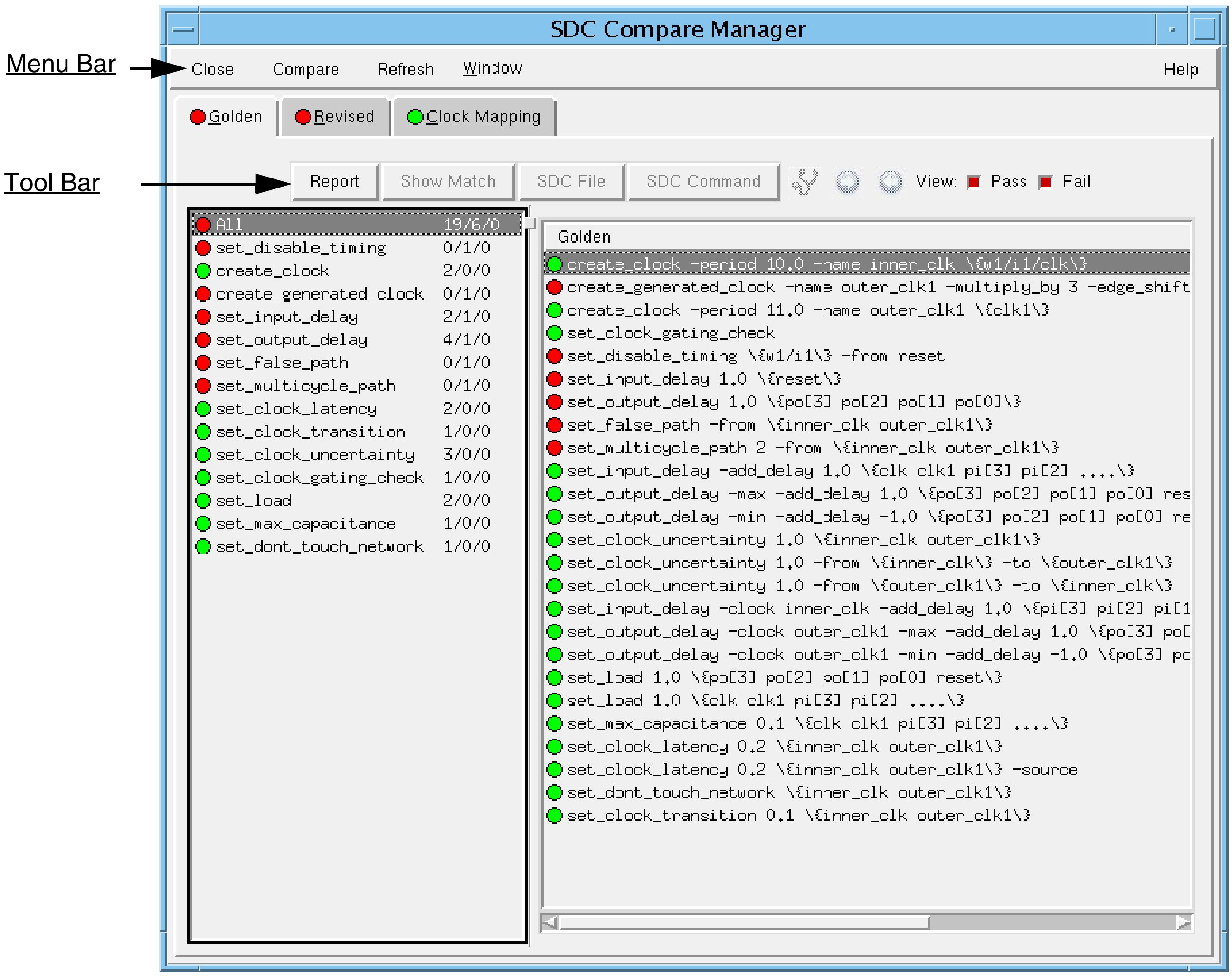
Use the SDC Compare Manager to generate SDC commands based on the provided candidate objects. There are two ways to open the SDC Compare Manager from the Main window (in Verify mode):

Note: To enable this feature, you must have both XL and MCC XL-Option licenses.

➤ Choose *Tools – SDC Compare*.

➤ Click the *SDC Compare* toolbar widget.

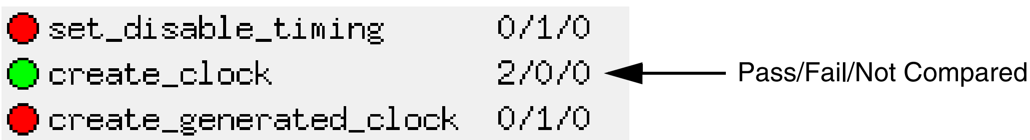




The SDC Compare Manager has three pages: Golden, Revised, and Clock Mapping.

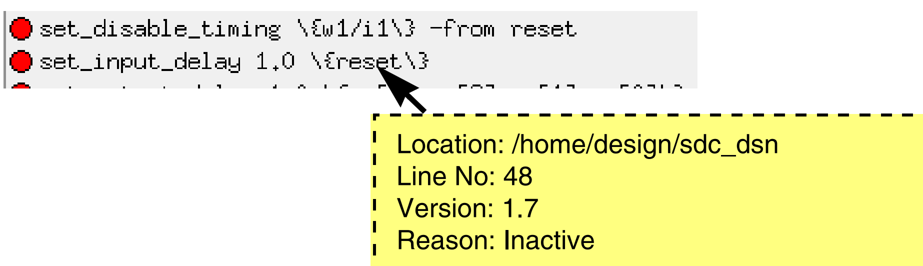
Golden and Revised Pages

The Golden and Revised pages show the SDC commands in the left window with their total results status of Fail, Not Compared, and Pass.



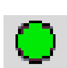
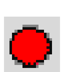
For example, `2/0/0` for `create_clock` means that two SDC statements have a status of Pass, none have a status of Fail, and none have a status of Not Compared.

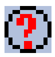
Clicking an SDC command will list the SDC statements on the right side of the SDC Compare Manager where you can identify the SDC statements with their status. Moving the cursor over the SDC statement will display the location, line number, and Version of the SDC statement. If it is in a status of Fail, it will also give the reason for the status.



SDC Status Icons

The following shows the status icons for SDC commands for the Golden and Revised pages:

Icon	Status	Description
	Pass	Indicates that all SDC statements have a status of Pass.
	Fail	Indicates that there is at least one SDC statement with a status of Fail. For a description of possible causes, see Causes for Fail Status on page 272.

Icon	Status	Description
	Not Compared	Indicates that there is at least one SDC statement with a status of Not Compared. For a description of possible causes, see Causes for Not Compared Status on page 273.

Menu Bar

The following lists the menu bar features of the SDC Compare Manager:

Close	Closes the SDC Compare Manager.
Compare	Compares the SDC designs. This runs the `COMPARE SDC` command.
Refresh	Refreshes the SDC Compare Manager window display and clears the SDC matching statement window.
Window	Lists all of the active windows for the session. Click on a window name to bring it to the front of your screen. Choose *Cascade Window* to refresh your desktop and display the main Constraint Designer window on top with all other open windows in a cascading view to the left of the main window.

Tool Bar

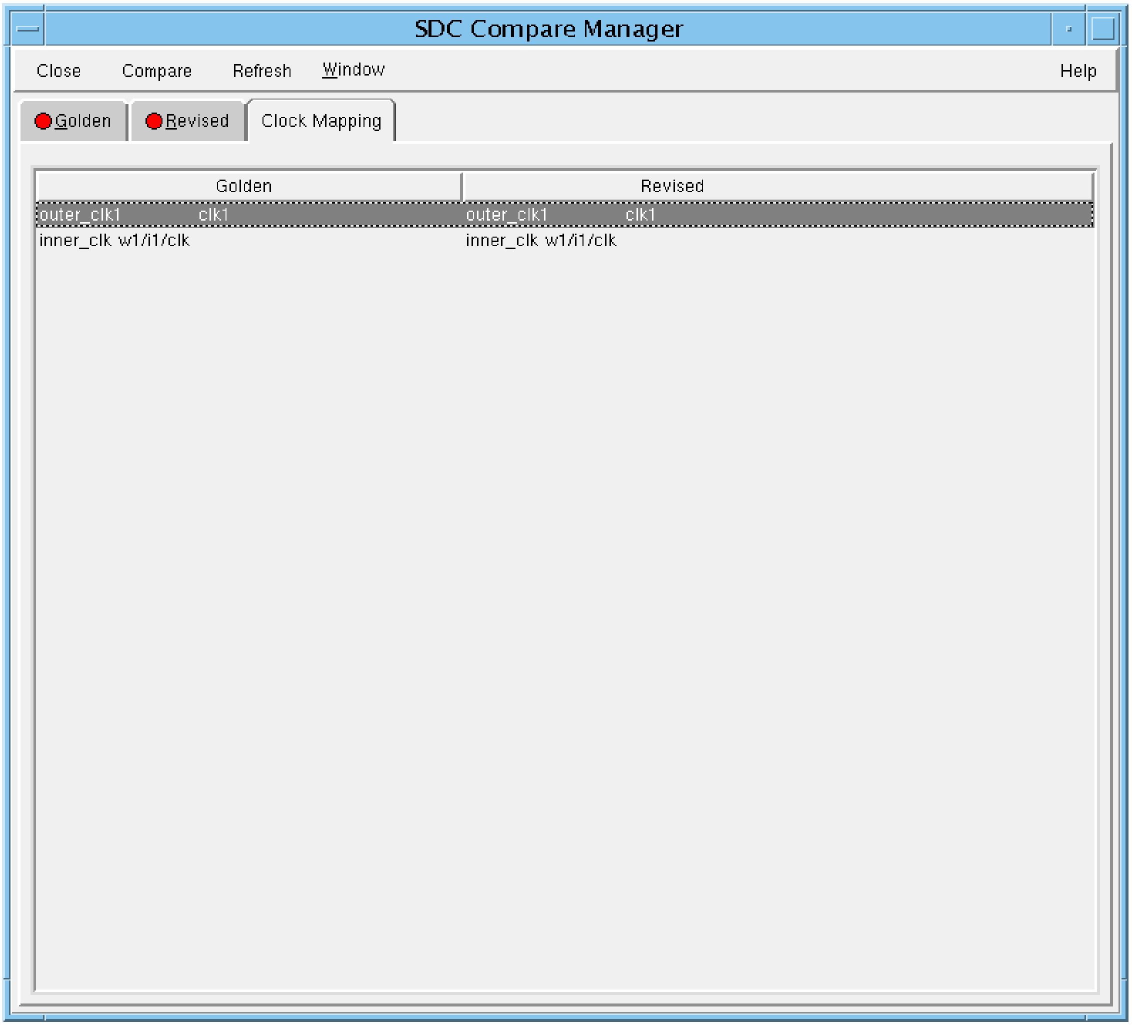
The following lists the tool bar features of the SDC Compare Manager for the Golden and Revised pages:

Report	Reports on the compared SDC design data. This runs the `REPORT COMPARED SDC` command.
Show Match	Opens a window that shows the matching Golden or Revised designs' SDC statement.
SDC File	Opens the SDC Command Browser.
	*Diagnose* icon—Runs diagnosis on the compared SDC design data. This runs the `DIAGNOSE COMPARED SDC` command.

	*Next*— Highlights the next SDC statement with a status of Fail.
	*Previous*— Highlights the previous SDC statement with a status of Fail.
View	Filters the SDC statements by status. Select a *Pass* or *Fail* view option.

Clock Mapping Page

Use the Clock Mapping page of the SDC Compare Manager to report final clock mapping information.



If clock mapping between the Golden and Revised designs is needed to continue SDC comparison, the software uses the `ADD CLOCK EQUIVALENCE` command.

Clocks that are mapped are marked green and unresolved clocks are marked red. Virtual clocks do not have expandable sign (+).

15

Accessing Objects

Using the Tcl Interface

Conformal supports two types of Tool Command Language (Tcl) commands: native Tcl commands and Conformal Tcl commands that have been tailored for use with Conformal to query the design database. Information retrieved from the design database is referenced by pointers (which are also called object handles in Tcl).

For a complete description of the Tcl design access commands and the Tcl Utility commands, see the Tclmode Commands chapter of the *Conformal Constraint Designer Reference Manual*. Each section includes the syntax for individual commands, definitions for the applicable arguments, command examples, and what Conformal returns.

The focus of the chapter is Conformal Tcl commands. Therefore, if you want to learn more about *native* Tcl commands, refer to the public Tcl manual widely available online. To see a list of supported Tcl commands, enter a question mark (?) at the Tcl prompt.

Note: This has no effect when Conformal is in the default command entry mode.

As you work with the Tcl commands, you will find that some of the commands invalidate the object handles you saved in Tcl variables. For example, when you change the design with `set root module`, every object handle is invalidated. When an object handle is invalidated, yet still referred to by a Tcl variable, the memory is not free until you reassign the Tcl variable to another value.

By its very nature, the Tcl command interface is not as efficient as internal C functions. Therefore, you will encounter some performance penalties when you access large amounts of information using Tcl commands. For example, most of the `get` commands return a `TCL LIST`, thus costing memory and speed.

Tcl Conventions

Conventions used in the Conformal Tcl command documentation differ somewhat from those used in the remainder of the manual. For example, Conformal Tcl commands are case-sensitive (you must type them in lowercase). Therefore, as a reminder, they appear in lowercase.

- `commands`
  Tcl commands appear in the text and in examples in lowercase with a Courier font. And since Conformal Tcl commands are case-sensitive, you must type them in lowercase. (However, options are not case-sensitive.) Default options are noted.
- Hierarchical context (/)
  If a name begins with a slash (/), Conformal considers the name in a hierarchical context. For example: `/U02/U199`

- Module context
  Module context operations always work on the current module. For example, `find -net zero` refers to a net named `zero` that is in the current module.
- Pin `object_type`
  Pin `object_types` appear in the format `instance_name/pin_name`. For example:
  - Pin `object_type` in module context:
    A pin named `data` on instance `U01` of the current module is specified as `U01/data`.
  - Pin `object_type` in hierarchical context:
    In hierarchical context, the string is preceded by a slash. Thus, the pin is specified as `/U01/data`.
- Wildcards: (*) and (?)
  Conformal supports the wildcard * or ? in an `object_name`, but only at the bottom hierarchical level:

  ```
  find -net /d*
  ```

  Return examples are:
  `/d1 and /d0`

Using Native Conformal Commands

When Conformal is in Tcl mode, you will run native Tcl and Conformal Tcl commands. However, you can also run native Conformal commands.

To run native Conformal commands, use an underscore for spaces in commands. With this feature, type the entire command; Conformal does not permit partial entry matching for native Conformal commands:

```
read_design counter.v
```

For more information on command entry modes, refer to "Command Entry Modes" on page 64.

To Get Quick Help for Native Conformal Command Names

If you type a native Conformal command incorrectly using the underscore, Conformal echoes commands with common prefixes. For example, type:

```
add_in
```

Conformal returns:

```
ambiguous command name "add_in": add_instance_attribute add_instance_constraints
add_instance_equivalences
```

Life Cycle of an Object Handle

As you work with the Tcl commands, you will find that some of the commands invalidate the object handles you saved in Tcl variables. For example, when you change the design with `set root module`, every object handle is invalidated. When an object handle is invalidated, yet still referred to by a Tcl variable, the memory is not free until you reassign the Tcl variable to another value.

Tcl Command Limitations

By its very nature, the Tcl command interface is not as efficient as internal C functions. Therefore, you will encounter some performance penalties when you access large amounts of information using Tcl commands. For example, most of the get commands return a Tcl LIST, thus costing memory and speed.

Ignoring Signals Along a Path

By default, when you use the `READ CRITICAL PATH` command, the Conformal Constraint Designer considers all paths within the given timing report. You can use the `skip_signal_pattern` variable before you use the `READ CRITICAL PATH` command, to ignore signals along paths that match a specified pattern.

Note the following when using this variable:

- `skip_signal_pattern` must be defined in the `cpa` namespace.
- For the specified pattern, the Conformal Constraint Designer supports glob-style string matching, where:
  - "`*`" matches 0 or more characters
  - "`?`" matches a single character
  - "`[...]`" matches a set and/or range of characters
  - "`\`" following a character indicates that it is not special
- The specified pattern is in effect for all subsequent `READ CRITICAL PATH` commands until you unset the variable.

For example, the following tells the Conformal Constraint Designer to ignore signals along paths that have `TEST_` as a part of their name.

```
TCL_SETUP>set cpa::skip_signal_pattern "*TEST_*"
```

Database Objects in Conformal Constraint Designer

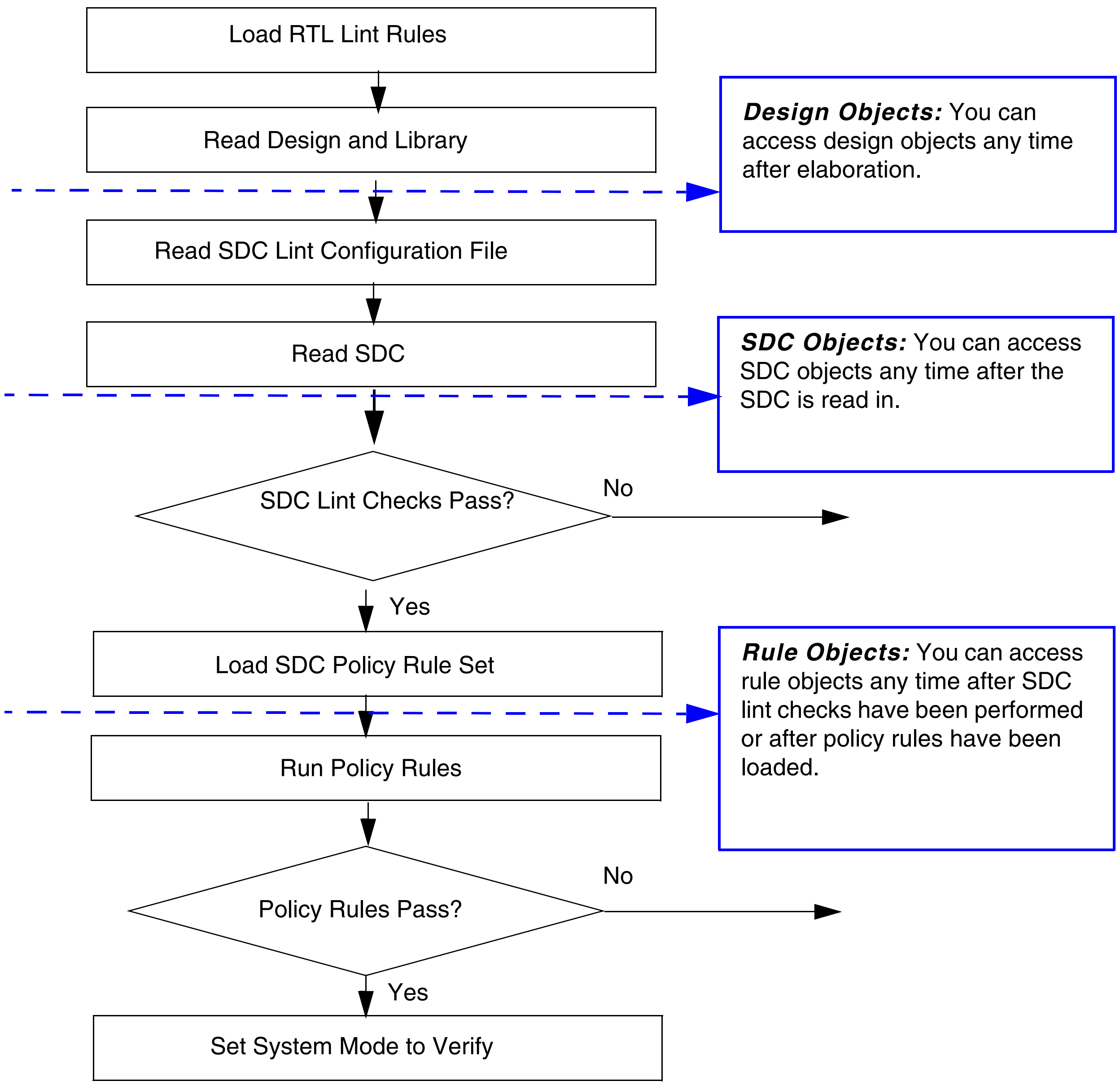
An *object* is anything Conformal Constraint Designer can manipulate, such as designs, ports, constraints, rules, and so on. Each object has a set of attributes. An *attribute* is a setting that controls how Conformal Constraint Designer operates on objects. For information on object attributes, refer to the *Conformal Constraint Designer Database Access Object and Attribute Reference*.

In Tcl mode, you have access to the following objects.

Design Related Objects	SDC Related Objects	Rule Related Objects
`design`	`sdcmode`	`ruleset`
`instance`	`sdcdsgn`	`rulegrp`
`port`	`sdcstmt`	`ruleinst`
`pin`	`sdcobj`	`rulesrc`
`net`		`occr`
`lib`		`rulefilter`
`libcell`		
`libpin`		

The following figure illustrates (using the “Rule Check Flow” on page 147) when each type of design object can be accessed.

Figure 15-1 Object Usage Model



Accessing Database Objects

This section describes how you can use various Tcl commands to access database objects and their attributes.

Using the Tcl find Command

You can use the Tcl find command to find a specific object type and to view the value of its attributes:

```
find -<object_type>
     [<patterns> | <object_list> | -of_objects <object_list>]
     [ -sensitive | -nosensitive ]
     [-hierarchical] [-filter <condition>]
```

Where `object_type` is one of the following database object types (such as `sdcdsgn`, `ruleset`, and so on).

For example:

```
set myinst [ find -ruleinst myruleset_1/grp_a/ri_1 ]
set all_rule_srcs [ find -rulesrc ]
set all_rule_insts [ find -ruleinst ]
```

Using the Tcl get_attribute and set_attribute Commands

Use the `get_attribute` Tcl command to retrieve the value of an attribute, and the `set_attribute` command to set the value of a specific attribute.

```
get_attribute <rule_obj_handle><attr_name>
set_attribute <rule_objc_handle><attr_name><value>
```

For example, the following changes the option for rule instance `s1` and then re-runs the instance:

```
#changes value of attribute cmd_name
set_attribute [find -ruleinst s1/g1/r1] cmd_name set_false_path
run_rule_check s1
```

For example:

```
get_attribute $myinst name //Retrieves instance name
get_attribute $myinst desc //Retrieves instance description
```

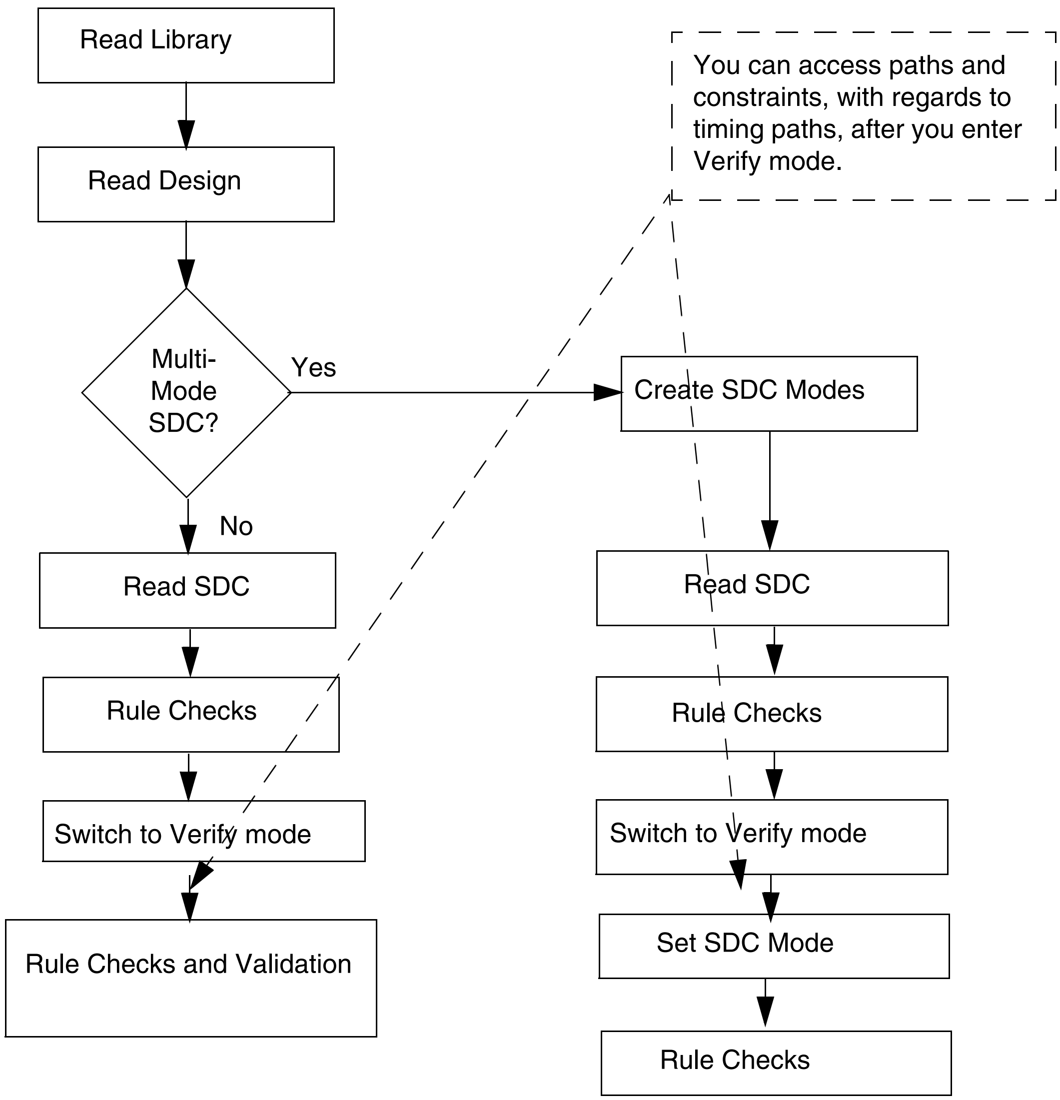
Retrieving Paths and Constraints

This section describes how you can use the `get_matching_paths` and `get_matching_constraints` Tcl commands to retrieve paths and constraints within the given SDC design, and in the current SDC mode.

Use these commands in Verify mode.

- For more information on each command's options, refer to the "Tcl Command Reference" chapter of the *Conformal Constraint Designer Reference*.
- For more information on SDC Objects (SDCOBJ), refer to the "Database Access Objects and Attributes" chapter of the *Conformal Constraint Designer Database Access Object and Attribute Reference*.

Flow for Retrieving Matching Paths and Constraints



Retrieving Paths from an SDC Design

The `get_matching_paths` Tcl command returns paths that are within a given region and that satisfy a given target path.

Syntax:

```
get_matching_paths
  <targets>
  [candidates]
  [-reference_only | -full_path]
  [-limit <natural_number>]
```

The returned value is a list of pairs of targets and their matching paths. For example:

```
{ <target1> { {from <startpoint1a> through <point1a> to <endpoint1a>} {from
<startpoint1b> through <point1b> to <endpoint1b>} } <target2> { {from
<startpoint2a> through <point2a> to <endpoint2a>} {from <startpoint2b> through
<point2b> to <endpoint2b>} } }
```

Specifying Targets

The targets should be specified as a list of exception SDCOBJ objects, a list of `group_path` SDCOBJ objects, or a list of explicit paths. No combination of these can be used at the same time. (See"Specifying Paths for get_matching_paths and get_matching_constraints" on page 293 for more information on specifying paths). For example:

a list of exception SDCOBJ objects:

```
[concat [find -sdcobj set_false_path_<id>] [find -sdcobj
set_multicycle_path_<id>]]
```

a list of group_path SDCOBJ objects:

```
[concat [find -sdcobj group_path_<id1>] [find -sdcobj group_path_<id2>]]
```

a list of explicit paths:

```
[list [list from [find -pin <startpoint>] to [find -pin <endpoint>]] [list through
[find -pin <point>]]]
```

Specifying Candidates

The candidates should be specified as a list of exception SDCOBJ objects, a list of `group_path` SDCOBJ objects, or a list of explicit paths.No combination of these can be used at the same time. For example:

a list of exception SDCOBJ objects:

```
[concat [find -sdcobj set_false_path_<id>] [find -sdcobj
set_multicycle_path_<id>]]
```

a list of group_path SDCOBJ objects:

```
[concat [find -sdcobj group_path_<id1>] [find -sdcobj group_path_<id2>]]
```

a list of explicit paths:

```
[list [list from [find -pin <startpoint>] to [find -pin
<endpoint>]] [list through [find -pin <point>]]]
```

Retrieving Constraints from and SDC Design

The `get_matching_constraints` Tcl command returns constraints that are within a specified region and that satisfy at least one path along a given target path.

Syntax:

```
get_matching_constraints
  <targets>
  [candidates]
  [-filter <filter_condition>]
  [-partial_overlap | -total_overlap]
  [-full_constraint | -winning_only]
```

The returned value is a list of pairs of targets and their matching constraints. For example:

```
{ <target1> { set_false_path_<id1> set_multicycle_path_<id1> } <target2> {
set_false_path_<id2> set_multicycle_path_<id2> } }
```

Specifying Targets

The targets should be specified as a list of exception SDCOBJ objects, a list of `group_path` SDCOBJ objects, or a list of explicit paths. (See "Specifying Paths for get_matching_paths and get_matching_constraints" on page 293 for more information on specifying paths). No combination of these can be used at the same time.

For example, a list of exception SDCOBJ objects:

```
[concat [find -sdcobj set_false_path_<id>] [find -sdcobj
set_multicycle_path_<id>]]
```

a list of group_path SDCOBJ objects:

```
[concat [find -sdcobj group_path_<id1>] [find -sdcobj group_path_<id2>]]
```

or, an explicit path:

```
[list [list from [find -pin <startpoint>] to [find -pin <endpoint>]] [list through
[find -pin <point>]]]
```

Specifying Candidates

The candidates should be specified as a list of exception SDCOBJ objects, a list of group_path SDCOBJ objects, or a list of explicit paths. No combination of these can be used at the same time. For example:

a list of exception SDCOBJ objects:

```
[concat [find -sdcobj set_false_path_<id>] [find -sdcobj
set_multicycle_path_<id>]]
```

a list of `group_path` SDCOBJ objects:

```
[concat [find -sdcobj group_path_<id1>] [find -sdcobj group_path_<id2>]]
```

a list of group_path SDCOBJ objects:

```
[list [list from [find -pin <startpoint>] to [find -pin <endpoint>]] [list through
[find -pin <point>]]]
```

Specifying Paths for get_matching_paths and get_matching_constraints

You can specify paths in two ways:

- Implicit specification through SDCOBJ objects. When you specify an SDCOBJ, the following attributes are extracted:
    - `from/through/to` attributes for the SDCOBJ object `set_false_path`
    - `from/through/to` attributes for the SDCOBJ object `set_max_delay`
    - `from/through/to` attributes for the SDCOBJ object `set_min_delay`
    - `from/through/to` attributes for the SDCOBJ object `set_multicycle_path`
    - `path_specs` attribute for the SDCOBJ object `group_path`
- Explicit specification using the following format:

```
{

[ <from | rise_from | fall_from> <object_list> ]

[ <through | rise_through | fall_through> <object_list> ]*

[ <to | rise_to | fall_to> <object_list> ]
}
```

Start points (if any)

Through points (if any)

End points (if any)

For more information on this format, see "Explicitly Specifying Paths" on page 294

Explicitly Specifying Paths

This section describes how to explicitly specify a path for the get_matching_paths and get_matching_constraints commands using the following format:

```
{
[ <from | rise_from | fall_from> <object_list> ]
[ <through | rise_through | fall_through> <object_list> ]*
[ <to | rise_to | fall_to> <object_list> ]
}
```

Start points (if any)

Through points (if any)

End points (if any)

The following table describes each portion of the path specification:

[<from | rise_from | fall_from> <object_list>]

Specifies the starting points to reference.

Where:

- rise_from indicates that a path has a rising transition at the specified object
- fall_from indicates that a path has a falling transition at the specified object

Supported design objects: ports, pins, and instances

Supported points: input/inout ports, clock pins of instances of sequential cells, data pins of instances of level-sensitive latches, output/inout pins of black boxes, pins with input delays specified, and instances of sequential cells

Supported constraint objects: SDCOBJ

Supported SDC commands: create_clock and create_generated_clock

Note: from, rise_from, and fall_from are mutually exclusive

[<through | rise_through | fall_through> <object_list>]*

Specifies the through points to reference.

Where:

- `rise_through` indicates that a path has a rising transition at the specified object
- `fall_through` indicates that a path has a falling transition at the specified object

Supported design objects: port, pin, instance, and net

Supported points: ports, pins, leaf instances, and nets

Note: `through`, `rise_through` and `fall_through` can be specified more than once

`[ <to | rise_to | fall_to> <object_list> ]`

Specifies the end points to reference.

Where:

- `rise_to` indicates that a path has a rising transition at the specified object
- `fall_to` indicates that a path has a falling transition at the specified object

Supported design objects: port, pin, and instance

Supported points: output/inout ports, input pins of instances of sequential cells, input/inout pins of black boxes, pins that have an output delay specified and instances of sequential cells

Supported constraint object: SDCOBJ

Supported SDC commands: `create_clock` and `create_generated_clock`

Note: `to`, `rise_to`, and `fall_to` are mutually exclusive

Conformal Primitive Gate Types

The following lists the primitive gate types that are used in the Conformal software:

Cell Type	Description
`ADD`	word-level addition primitive
`AND`	AND gate
`BUF`	buffer
`BUFIF0`	buffer if output is zero
`BUFIF1`	buffer if output is one
`CD`	2-input AND gate. First input is control, second input is data.
`CMOS`	complementary-symmetry metaloxidesemiconductor
`DFF`	delay flip-flop
`DIV`	divider
`DLAT`	delay latch
`EQ`	logical equality
`GE`	greater than or equal
`GT`	greater than
`INV`	inverter
`LE`	less than or equal
`LT`	less than
`MODULUS`	modulus
`MUX`	multiplexer
`MULT`	multiplier
`NAND`	NAND gate
`NE`	logical not equal
`NMOS`	n-type metal-oxide-semiconductor
`NOR`	NOR gate
`NOTIF0`	NOT if output is zero

Cell Type	Description
NOTIF1	NOT if output is one
ONECOLD	One-cold condition
ONECOLD0	Zero-one-cold condition
ONEHOT	One-hot condition
ONEHOT0	Zero-one-hot condition
OR	OR gate
PMOS	p-type metal-oxide-semiconductor
PULLDOWN	pull-down resistor
PULLUP	pull-up resistor
RCMOS	primitive which is same as Verilog’s rcmos primitive gate
REM	remainder
RNMOS	primitive which is same as Verilog’s rnmos primitive gate
ROL	rotate left
ROR	rotate right
RPMOS	same as Verilog’s rpmos primitive gate
RTRAN	same as Verilog’s rtran primitive gate
RTRANIF0	same as Verilog’s rtranif0 primitive gate
RTRANIF1	same as Verilog’s rtranif1 primitive gate
SLA	shifter left arithmetic
SLL	shifter left logical
SRA	shifter right arithmetic
SRL	shifter right logical
SUBTRACT	subtractor
TIE0	constant 1’b0
TIE1	constant 1’b1
TIEX	constant 1’bx
TIEZ	constant 1’bz
TRAN	transistor

Cell Type	Description
TRANIF0	transistor if output is zero
TRANIF1	transistor if output is one
WAND	word-level AND
WBUF	word-level buffer
WBUFIF0	word-level bufif0
WBUFIF1	word-level bufif1
WCD	word-level CD, m-bit Data, 1-bit Control
WDC	word-level DC, m-bit Data, 1-bit Control
WDFF	word-level D Flop
WDLAT	word-level D Latch
WINV	word-level inverter
WMUX	word-level MUX
WNAND	word-level NAND
WNOR	word-level NOR
WOR	word-level OR
WSEL	word-level selector
WXNOR	word-level XNOR
WXOR	word-level XOR
XNOR	XNOR gate
XOR	XOR gate

Fanin and Fanout Traversal

Fanin and Fanout traversal traces paths through combinational logic to collect the objects in the fanin or fanout of a target point. Path information can be extracted from functions and timing arcs. With fanin/fanout traversal feature, you can access the objects in the fanin/fanout of a target point, write methodology specific scripts to fulfill your implementation flows, and create, check and diagnose rules.

There are two types of fanin and fanouts: transitive and immediate. For transitive types, an object is considered to be in the transitive fanin or fanout of a target point if there is a path through combinational logic between the object and that target point. Immediate types are similar to transitive fanin and fanout, except the traversal stops at the first tier of the object level.

This document focuses on transitive fanin and fanout types, where the object level is used to manipulate the collected objects in the transitive fanin and fanout.

Note: Fanin/Fanout traversal is done for current SDC design and current SDC mode.

Traversal Categories

In structural traversal, functions take precedence over timing arcs when both are available. It traces all combinational arcs and ignores hard constant propagation and timing constraints are ignored. For example:

```
get_fanin/get_fanout <object> -structural
```

In timing traversal, timing arcs take precedence over functions when both are available. It traces all combinational arcs and considers hard constant propagation and timing constraints. For example:

```
get_fanin/get_fanout <object> -timing [-ignore_sca] [-ignore_sdt]
```

Fanin Traversal Startpoints

Fanin traversal should stop at input/inout ports, clock pins of instances of sequential cells, non-timing blackboxes (where the type is not timing), or startpoints in accordance with specified fanin traversal category.

The following objects can be valid startpoints:

- An input/inout port
- A clock pin of an instance of a sequential cell
- A data pin of an instance of a level-sensitive latch
- An output/inout pin of a blackbox
- An instance that has a pin that is considered a valid startpoint along the traversal paths

In structural traversal, the following objects can also be valid startpoints:

- A dead-end pin
- A dead-end net

In timing traversal, the following objects can also be valid startpoints:

- A clock definition point
- A pin that has input/output delay

In timing traversal which does not ignore the effect of `set_disable_timing`, the following objects can also be valid startpoints:

- A hierarchical pin that has disabled timing
- A leaf output pin that has disabled timing
- A leaf output pin whose related leaf input pins of the same instance have disabled timing

Fanout Traversal Endpoints

Fanout traversal should stop at output/inout ports, input pins of instances of sequential cells, non-timing blackboxes (where the type is not timing), or endpoints in accordance with specified fanout traversal category

The following objects can be valid endpoints:

- An output/inout port
- An input pin of an instance of a sequential cell
- An input/inout pin of a blackbox
- An instance that has a pin that is considered a valid endpoint along the traversal paths

In structural traversal, the following objects can also be valid endpoints

- A dead-end pin
- A dead-end net

In timing traversal, the following objects can also be valid endpoints

- A clock definition point
- A gating pin of an instance that has clock gating check specified
- A pin that has output/input delay

In timing traversal which does not ignore the effect of `set_disable_timing`, the following objects can also be valid endpoints:

- A hierarchical pin that has disabled timing
- A leaf input pin that has disabled timing
- A leaf input pin whose related leaf output pins of the same instance have disabled timing

Fanin and Fanout Traversal Operation Modes

Operation modes are used to determine whether fanin/fanout traversal should be performed hierarchically or not. In Hierarchical mode, both hierarchical and leaf objects can be returned. For example:

```
get_fanin/get_fanout <object> -hierarchical
```

In non-hierarchical mode, only objects at the same hierarchy level as a target point can be returned. Hierarchical objects at the same hierarchy level as a target point are considered virtual leaf objects. For example:

```
get_fanin/get_fanout <object>
```

Fanin and Fanout Traversal Object-Level Counting

Object level is used to stop traversal when a depth of search of objects is reached. For example:

```
get_fanin/get_fanout <object> ... -instance_levels <instance_depth>
get_fanin/get_fanout <object> ... -pin_levels <pin_depth>
```

Object level counting is run over objects along the traversal paths, and can be specified to run over leaf objects, hierarchical objects, or both. For example,

```
get_fanin/get_fanout <object> ... [-count_leaf | -count_hier | -count_all]
```

By default, in hierarchical mode, only leaf objects are considered in object level counting to determine the tier of object levels. For example:

```
get_fanin/get_fanout <object> -hierarchical ... \
                                    [-count_leaf | -count_hier | -count_all]
```

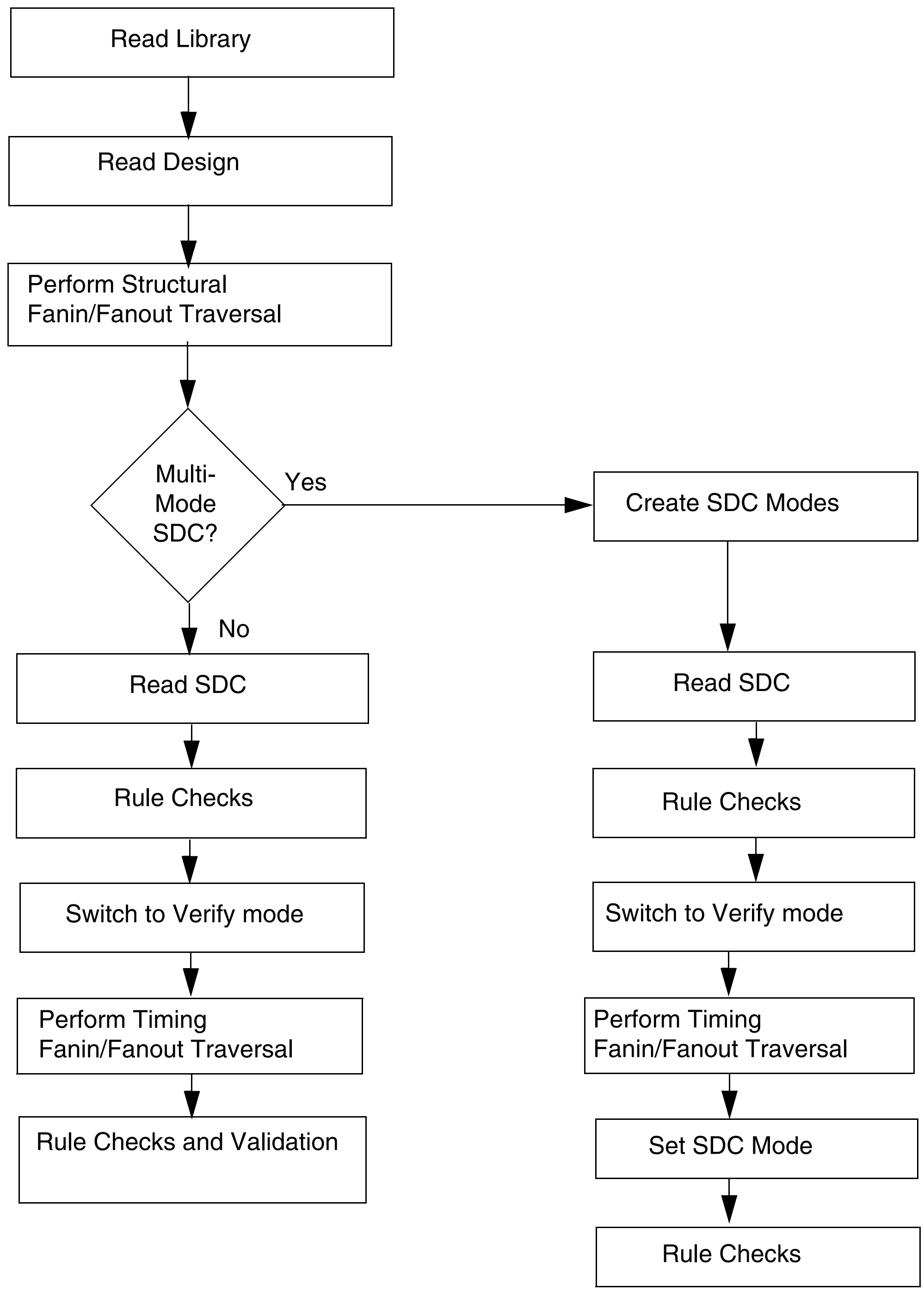
In non-hierarchical mode, only objects at the same hierarchy level as a target point are considered in object level counting to determine the tier of object levels. Hierarchical objects at the same hierarchy level as a target point are considered virtual leaf objects. For example:

```
get_fanin/get_fanout <object> ...
```

By default, in non-hierarchical mode with stepping into hierarchy, hierarchical objects at the same hierarchy level as a target point are ignored in object level counting, and leaf objects at a hierarchy level lower than that of a target point are considered in object level counting to determine the tier of object levels. For example,

```
get_fanin/get_fanout <object> ... -step_into_hierarchy \
                                    [-count_leaf | -count_hier | -count_all]
```

Fanin and Fanout Traversal Flow



16

Integration with Other Tools

RTL Compiler Integration

With RTL Compiler, you can validate and generate design constraints natively, through RTL Compiler commands, or through dofiles that can be loaded into Conformal Constraint Designer. There are multiple methods, or flows, that are available to validate constraints. These flows are discussed in the following sections.

- Validating Constraints on page 307
  - Validating Constraints with Dofiles on page 307
  - Validating Constraints Natively on page 308
- Generating Constraints on page 309
  - Generating Constraints with Dofiles on page 309
  - Generating Constraints Natively on page 311

The `write_do_ccd` command creates dofiles that Conformal Constraint Designer uses to validate and generate design constraints:

```
write_do_ccd validate [-design string] [-logfile string] [-netlist string]
    -sdc string [-init_sequence_file string] [-no_exit] [> file]
```

```
write_do_ccd generate [-design string] [-logfile string] [-netlist string]
    [-slack integer] [-report string] -in_sdc string [-out_sdc string] [-trv]
    [-fpgen] [-dfpgen] [-no_exit] [> file]
```

The `validate_constraints` and `generate_constraints` commands are native, RTL Compiler commands that validate and generate constraints:

```
validate_constraints [-rtl] [-netlist string] [-init_sequence_file string]
    [-sdc string] [> file]
```

```
generate_constraints [-rtl] [-netlist string] [-slack integer]
    [-report string] [-in_sdc string] [-out_sdc string] [-trv] [-fpgen]
    [-dfpgen] [> file]
```

Validating Constraints

- Validating Constraints with Dofiles on page 307
- Validating Constraints Natively on page 308

Validating Constraints with Dofiles

To generate a dofile for the validate flow, use the following command:

```
rc:/> write_do_ccd validate -logfile log_file -sdc list_of_SDC_files > \
    dofile_for_validation
```

In the validate flow, the `write_do_ccd` command generates a dofile that can be read by Conformal Constraint Designer. The dofile validates SDC against the RTL or netlist. By default, Conformal Constraint Designer validates the SDC against the RTL.

Specifically, Conformal Constraint Designer uses the generated dofile to:

- Perform semantic checks for the SDC
- Report any missing constraints such as loads, transitions, clock uncertainty, clock latency, or any overlapping exceptions
- Report if any paths are incorrectly specified as false paths (if a true path is specified as a false path)

To validate the SDC against a netlist, use the `-netlist` option:

```
rc:/> write_do_ccd validate -logfile log_file -sdc list_of_SDC_files
    -netlist UNIX_path_to_the_netlist > dof ile_for_validation>
```

The following example illustrates a sample generated dofile:

```
read library -liberty library
read design -verilog file
read sdc sdc_file
set system mode verify
commit clock
run rule check
validate
report rule check
report validated sdc -fail
```

Validating Constraints Natively

To validate SDCs natively (without a dofile) in the RTL Compiler, use the `validate_constraints` command. This command validates the SDCs specified in the SDC files against the RTL or netlist. You can use this command any time after elaborating the design. If you do not specify either the `-rtl`, `-netlist`, or `-sdc` options, RTL Compiler will validate the internally generated constraints against the design at its current state.

The following sections provide examples for:

- Validating False Paths Natively on page 308
- Validating Multi-Cycle Paths Natively on page 308

Validating False Paths Natively

The following example uses the `validate_constraints` command to validate false paths:

```
rc:/> read_hdl design_file
rc:/> elaborate
rc:/> read_sdc sdc_file
rc:/> write > generic.nl.v
rc:/> validate_constraints -rtl -sdc sdc_file
rc:/> validate_constraints -netlist generic.nl.v -sdc sdc_file
rc:/> synthesize -to_mapped
rc:/> write -m > mapped.nl.v
rc:/> validate_constraints -netlist mapped.nl.v -sdc sdc_file
```

Validating Multi-Cycle Paths Natively

The following command uses the `validate_constraints` command to include multi-cycle path validation against the RTL:

```
rc:/> read_hdl design_file
rc:/> elaborate
rc:/> read_sdc sdc_file
rc:/> write > generic.nl.v
rc:/> validate_constraints -rtl -sdc sdc_file \
    -init_sequence_file <initialization sequence_file
rc:/> validate_constraints -netlist generic.nl.v -sdc sdc file
    -init_sequence_file <initialization sequence_file
rc:/> synthesize -to_mapped
rc:/> write -m > mapped.nl.v
rc:/> validate_constraints -netlist mapped.nl.v -sdc sdc_file
    -init_sequence_file <initialization sequence_file
```

Generating Constraints

- Generating Constraints with Dofiles on page 309
- Generating Constraints Natively on page 311

Generating Constraints with Dofiles

The generate flow with dofiles is categorized into three flows:

- False Path Generation on page 309
- Directed False Path Generation on page 310
- Timing Report Validation on page 310

False Path Generation

To generate a dofile for the false path generation flow, use the following command:

```
write_do_ccd generate -fpgen -netlist UNIX_path_to_the_netlist
    -in_sdc list_of_SDC_files -out_sdc Output_SDC_file > Dofile
```

In the false path generation flow, the `write_do_ccd_generate` command generates a timing report in a format readable by Conformal Constraint Designer. The timing report contains all the paths in the design where the number of logic levels is above a certain threshhold. Conformal Constraint Designer works on all these paths to verify that they are true paths. Any false paths are written out in SDC format in the file specified with the `-out_sdc` option.

The following example illustrates a sample dofile that `write_do_ccd` will generate in this case:

```
read library -liberty library
read design -verilog file
read sdc sdc_file
set system mode verify
add sdc generation
set ccd option -generation -threshold_percentage 0.8
generate
usage
write generated sdc Output_SDC_file -replace
```

Directed False Path Generation

To generate a dofile for the directed false path flow, use the following command:

```
write_do_ccd generate -dfpgen -netlist UNIX_path_to_the_netlist
-in_sdc list_of_SDC_files -out_sdc Output_SDC_file
-report CCD_timing_report_file -slack integer > Dofile
```

In the directed false path generation flow, the `write_do_ccd` command generates a timing report in a format readable by Conformal Constraint Designer. This timing report contains all the paths with a slack less than the number specified with the `-slack` option. The Conformal Constraint Designer works on all paths in the fanin cone of the endpoints of the timing critical paths to identify false paths. Any false paths are written out in SDC format in the file specified with the `-out_sdc` option.

The following example illustrates a sample dofile that `write_do_ccd` will generate in this case:

```
read library -liberty library
read design -verilog file
read sdc sdc_files
read critical path CCD_timing_report_file -generation -replace
set system mode verify
add sdc generation
set ccd option -generation -threshold_precentage 0.8
generate
write generated sdc Output_SDC_file -replace
```

Timing Report Validation

To generate a dofile for the timing report validation flow, use the following command:

```
write_do_ccd generate -trv -netlist UNIX_path_to_the_netlist
-in_sdc list_of_SDC_files -out_sdc Output_SDC_file
-report CCD_timing_report_file -slack integer > Dofile
```

In the timing report validation flow, the `write_do_ccd` command generates a timing report in a format readable by the Conformal Constraint Designer. This timing report contains all the paths with a slack less than the number specified with the `-slack` option. The Conformal Constraint Designer then works exclusively on the timing critical paths listed in the report, and verifies that all of them are true paths. That is, it ensures that none of the timing critical paths are actually false paths. Any false paths are written out in SDC format in the file specified with the -out_sdc option.

The following example illustrates a sample dofile that `write_do_ccd` will generate in this case:

```
read library -liberty <library>
```

```
read design -verilog <file>
read sdc <SDC files>
read critical path <CCD timing report file> -replace
set system mode verify
add sdc check trv
validate
write trv sdc <Output SDC file> -replace
```

Note: In the generate flow, if no *Output_SDC_file* is specified, the default SDC file that will be generated is `cpf.sdc`.

Generating Constraints Natively

Instead of creating dofiles for Conformal Constraint Designer, you can generate missing constraints natively in RTL Compiler using the `generate_constraints` command. The command verifies the false paths and multi-cycle paths in the SDC files against the RTL or netlist and then generates any missing functional false paths or multi-cycle paths. The command can be used any time after elaboration. If you do not specify either the `-rtl`, `-netlist`, or `-in_sdc` options, RTL Compiler will internally generate the SDCs and verify them against the design at its current state. RTL Compiler will generate any missing constraints.

The generate flow without dofiles is categorized into the same three flows found in the generate with dofiles flow:

- False Path Generation on page 311
- Directed False Path Generation on page 312
- Timing Report Validation on page 312

False Path Generation

In this flow, RTL Compiler works on all the paths in the design for which the number of logic levels is above a certain threshold. These paths are checked to verify that they are all true paths. That is, RTL Compiler will ensure that none of the paths are actually a false paths. If any false paths are detected, they will be written out in the file specified with `-out_sdc` option.

```
rc:/> read_hdl design_file
rc:/> elaborate
rc:/> read_sdc Input_SDC_file
rc:/> write > generic.nl.v
```

```
rc:/> generate_constraints -fpgen -rtl -in_sdc Input_SDC_file \
    -out_sdc Output_SDC_file
rc:/> generate_constraints -fpgen -netlist generic.nl.v -in_sdc Input_SDC_file \
    -out_sdc Output_SDC_file
rc:/> write -m > mapped.nl.v
rc:/> generate_constraints -fpgen -netlist mapped.nl.v -in_sdc Input_SDC_file \
    -out_sdc Output_SDC_file
```

Directed False Path Generation

In this flow, RTL Compiler creates a timing report in Conformal Constraint Designer understandable format. This timing report will have all the paths that have a slack less than the slack number specified with
`-slack` option of the `generate_constraints` command. Conformal Constraint Designer will then work exclusively on paths ending at these critical endpoints to check that they are all are true paths. Any false paths will be written out in SDC format in the file specified with `-out_sdc` option.

```
rc:/> read_hdl design_file
rc:/> elaborate
rc:/> read_sdc sdc_file
rc:/> write > generic.nl.v
rc:/> generate_constraints -dfpgen -rtl -in_sdc sdc_file \
    -report <CCD timing report file> -slack Slack_number \
    -out_sdc Output_SDC_file
rc:/> generate_constraints -dfpgen -netlist generic.nl.v \
    -in_sdc <sdc file> -report CCD_timing_report_file \
    -slack Slack_number -out_sdc Output_SDC_file
rc:/> write -m > mapped.nl.v
rc:/> generate_constraints -dfpgen -netlist mapped.nl.v \
    -in_sdc sdc_file -report CCD_timing_report_file \
    -slack Slack_number -out_sdc Output_SDC_file
```

Timing Report Validation

In this flow, RTL Compiler creates a timing report in Conformal Constraint Designer understandable format. This timing report has all the paths that have a slack less than the slack number specified with `-slack` option of the `generate_constraints` command. Conformal Constraint Designer then works exclusively on these timing critical paths to check that they are all true paths. That is, RTL Compiler ensures that none of the timing critical paths are actually false paths. If any false paths are detected, they are written out in the output SDC file (*`Output_SDC_file`* below). This flow could validate and generate the exceptions only with respect to the netlist and not the RTL.

```
rc:/> read_hdl design_file
rc:/> elaborate
```

```
rc:/> read_sdc sdc_file

rc:/> write > generic.nl.v

rc:/> generate_constraints -trv -netlist generic.nl.v \
    -in_sdc sdc_file -report_CCD_timing_report_file \
    -slack Slack_number -out_sdc Output_SDC_file

rc:/> write -m > mapped.nl.v

rc:/> generate_constraints -trv -netlist mapped.nl.v \
    -in_sdc sdc_file -report CCD_timing_report_file \
    -slack Slack_number -out_sdc Output_SDC_file
```

■

SoC Encounter Integration

SoC Encounter gives you the ability to validate and generate design constraints natively, through SoC Encounter commands, or through dofiles that can be loaded into Conformal Constraint Designer (CCD).

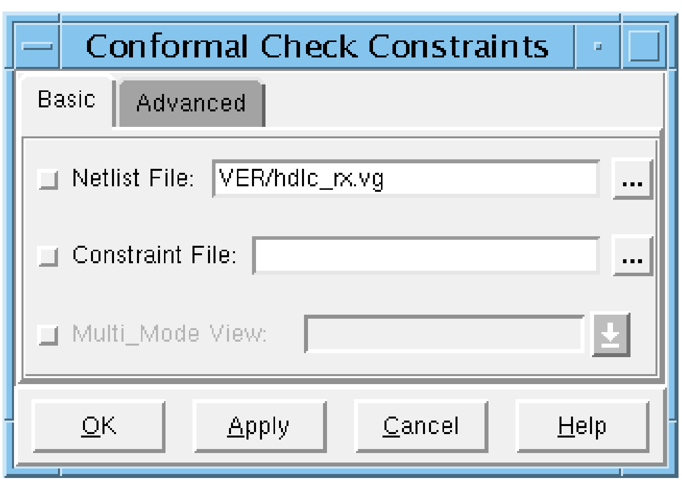
- Checking Constraints on page 314
- Checking Budget Constraints on page 317
- Checking Assembled Constraints on page 323
- Deriving Critical False Paths on page 326
- Promoting Constraints on page 331

For information on the related commands in this chapter, see “Conformal Commands” in SoC Encounter’s Text Command Reference.

Checking Constraints

Use SoC Encounter’s `checkSdcCCD` command or Conformal Check Constraints form to check the quality of constraints for the current design using the Conformal Constraint Designer software.

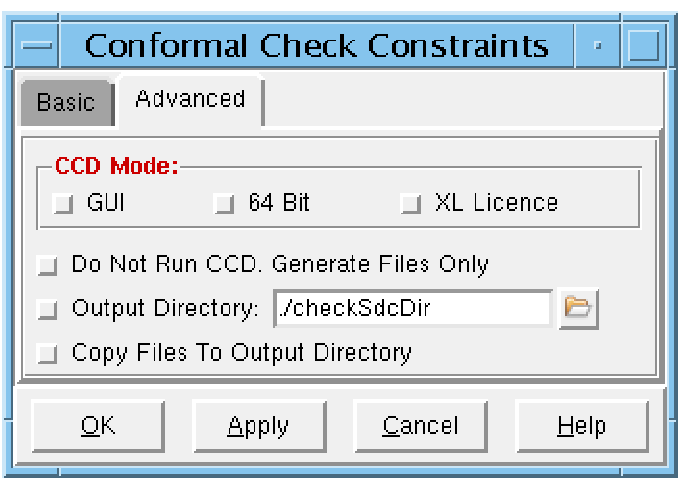
➤ From the SoC Encounter main window, choose *Tools – Conformal – Check Constraints*.



Conformal Check Constraints – Basic Fields and Options

Netlist File	Specifies the name of the netlist file. For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Netlist Files browser window.
Constraint File	Specifies the constraint file(s) to analyze using Conformal Constraint Designer. For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Constraint Files browser window.
Multi_Mode View	Specifies the multi-mode view name. If a design has multiple modes, you can specify which set of view constraints to pass to Conformal with this option. **Note:** Multi-mode constraints and views must already be specified. If you do not specify a constraint file, the current design SDC file is passed to the Conformal software for analysis

Use the *Advanced* page of the Conformal Check Constraints form to specify additional constraint checking options.



Conformal Check Constraints – Advanced Fields and Options

GUI	Runs the Conformal Constraint Designer software in GUI mode. This runs as a parallel job separate from the SoC Encounter session—you can continue to run additional SoC Encounter commands while the CCD GUI mode session is running in parallel. The software does not exit at the end of the session, so you can continue interactive debugging in the standalone Conformal GUI after completion of the CCD script. Conformal log messages are not echoed to the SoC Encounter log file. The software creates a separate Conformal log file in the CCD run directory (see the *Output Directory* option). *Default:* Off. The Conformal Constraint Designer software exits at the end of the session. In non-GUI mode, the software is not run as a parallel job, therefore no SoC Encounter command is executed until the CCD script has completed.
64 Bit	Specifies 64-bit CCD. *Default:* 32-bit CCD, or 64-bit if the SoC Encounter software starts in 64-bit mode.
XL License	Runs the Conformal Constraint Designer software with the XL license. *Default:* CCD L license
Do Not Run CCD. Generate Files Only	Specifies that the Conformal Constraint Designer software does not start, however, the CCD script is generated. You can use this option if you need to customize CCD run scripts.
Output Directory	Specifies the name of the directory in which to generate CCD script and log files. You can enter the name or click the *Browser* icon select a directory from the Output Directory browser window. *Default*: `./checkSdcDir`

Copy Files to Output Directory	Copies all design files present in the Conformal script to the Conformal run directory. You can use the *Output Directory* option to specify the run directory.

Checking Budget Constraints

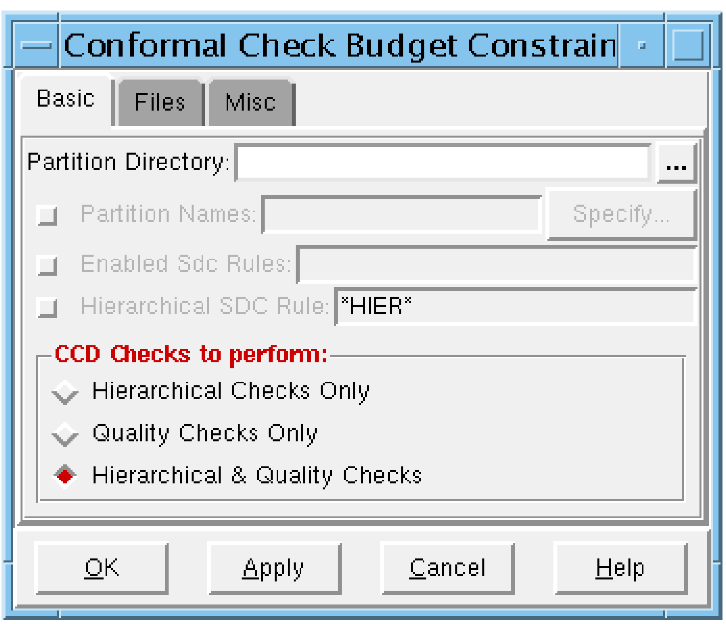
Use SoC Encounter's `checkBudgetSdcCCD` command or Conformal Check Constraints form to check the time budget directory's top and block constraints against their pre-partitioned original chip SDCs. Conformal Constraint Designer performs checks for hierarchical constraint mismatches, exceptions, clocks, unconstrained ports, and invalid SDC command syntax.

For more information on SDC checks that the Conformal Constraint Designer uses to verify SDC data, see the "SDC Rule Checks" chapter of the *SoC Encounter Conformal Constraint Designer Reference Manual*.

To use this feature, you must specify the path to the Conformal Constraint Designer installation before running SoC Encounter, and then after running SoC Encounter's `savePartition` command.

Note: You must have previously SoC Encounter's `deriveTimingBudget -ccd` to generate time budget CCD dofiles and CCD clock map files. These dofile scripts are automatically detected by this form in the *Partition Directory*.

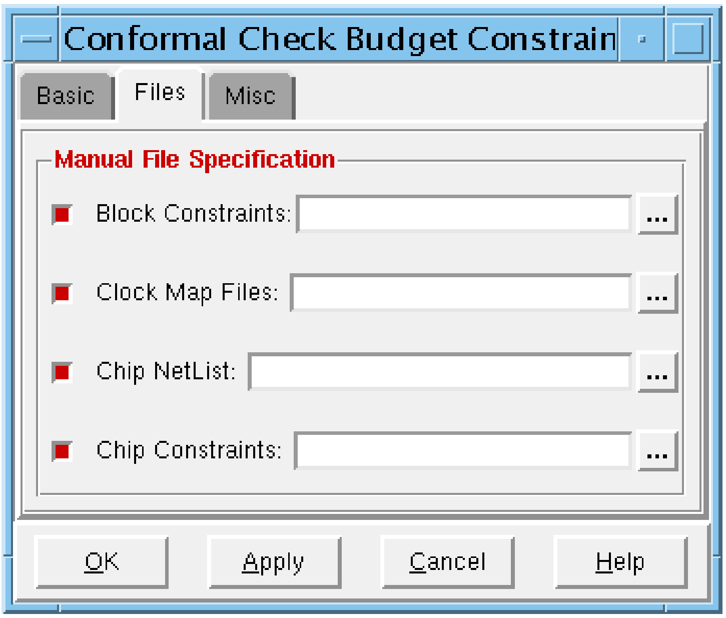
➤ From the SoC Encounter main window, choose *Tools – Conformal – Check Budget Constraints*.



Conformal Check Budget Constraints – Basic Fields and Options

Partition Directory	Specifies the budget or partition directory. Time budget constraints and clockmap files are automatically detected by the `checkBudgetSdcCCD` command in the specified partition directory. You can enter the name or click *...* to select a directory.
Partition Names	Specifies the partition names whose constraints should be checked against the original pre-partition chip SDCs. You can enter the name or click the *Specify* button to select a partition name. *Default*: All top and block level partitions under the user-specified partition directory are checked by the CCD software against their original pre-partition chip constraints.
Enabled SDC Rules	Allows specification of user-enabled or disabled rules in CCD to be used during quality rule checking of partition constraints. You can define multiple added and deleted disabled rules. *Default*: A default set of rules are defined on script initialization. The user-enabled rules are added after this default set of rules.
Hierarchical SDC Rule	Specifies the rules to be checked during hierarchical rule checking. *Default*: `*HIER*`
Hierarchical Checks Only	Specifies that only hierarchical rule checking be performed by the CCD software.
Quality Checks Only	Specifies that only quality rule checking for each partition be performed by the CCD software.
Hierarchical & Quality Checks	Specifies that both hierarchical and quality rule checking be performed by the CCD software.

Use the *Files* page of the Conformal Check Constraints form to specify constraint, mapping, and netlist files manually. You can use this page after saving the partition information to the current or specified directory

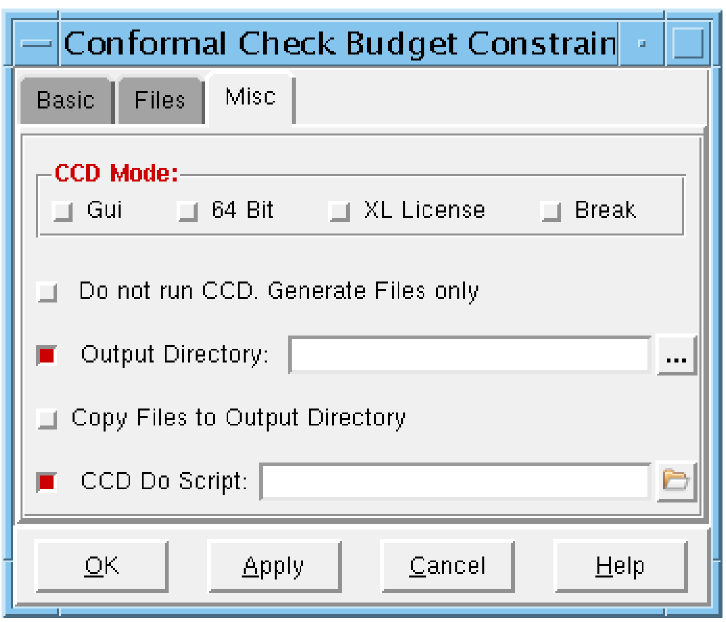


Conformal Check Budget Constraints – Files Fields and Options

Block Constraints	Allows manual specification of the budgeted constraints files to be used during Conformal Constraint Designer analysis. You can enter the name or click *...* to select a file or files. *Default*: Constraints are automatically detected in the partition directory.
Clock Map Files	Allows manual specification of the clock map file(s) that contain hierarchically equivalent clocks. This option is useful for overriding clock map files. You can enter the name or click *...* to select a file or files. *Default*: The netlist is automatically detected from the SoC Encounter database. In most cases, the clock map file(s) are automatically detected from the SoC Encounter database and passed to CCD without having to use this option.

Chip Netlist	Allows manual specification of the original chip netlist file(s) to analyze using CCD. This option is useful for overriding chip netlist settings You can enter the name or click *...* to select a file or files. *Default*: The netlist is automatically detected from the SoC Encounter database.
Chip Constraints	Allows manual specification of the original chip constraints file(s) to analyze using CCD. This option is useful for overriding chip constraints. You can enter the name or click *...* to select a file or files. *Default:* Constraints are automatically detected from the SoC Encounter database.

Use the *Misc* page of the Conformal Check Constraints form to specify additional options for checking the time budget directory's top and block constraints against their pre-partitioned original chip SDCs. You can use this page after saving the partition information to the current or specified directory.



Conformal Check Budget Constraints – Misc Fields and Options

Gui	Runs the Conformal Constraint Designer software in GUI mode. This runs as a parallel job separate from the SoC Encounter session—you can continue to run additional SoC Encounter commands while the CCD GUI mode session is running in parallel. The software does not exit at the end of the session, so you can continue interactive debugging in the standalone Conformal GUI after completion of the CCD script. Conformal log messages are not echoed to the SoC Encounter log file. The software creates a separate Conformal log file in the CCD run directory (see the *Output Directory* option). *Default:* Off. The Conformal Constraint Designer software exits at the end of the session. In non-GUI mode, the software is not run as a parallel job, therefore no SoC Encounter command is executed until the CCD script has completed.
64 Bit	Specifies 64-bit CCD. *Default:* 32-bit CCD, or 64-bit if the SoC Encounter software starts in 64-bit mode.
XL License	Runs the Conformal Constraint Designer software with the XL license. *Default:* CCD L license

Break	Specifies that the generated dofile should contain a “break” command after checking each partition constraints file. This is useful for interactive debug of partition constraints using the SDC Rule Manager. After browsing quality checks in the SDC Rule Manager, type `continue` to proceed on the CCD command line.
	
	The CCD software will then proceed to check the next partition and then break again, until finally all partition constraints have been checked.
	
	Note: This option is only available with the *Gui* option.
	
	Default: CCD will not perform any breaks. Interactive debug of partition constraint’s quality checks using the SDC Rule Manager will not be possible. The SDC Rule Manager will only display hierarchical rule check results.
	
	Quality checks can be examined instead in the individual partition reports in the following file:
	
	`checkBudgetSdcDir/`
	`rule_check.quality.<partitionName>.rpt`

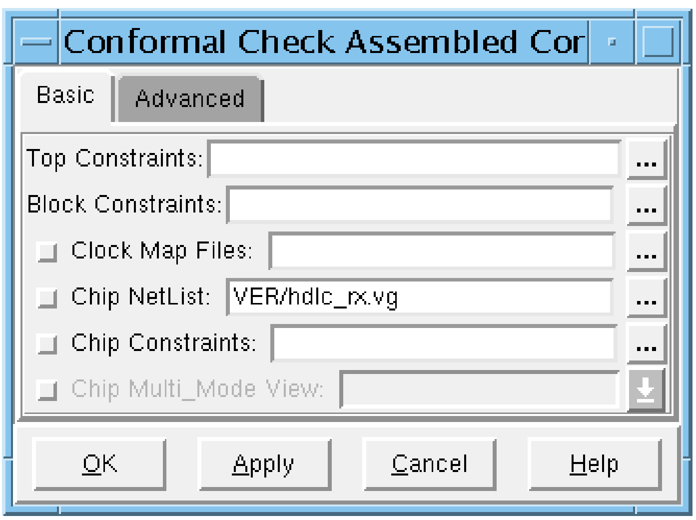
Do Not Run CCD. Generate Files Only	Specifies that the Conformal Constraint Designer software does not start, however, the CCD script is generated. You can use this option if you need to customize CCD run scripts.
Output Directory	Specifies the name of the directory in which to generate CCD script and log files. You can enter the name or click *...* to select a directory. *Default*: `./checkBudgetSdcDir`
Copy Files to Output Directory	Copies all design files present in the Conformal script to the Conformal run directory. You can use the *Output Directory* field to specify the run directory.
CCD Do Script	Specifies an existing dofile script that should be passed to the CCD software for execution. You can enter the name or click the *Browser* icon select a file. *Default*: A new dofile script is generated based on user specified `checkBudgetSdcCCD` options and used to run the CCD software.

Checking Assembled Constraints

Use SoC Encounter's `checkAssembledSdcCCD` command or Conformal Assembled Check Constraints form to to check pre-assembled design top and block constraints against post-assembled design chip constraints. The Conformal Constraint Designer performs checks for for hierarchical constraint mismatches, exceptions, clocks, unconstrained ports, and invalid SDC command syntax. You can use this form after running SoC Encounter's `assembleDesign` command.

For more information on SDC checks that the Conformal Constraint Designer uses to verify SDC data, see the "SDC Rule Checks" chapter of the *SoC Conformal Constraint Designer Reference Manual*.

➤ From the SoC Encounter main window, choose *Tools – Conformal – Check Assembled Constraints*.

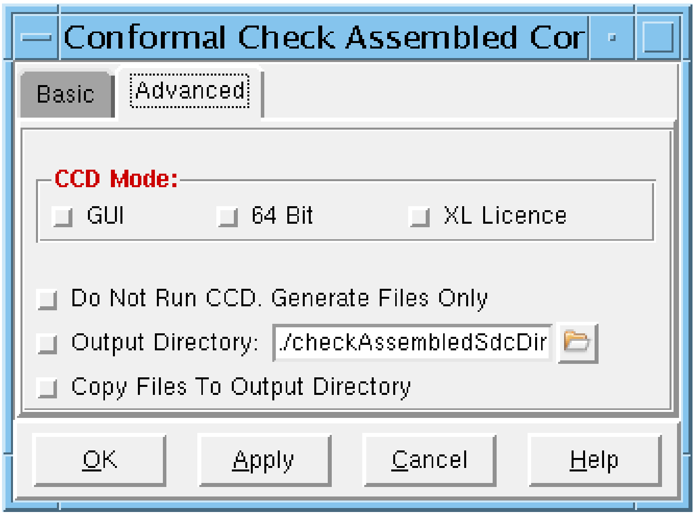


Conformal Check Assembled Constraints – Basic Fields and Options

Top Constraints	Specifies the pre-assembly top level constraint file(s). For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Top Constraints Files browser window.

Block Constraints	Specifies the pre-assembly block constraints. You must provide either one combination or multiple combinations of both instance name(s) and corresponding constraint file name(s). For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Block Hierarchical Instances & Constraints browser window.
Clock Map Files	Specifies clock map file(s) that contain hierarchically equivalent clocks. For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Clock Map File browser window. **Note:** Use the clock map files previously generated when running the `deriveTimingBudget -ccd` command. *Default:* Partition map file.
Chip Netlist	Specifies the chip netlist file(s). For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Chip Netlist browser window. *Default:* Current netlist file. If the design has changed since loading or saving the design, a new netlist is written out and passed to the software.
Chip Constraints	Specifies the chip constraint file(s). For multiple files, separate each filename with a space. You can enter the name(s) or click … to select them from the Chip Constraint File browser window. *Default:* Current SDC file.
Chip Multi_Mode View	Specifies the multi-mode chip-level view name. If a design has multiple modes, you can specify which set of view constraints to pass to Conformal with this option. **Note:** Multi-mode constraints and views must already be specified.

Use the *Advanced* page of the Conformal Check Assembled Constraints form to specify additional options for checking pre-assembled design top and block constraints against post-assembled design chip constraints.



Conformal Check Assembled Constraints – Advanced Fields and Options

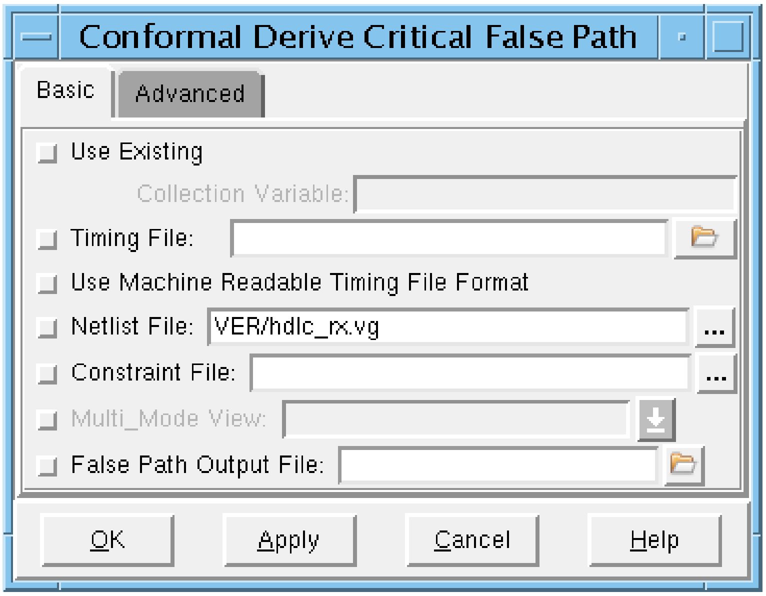
GUI	Runs the Conformal Constraint Designer software in GUI mode. This runs as a parallel job separate from the SoC Encounter session—you can continue to run additional SoC Encounter commands while the CCD GUI mode session is running in parallel. The software does not exit at the end of the session, so you can continue interactive debugging in the standalone Conformal GUI after completion of the CCD script. Conformal log messages are not echoed to the SoC Encounter log file. The software creates a separate Conformal log file in the CCD run directory (see the *Output Directory* option). *Default:* Off. The Conformal Constraint Designer software exits at the end of the session. In non-GUI mode, the software is not run as a parallel job, therefore no SoC Encounter command is executed until the CCD script has completed.

64 Bit	Specifies 64-bit CCD. *Default:* 32-bit CCD, or 64-bit if the SoC Encounter software starts in 64-bit mode.
XL License	Runs the Conformal Constraint Designer software with the XL license. *Default:* CCD L license
Do Not Run CCD. Generate Files Only	Specifies that the Conformal Constraint Designer software does not start, however, the CCD script is generated. You can use this option if you need to customize CCD run scripts.
Output Directory	Specifies the name of the directory in which to generate CCD script and log files. You can enter the name or click the *Browser* icon select a directory from the Output Directory browser window. *Default*: `./checkAssembledSdcDir`
Copy Files to Output Directory	Copies all design files present in the Conformal script to the Conformal run directory. You can use the *Output Directory* option to specify the run directory.

Deriving Critical False Paths

Use SoC Encounter's `deriveFalsePathCCD` command or Conformal Derive Critical False Path form to analyze critical false paths based on SoC Encounter CTE timing information and constraints. A set of false paths are output which can be loaded into SoC Encounter. These false paths can eliminate unnecessary netlist optimizations and can improve design area and timing. You can use this form after loading design with timing constraints.

➤ From the SoC Encounter main window, choose *Tools – Conformal – Derive Critical False Paths*.



Conformal Derive Critical False Path – Basic Fields and Options

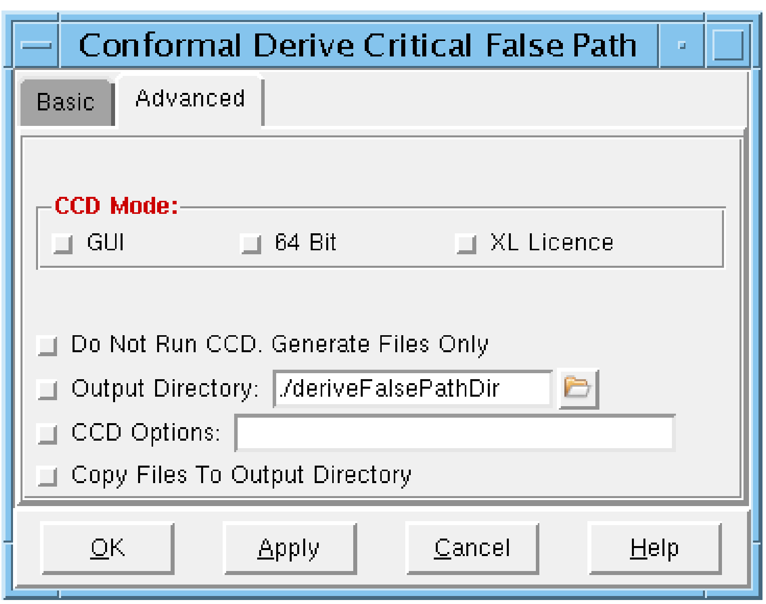
Use Existing	Specifies a previously generated collection of timing paths to be used to pass timing information to the CCD software. The timing paths contained in the user-defined collection are then passed to the CCD software in a Standard Format Timing File. To generate this collection of timing paths, use the `report_timing -collection` option along with your own set of reporting options as required. *Default*: The software performs timing analysis. A collection containing the timing paths is used to generate a standard format timing file to pass to the CCD software.

Timing File	Specifies previously generated timing debug file to pass to the Conformal Constraint Designer software. You can enter the name or click the *Browser* icon select a file from the Timing File browser window.
	
	You can generate this machine readable timing file. For example, for maximum slack value, you can run the following command:
	
	`report_timing -machine_readable -max_points 30000 \`
	`-nworst 100 -max_slack 0.20 > timing_debug.rpt`
	
	Default: Timing analysis is performed to generate a machine readable timing file to pass to the software.

Use Machine Readable Timing File Format	Specifies that the machine readable format should be used when passing timing file information to the CCD software. *Default*: The CCD standard format is used to pass timing file information to the CCD software.
Netlist File	Specifies the name of the netlist file. For multiple files, separate each filename with a space. You can enter the name(s) or click *...* to select them from the Netlist Files browser window. *Default*: Passes the existing design netlist to the software. If the design has changed since previously loading or saving the design, a new netlist is written out and passed to the Conformal Constraint Designer software.
Constraint File	Specifies the constraint file(s) to analyze using Conformal Constraint Designer. For multiple files, separate each filename with a space. You can enter the name(s) or click *...* to select them from the Constraint Files browser window. *Default*: Passes the existing design constraints to the software. If the design has changed since previously loading or saving the design, a new constraint file is written out and passed to the software.

Multi_Mode View	Specifies the multi-mode view name. If a design has multiple modes, you can specify which set of view constraints to pass to Conformal with this option. This view is also used by CTE to generate the timing debug file to pass to the Conformal Constraint Designer software. **Note:** Multi-mode constraints and views must already be specified. *Default*: If you do not specify the `-constraints` option, the current design SDC file is passed to the Conformal software for analysis.
False Path Output File	Specifies the name of the output false-path file. You can enter the name or click the *Browser* icon select a file from the False Path Output File browser window. *Default*: `criticalFalsePaths.sdc`. This file is generated in output directory specified by the *Output Directory* option.

Use the *Advanced* page of the Conformal Derive Critical False Path form to specify additional options to analyze critical false paths in CCD based on SoC Encounter-CTE timing information and constraints.



Conformal Derive Critical False Path – Advanced Fields and Options

GUI	Runs the Conformal Constraint Designer software in GUI mode. This runs as a parallel job separate from the SoC Encounter session—you can continue to run additional SoC Encounter commands while the CCD GUI mode session is running in parallel. The software does not exit at the end of the session, so you can continue interactive debugging in the standalone Conformal GUI after completion of the CCD script. Conformal log messages are not echoed to the SoC Encounter log file. The software creates a separate Conformal log file in the CCD run directory (see the *Output Directory* option). *Default:* Off. The Conformal Constraint Designer software exits at the end of the session. In non-GUI mode, the software is not run as a parallel job, therefore no SoC Encounter command is executed until the CCD script has completed.
64 Bit	Specifies 64-bit CCD. *Default:* 32-bit CCD, or 64-bit if the SoC Encounter software starts in 64-bit mode.
XL License	Runs the Conformal Constraint Designer software with the XL license. *Default:* CCD L license
Do Not Run CCD. Generate Files Only	Specifies that the Conformal Constraint Designer software does not start, however, the CCD script is generated. You can use this option if you need to customize CCD run scripts.
Output Directory	Specifies the name of the directory in which to generate CCD script and log files. You can enter the name or click the *Browser* icon select a directory from the Output Directory browser window. *Default*: `./deriveFalsePathDir`

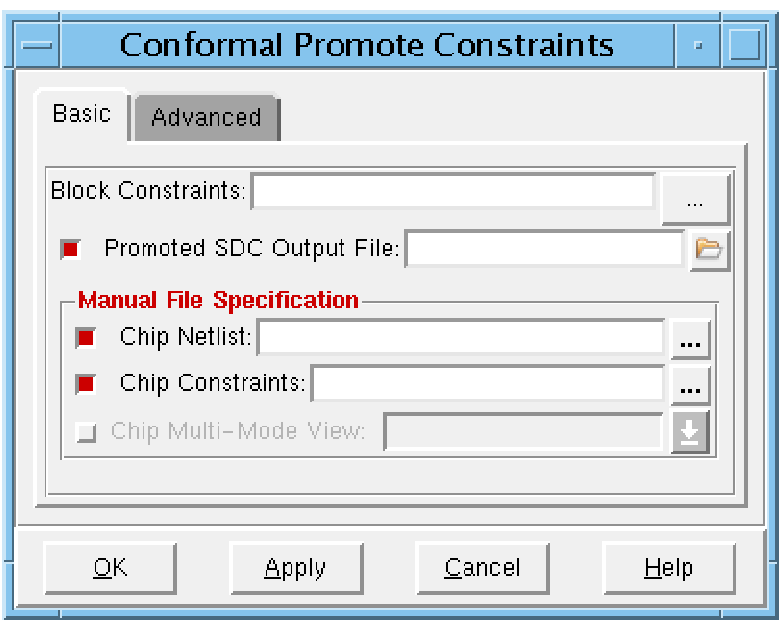
CCD Options	Specifies how you want the Conformal Constraint Designer software to handle exception validation and exception generation. For a table of available strings, see SoC Encounter's `deriveFalsePathCCD` command documentation.
Copy Files to Output Directory	Copies all design files present in the Conformal script to the Conformal run directory. You can use the *Output Directory* option to specify the run directory.

Promoting Constraints

Use SoC Encounter's `promoteSdcCCD` command or Conformal Promote Constraints form to generate top-level constraints using Conformal Constraint Designer by promoting block-level constraints to the top level, integrating them with any existing chip-level constraints.

To use this feature, specify the path to the Conformal Constraint Designer installation before running SoC Encounter.

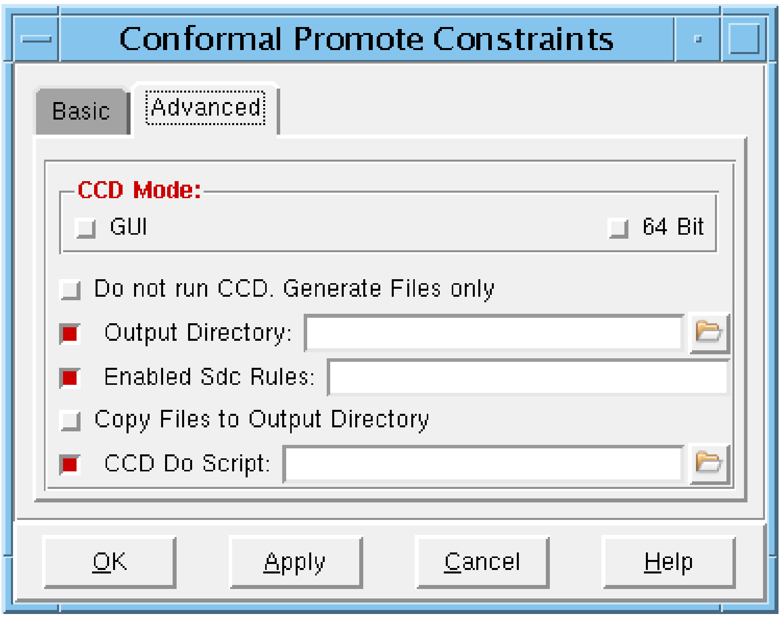
➤ From the SoC Encounter main window, choose *Tools – Conformal – Promote Constraints*.



Conformal Promote Constraints – Basic Fields and Options

Block Constraints	Specifies the block constraints to be passed to Conformal Constraint Designer for SDC promotion. You must provide either one combination or multiple combinations of both instance name(s) and corresponding constraint file name(s).
Promoted SDC Output File	Specifies the output SDC file. *Default*: `promotedChip.sdc`. This file is generated in output directory specified by the *Output Directory* option in this form's *Advanced* page.
Chip Netlist	Allows manual specification of the original chip netlist file(s) to analyze using Conformal Constraint Designer. This option is useful for overriding chip netlist settings. You can enter the name or click *...* to select a file or files. *Default:* The netlist is automatically detected from SoC Encounter database.
Chip Constraints	Specifies the SDC chip constraint file(s) to be passed to Conformal Constraint Designer for SDC promotion. You can enter the name or click *...* to select a file or files. **Note:** This option cannot be used in conjunction with *Chip Multi-Mode View.* *Default:* Passes the existing design constraints to the software. If the design has changed since loading or saving the design, a new constraint file is written out and passed to the software.
Chip Multi-Mode View	Specifies the multi-mode chip-level view name. If a design has multiple modes, you can specify which set of view constraints to pass to Conformal with this option. **Note:** Multi-mode constraints and views must already be specified. *Default:* If you do not use the *Chip Constraints* option, the current design SDC file is passed to the Conformal software for analysis.

Use the *Advanced* page of the Conformal Promote Constraints form to specify additional options to promote block-level constraints to the top level.



Conformal Promote Constraints – Advanced Fields and Options

GUI

Runs the Conformal Constraint Designer software in GUI mode. This runs as a parallel job separate from the SoC Encounter session—you can continue to run additional SoC Encounter commands while the CCD GUI mode session is running in parallel.

The software does not exit at the end of the session, so you can continue interactive debugging in the standalone Conformal GUI after completion of the CCD script.

Conformal log messages are not echoed to the SoC Encounter log file. The software creates a separate Conformal log file in the CCD run directory (see the *Output Directory* option).

Default: Off. The Conformal Constraint Designer software exits at the end of the session. In non-GUI mode, the software is not run as a parallel job, therefore no SoC Encounter command is executed until the CCD script has completed.

64 Bit	Specifies 64-bit CCD. *Default:* 32-bit CCD, or 64-bit if the SoC Encounter software starts in 64-bit mode.
Do not run CCD. Generate Files only	Specifies that Conformal Constraint Designer software does not start, however, the CCD script is generated. You can use this option if you need to customize CCD run scripts.
Output Directory	Specifies the name of the directory in which to generate CCD script and log files. *Default*: `./promoteSdcDir`.
Enabled SDC rules	Allows specification of user-enabled or disabled rules in CCD to be specified during CCD do file initialization. You can define multiple added and deleted disabled rules. *Default*: The following default set of rules are defined on script initialization: `add integration_rule instance -def` The user-enabled rules are added after this default set of rules.
Copy Files to Output Directory	Copies all design files in the Conformal script to the Conformal run directory. You can use the *Output Directory* option to specify the run directory. *Default*: Design files are not copied to the CCD run directory.
CCD Do Script	Specifies an existing dofile script that should be passed to the CCD software for execution. *Default*: A new dofile script is generated based on user specified `checkBudgetSdcCCD` options and used to run the CCD software.

17

Troubleshooting

Clock Errors

If you are experiencing a large number of errors or warnings that are caused by a single clock, try the following:

- Ensure that your clock groups are correct. Your clock groups affect the rule checks that the Conformal Constraint Designer flags. See “Checking Clock Groups” on page 90.
- Ensure that all of the generated clocks are correctly associated to their master clock.
- Check that all the clock domain assignments are correct (using the `REPORT CLOCK GROUP` command)
- Try constraining all registers reported by `CCD_CLK_DEF1`

SDC Errors

If you are experiencing a large number of SDC errors, the naming for registers, ports, or pins may have been modified (if you are working on the original design). Try the following:

- Ensure that your design matches with the SDC file.
- Apply renaming rules. Renaming rules are applied to the SDC file—not to the design. See “Specifying Renaming Rules” on page 111.

18

Running Reports

Use the *Report* menu of the Conformal GUI to open the Report form to display extensive design information in the Transcript window of the main Conformal GUI window.

Note: You can run some of these reports in Setup and mode.

The *Report* menu and Report form contains the following categories:

Design Data Report

Use the `REPORT DESIGN DATA` command or the Design Data Report form (*Report – Design Data*) to specify and run a report of current design information, including word-level information.

Design Data Report Form Fields and Options

Summary	Summarizes the design data.
Verbose	Verbose reports a detailed list of the design data.

Environment Report

Use the `REPORT ENVIRONMENT` command or the Environment Report form (*Report – Environment*) to display global settings for the design and system settings.

There are no customized options for the Environment Report form. Click *Apply* to view the report in the Transcript window.

Floating Signals Report

Use the `REPORT FLOATING SIGNALS` command or the Floating Signals Report form (*Report – Floating Signals*) to display all floating signals in the design or in specified modules of a design. The reported floating signals are either nets or pins and are either undriven or unused.

Floating Signals Report Form Fields and Options

Category	*Undriven* displays only undriven floating signals (the default). *Unused* displays only unused floating signals.
Signal	*Net* displays only floating nets, *Pin* displays only floating pins, and *Full* displays both floating nets and floating pins.
All	Display all floating signals in all modules

Instance Constraints Report

Use the `REPORT INSTANCE CONSTRAINTS` command or the Instance Constraints Report form (*Report – Instance Constraints*) to display constraints placed on instances in the design.

There are no customized options for the Instance Constraints Report form. Click *Apply* to view the report in the Transcript window.

Reporting Feedback Paths

Conformal inserts CUT gates to break combinational feedback paths. Then, it displays a summary warning message during flattening and modeling to tell how many CUT gates were inserted. Display the feedback paths of all CUT gates using the `REPORT PATH` command with the `-feedback` option. Also use this command to display the path between two key points.

Modules Report

Use the `REPORT MODULES` command or the Modules Report form (*Report – Modules*) to display the module hierarchy for the design.

Modules Report Form Fields and Options

Source	Displays the source-code information identifying where the module is located.
Library	Displays all of the library cells that are in the module hierarchy.
All	Displays all the modules. The top root module is denoted by (T).
Direction	*Up* (the default) reports on modules and library cells up the hierarchy of the specified module name. *Down* reports on modules and library cells down the hierarchy of the specified module name.

Notranslate Modules Report

Use the `REPORT NOTRANSLATE MODULES` command or the Notranslate Modules Report form (*Report – Notranslate Modules*) to display all library and design modules that were originally added with the Conformal software.

Note: The software will not compile these modules when reading in libraries and designs.

There are no customized options for the Modules Report form. Click *Apply* to view the report in the Transcript window.

Pin Constraints Report

Use the `REPORT PIN CONSTRAINTS` command or the Pin Constraints Report form (*Report – Pin Constraints*) to display constraints placed on primary input pins in the design.

Pin Constraints Report Form Fields and Options

All	Displays pin constraints in all modules (within the given defaults).
Root	Displays the pin constraints from the root module.

Pin Equivalences Report

Use the `REPORT PIN EQUIVALENCES` command or the Pin Equivalences Report form (*Report – Pin Equivalences*) to display all defined pin equivalences and inverted pin equivalences.

Inverted pin equivalences are distinguished by a “`-`” next to the primary input pin name.

Pin Equivalences Report Form Fields and Options

All	Displays pin equivalences in all modules (within the given defaults).
Root	Displays the pin equivalences from the root module.

Primary Inputs Report

Use the `REPORT PRIMARY INPUTS` command or the Primary Inputs Report form (*Report – Primary Inputs*) to display all defined primary inputs.

There are no customized options for the Primary Inputs Report form. Click *Apply* to view the report in the Transcript window.

Primary Outputs Report

Use the `REPORT PRIMARY OUTPUTS` command or the Primary Outputs Report form (*Report – Primary Outputs*) to display all defined primary outputs.

There are no customized options for the Primary Outputs Report form. Click *Apply* to view the report in the Transcript window.

Search Paths Report

Use the `REPORT SEARCH PATH` command or the Search Path Report form (*Report – Search Path*) to display all paths used to search for library and design files.

There are no customized options for the Search Path Report. Click *Apply* to view the report in the Transcript window.

Tied Signals Report

Use the `REPORT TIED SIGNALS` command or the Tied Signals Report form (*Report – Tied Signals*) to display tied signals from the design.

Tied Signals Report Form Fields and Options

Signal	*Net* displays net names that have tied signals assigned to them, *Pin* pin names that have tied signals assigned to them, and *All* displays net and instance names that have tied signals assigned to them (within the given defaults).
Class	*Full* (the default) displays tied signals from both the User and System classes. *System* displays tied signals from the original design. *User* displays tied signals added with the Conformal software.

A

Initialization Sequence File

This section defines the syntax of an initialization sequence file and provides examples. It includes the following sections:

Syntax

The following syntax must appear on each line of the initialization sequence file:

```
<time> <value> < <cell_name*>…| < <type_specifier> < | cell_name*>… >… >
```

```
<time> =  <const_int>
          | < <begin_const_int> - <end_const_int> >
```

Note: No expression or wildcard is allowed in the <time> specification.

`<const_int>` — The `<const_int>` is a Conformal Constraint Designer time unit, which must be greater than or equal to 0.

The Conformal Constraint Designer allows the keyword `$init_end[_time]` to represent the time unit at the end of the initialization (that is, maximum `<const_int>` in the initialization sequence file). If no explicit `<const_int>` is specified, then `$init_end[_time] = 0`.

Note: `[_time]` is optional.

`< <begin_const_int> - <end_const_int> >`

Example: `0 - 10`

This means for each time unit from 0 to 10 (inclusive).

```
<value> = < <bit_string>
          |<verilog_value_string>
          | $random >
```

`<bit_string>` This is equal to: `[01_]+`,

It will be converted to: `<length>'b<bit_string>`

`<verilog_value_string>`

This is equal to:

```
<length>'[bBdDoOhH][0-9a-fA-FzZ_]+
```

Examples:

```
8'b0
16'd12345
32'habcdef00
```

`$random` This string will trigger the random number generation between 0 and 1. This is pseudo-random; that is, the Conformal Constraint Designer always starts with a fixed random seed, which will guarantee the random assignments are always the same for each execution.

`<  <cell_name*>… | < <type_specifier> < | cell_name*>… >…`

`<cell_name*>…` Specify cell_list by cell name.

```
<cell_name*>     = <  name_string>*
                    |<name_string>*[<idx>]
                    |<name_string>*[<msb>:<lsb>]
```

`< <  type_specifier> <| cell_name*>… >…`

Specify cell_list by type specifier and cell name.

```
<type_specifier> = < -PI |-DFF |-DLAT |-BBOX
                    |-X_CONST  |-Z2LOGIC
                    |-BUS_CONTention>
```

Refer to Requirements, below for additional information.

```
<cell_name*>     = <  name_string>*
                    |<name_string>*[<idx>]
                    |<name_string>*[<msb>:<lsb>]
```

Requirements

- DFF and DLAT assignments are allowed only at time 0, while others can be at any time unit.

  Note: Apply assignments on DFFs and DLATs in the beginning of the simulation at this time unit. However, the simulation value of their inputs may overwrite the assignment.

- Wildcard refers to *legal* cells allowed at that time unit. If the file uses `type_specifier`, only cells of that type are included.
- The Conformal Constraint Designer accepts word-level cell names. For example, let `a` be an 8-bit signal. You can then specify `0 0 a`, which has the same meaning as `0 0 a[7:0]` *if* the `a` is declared as `a[7:0]`.

Important

Cadence strongly recommends that you specify the vector range (that is, `a[msb:lsb]` instead of `a`) to ensure the assignment order matches your intent.

Note: The Conformal Constraint Designer assigns the least-significant bit of the given constant to the lsb of the vector, the second lsb to the second lsb of the vector, and so on.

For example:
Let `a` be a 2-bit signal declared as `a[0:1]`. Given the constant `2'b10`, `0` is assigned to `a[1]` and `1` is assigned to `a[0]`. Similarly, if `a` is declared as `a[1:0]`, then `0` is assigned to `a[0]` and `1` is assigned to `a[1]`.

Note: If the width of the given constant is less than the vector width, the Conformal Constraint Designer does zero extension of that constant up to the vector width:

Examples for `a[0:3]`:

```
0 0 a      // 0 4'b0000 a[0:3]
0 1 a      // 0 4'b0001 a[0:3], a[3] = 1, a[0:2] = 0
0 3'b111 a // 0 4'b0111 a[0:3], a[0:2] = 1, a[3] = 0
```

- Wildcards are not allowed in an array index.
- `X` constant has no name. There is no way to set a specific `X` constant (`-X_CONST`).
- If the Conformal Constraint Designer cannot find or does not allow the specified `cell_name`, or if the specified `cell_name` is not consistent with the `type_specifier`, the Conformal Constraint Designer issues a parse error and returns the command prompt.

- Some assignments are conditional, for example, `-Z2LOGIC` and `-BUS_CONTention`. They are assigned with values only when `x` occurs (that is, `z` in `Z2X` fan-in, and bus contention).

  If you write:

  ```
  10 $random -Z2LOGIC
  ```

  the `Z2X` gates will be assigned with random numbers *at time 10* if the fan-ins are `z`.

  Treatment of Z2Logic and Z2BUS
  While `Z` acts as high impedance to Bus gates, `Z` is treated as `x` (unknown) when it goes to a logic gate (for example, `AND(a, 1'bz) ==> AND (a, 1'bx))`. Therefore, the Conformal Constraint Designer models the wire between `Z` and a logic gate as a `Z2X` gate and calls this condition `Z2Logic` (as opposed to `Z2BUS`).

  While `Z` in a `Z2BUS` condition should always be treated as `Z` (high impedance), `Z` in `Z2Logic` has an option to be treated as assignable (unknown `X`) or unassignable (don't-care `X`). The type specifier `-Z2Logic` in the initialization sequence file turns the `Z2Logic` case into assignable `X` (without this specification, the default in the Conformal Constraint Designer is an unassignable `X`). The purpose of this assignment is to reduce the `X` occurrence in the initialization.

  Note: Assignments on `-Z2LOGIC` and `-BUS_CONTention` should be applied in the end of the simulation at this time unit. Apply the assignments of other signals first, and after the combinational simulation of this time unit, check whether any of these gates are `x` (due to `z` or bus contention). Then apply these `x` assignments and perform the propagation again. If these assignments produce a new `x` on the specified `Z2X` or `BUS` cells, repeat this process until it converges.

Special Notes

- Keywords are case insensitive and always begin with `$`. The Conformal Constraint Designer supports minimal matching (for example, `$rand`). Below is the list of supported keywords. The number in ( ) is the required minimum leading characters (including the `$`):

  ```
  $RANDom (5)
  $INIT_END_time (9)
  ```

- Lines of the file do not have to be chronologically ordered (that is, lines with a smaller time unit can be specified later in the init file).
- If there exist two lines with an overlapping time unit and overlapping cell(s), but different value assignments, the latter one in the init file will overwrites the previous one. This is to allow, for example, `0 0 *`, followed by `0 1 reset`.

- Be aware of the difference in the following:

  `0 $random a` // Generates a random number on `a` at time 0
  `0-250 $random a` // Generates random numbers on `a` for time `0` to `250`

- To specify the `$random` on `Z2LOGIC` or `BUS_CONTention` for all time units, use:

  `0-$init_end_time $random  -Z2LOGIC`

  Note: `0 $random  -Z2LOGIC` only applies to `Z2LOGIC` at time `0`.

- The Conformal Constraint Designer ignores the backspace, tab, and new-line keys when used in the syntax. You may specify in free style (that is, there is no need for all characters to fit on one line).

- The Conformal Constraint Designer allows the following to denote comments: `//` *OR* `/* … */`

Examples

Legal Examples

```
0   0                          in1 a_reg   // can specify multiple signals in a line
0   0                          in*         // all signals started with "in"
0   0                          in* a*      // all signals started with "in" or "a"
0   0                          *           // all signals allowed at time 0
0   x                          in1[0]      // assign in1[0] = 1'bx
0   0101                       in3[3:0]    // assign in3[3:0] = 4'b0101
0   x                          in*[0]      // assign 1'bx to all in*[0]
0   0101                       in*[3:0]    // assign 4'b0101 to all in*[3:0]
0   4'b1x0                     in4         // assign in4 = 4'b01x0 (0 extension)
0   4'bx0                      in4         // assign in4 = 4'bxxx0 (x extension)
0   32'd500                    in5         // assign in5 = 32'd500
0   0                          in6[31:0]   // assign in6[31:0] = 32'b0
0   3'b101                     in6[31:0]   // assign in6[31:0] = 32'b101 (extended)
0   8'b11110000                in7[5:2]    // assign in7[5:2] = 4'b0000 (truncated)
0   0   -PI                                // all PIs
0   0   -DFF                               // all DFFs
```

```
0   0   -X_CONST                         // all X constant assignments
0   0   -PI -DFF -DLAT -BBOX             // all PIs DFFs DLATs BBOXs
0   0   -PI in1                          // in1 must be a PI; same as "0 0 in1"
0   0   -PI in1 -DFF -a_reg              // same as "0 0 in1 a_reg"
0   0   -PI * -DFF *                     // all PIs and DFFs; same as "0 0 -PI -DFF"
0   0   -PI -DFF a_reg                   // all PIs and a_reg DFF
0   0   -DFF a_reg -PI                   // same as "0 0 -PI -DFF a_reg"
0   0   -PI in* -DFF a*                  // all PIs started with "in" and DFFs with
                                         // "a"; different from "0 0 in* a*"
0   $random               -PI *          // assign random number to PIs
0-10 $random              -PI            // assign random number to PIs
                                         // for time 0 to 10
0 - 10  $random           -PI            // same as above; space is ignored
```

Illegal Examples

```
0   0                     a -PI b        // a needs to have type specifier
1   0                     -DFF           // cannot specify DFF other than time 0
0-10   0                  -DFF           // cannot specify DFF other than time 0
```

Risks

- Random assignment cannot guarantee to satisfy the constraints
- Random assignment cannot guarantee legal operation. The circuit may be brought into an illegal state after initialization. In this case, the Conformal Constraint Designer issues an INIT2 error.
- Random assignment may produce unwanted behavior of the design, for example, bus contention and internal don't-care. You can specify the x handling with `-Z2LOGIC` and `-BUS_CONTention` at different time frames.
- `-Z2LOGIC` and `-BUS_CONTention` are more suited to global than individual cell settings.
- When you try to assign a sequence of values using word-level information, these values can get lost. If this happens, use the following workaround:

  For example, instead of using:

```
for (1,15) 8'b10101010 din
```

Use:

```
$for (1,15) 8'b10101010 din[7:0]
```

Applying an Initialization Sequence

Choose one of the following methods to specify initial state values for the state elements in a design.

- Read a VCD dump file from a previously simulated initialization sequence using the READ INITIAL STATE command. The `-root` option specifies the instance path to the key module that corresponds to the root you specified when you read in the design. For example:

  ```
  read initial state myinit.vcd -vcd -root testbench/dut_instance -time 750
  ```

- Use the READ INITIAL STATE command to specify an initialization sequence defined in an input file. This is especially useful for multiple sessions where the size of the initialization sequence is large. For example:

  ```
  read initial state init.seq -sequence
  ```

  The following are examples of initialization sequence file lines:

  ```
  0   8'b11110000   in7[5:2]   // assign in7[5:2] = 4'b0000 (truncated)
  0   0   -PI                  // all PIs
  0   0   -DFF                 // all DFFs
  0   $random       -PI *      // assign random number to PIs
  ```

  If you include only the time on a line, simulation is forced to advance to the specified time unit without changing the initial values. At time 0, storage elements (flip-flops and latches) can be specified as well.

  Note: The values specified in the initialization sequence file do not need to follow chronological order.

- Add initial states to individual flip-flops using the ADD INITIAL STATE command.

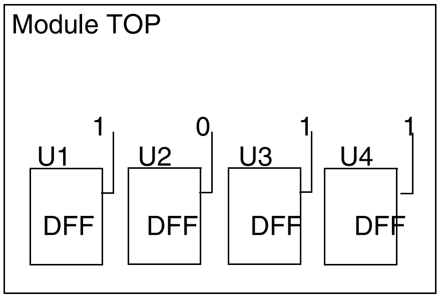
  Use the ADD INITIAL STATE command to initialize flip-flops and latches that were not initialized using another method. If you use this command for a flip-flop or latch that was already initialized, it creates a conflicting assignment. Then, the Conformal Constraint Designer issues a warning and *does not* overwrite the assignment.

  The following is an example of a conflict and the warning that the Conformal Constraint Designer issues:

  ```
  SETUP> add initial state 1 state_reg
  // Warning: Initial state of 'state_reg' has already assigned to 0
  ```

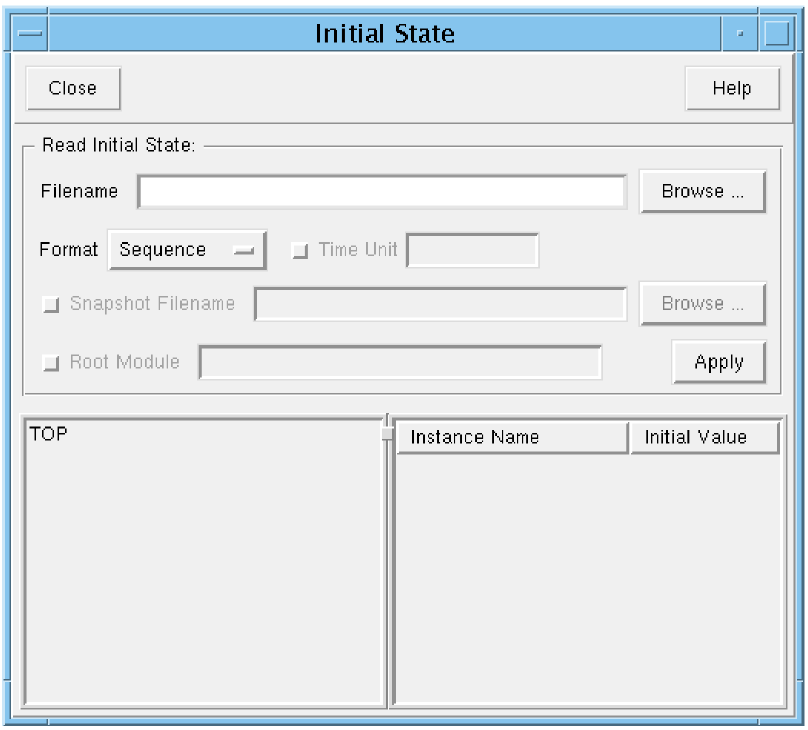
To avoid this situation, use the following command prior to making the new assignment:

```
SETUP> delete initial state state_reg
```




```
SETUP> add initial state 1 /TOP/U1
SETUP> add initial state 0 /TOP/U2
SETUP> add initial state 1 /TOP/U3
SETUP> add initial state 1 /TOP/U4
```

Alternatively, you can use the Initial State form (*Setup – Initial State*) to specify an initialization sequence for a circuit through a VCD dump file or an initial sequence file. You can also use this form to add individual initial states.



Initial State Fields and Options

Filename	Specifies the name of the VCD dump file. You can enter the name of the file or click *Browse* and select a file from the Init State File window.
Format	Specifies the VCD dump initialization mode.
Time Unit	Specifies the time unit.
Snapshot Filename	Specifies the name of the snapshot file. You can enter the name of the file or click *Browse* to locate a snapshot file. The snapshot is a reduced VCD file that stores only the information needed for the initialization time you specified. It can be dramatically smaller than a normal VCD file.
Root Module	Specifies the level in the VCD file that is to be used as the root level.
Format	Specifies the file format.

Specifying a VCD Dump for the Circuit Initialization

1. Do one of the following:
   - ❑ Click the *Browse* button to open the Init State File browser window and locate the desired VCD dump file.
   - ❑ Enter the name in the *Filename* field.
2. Click the *Format* button and choose *VCD* dump initialization mode.
3. Specify the time point at which the design is considered initialized as follows:
   a. Click the *Time Unit* check box.
   b. Click in the adjacent field and enter an integer.
4. To save a snapshot file, do the following:
   a. Click the *Snapshot Filename* check box.
   b. Click in the adjacent field and enter a filename (or use the *Browse* button to locate a snapshot file).

The snapshot is a reduced VCD file that stores only the information needed for the initialization time you specified. It can be dramatically smaller than a normal VCD file.

5. Specify the level in the VCD file that is to be used as the root level:
   a. Click the *Root Module* button.
   b. Enter the instance name in the *Root Module* field.
6. Click *Apply*.

Specifying an Initialization Sequence File

Use the following procedures to specify an initialization sequence file for circuit initialization.

1. Specify the initialization sequence file by doing one of the following:
   - ❑ Left click the *Browse* button to locate the desired initialization sequence file.
   - ❑ Enter the filename in the *Filename* field.
2. Click the *Format* field and choose *Sequence* initialization mode.
3. Click *Apply*.

Adding an Initial State

1. Click a module in the display located below the *Root Module* button.

   The module's instances appear in the adjacent display.
2. Click an instance name to select it.
3. Right-click and choose one of the initial states from the pop-up menu.
4. Click *Apply*.

Deleting Previously Specified Initial States

Use the following procedure to delete previously specified initial states.

1. Click an instance name in the *Instance Name* list.
2. Right click and choose one of the following from the pop-up menu:
   - ❑ To delete an initial state from a single instance, choose *Delete Initial State*.

- ❑ To delete initial states from all instances, choose *Delete All Initial States*.

B

Command Line Features

Command Line Editing

The non-GUI terminal of any Conformal tool supports the following editing functions:

In the following table, `^F` indicates pressing the `Ctrl` key and the `F` key simultaneously. Function keys have their names enclosed in angle brackets, for example, `<ESC>` is the Escape key.

The key sequences for basic editing functions are summarized in the following table.

Figure B-1 Basic Editing Functions

Keys	Function
`^F` or `<right-arrow>`	Move the cursor one character to the right
`^B` or `<left-arrow>`	Move the cursor one character to the left
`^A`	Move the cursor to the beginning of the line
`^E`	Move the cursor to the end of the line
`^U`	Delete the entire line
`^K`	Delete all characters from the cursor position to the end of the line
`<ESC>f`	Move the cursor forward by one word
`<ESC>b`	Move the cursor backward by one word
`^D`	Delete the character under the cursor

`^H` or `<Backspace>` or `^?`	Delete the character to the left of the cursor
`^R`	Redisplay the current line
`^L`	Clear the screen and show the current line at the top of the screen
`^I` or `<TAB>`	Complete word (See section below)

Every command that is successfully entered is saved in a history list. You can recall commands in the history list to avoid repeated typing. The history list has a size limit of 256k bytes and the oldest commands in the list will be discarded when this limit is exceeded.

Figure B-2 Command Line History

Key	Function
`^P` or `<up-arrow>`	Recall the previous history line
`^N` or `<down-arrow>`	Recall the next history line
`<ESC>p`	History search backward
`<ESC>n`	history search forward
`^Xh`	List the history
`<ESC><`	Recall the first history line
`<ESC>>`	Recall the last history line
`^D`	List all possible completions when cursor is at end of line

The history search capability looks into the history list for one that matches the beginning of the current line. If the search string contains wildcards (`*`, `?`), then the entire pattern is matched. This is useful for searching patterns in the middle of a line.

For example:

```
SETUP> usage
SETUP> echo hello
SETUP> echo bye
SETUP> us<ESC>p
SETUP> usage
SETUP> *hello*<ESC>p
SETUP> echo hello
```

The command line history can also be saved into a file using the command `SAVE DOFILE`.

Command Line Completion

Command line completion (or tab completion) is when the tool automatically fills in partially typed commands. The tool supports command line completion in VPX and TCL mode (using the appropriate commands for each mode).

Using Command Line Completion

Completion mode is activated by the `<TAB>` key. For example, if you press the `<TAB>` key after typing "`re`", you will get the following possible command completions:

```
SETUP> re<TAB>
read... remodel* report... reset... restore... reduce... remove* reset*
```

Partial command completions are listed with the postfix "`...`"; complete command completions are listed with the postfix "`*`". In the example above, the `remodel` command is complete, commands like "`read`" are not. To narrow the choices, type more characters. For example, press `<TAB>` after typing "`read`" will show the commands that begin with "`read`":

```
SETUP> read<TAB>
read cpf* read lef... read memory... read rule... read design* read library* read
pattern* read testcase* read fsm... read mapped... read rom...
```

Valid abbreviated commands are understood during command completions as illustrated below:

```
SETUP> ana m<TAB>
ana module* ana multiplier*
```

When there is only one choice, the command is completed automatically. For example, pressing `<TAB>` after typing "`read li`" will complete the command "`read librar`y". Typing `^D` (when the cursor is at the end of a line) lists the possible completions without making any completions. *Beware that using ^D on an empty line will terminate the tool*.

Command completion understands the `VPX` and `MAN` commands, and will complete the commands that come after. For example,

```
SETUP> man write l<TAB> SETUP> man write library
```

After the command is completed, pressing `<TAB>` will activate filename completion.

Repeating Actions

The effect of pressing a key can be repeated automatically by giving it a repeat count using the key sequence `<ESC>`*`numberX`* where *`number`* is the count in one or more digits, and `X` is the key to be repeated. For example, the following example repeats the single character deletion using the `<Backspace>` key 20 times.

```
SETUP> abcdef01234567890123456789<ESC>20<Backspace>
SETUP> abcdef
```

Notes

- Command completion completes one word at a time. For example, the partial input `"write ru"` needs two completions, one for `"rule"`, and one for `"check"` to result in the completed `"write rule check"` command. However, since there are no other commands that begin with `"write ru"`, only one completion should be necessary.
- Options are not completed.

C

Verilog Support

Verilog Configurations

A configuration is an explicit set of rules that specify the exact source description to be used to represent each instance in a design. There could be more than one model describing the same module if they are at different levels of abstraction, such as behavioral, synthesis, and simulation. A configuration allows you to specify which model is to be used for each instance (or selected instances) in the design.

The Conformal software supports a subset of Verilog configurations, as defined in the *Verilog 2005 Language Reference Manual*. In particular, 'hierarchical use configurations,' liblist ordering, and cell configurations are not supported. The namespace mapping is supported through the `READ DESIGN` command's `-map` and `-mapfile` options. The Conformal software supports the configuration of several levels of hierarchy through instance configurations.

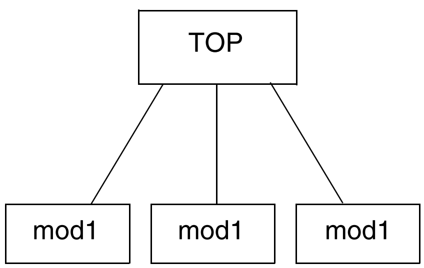
The Conformal software allows Verilog modules read during `READ DESIGN` to be stored in a user defined design namespace using the `-map` or `-mapfile` option. If either option is not specified, the Verilog modules are stored in the default design namespace called '`work`'. The verilog modules read in during `READ LIBRARY` are automatically stored in a default library namespace also called '`work`'. Thus, '`work`' is the name of two default namespaces in Conformal: default design namespace and default library namespace. For Liberty library cells, the name of the library is itself the name of the namespace.

If a module is specified in the configuration along with a library name, for example, `clock_lib.clock_mod_1`, then the module `clock_mod_1` is searched only in the namespace `clock_lib`. However, if a module `work.mod_2` or simply `mod_2` was specified, then the design namespace '`work`' is first searched for module `mod_2`. Only if the module is not found, is the library space '`work`' searched.

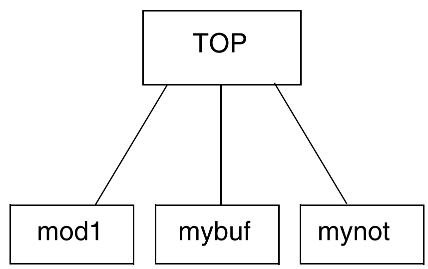
For example, if a design whose top module top has three instances `i1`, `i2` and `i3` of module `mod1`, and you read in the configuration `cfg1`, the following shows the design hierarchy before and after applying the configuration:

```
config cfg1;
    design work.top;
    instance top.i2 use mybuf;
    instance top.i3 use mynot;
endconfig
```

Before configuration

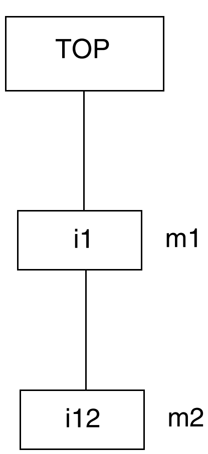


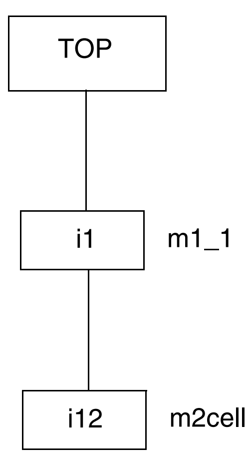
After configuration



In another example, if a design whose top module has an instance `i1` of module `m1`, and you want to configure the instance `i12` of `i1` to use liberty cell `m2cell` from the Liberty library `cell_lib_W125_V1`, you can use the following configuration:

```
config cfg1;
    design work.top;
    instance top.i1.i12 use cell_lib_W125_V1.m2cell;
endconfig
```





Verilog 2001 Support Tables

Supported

ANSI C Style Module Declarations
ANSI C Style Task/Function Declarations
ANSI C Style UDP Declarations
Arithmetic Shift Operators
Array Bit And Part Selects
Arrays Of Net
Assignment Width Extension Past 32 Bits
Automatic (Recursive) Functions
Automatic (Re-Entrant) Tasks
Combinatorial Logic Sensitivity Lists
Combined Port And Data Type Declarations
Comma Separated Sensitivity Lists
Constant Functions
Default Net Type None
Disabling Implicit Net Declarations
Enhanced Conditional Compilation
Explicit Inline Parameter Redefinition
Fixed Local Parameters
Generate Blocks
Implicit Nets For Continuous Assignments
Implicit Port Connections
Module Parameter Port Lists
Multi-Dimensional Arrays
Operator: <<< : Shift Left (Signed Data Type)
Operator: >>> : Shift Right (Signed Data Type)

Power Operator

Sign Conversion System Functions

Signed Based Integer Numbers

Signed Functions

Signed Reg, Net And Port Declarations

Sized And Typed Parameter Constants

Variable Vector Part Selects

Limited Support

Arrays of Instance

Conformal supports global hierarchical references to an instance of the `array_instance` (for example, `array_instance[0].reg1`)

Attributes

Conformal supports the following attribute pragmas:

- (* synthesis, full_case [= <optional_value>] *)
- (* synthesis, parallel_case [= <optional_value>] *)
- (* synthesis, black_box *) - Partial support: black box will apply to the followed module.
- (* synthesis, async_set_reset [="signal_name1, signal_name2, ..."] *)
- (* synthesis, fsm_state [=<encoding_scheme>] *)
- (* synthesis, implementation = "<value>" *)

Real Data Types

Conformal supports real type literals mixed with integer type in constant expression

Ignored

Reg Declaration Initial Assignments

Source File And Line Compiler Directive

Variable Initial Value At Declaration

Not Applicable

Enhanced File I/O

Enhanced Input Timing Checks

Enhanced Invocation Option Testing

Enhanced PLA System Tasks

Enhanced SDF File Support

Enhanced Verilog PLI Support

Extended Number Of Open Files

Extended VCD Files

Negative Input Timing Constraints

Negative Pulse Detection

On-Detect Pulse Error Propagation

Standard Random Number Generator

String Read And Write System Tasks

D

SystemVerilog Support

SystemVerilog Support Tables

The following tables are sorted by category.

Literals

IEEE 1800		Status
3.3	unsized literals	Supported
3.4	shortreal literals	Supported
3.5	time units in literals	Supported
3.5	time units in literals (step)	Unsupported
3.6	string literals	Supported
3.7	array literals	Supported
3.8	structure literals	Supported

Data Types

IEEE 1800		Status
4.3	logic (4-state) data types	Supported
4.3	integer and bit (2-state) data types	Supported
4.3	byte, shortint, longint	Supported
4.4	short real data types	Round to Int value
4.5	void data type (see void functions 12.3.1)	Void function
4.6	chandle data type	Unsupported
4.7	string data type	Unsupported
4.7	Parameters and localparams of strings	Unsupported
4.7	string data arrays	Unsupported
4.7	string operator: !=	Unsupported

IEEE 1800		Status
4.7	string operator: ==	Unsupported
4.7	string operator: <	Unsupported
4.7	string operator: <=	Unsupported
4.7	string operator: >	Unsupported
4.7	string operator: >=	Unsupported
4.7	string operator: concat {s1,s2}	Unsupported
4.7	string operator: {multiplier{s1}}	Unsupported
4.7	stringo perator: str[i]	Unsupported
4.7.1	string len()	Unsupported
4.7.2	string putc()	Unsupported
4.7.3	string getc()	Unsupported
4.7.4	string toupper()	Unsupported
4.7.5	string tolower()	Unsupported
4.7.6	string compare()	Unsupported
4.7.7	string icompare()	Unsupported
4.7.8	string substr()	Unsupported
4.7.9	string atoi()	Unsupported
4.7.9	string atohex()	Unsupported
4.7.9	string atooct()	Unsupported
4.7.9	string atobin()	Unsupported
4.7.10	string atoreal()	Unsupported
4.7.11	string itoa()	Unsupported
4.7.12	string hextoa()	Unsupported
4.7.13	string octtoa()	Unsupported
4.7.14	string bintoa()	Unsupported
4.7.15	string realtoa()	Unsupported
4.8	event data type	Unsupported

IEEE 1800		Status
4.9	User-defined types (use before called)	Supported
4.9	User-defined types (interface typedef scoping)	Supported
4.10	enumeration data type - 2 state	Supported
4.10	enumeration data type - 4 state	Supported
4.10.2	Enumeration data type (OOMR to enum constants)	Supported
4.10.2	Enumeration data type shorthand (name[N])	Supported
4.10.2	Enumeration data type shorthand (name[N]=C)	Supported
4.10.2	Enumeration data type shorthand (name[N:M])	Supported
4.10.2	Enumeration data type shorthand (name[N:M]=C)	Supported
4.10.1	typedef enum	Supported
4.10.2	enum type ranges	Supported
4.10.3	enum type checking	Supported
4.10.4	enum methods - numerical expressions	Supported
4.10.4	enum methods - constant expression for enum constant	Supported
4.10.4.1	enum methods - first	Supported
4.10.4.2	enum methods - last	Supported
4.10.4.3	enum methods - next	Supported
4.10.4.6	enum methods - name	Supported
4.10.4.4	enum methods - prev	Supported
4.10.4.5	enum methods - num	Supported
4.11	Packed structure data type	Supported
4.11	Packed structure data type (initializing members)	Supported
4.11	Unpacked structure data type (static arrays/reals)	Supported
4.11	Unpacked structure data type (string)	Unsupported
4.11	Dynamic objects inside unpacked structs	Unsupported
4.11	Unpacked structure data type (OOMR's to members)	Supported

IEEE 1800		Status
4.11	Packed union data type	Supported
4.11	Packed union data type (tagged)	Unsupported
4.11	Unpacked Union data type	Unsupported
4.11	Unpacked Union data type (tagged)	Unsupported
4.12	Class data type - store object handles in dynamic arrays (see 5.6)	Unsupported
4.12	Class data type - store object handles in queues (see 5.14)	Unsupported
4.12	Class data type - store object handles in associative arrays (see 5.9)	Unsupported
4.12	Class data type - store object handles in mailbox	Unsupported
4.14	Enum static casting	Supported
4.15	$cast dynamic casting (class type)	Unsupported
4.15	$cast dynamic casting (non-class type)	Unsupported
4.15	$cast dynamic casting (enums)	Supported
4.16	bit stream casting	Supported
4.17	Default attribute type	Unsupported

Arrays

IEEE 1800		Status
5.6,9,14,4.7	Public access to QDAs and strings	Unsupported
5.6,9,14,4.7	Local and protected access to QDAs and strings	Unsupported
5.6,9,14,4.7	QDAs as local variables in tasks/functions/methods	Unsupported
5.6	Dynamic arrays of strings	Unsupported
5.9,14,4.7	Queues and associative arrays of strings	Unsupported
5.6,9,14	Hierarchical (OOMR) refereces to QDAs	Unsupported
5.2	array of mailboxes	Unsupported

IEEE 1800		Status
5.2	packed arrays	Supported
5.2	unpacked arrays	Supported
5.2	packed arrays (slicing any dimension of multi-dimensional array)	Supported
5.4	unpacked arrays (slices)	Supported
5.4	indexing of arrays	Supported
5.5	array query functions	Supported
5.6	dynamic arrays (details below)	Unsupported
5.6	dynamic arrays of classes	Unsupported
5.6	dynamic arrays in classes (public/local)	Unsupported
5.6	dynamic arrays in packages	Unsupported
5.6	dynamic arrays i-- multidimensional	Unsupported
5.6.1	dynamic arrays with copy, resize	Unsupported
5.6.1	dynamic arrays - new[]	Unsupported
5.6.2	dynamic arrays - size()	Unsupported
5.6.3	dynamic arrays - delete()	Unsupported
5.7	array assignment	Supported
5.8	arrays as arguments	Supported
5.9	associative arrays	Unsupported
5.9	associative arrays of classes	Unsupported
5.9	associative arrays in classes (public/local)	Unsupported
5.9	associative arrays in packages	Unsupported
5.9.1	wildcard index types for integral types	Unsupported
5.9.2	string index types	Unsupported
5.9.3	class index types	Unsupported
5.9.4	integer/int index types	Unsupported
5.9.5	signed packed array index types	Unsupported
5.9.6	unsigned packed array index types	Unsupported

IEEE 1800		Status
5.9.7	associative arrays - indextype=other user defined type	Unsupported
5.1	associative array methods	Unsupported
5.1	associative array locator methods	Unsupported
5.11	associative array assignment	Unsupported
5.12	associative array arguments - pass by reference	Unsupported
5.12	associative array arguments - pass by value	Unsupported
5.13	associative array literals	Unsupported
5.14	queues	Unsupported
5.14	queues of classes	Unsupported
5.14	queues in classes (public/local)	Unsupported
5.14	queues in packages	Unsupported
5.15	array manipulation methods	Unsupported

Data Declarations

IEEE 1800		Status
6.3	constants (in classes)	Unsupported
6.3.3	parameterized types	Supported
6.3.5	const keyword parse and ignore (non-classes)	Supported
6.4	variables (var keyword support)	Supported
6.6	scope/lifetime (global scope - see $root)	Supported
6.6	scope/lifetime (unnamed blocks)	Supported
6.6	scope/lifetime (static/auto task/function/block data)	Supported
6.7	continuous assign to vars	Supported
6.8	signal aliasing	Supported
6.9	Type compatibility - incl passing subclass arg to superclass formal	Unsupported

IEEE 1800		Status
6.10	Type operator	Supported

Attributes

IEEE 1800		Status
	Default attribute type	Unsupported

Operators & Expressions

IEEE 1800		Status
8.2	Constraint operators shift, division, modulus, exponent, logical, concat	Supported
8.3	assignment operators as statements	Supported
8.3	assignment operators as expressions	Supported
8.3	postincrement/decrement statements	Supported
8.3	preincrement/decrement statements	Supported
8.3	++ and -- as expressions	Supported
8.5	wild equality/inequality	Supported
8.6	short real operators	Supported
8.12	concatenation	Supported
8.13	Unpacked array and structure assignment patterns except below:	Supported
8.13	assignment patterns - unpacked array	Supported
8.13	assignment patterns - unpacked structure	Supported
8.13	assignment patterns - left hand side assignment	Supported
8.13	assignment patterns - associations by type	Supported
8.13	assignment patterns - replications factors	Supported

IEEE 1800		Status
8.13	assignment patterns - simple ’ type qualification for assignments to OOMRs	Supported
8.13	assignment patterns - simple ’ type qualification for port connection expressions	Supported
8.13	Structure assignment expressions	Supported
8.14	Tagged unions	Unsupported
8.15	Aggregate expressions	Supported
8.16	Operator Overloading	Unsupported
8.17	Streaming Operators	Supported
8.18	Conditional operator	Supported
8.19	Set membership	Unsupported

Procedural Statements

IEEE 1800		Status
10.4	Selection statements - if	Supported
10.4	Selection statements - case	Supported
10.5.1	do while loop	Supported
10.5.2	enhanced for loop	Supported
10.5.3	foreach loop	Supported
10.5.3	foreach loop with procedural assignment	Supported
10.6	jump statements (return, break, continue)	Supported
10.7	final blocks	Supported
10.7	final blocks in programs	Unsupported
10.8	named blocks (matching end block name)	Supported
10.8	named blocks (statement labels)	Supported
10.10	iff event control	Supported
10.11	Level-sensitive sequence controls	Unsupported

Processes

IEEE 1800		Status
11.2	always_comb	Supported
11.3	always_latch	Supported
11.4	always_ff	Supported
11.5	continuous assignments (to variables)	Supported
11.6	join_none	Unsupported
11.6	join_none (disable)	Unsupported
11.6	join_none (wait on automatic variables with wait or event controls)	Unsupported
11.6	join_any	Unsupported
11.8	process control (wait fork)	Unsupported
11.8	process control (disable fork)	Unsupported
11.9	Fine grain process control	Unsupported

Tasks and Functions

IEEE 1800		Status
12.1	Task/func called via OOMR	Unsupported
12.1	Task/func - return object handle	Unsupported
12.1	Function return - string	Unsupported
12.1	Pass by value - object handles	Unsupported
12.2	default function argument types	Supported
12.2	default task argument types direction	Supported
12.2	multiple statements without begin/end	Supported
12.3	function output arguments	Supported
12.3.2	void functions	Supported

IEEE 1800		Status
12.3.2	discarding func return	Supported
12.4.2	Pass dynamic types by reference	Supported
12.4.2	Pass strings as arguments to tasks/functions	Supported
12.4.2	Pass mailboxes as arguments to tasks/functions	Supported
12.4.3	Default argument values	Supported
12.4.3	Default argument values (task/func referenced by OOMR)	Unsupported
12.4.4	Argument passing by name	Supported
12.4.5	Optional argument list	Supported
12.5	Pass ref arg of type array as actual to imported task/func having formal argument of type open array	Unsupported
12.5	Import tasks/functions (DPI)	Unsupported
12.5	Export tasks/functions (DPI)	Unsupported

Classes

IEEE 1800		Status
7.4	Objects (class instance) - null object handling; can pass as arg, etc.	Unsupported
7.4	Pass classes by ref to tasks/functions	Unsupported
7.4	Pass classes through module ports	Unsupported
7.4	Out Of Module References to class instances	Unsupported
7.4	Class instances passed to Out Of Module Reference tasks or functions	Unsupported
7.4	class instances passed to tasks/functions declared in a package	Unsupported
7.5	Object properties - dynamic arrays	Unsupported
7.5	Object properties - queues	Unsupported

IEEE 1800		Status
7.5	Object properties - assoc. arrays	Unsupported
7.5	Object properties - mailboxes (typeclass)	Unsupported
7.5	Object properties - event vars	Unsupported
7.5	Object properties - semaphores	Unsupported
7.5	Object properties - strings	Unsupported
7.5	Object properties - unpacked structs	Unsupported
7.5	Object properties - int types, packed structs	Unsupported
7.6	Object methods	Unsupported
7.7	Constructors - must support all arg types as in any function	Unsupported
7.8	Static class properties - of same object type as SV3.1a 11.5	Unsupported
7.9	Static methods	Unsupported
7.10	This - needs to be poymorphic; must be able to pass args	Unsupported
7.11	Assignment, renamic, and copying; myClass c = myOtherClass new;	Unsupported
7.12	Intstance and subclasses	Unsupported
7.13	Overridden members	Unsupported
7.14	Super - need to call with args	Unsupported
7.15	$cast - need for downcasting from base calass object handle (see also 3.15)	Unsupported
7.16	Chaining constructors - passing args to super.new(…)	Unsupported
7.17	Data hiding and encapsulation	Unsupported
7.17	Data hiding and encapsulation - parsing support	Unsupported
7.18	Constant class properties	Unsupported
7.18	Constant class properties - parsing support	Unsupported
7.19	Abstract classes - can use empty virtual methods	Unsupported

IEEE 1800		Status
7.19	Virtual methods	Unsupported
7.20	Polymorphism; dynamic method lookup	Unsupported
7.21	Class scope resolution operator ::	Unsupported
7.22	Out of block declarations	Unsupported
7.23	Parameterized classes	Unsupported
7.24	Typedef classes - forward referencing	Unsupported

Randomization & Constraints

IEEE 1800		Status
13.3	Random Variables - rand (class handles)	Unsupported
13.3	Random Variables - rand (unpacked structures)	Unsupported
13.3	Random Variables - rand (unpacked arrays)	Unsupported
13.3	Random Variables - rand (associative arrays)	Unsupported
13.3	Random Variables - rand (static arrays)	Unsupported
13.3	Random Variables - rand (dynamic arrays)	Unsupported
13.3	Random Variables - rand (dynamic array size)	Unsupported
13.3	Random Variables - rand (queues)	Unsupported
13.3	Random Variables - rand (enum support)	Unsupported
13.3	Random Variables - rand (multidimensional packed arrays)	Unsupported
13.3	Random Variables - rand (multidimensional arrays)	Unsupported
13.3	Random Variables - (packed structs)	Unsupported
13.3	Random Variables - (int types)	Unsupported
13.3	Random Variables - (array randomization: using arr.size as rand var)	Unsupported
13.4	Constraint blocks - concatenation within a constraint	Unsupported

IEEE 1800		Status
13.4	Constraint blocks - support for operators (/ % ** << >> <<< >>> ^~ | & ^?:)	Unsupported
13.4	Constraint blocks - var ordering	Unsupported
13.4	Constraint blocks - external	Unsupported
13.4	Constraint blocks - global (contain variables declared in other classes)	Unsupported
13.4	Constraint blocks -interative	Unsupported
13.4	Constraint blocks - distribution (rand with more than 1 dist constrain)	Unsupported
13.4	Constraint blocks - distribution (combo of weighted and complex constraints)	Unsupported
13.4	Constraint blocks - distribution (range an weight any integral SV expression)	Unsupported
13.4	Constraint blocks - distribution (dist expression any integral SV expression)	Unsupported
13.4	Constraint blocks - guards - compare handle with null (class handles in expressions)	Unsupported
13.4	Constraint blocks - guards (4 state logic evaluation)	Unsupported
13.4	Constraint blocks - inheritance (constrain name same in parent and derived)	Unsupported
13.4	Constraint blocks - implication (contain dist constraints)	Unsupported
13.4	Constraint blocks - static	Unsupported
13.4	Constraint blocks - override	Unsupported
13.4	Constraint blocks - named	Unsupported
13.4	Constraint blocks - 1d array for values of constraint inside operator	Unsupported
13.4.11	Constraint blocks - functions in constraints	Unsupported
13.4.3	Constraint blocks - set membership (arrays)	Unsupported
13.4.6	Constraint blocks - if … else (contain dist constraints)	Unsupported

IEEE 1800		Status
13.4.7	Constraint blocks - foreach	Unsupported
13.4.9	Constraint blocks - solve before	Unsupported
13.5	Randomization methods incl pre/post randomize	Unsupported
13.6	In-line constraints - randomize() with	Unsupported
13.7	rand_mode() - members of unpacked arrays	Unsupported
13.7	rand_mode() - members of unpacked structures	Unsupported
13.8	constraint_mode	Unsupported
13.10	In-line randomd variable control	Unsupported
13.10.1	In-line constraint checker	Unsupported
13.11	Randomization of scope Vars - (except below)	Unsupported
13.11	Randomization of scope Vars - (packed structs defined in classes)	Unsupported
13.11	Randomization of scope Vars - (packed structs defined in a package)	Unsupported
13.11	Randomization of scope Vars - (packed structs in dist expression)	Unsupported
13.11	Randomization of scope Vars - (packed structs in set membership)	Unsupported
13.11	Randomization of scope Vars - (bit or part select of packed structure array member)	Unsupported
13.11	Randomization of scope Vars - (class members in constraints or arguments)	Unsupported
13.11	Randomization of scope Vars - (multiple constrain expressions after an if or else)	Unsupported
13.12.1	$urandom (in classes)	Unsupported
13.12.2	$urandom_range (in classes)	Unsupported
13.12.3	$srandom	Unsupported
13.12.4	get_randstate()	Unsupported
13.12.5	set_randstate()	Unsupported
13.13	Random stability	Unsupported

IEEE 1800		Status
13.14	manually seeding randomize	Unsupported
13.15	Randcase	Unsupported
13.16.1	Randsequence - randome production weights	Unsupported
13.16.2	Randsequence - if …else production statements	Unsupported
13.16.3	Randsequence - case production statements	Unsupported
13.16.4	Randsequence - repeat production statements	Unsupported
13.16.5	Randsequence - interleaving production - rand join	Unsupported
13.16.6	Randsequence - aborting productions - break and return	Unsupported
13.16.7	Randsequence - value passing between productions	Unsupported

Synchronization

IEEE 1800		Status
14.2	semaphores	Unsupported
14.2	semaphores in packages	Unsupported
14.2	semaphores as protected/public in classes in packages	Unsupported
14.3	mailboxes	Unsupported
14.4	parameterized mailboxes	Unsupported
14.5.1	Triggering a named event	Unsupported
14.5.2	Non-blocking event triggering	Unsupported
14.5.3	Waiting for a named event	Unsupported
14.5.4	Persistent trigger: Triggered property	Unsupported
14.6	Event sequencing	Unsupported
14.7	Event variables without assignments	Unsupported

Scheduling Semantics

IEEE 1800		Status
9.3	Stratified event scheduler	Unsupported

Clocking Blocks

IEEE 1800		Status
15.2	Clocking blocks (in generate loops)	Unsupported
15.3	Input/output skews	Unsupported
15.4	Hierarchical expressions	Unsupported
15.5	Signals in multiple clocking blocks	Unsupported
15.6	Clocking block scope and lifetime	Unsupported
15.7	Multiple clocking blocks	Unsupported
15.8	Clocking blocks inside interfaces	Unsupported
15.9	Clocking block events	Unsupported
15.10	Cycle delay:##	Unsupported
15.11	Default clocking	Unsupported
15.12	Input sampling	Unsupported
15.13	Synchronous events	Unsupported
15.14	Synchronous drives	Unsupported

Program Blocks

IEEE 1800		Status
16.2-06	Program Blocks	Unsupported

Assertions

For more information, see System Verilog Assertions (SVA) on page 393.

IEEE 1800		Status
17.2	Immediate Assertions	Unsupported
17.4	Boolean Assertions	Supported
17.5	Sequences (see specifics under 17.6 and 17.7)	Unsupported
17.6	Declaring sequences	Unsupported
17.6.1	Typed formal argument in sequences	Unsupported
17.7.1	Sequence operator precedence	Unsupported
17.7.2	Sequence repetition in sequences	Unsupported
17.7.3	Sequence sampled value functions ($rose, $fell, $stable)	Supported
17.7.4	Sequence AND operation	Unsupported
17.7.5	Sequence INTERSECT operation	Unsupported
17.7.6	Sequence OR operation	Unsupported
17.7.7	Sequence first_match operation	Unsupported
17.7.8	Sequence throughout operation	Unsupported
17.7.9	Sequence within operation	Unsupported
17.7.10	Sequence ended, matched, and triggered	Unsupported
	Manipulating data in a sequence	Unsupported
17.8	Local variables of complex data types	Unsupported
17.8	Local variables	Unsupported
17.9	Calling subroutines on match of a sequence	Unsupported
17.10	system functions ($onehot, $inset, etc)	Supported
17.11	Declaring properties (see below)	Unsupported
17.11	Decaring properties in a module	Unsupported
17.11	Decaring properties in an interface	Unsupported
17.11	Declaring properties in a clocking block	Unsupported

IEEE 1800		Status
17.11	Declaring properties in a compilation unit scope	Unsupported
17.11	Declaring properties: operators NOT	Supported
17.11	Declaring properties: operators AND	Supported
17.11	Declaring properties: operators OR	Supported
17.11	Declaring properties: operators IFELSE	Supported
17.11	Declaring properties: operators |->	Supported
17.11	Declaring properties: operators |=>	Supported
17.11	Typed formal arguments in property	Unsupported
17.11.2	Implication	Supported
17.11.4	recursive properties	Unsupported
17.12.1	multiple clock support	Unsupported
17.13	concurrent assertions (see below)	Unsupported
17.13.1	assert statement	Supported
17.13.2	assume statement	Supported
17.13.3	cover statement	Unsupported
17.13.5	concurrent assertions in procedural code	Supported
17.14.1	clock resolution	Unsupported
17.14	clocked sequences	Unsupported
17.14	clock inferred from always block	Supported
17.14	Default clocking	Unsupported
17.15	bind directive (not including compilation unit)	Unsupported
17.28	assertion control tasks ($assertion/off/kill)	Unsupported
17.16	expect statement	Unsupported

Coverage

IEEE 1800		Status
18.2	Covergroups	Unsupported
18.3	Covergroup in classes	Unsupported
18.2	Covergroup in interfaces	Unsupported
18.2	Covergroup in program blocks	Unsupported
18.4	Defining coverage points	Unsupported
18.4.1	Specifying bins for transitions	Unsupported
18.4.2	Automatic bin creation for coverpoints	Unsupported
18.4.4	ignore_bins for coverpoints	Unsupported
18.4.5	illegal_bins for coverpoints	Unsupported
	Assertions in generates	Unsupported
	open-ended bins for coverpoints	Unsupported
	oOptions: name, comment, weight, per_instance, at_least, and goal	Unsupported
18.4.3	Wildcard specification of bins	Unsupported
18.4.4	Exclusion of coverpoints or transitions	Unsupported
18.4.5	Specifying illegal coverpoints or transitions	Unsupported
18.5	Cross products	Unsupported
18.5.2	Excluding cross products	Unsupported
18.5.3	Specifying illegal cross products	Unsupported
18.6	Procedural setting of options	Unsupported
18.7	Predefined coverage methods (start() and sample())	Unsupported
18.8	coverage system tasks/functions	Unsupported

Hierarchy

IEEE 1364-2005		Status
12.4.1	Generate support in if-generates	Supported
12.4.2	Generate support in for-generates	Supported

IEEE 1800		Status
19.2	packages - classes in packages	Unsupported
19.2	packages - extern_constraint declaration	Unsupported
19.2	packages - covergroup declaration	Unsupported
19.2	Packages - overload declaration	Unsupported
19.2	Packages - anonymous_program	Unsupported
19.3	compilation unit support	Supported
19.4	Top-level instance ($root)	Supported
19.6	nested modules	Supported
19.7	Extern modules	Unsupported
19.8	default port type/direction	Supported
19.8	event ports	Unsupported
19.8	interface ports	Supported
19.8	variable ports (logic, bit, byte, int, enum)	Supported
19.8	packed arrays on ports	Supported
19.8	unpacked arrays on ports	Supported
19.8	packed structures on ports	Supported
19.8	unpacked structures on ports	Supported
19.8	queues, dynamic/associative arrays, classes ports	Unsupported
19.8	strings ports	Unsupported
19.8	union ports	Unsupported
19.9	list of port expressions	Supported

IEEE 1800		Status
19.10	timeunitand timeprecision	Supported
19.11.3	implicit .name port connections	Supported
19.11.4	implicit .* port connections	Supported
19.11.4	implicit .* port connections (use in generate block)	Supported
19.12	ref ports	Supported
19.12	port connection rules (see below)	Supported
19.12	input ports declared as variables (built-in or user-defined types)	Supported
19.12	output ports connected to variables	Supported
19.12	output ports declared as variables	Supported
19.13	Extended name spaces	Supported
19.14	Hierarchical names (see 18.4)	Unsupported

Interfaces

IEEE 1800		Status
20.2	Attributes on interfaces	Unsupported
20.2	Nested interfaces	Unsupported
20.2	Nested interface instances	Unsupported
20.2	Interfaces connected to ports of interface	Unsupported
20.2	Interface arrays	Supported
20.2	Interfaces in generates	Unsupported
20.2.2	Named Bundles	Supported
20.2.3	Generic Bundles	Supported
20.3	Ports in Interfaces	Supported
20.4	Modports	Supported
20.5	Interfaces and Specify Blocks	Supported

IEEE 1800		Status
20.6	Modports - clocking keyword on Modports	Supported
20.6	Modports - task/function export	Supported
20.6	Extern fork/join tasks	Supported
20.6	Modports - task/function import	Supported
20.6	Tasks and functions on interfaces	Supported
20.6	Parameterized Interfaces	Supported
20.7	Access without ports (static interface)	Unsupported
20.8	Virtual Interfaces	Unsupported
20.9	Access to interface objects (through OOMRs) both port and hierarchical refs	Unsupported

Configuration Libraries

IEEE 1800		Status
21.2	config support for interfaces	Unsupported

System Tasks and Functions

IEEE 1800		Status
22.2	Elaboration time ’type’ keyword	Unsupported
22.3	expression size ($bits)	Supported
22.4	Range function $isunbounded	Unsupported
22.5	shortreal conversions $shortrealtobits, $bitstoshortreal	Unsupported
22.6	array querying (see 4.5)	Supported
22.7	assertion severity functions	Unsupported
22.8	assertion control functions ($asserton $assertoff $assertkill)	Unsupported

IEEE 1800		Status
22.9	assertion system functions	Unsupported
22.10	Random number system functions $urandom, $urandom_range	Unsupported
22.11	Program control - need $exit()	Unsupported
22.12	Coverage system functions	Unsupported
22.13	New format specifiers %u %z	Unsupported
22.13	$fread extensions	Unsupported
22.14	$readmemb, $readmemh,	Unsupported
22.15	$writememb, $writememh	Unsupported
22.16	file format considerations for multidimensional unpacked arrays (MDUAs)	Unsupported
22.17	system task args for MDUAs	Unsupported

VCD Data

IEEE 1800		Status
24.0	Mapping SV types to VCD format	Unsupported

Macros and Compiler Directives

IEEE 1800		Status
23.2	`define macros	Supported
23.3	‘include (angle brackets <filename>)	Supported
23.4	`being_keywords/`end_keywords (allow expanding set of keywords implied by command line)	Supported

Support for Macro Expansions

The SystemVerilog `` `define`` macro expansion is described in Section 23.2 of the SystemVerilog 1800 Standard.

In Verilog, the `` `define`` macro text can include a backslash (\ ) at the end of a line to show continuation on the next line. SystemVerilog enhances the Verilog `` `define`` text substitution macro compiler directive as follows:

- The macro text can include `` `" ``. An `` `" `` overrides the usual lexical meaning of `"` and indicates that the expansion should include an actual quotation mark. This allows string literals to be constructed from macro arguments.
- The macro text can include `` `\`" ``. A `` `\`" `` indicates that the expansion should include the escape sequence `\"`.
- The macro text can include ` `` `. This is used to delimit an identifier name without introducing white space. A ` `` ` delimits lexical tokens without introducing white space, allowing identifiers to be constructed from arguments.

However, the above specification does not describe how to treat escaped Verilog 2001 identifiers that contain macro parameters.

To work around this, the Conformal software has the `-KEEP_ESCAPED_ID` option for the `READ DESIGN` or `READ LIBRARY` commands.

When used, the `-KEEP_ESCAPED_ID` option keeps escaped identifiers, as in Verilog 2001.

For example, you have the following macro definition:

```
`define MACRO_TEST(head, tail) \head``_``tail
```

- If you use the `-KEEP_ESCAPED_ID` option, the escaped identifier is kept as ``\head``_``tail``.
- Without the `-KEEP_ESCAPED_ID` option, Conformal treats ``\head``_``tail`` as the concatenation of three parts:
  - `\head` an escaped identifier
  - `_` as an underscore
  - `tail` as a macro parameter that will be replaced by actual text passing to the `MACRO_TEST()`

  Note: `\head` is not recognized as the first argument (`head`) because it is not preceded with the ` `` ` operator. Instead, Conformal will expand

  ```
  `MACRO_TEST(aa, bb)
  ```

to

```
\head_bb
```

To treat `head` as a macro parameter:

```
`define MACRO_TEST(head, tail) \``head``_``tail
```

Conformal treats `` \``head``_``tail `` as the concatenation of four parts:

- ❑ `\` a backslash character
- ❑ `head` a macro parameter
- ❑ `_` an underscore
- ❑ `tail` a macro parameter to be replaced by an actual text passing to the `MACRO_TEST()`

and the `` `MACRO_TEST(aa, bb) `` call will be expanded to

```
\aa_bb
```

APIs

IEEE 1800		Status
26.0	Import tasks/functions; context; pure	Unsupported
26.4.2	Pure functions (optimizations)	Unsupported
26.4.6	Import/export function return types - longint	Unsupported
26.4.6	Import/export function return types - shortreal	Unsupported
26.4.6	Import/export function return types - chandle	Unsupported
26.4.6	Import function return types - string	Unsupported
26.4.6	Export function return types - string	Unsupported
26.4.6	Import/export function/task args - packed union of type bit/long	Unsupported
26.4.6	Import/export function/task args -enums	Unsupported
26.4.6	Import/export function/task dynamic array args (non-strings)	Unsupported
26.4.6	Import/export function/task dynamic array args	Unsupported

IEEE 1800		Status
26.4.6	export function/task strings	Unsupported
26.4.6	Import/export function/task open array handles for ints	Unsupported
26.4.6	Import/export function/task open array handles for strings	Unsupported
26.4	Export function/tasks - time consuming	Unsupported
28	Assertion API	Unsupported
29	Coverage API	Unsupported
30	Data Read API	Unsupported
	VPI Object Model	Unsupported

Annexes

IEEE 1800		Status
C	Std Package	Unsupported
D	Link Lists	Unsupported

Non-std

IEEE 1800		Status
	$psprintf	Unsupported

System Verilog Assertions (SVA)

The Conformal software accepts all syntactically correct SystemVerilog Assertion (SVA) constructs, including property and sequence declarations. However, only Boolean-level constraints are supported by Conformal. Other SVA constructs are ignored. Conformal supports 'assert' and 'assume' properties for Boolean expressions, and treats them as constraints. The software considers 'assert' and 'assume' as interchangeable, and treats both as constraints.

To read in SVA constructs in the design, run the `READ DESIGN` command with the `-sva` option. The following example shows a command sequence for reading in the SVA constructs in the `barrel_shifter.v` file:

```
read design -golden barrel_shifter.v -root barrel_shifter -sva
read design -revised shifter.v
set system mode lec
add compare point -all
compare
```

The following brief describes what the Conformal software supports for System Verilog:

- Conformal will read all SVA constructs including property and sequence declarations without erroring out during parsing.
- Conformal will support assert/assume property for boolean expressions. These assertions will be treated as constraints.
- Simple named property instantiations are also supported. Properties can be referred to by name inside another assertion. However, formal/actual argument passing in the named property is not currently supported.
- Assertions can also be embedded inside clocked always blocks.
- Simple default clocking block is supported.
- Sampled value functions are supported including `$rose`, `$past`, `$fell`, `$onehot`, `$stable`, and so on.
- Property operators are supported including negation, disjunction, conjunction, implication, and if-else.
- Binding properties are supported for properties declared inside/outside module declaration, and in a separate file.

The following brief describes what the Conformal software does not support for System Verilog:

- Complex sequential assertions that span over time
- Immediate assertions
- Conformal ignores cover statements
- Sequence operators

Supported SVA System Functions

The following system functions are supported by Conformal, but they are not sampled functions because no sampled clocks are used.

```
$onehot (<expression>) returns true if only 1 bit of the expression is high.
$onehot0 (<expression>) returns true if at most 1 bit of the expression is high.
$countones (<expression>) returns the number of ones in a bit vector expression.
```

For more details, see SV-1800 LRM, 17.10 System functions.

The following sampled value system functions are supported:

```
$sampled(expression [, clocking_event])
$rose( expression [, clocking_event])
$fell( expression [, clocking_event])
$stable( expression [, clocking_event])
$past( expression1 [, number_of_ticks] [, expression2] [,clocking_event])
```

For more details, see SV-1800 LRM, 17.7.3 Sampled value functions.

Default Clocking

The `clocking_event` option specifies the sampling clock edge for the Boolean expressions. Conformal only handles `@(posedge clk)` and `@(negedge clk)`. If the clocking event is not specified, a default clocking must be specified. Conformal supports the following SVA clocking statements:

```
  clocking clocking_identifier clocking_event ;
endclocking [ : clocking_identifier ]
```

```
default clocking clocking_identifier ;
```

```
  default clocking clocking_event ;
endclocking
```

```
  default clocking clocking_identifier clocking_event ;
endclocking [ : clocking_identifier ]
```

Property Declaration

Property can be specified using the following property declaration:

```
     property_declaration ::=
          property property_identifier [ ( [ tf_port_list ] ) ] ;
          [@(<clocking_event>] [ disable iff (<expression>) ] <property_expr>;
          endproperty [ : property_identifier ]
```

For example:

```
property GT (x, y); @posedge clk (x > y);
endproperty

p1: assume property (GT (vec1, vec2));
// constraint (vec1 > vec2) @posedge clk

p2: assume property @ posedge clk (vec1 > vec2);
// same effect as p1 above
```

Property Binding

Conformal supports property binding to specific modules or instances. For more details, see SystemVerilog LRM 17.15 “Binding properties to scopes or instances”

Binding properties are supported for properties declared, inside/outside module declaration, and in separate file.

Supported SVA Properties

Conformal supports ’assert property (`<property_spec>`) as ’assume property (`<property_spec>`), which is used as constraint. The `<property_spec>` can be in several forms which are described in the following sections.

Clocked Boolean Expression

The `<property_spec>` can be specified as one of the following clocked Boolean expression. If `b1`, `b2`, `b3`, `b4` are `<property_spec>`, then Conformal supports:

```
assert property ([@(<clocking_event>] [disable iff (b1)] b2);
assert property ([@(<clocking_event>] [disable iff (b1)] if (b2) b3 else b4);
```

The `<property_expr>` can be any Boolean expressions, SVA system functions, SVA sampled value functions connected by Verilog operators and the property operators: `not`, `or`, `and`, `|->`, `if-else`.

The following example shows default clocking applied to the assert property:

```
default clocking master_clk @(posedge clk);
endclocking
assert property (b2);
```

The Boolean expression `b2` is sampled at the `(posedge clk)`.

Property Specification

Conformal supports the following property declaration and assert or assume property using the declared property name. For example:

```
property GT (x, y);
  @posedge clk (x > y);
endproperty
p1: assume property (GT (vec1, vec2)); // constraint (vec1 > vec2) @posedge clk
```

SVA Extension Using assert final Statement

Conformal also supports the following extension to the SVA:

```
assert final(<property_expr>) [statement_or_null] [ else statement_or_null ]
     e.g. assert final(rst || $onehot(sig))
```

The [else statement_or_null] portion is ignored when translating assertion statement to constraint. The assert final is treated as combinational constraint, which can be used in both concurrent code and procedural code.

SVA Embedded in Procedural Code

Conformal supports embedded SVA inside clocked always blocks. For more details, see SystemVerilog LRM, 17.13.5 “Embedding concurrent assertions in procedural code.”

Examples

The following are two shifters. One is the normal shifter and the other is a barrel shifter.

```
///////////////// normal shifter ///////////////////////////
module shifter(clk, rst_n, in, shift_amoung, out);

  input       clk;
  input       rst_n;
  input [7:0] in;
  input [2:0] shift_amoung;

  output [7:0] out;

  reg [7:0] out, out_t;

  always@(posedge clk or negedge rst_n) begin
    if(~rst_n)
        out <= #(`delay) 8'd0;
      else
        out <= #(`delay) out_t;
  end

  always@(shift_amoung or in) begin
    if(shift_amoung[0])
        out_t = in << 1;
      else if(shift_amoung[1])
        out_t = in << 2;
      else if(shift_amoung[2])
        out_t = in << 4;
      else
        out_t = in;
  end

endmodule
///////////////// normal shifter ///////////////////////////

///////////////// barrel shifter ///////////////////////////
module barrel_shifter(clk, rst_n, in, shift_amoung, out);

  input       clk;
  input       rst_n;
  input [7:0] in;
  input [2:0] shift_amoung;

  output [7:0] out;

  reg [7:0] temp;
  reg [7:0] out, out_t;

  always@(posedge clk or negedge rst_n) begin
    if(~rst_n)
        out <= #(`delay) 8'd0;
      else
        out <= #(`delay) out_t;
  end

  integer i;

  always@(i or shift_amoung or in or temp or out_t) begin
    temp = in;

    for(i=0;i<3;i=i+1) begin
        if( shift_amoung[i] )
          out_t = temp << (1<<i);
      else
          out_t = temp;
```

```
        temp = out_t;
      end
    end
endmodule
///////////////// barrel shifter ///////////////////////////
```

The Conformal software reports Non-EQ results for the two shifters. It reports EQ results if you add an "`assert property ($onehot(shift_amoung));`" statement to the `barrel_shifter` module as follows:

```
///////////////// barrel shifter ///////////////////////////
module barrel_shifter(clk, rst_n, in, shift_amoung, out);
  input       clk;
  input       rst_n;
  input [7:0] in;
  input [2:0] shift_amoung;
  output [7:0] out;
  reg [7:0] temp;
  reg [7:0] out, out_t;
  always@(posedge clk or negedge rst_n) begin
    if(~rst_n)
        out <= #(`delay) 8'd0;
      else
          out <= #(`delay) out_t;
  end
  integer i;
  always@(i or shift_amoung or in or temp or out_t) begin
    temp = in;
    for(i=0;i<3;i=i+1) begin
        if( shift_amoung[i] )
          out_t = temp << (1<<i);
    else
          out_t = temp;
        temp = out_t;
    end
end
  assert property ($onehot(shift_amoung));
endmodule
///////////////// barrel shifter ///////////////////////////
```

Sample for Default Clocking

```
///////////////// default clocking //////////////////////
module m_unique(q, d, sel, clk);
input  [1:0] d;
input  [2:0] sel;
input        clk;
output [1:0] q;
reg    [1:0] q;

always @(posedge clk) begin
  case (sel)
    3'b001: q <= d;
    3'b010: q <= ~d;
    3'b100: q <= {d[0], d[1]};
    3'b111: q <= 'b0;
  endcase
end

default clocking CLK @(posedge clk); endclocking
assert property ( $onehot(sel) );

endmodule

///////////////////////////////////////////////////////////
```

With the default clocking declaration, the combination property assert property `( $onehot(sel) )` will be translated to clocked constraint assert property `( @(posedge clk) $onehot(sel) )` the same as the following assert property specified.

```
///////////////////////////////////////////////////////////
module m_unique(q, d, sel, clk);
input  [1:0] d;
input  [2:0] sel;
input        clk;
output [1:0] q;
reg    [1:0] q;

always @(posedge clk) begin
  case (sel)
    3'b001: q <= d;
    3'b010: q <= ~d;
    3'b100: q <= {d[0], d[1]};
    3'b111: q <= 'b0;
  endcase
end

assert property ( @(posedge clk) $onehot(sel) );

endmodule
///////////////// default clocking //////////////////////
```

A

VHDL Support

Supported and Unsupported IEEE Packages

The following table lists standard and IEEE packages in two columns: Supported and Not Supported (Ignored).

Standard and IEEE Packages	
Supported:	**Partially Supported:**
standard.vhdl	vital_primitives-body.vhdl
textio.vhdl	vital_primitives.vhdl
std_logic_1164.vhd	vital_timing-body.vhdl
std_logic_arith.vhd	vital_timing.vhdl
std_logic_misc.vhd	
std_logic_signed.vhd	
std_logic_unsigned.vhd	
std_logic_textio.vhd	

For a list of Vital packages that are supported, see Vital Package Support on page 407.

The following table lists the RTL VHDL synthesis subset constructs that are:

- Supported
- Ignored
- Unsupported

Support Status for RTL VHDL Synthesis Subset Constructs		
Design Units:	entity	supported
	generics	supported
	port default value	supported for undriven submodule input ports.
Architectures:	multiple architectures	supported

Support Status for RTL VHDL Synthesis Subset Constructs		
	global signals	supported* See Global Signal on page 410.
Configurations:	configuration declaration	supported
	block configuration	supported
	use	supported
	attribute specifications	ignored
	component configurations	supported* See Component Configuration on page 411.
	hierarchical block configuration	ignored
Packages:	standard/predefined packages	supported
	IEEE arith/signed/unsigned packages	supported
	Libraries	supported
Subprograms:	default value	ignored
	unconstrained parameters	supported
	subprogram recursion	supported
	resolution functions	supported
Data Types:	enumeration	supported
	integer	supported
	physical	ignored
	floating	ignored
	one-dimensional array	supported
	two-dimensional array	supported
	three-dimensional array	supported
	multi-dimensional array	supported
	record	supported

Support Status for RTL VHDL Synthesis Subset Constructs		
	access	ignored
	file	ignored
	incomplete type declaration	unsupported
Declarations:	constant	supported
	deferred constant	unsupported
	signal	supported
	register	unsupported
	bus	supported
	initial value	supported* See Initial Value on page 414.
	variable	supported
	shared variable	supported* See Shared Variable on page 414.
	file	ignored
	buffer port	supported
	linkage port	supported
	alias	supported
	component	supported
	attribute	supported
Specifications:	attribute others/all	supported
	configuration specifications	supported
	disconnection specifications	unsupported
Names:	simple names	supported
	selected names	supported
	operator symbols	supported
	indexed names	supported

Support Status for RTL VHDL Synthesis Subset Constructs		
	sliced names	supported* See Sliced Names on page 415.
	predefined attributes	supported* See Predefined Attributes on page 416.
	user-defined attributes	supported* See User-Defined Attributes on page 420.
Operators:	logical	supported
	relational	supported
	addition	supported
	signing	supported
	multiplying	supported
	miscellaneous	supported
	operator overloading	supported
	short-circuit operations	unsupported
Expressions:	based literals	supported
	null literals	unsupported
	physical literals	ignored
	strings	supported
	aggregates	supported
	function calls	supported* See Function Calls on page 420.
	qualified expressions	supported
	type conversions	supported
	allocators	unsupported
	static expressions	supported
	universal expressions	supported

Support Status for RTL VHDL Synthesis Subset Constructs		
Sequential Statements:	wait	supported* See Wait Statements on page 421.
	assertion	ignored
	report	ignored
	guarded signal assignment	supported* See VHDL GUARDED Block Support on page 423.
	transport / after	ignored
	signal assignment	supported* See Signal Assignment on page 423.
	variable assignment	supported
	procedure call	supported* See Procedure Calls on page 425.
	if statement	supported
	case statement	supported
	for loop statement	supported* See For Loops on page 425.
	while loop statement	unsupported
	next statement	supported
	exit statement	supported
	return statement	supported
	null statement	supported
Concurrent Statements:	block guard	supported* See VHDL GUARDED Block Support on page 423.
	block	supported
	process	supported
	sensitivity list	ignored

Support Status for RTL VHDL Synthesis Subset Constructs		
	concurrent procedure call	supported
	concurrent assertion	ignored
	concurrent signal assignment	supported* See Signal Assignment on page 423.
	guarded concurrent signal assignment	supported* See VHDL GUARDED Block Support on page 423.
	multiple waveforms	unsupported
	component instantiation	supported
	generate	supported

Note: The * denotes limited support. See the following section for information about restrictions on these constructs.

Vital Package Support

The Conformal software support the following functions and procedures with ignored delay values:

`VitalAND`	`VitalXOR2`	`VitalNOR3`	`VitalBUF`	`VitalDECODER2`
`VitalOR`	`VitalNAND2`	`VitalXNOR3`	`VitalINV`	`VitalDECODER4`
`VitalXOR`	`VitalNOR2`	`VitalAND4`	`VitalMUX2`	`VitalDECODER8`
`VitalNAND`	`VitalXNOR2`	`VitalOR4`	`VitalMUX4`	`VitalDECODER`
`VitalNOR`	`VitalAND3`	`VitalXOR4`	`VitalMUX8`	`VitalPathDelay`
`VitalXNOR`	`VitalOR3`	`VitalNAND4`	`VitalMUX`	`VitalPathDelay01`
`VitalAND2`	`VitalXOR3`	`VitalNOR4`		`VitalPathDelay01Z`
`VitalOR2`	`VitalNAND3`	`VitalXNOR4`		`VitalWireDelay`

Read Design

Important

You must specify all necessary VHDL files explicitly in the `READ DESIGN` command. In addition, you must read in all related VHDL files in a single `READ DESIGN` command.

Library Mapping

You can specify how VHDL libraries are mapped using the `READ DESIGN` command's `-map`, `-mapfile`, or `-library` options.

The `-map` and `-library` options work the same in that they map logical library names to physical directories. You can use multiple `-map` commands to map multiple physical directories to one logical library. Use the `-mapfile` option for more specific library mapping, such as specifying that a list of files must be compiled into a specified library. If you read in a file without specifying its library mapping, that file is stored in a default library called `work` in a design space or `worklib` in a library space.

Note: You can map a file into more than one library. In this case, the file is stored in each library for which it is mapped.

Performing Library Mapping

This section demonstrates how to use the `READ DESIGN` command to perform library mapping.

For example, your current directory contains the following files:

Physical File/Directory	Contents
`top.vhd`	See Example A-1.
`lib1/pkg1.vhd`	Package `package1`
`lib1/pkg1_body.vhd`	Package body of `package1`
`lib2/pkg2.vhd`	Package `package2`
`lib2/pkg2_body.vhd`	Package body of `package2`

Table A-1 Desired Library Mapping

Logical Library Name	Physical File/Directory
LIB1	lib1
LIB2	lib2
work	top.vhd (implicit)

Example A-1 Contents of top.vhd

```
-------- top.vhd begin --------
library LIB1;
use LIB1.package1.all;

library LIB2;
use LIB2.package2.all;

entity top ...;
architecture rtl of top ...;
-------- top.vhd end --------
```

To achieve the Desired Library Mapping outlined in Table A-1, the `READ DESIGN` command should look like one of the following:

- `read design -vhdl top.vhd -map LIB1 lib1 -map LIB2 lib2`
- `read design -vhdl top.vhd -library LIB1 lib1 -library LIB2 lib2`
-
```
read design -vhdl top.vhd \
      -mapfile LIB1 lib1/pkg1.vhd lib1/pkg1_body.vhd \
      -mapfile LIB2 lib2/pkg2.vhd lib2/pkg2_body.vhd
```

Note: The tool terminates the <file_list> for `-mapfile` when it encounters the next option or the end of the `READ DESIGN` command. For example, the following command does not generate the desired library mapping for this example. The tool terminates the file list at `top.vhd`; because of this, `top.vhd` is added to the `LIB2` library—not the `work` directory.

```
read design -vhdl \
     -mapfile LIB1 lib1/pkg1.vhd lib1/pkg1_body.vhd \
     -mapfile LIB2 lib2/pkg2.vhd lib2/pkg2_body.vhd \
      top.vhd
```

In the following example, `top.vhd` is correctly added to the `work` library because the `LIB2` file list terminates at `lib2/pkg2_body.vhd`.

```
read design -vhdl \
     -mapfile LIB1 lib1/pkg1.vhd lib1/pkg1_body.vhd \
     -mapfile LIB2 lib2/pkg2.vhd lib2/pkg2_body.vhd \
```

```
-golden \
top.vhd
```

Handling Unspecified Library Mappings

The tool handles `library.declaration` references as follows:

- If the library is defined and the declaration exists, the tool returns the declaration. Otherwise, the tool searches for the declaration in the `work` directory. If the tool finds the declaration, it returns the declaration.
- If the library is undefined, because of unspecified library mappings, the tool searches through the `work` library. If the tool finds the declaration in the `work` library, it returns the declaration with a note; otherwise, the tool returns an error message.
- If the tool finds a `work.declaration` reference while parsing a file that is stored in a logical library (for example, `lib1`), the tool searches through `lib1`, and then through the default `work` library for the declaration. Once the tool finds the declaration, it returns the declaration. The tool notifies you when it returns a declaration from the default `work` library.

Architectures

Global Signal

Restriction

The Conformal software does not support a Global Signal when the design includes it in multiple entities. When the design uses a Global Signal within an entity, it is treated as a local signal.

Example

In the following example, the Conformal software does not support the Global Signal `glob1` because it is used in two entities. See lines 10 and 19 in bold.

```
1.   PACKAGE pack IS
2.   SIGNAL glob1 : BOOLEAN;
3.   END pack;
```

```

USE work.pack.all;
ENTITY test IS
… END test;
ARCHITECTURE arch OF test IS
…
glob1 <= in0 OR in1;
END arch;

USE work. pack.all;
ENTITY test2 IS
…
END test2;
ARCHITECTURE arch OF test2 IS
…
glob1 <= in0 AND in1;
END arch;
```

Configurations

Component Configuration

The Conformal `GENERATE` support Component Configurations for references to labels and indices of GENERATE statements. In the following example, the Component Configuration uses `GENERATE` labels and indices. See bold lines 17, 20, and 24.

```
1.  ARCHITECTURE rtl_arch OF design IS
2.    COMPONENT comp_a PORT( … ) END COMPONENT;
3.    COMPONENT comp_b PORT( … ) END COMPONENT;
4.  BEGIN
```

```
  gen_label_1: FOR idx IN 0 TO 255 GENERATE
    comp_a (…);
  END FOR;
  gen_label_2: FOR idx IN 0 TO 255 GENERATE
    comp_b (…);
  END FOR;
END

CONFIGURATION real_config OF design IS
  USE work.all;
  FOR rtl_arch
     -- using generate statement label and indices
     FOR gen_label_1(255 DOWNTO 1)
      USE CONFIGURATION my_lib.comp_a_config;
     END FOR;
     FOR gen_label_1(0)
       USE CONFIGURATION my_lib.comp_a_config_2;
     END FOR;
     -- using generate statement label w/o indices
     FOR gen_label_2
      USE CONFIGURATION my_lib.comp_b_config;
    END FOR;
  END FOR;
END;
```

Nested Configurations

The Conformal software supports nested configurations and configurations with more than one level of hierarchy. It allows multiple architectures of an entity to exist by creating new modules with composite names created from the entity and architecture names. The Conformal software also allows the same architecture to be configured differently internally for different instances by creating a unique composite name for each such differing sub-configuration.

The following is an example:

```
-- entity e1 has two architectures e1a0 and e1a1
entity e1 is
    port (e1out : out BIT);
end e1;

architecture e1a0 of e1 is
begin
    e1out <= '0';
end e1a0;

architecture e1a1 of e1 is
begin
    e1out <= '1';
end e1a1;

-- entity e2 has an architecture e2arch which has a component C1
entity e2 is
    port (e2out : out BIT);
end e2;

architecture e2arch of e2 is
    component C1 is
        port (e1out : out BIT);
    end component C1;
begin
    e2i1 : C1 port map (e1out => e2out);
end e2arch;

-- a configuration for e2 which binds component C1 to
entity/architecture e1(e1a0)
use work.e1;
configuration e2conf of e2 is
    for e2arch
        for e2i1 : C1 use entity e1(e1a0);
        end for;
    end for;
end configuration e2conf;

-- entity e3
entity e3 is
    port (e3out1, e3out2 : out BIT);
end e3;

architecture e3arch of e3 is
    component C2 is
        port (e2out : out BIT);
    end component C2;
begin
    e3i1 : C2 port map (e2out => e3out1);
    e3i2 : C2 port map (e2out => e3out2);
end e3arch;

configuration e3conf of e3 is
    for e3arch
        for e3i1 : C2
```

```
            use configuration work.e2conf;  -- nested configuration
        end for;
        for e3i2 : C2
            use entity work.e2(e2arch); -- further configuring sub-hierarchy
            for e2arch
                for e2i1 : C1 use entity work.e1(e1a1);
                end for;
            end for;
        end for;
    end for;
end configuration e3conf;
```

Declarations

Initial Value

The Conformal software supports Initial Value variables or signals with the `READ DESIGN` command's `-initial_value` option.

Example

In the following example, signal `out1` will get the initial value of high, which is '`1`'. This initial value '`1`' will be discarded if the variable `high` is assigned.

```
proc1 : PROCESS is
    VARIABLE high : BIT := '1';
    BEGIN
        out1 <= high;
END PROCESS proc1;
```

Shared Variable

Restriction

The Conformal software does not support Shared Variables when they are declared inside a package.

Example

In the following example, the Conformal software does not support the `SHARED VARIABLE counter` because it is declared inside a package. See line 5, in bold.

```
1.   LIBRARY IEEE;
2.   USE IEEE.STD_LOGIC_1164.ALL;
3.
4.   PACKAGE pack1 IS
5.   SHARED VARIABLE counter: INTEGER RANGE 0 TO 99 := 0;
6.   END PACKAGE pack1;
```

Names

Sliced Names

Restriction

The Conformal software does not support Sliced Names when their ranges are not computable.

Example

In the following example, the Conformal software does not support Sliced Name `in0 (idx DOWNTO 0)` because input `idx` is not computable. See line 15, in bold.

```
1.   ENTITY test IS
2.     PORT (
3.       clk : IN BIT;
4.       idx : IN INTEGER RANGE 0 TO 3;
5.       in0 : IN BIT_VECTOR(3 DOWNTO 0);
6.       out0 : OUT BIT_VECTOR(3 DOWNTO 0)
```

```
  );
end test;

ARCHITECTURE arch OF test IS
BEGIN
  proc1 : PROCESS (clk)
    BEGIN
      IF clk‘EVENT AND clk=‘1’ THEN
        out0 <= in0(idx DOWNTO 0);
      END IF;
    END PROCESS;
END arch;
```

Predefined Attributes

The Conformal software only supports the following Predefined Attributes:

- LEFT
- RIGHT
- HIGH
- LOW
- RANGE
- REVERSE_RANGE
- LENGTH
- ASCENDING
- LEFTOF
- RIGHTOF
- PRED
- SUCC
- POS
- VAL
- BASE
- EVENT*
- STABLE*
- LAST_VALUE*
- TRANSACTION*

*See Restriction 2 on page 417.

Restriction 1

The return value of a Predefined Attribute must be globally computable.

Example 1

In the following example, the Conformal software does not support the Predefined Attribute `RANGE` because the variable `tmp` is not computable. See line 13, in bold.

```
1.  ENTITY attributes3 IS
2.    PORT ( input : IN BIT_VECTOR(7 DOWNTO 0);
3.           output1 : OUT BIT_VECTOR(7 DOWNTO 0);
4.           idx : In INTEGER RANGE 0 TO 7
5.         );
6.  END attributes3;
7.
8.  ARCHITECTURE arch OF attributes3 IS
9.  BEGIN
10.   PROCESS ( input )
11.   VARIABLE tmp: BIT_VECTOR(idx DOWNTO 0);
12.   BEGIN
13.      FOR i IN tmp'RANGE LOOP
14.        output1(i) <= input(i) XOR '1';
15.      END LOOP;
16.   END PROCESS;
17. END arch;
```

Restriction 2

The Conformal software supports the asterisked (*) predefined attributes (shown above), but only when they are used with synthesizable clock expressions.

Example 2a

In this example, the Conformal software supports the Predefined Attribute `STABLE` because it is used in a synthesizable clock expression on line 3 (in bold).

```
1.    PROCESS
2.     BEGIN
3.       IF NOT clk'STABLE AND clk = '1' THEN
4.    ...
```

Example 2b

In this example, the Conformal software does not support the Predefined Attribute `EVENT` because the design uses it in a non-synthesizable clock expression on line 3 (in bold).

```
1.  PROCESS
2.   BEGIN
3.     IF clk'EVENT THEN
4.  …
```

Out-of-Range Handling

For the following attributes:

- LEFTOF
- RIGHTOF
- PRED
- SUCC
- VAL

If variable x is out-of-range, the Conformal software has two choices to interpret the attribute value:

- If -RANGECONSTRAINT is specified in the READ DESIGN command, (or `set hdl compiler rangeconstraint`), Conformal will result dont care for attributes when 'x' if out-of-range

- If -RANGECONSTRAINT is not specified, the Conformal software will result a value for attribute as following:

`T'VAL(x)`	x itself
`T'SUCC(x)`	`T'RIGHT` if T is ascending, `T'LEFT` is T is descending
`T'PRED(x)`	`T'LEFT` if T is ascending, `T'RIGHT` is T is descending
`T'RIGHTOF(x)`	`T'RIGHT`
`T'LEFTOF(x)`	`T'LEFT`

For example:

```
type T is (e0, e1, e2, e3, e4, e5);
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of T: type is "1100 0110 1000 0110 0100 0001";
subtype ST is T range e4 downto e1;

-- p = 0
ST'val(p) = 0000
-- x = e0 (1100)
ST'succ(x) = e4 (0100)
ST'pred(x) = e1 (0110)
ST'rightof(x) = e1 (0110)
ST'leftof(x) = e4 (0100)
-- x = e1 (0110)
ST'pred(x) = ST'rightof(x) = e1 (0110)
-- x = e4 (0100)
ST'succ(x) = ST'leftof(x) = e4 (0100)
```

Note: The default values of the attributes are only for the scenarios where x is a variable. If the argument of these attributes is a constant which is out-of-range, the Conformal software will error it out.

If you use the `READ DESIGN` command with the `-architecture`, `-configuration`, `-rootconfig`, and `-lastmod` options, the Conformal software links the entity/architecture based on the following priorities:

1. `-rootconfig` has the highest priority.
2. `-configuration` has the second priority.
3. `-architecture` has the third priority.
4. `-lastmod` is the fourth priority.

User-Defined Attributes

Restriction

Conformal ignores all User-Defined Attributes except when they are used in one of the following forms:

- Form A:

```
TYPE state_type IS (Init, State1, State2, State3);
ATTRIBUTE enum_encoding : STRING;
ATTRIBUTE enum_encoding OF state_type :
     TYPE IS "0001 0010 0100 1000";
SIGNAL current_state, next_state: state_type;
```

- Form B:

```
TYPE state_type IS (Init, State1, State2, State3);
ATTRIBUTE enum_encoding : STRING;
ATTRIBUTE enum_encoding OF state_type :
     TYPE IS      "Init=0001,State1=0010,State2=0100,State3=1000";
SIGNAL current_state, next_state: state_type;
```

Expressions

Function Calls

Restriction

The Conformal software does not support a Function Call when the function includes a `WAIT` construct or clock signal.

Example

In the following example, the Conformal software does not support `FUNCTION func1` because it contains a clock expression on line 3 (in bold).

```
1.    FUNCTION func1 (in0, clk : IN STD_LOGIC) RETURN STD_LOGIC IS
2.    BEGIN
```

```
    IF clk'EVENT AND clk = '1' THEN
    …
```

Sequential Statements

Wait Statements

Caution

The Conformal software supports multiple `WAIT` statements. However, if you choose to use multiple `WAIT` statements, Cadence recommends verifying that the FSM encoding is what you expected.

Restriction 1

The Conformal software does not support `WAIT` statements used within subprograms.

Example 1

In this example, the Conformal software does not support the `WAIT` statement because it is used within a procedure. See line 3, in bold.

```
1. PROCEDURE pro1 (in0, clk : IN STD_LOGIC) IS
2. BEGIN
3.   WAIT UNTIL clk'EVENT AND clk = '1';
4.   …
```

Restriction 2

When a design uses a `WAIT` statement within one path of a process, all other paths of the same process must have at least one `WAIT` statement.

Example 2

In this example, the Conformal software does not support the `WAIT` statement inside process `proc1` because the `WAIT` statement is used in the `ELSE` branch (line 18), but not in the `IF` branch.

```
1.  ENTITY test IS
2.    PORT (
3.      clk  : IN BIT;
4.      x    : IN BIT;
5.      in0  : IN BIT_VECTOR(3 DOWNTO 0);
6.      out0 : OUT BIT_VECTOR(3 DOWNTO 0);
7.    );
8.  END test;
9.
10. ARCHITECTURE arch OF test IS
11. BEGIN
12.
13.   proc1 : PROCESS (clk,x)
14.     BEGIN
15.       IF (x='1') THEN
16.           out0 <= (others => '0');
17.        ELSE
18.            WAIT UNTIL clk'EVENT AND clk='1' ;
19.            out0 <= in0;
20.        END IF;
21.
22.     END PROCESS;
23. END arch;
```

Restriction 3

The Conformal software does not support the `WAIT FOR` statement.

Example 3

In this example, the Conformal software does not support the `WAIT FOR` statements in lines 4 and 6.

```
1.  clock_gen: PROCESS
2.  BEGIN
3.          iclk <='0';
4.          WAIT FOR clk_prd/2;
5.          iclk <='1';
6.          WAIT FOR clk_prd/2;
7.  END PROCESS clock_gen;
8.  clk <= iclk;
9.  …
```

Signal Assignment

The following Signal Assignment information applies to Sequential Statements *and* Concurrent Statements.

VHDL GUARDED Block Support

The Conformal software supports `GUARDED` Signal Assignments. A `GUARDED` signal is a signal for which several drivers exist. The synthesis interpretation and limitations are:

1. Latch devices will be synthesized.
2. No tri-state devices will be synthesized. For `BUS` or `REGISTER` signal types, the software issues the following warning message:

   ```
   RTL2.8: 'BUS' and 'REGISTER' signal type are not supported for synthesis.
   ```
3. For guarded assignment without guard signal, the Conformal software issues the following errors:

```
RTL2.9: Guarded assignment requires GUARD signal
RTL2.10: Guard is not declared
```

4. You can use the `multi_port portname` pragma to specify multi-port latches. In the following example, port `l1` is the `multi_port`:

```
library ieee;
use ieee.std_logic_1164.all;
entity test is
port (
  a, c, g,b_init, d, s_in, inv_c :in std_logic;
  l1_out, l2_out, s_out: out std_logic);
end test;

architecture arch of test is
  signal l1: std_logic register := '0';
  signal l2: std_logic register := '0';

begin
  -- pragma multi_port l1
  load : block(c = '1' and g = '1') begin
    l1 <= guarded d;
  end block;
  scan : block(a = '1') begin
    l1 <= guarded s_in xor inv_c;
  end block;
  shft : block(b_init = '1') begin
    l2 <= guarded l1;
  end block;

  l1_out <= l1;
  l2_out <= l2;
  s_out <= l2 xor inv_c;
end arch;
```

Example 1

In the following example, the Conformal software does not support the `GUARDED` signal. See line 2, in bold.

```
1.  bb:   BLOCK ( RISING_EDGE ( clock ) )
2.          z <= GUARDED x;
3.  END bb;
4.  …
```

Restriction 1

The Conformal software ignores delay mechanisms used in signal assignments; for example, `AFTER`, `TRANSPORT` and `INERTIAL`.

For example, the Conformal software ignores `AFTER`, `INERTIAL`, and `TRANSPORT`. See lines 1, 2, and 3.

```
1.  Output_pin1 <= Input_pin AFTER 10 ns;
2.  Output_pin2 <= INERTIAL Input_pin AFTER 30 ns;
3.  Output_pin3 <= TRANSPORT Input_pin AFTER 40 ns,  NOT Input_pin AFTER 70 ns;
4.  …
```

Procedure Calls

Restriction

The Conformal software does not support Procedure Calls when the procedure includes a `WAIT` construct or clock signal.

Example

this example, the Conformal software does not support `PROCEDURE proc1` because it contains a clock expression. See line 3, in bold.

```
1.  PROCEDURE proc1 (in0 : INOUT STD_LOGIC; clk : IN STD_LOGIC) IS
2.  BEGIN
3.    IF clk’EVENT AND clk = ’1’ THEN
4.  …
```

For Loops

Restriction

The Conformal software does not support `FOR-LOOP` when the loop index range is globally non-computable.

Example

In the following example, the Conformal software does not support the `FOR-LOOP` because the input `count` is not computable. See line 17, in bold.

```
1.  LIBRARY IEEE;
2.  USE IEEE.STD_LOGIC_1164.ALL;
3.
4.  ENTITY for_loop IS
5.   PORT (data     : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
6.    clk      : IN STD_LOGIC;
7.    count    : IN INTEGER RANGE 0 TO 5;
8.    data_out : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
9.    END for_loop;
10.
11. ARCHITECTURE rtl OF for_loop IS
12. BEGIN
13. PROCESS (clk, count)
14. VARIABLE data_temp : STD_LOGIC_VECTOR(3 DOWNTO 0);
15.  BEGIN
16.   IF (CLK'EVENT AND CLK = '1') THEN
17.     FOR i IN 0 TO count LOOP
18.        data_temp(i) := data(i);
19.     END LOOP;
20.   END IF;
21.   data_out <= data_temp;
22.  END PROCESS;
23. END rtl;
```

Concurrent Statements

Signal Assignment

See Signal Assignment on page 423.

B

Supported Non-SDC Commands/Options

Supported Non-SDC Commands

The following lists the non-SDC commands that the Conformal software supports:

```
add_to_collection
all_connected
all_fanin
all_fanout
append_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
get_attribute
get_designs
get_generated_clocks
get_object_name
index_collection
load_of
remove_from_collection
sdc_active_clocks
```

`sdc_disable_clock_gating_check`	Supported only by SDC Lint checks
`set_annotated_transition`	
`set_dont_touch`	
`set_dont_touch_network`	
`set_dont_use`	
`set_ideal_net`	
`set_lib_pin`	Supported only by SDC Lint checks
`set_scan_signal`	
`sizeof_collection`	
`sort_collection`	

Supported Non-SDC Command Options

The following lists the non-SDC command options that the Conformal software supports:

`add_to_collection -unique`

`all_fanin -flat`

`all_fanin -levels`

`all_fanin -only_cells`

`all_fanin -pin_levels`

`all_fanin -startpoints_only`

`all_fanin -step_into_hierarchy`

`all_fanin -to`

`all_fanin -trace_arcs`

`all_fanout -clock_tree`

`all_fanout -endpoints_only`

`all_fanout -flat`

all_fanout -from

all_fanout -levels

all_fanout -only_cells

all_fanout -pin_levels

all_fanout -step_into_hierarchy

all_fanout -trace_arcs

all_inputs -no_clocks

append_to_collection -unique

compare_collections -order_dependent

filter_collection -regexp

get_attribute -class (except for timing_path and timing_point)

get_attribute -quiet

get_cells -filter

get_cells -quiet

get_clocks -quiet

get_generated_clocks -nocase

get_generated_clocks -quiet

get_generated_clocks -regexp

get_lib_cells -quiet

get_lib_pins -quiet

get_libs -quiet

get_nets -filter

get_nets -quiet

get_pins -filter

```
get_pins -leaf

get_pins -quiet

get_ports -filter

get_ports -of_objects

get_ports -quiet

group_path -critical_range

set_annotated_transition -delta_only

set_annotated_transition -fall

set_annotated_transition -max

set_annotated_transition -min

set_annotated_transition -rise

set_dont_touch value

set_dont_use -power

set_dont_use value

set_load -fall

set_load -rise

set_max_capacitance -clock_path

set_max_capacitance -data_path

set_max_capacitance -fall

set_max_capacitance -rise

set_port_fanout_number -max

set_port_fanout_number -min

set_scan_signal -chain

set_scan_signal -hookup
```

```
set_scan_signal -port

set_scan_signal -sense

set_timing_derate -max

set_timing_derate -min

set_voltage -cell

set_voltage -dynamic

set_voltage -min_dynamic

set_voltage -pg_pin_name

sort_collection -descending
```