Conformal® Verify Extended Checks User Guide

Product Version 23.2
October 2023

Cadence Design Systems, Inc., 2655 Seely Avenue, San Jose, CA 95134, USA

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. (Cadence) contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

All other trademarks are the property of their respective holders.

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. The information contained herein is the proprietary and confidential information of Cadence or its licensors, and is supplied subject to, and may be used only by Cadence's customer in accordance with, a written agreement between Cadence and its customer. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Cadence is committed to using respectful language in our code and communications. We are also active in the removal and/or replacement of inappropriate language from existing content. This product documentation may however contain material that is no longer considered appropriate but still reflects long-standing industry terminology. Such content will be addressed at a time when the related software can be updated without end-user impact.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

Contents

4

5

B
VHDL Support . . . 307

C
System Verilog Support . . . 313

1

Introduction to Conformal Extended Checks

Overview

The Conformal Extended Checks software is a formal HDL verification tool that incorporates multiple proof engines and greatly reduces extensive dependency on simulations. It addresses the needs of design engineers, as well as verification engineers, in ensuring design quality. Conformal is designed for use with Verilog and VHDL RTL and gate-level designs.

Conformal Extended Checks supports standard library and design formats and integrates readily into existing design flows. Conformal is an independent verification tool (that is, it is not tied to any particular synthesis environment). Thus, it provides a superior degree of confidence.

Conformal Extended Checks incorporates many features that streamline and authenticate the design process. Some examples follow:

- Full-Chip Verification

  The Conformal processing speed significantly reduces verification time for high-capacity and highly complex designs.

- Standard Design Formats

  Conformal supports mixed Verilog and VHDL designs.

- Standard Library Formats

  Conformal supports Verilog simulation libraries as well as the Synopsys Liberty Format Libraries.

- Automatically Verifies Semantic and Structural Inconsistencies

  The extensive set of Conformal Extended Checks enables coverage of many common design errors. Conformal Extended Checks automatically extracts properties from the design without user guidance, and provides the most extensive results with minimal user effort.

- Multiple Formal Verification Proof Engines

  Conformal Extended Checks efficiently employs multiple state-of-the-art proof engines to ensure full coverage verification of automatic checks using reasonable resources. Test vectors are not required.

- Automatic Diagnosis and Integrated Debugging

  Conformal Extended Checks automatically generates counterexamples to aid in diagnosing property checking violations. Counterexample display and diagnosis takes place in an integrated Conformal Extended Checks debugging environment that includes a schematic viewer, source code browser, and waveform viewer.

- Simulation-Extracted Initialization Conditions

  Conformal Extended Checks reads in VCD format files generated by simulators. This enables you to employ event-based simulators to initialize the design under verification. Alternatively, you can specify manual initialization sequences for Conformal Extended Checks.

- Complements Conformal Equivalence Checking Technology

  The Conformal Equivalence Checking solution offers a complete formal solution for functional closure: from RTL to layout.

Complete Formal Solution

Conformal Extended Checks is an integral part of the complete formal solution offered by Cadence. Semantic inconsistencies cause most design re-spins today and are created by the difference in the semantic interpretation by RTL simulation tools and synthesis tools. Only full sequential analysis ensures that these problems are consistently caught. Conformal Extended Checks helps ensure that the RTL is golden by catching semantic and structural inconsistencies early in the design cycle. Then, the Conformal software ensures logical consistency to the gate level, and Conformal GXL ensures logical consistency to the final layout.

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`

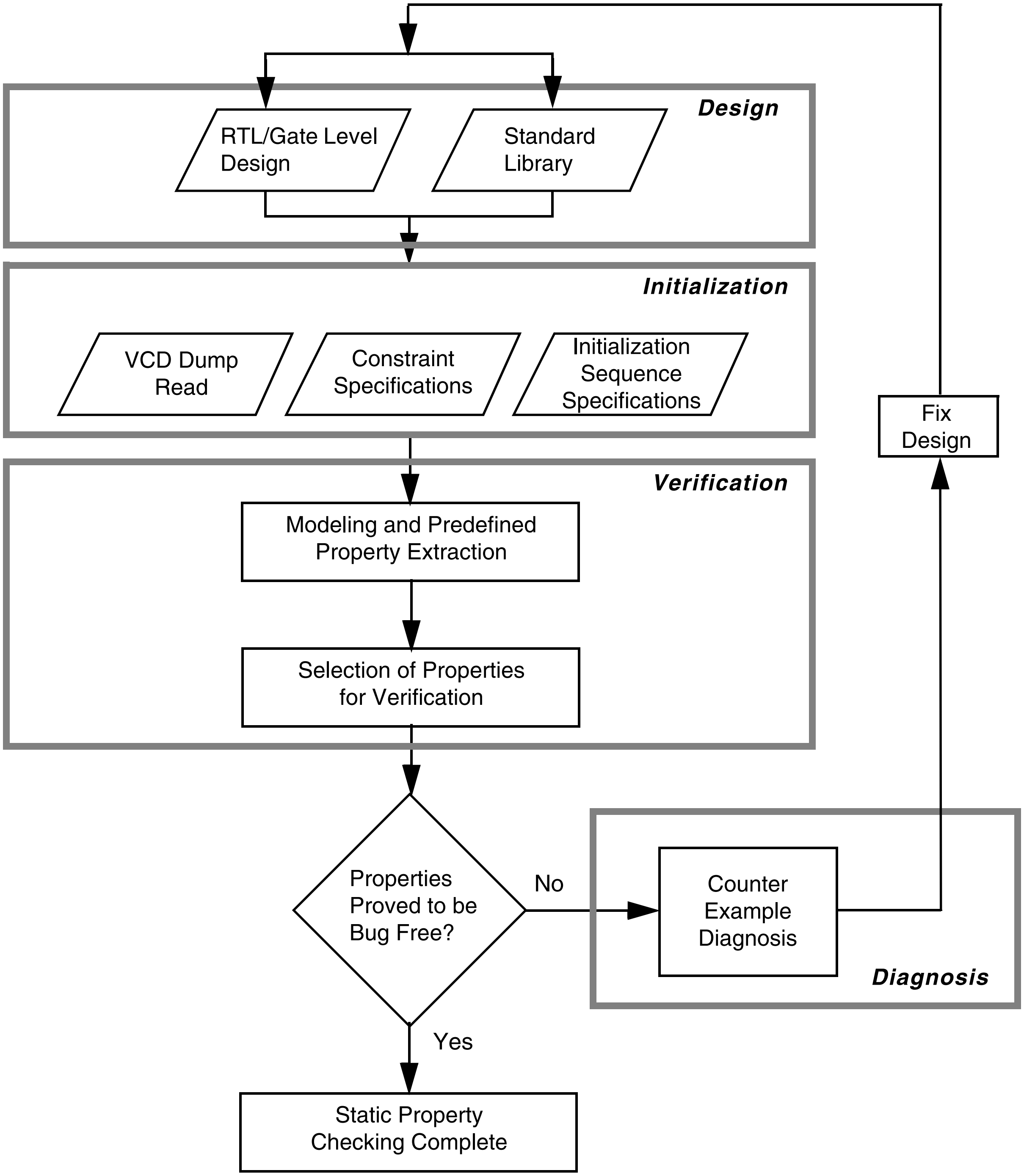
SystemVerilog	IEEE 1800-2005	`READ DESIGN -systemverilog`
		`READ LIBRARY -systemverilog`

Liberty	2007.3	`READ DESIGN -liberty`
		`READ LIBRARY -liberty`

CPF	1.0 and 1.0e	`READ CPF`

Extended Checks Methodology

The flow chart in the following figure illustrates the verification flow through a Conformal Extended Checks session.


Design
RTL/Gate Level Design
Standard Library
Initialization
VCD Dump Read
Constraint Specifications
Initialization Sequence Specifications
Fix Design
Verification
Modeling and Predefined Property Extraction
Selection of Properties for Verification
Properties Proved to be Bug Free?
No
Counter Example Diagnosis
Diagnosis
Yes
Static Property Checking Complete

Preparing the Design

During the first phase of the verification flow, Conformal reads a design that is specified in RTL, gate-level, or a combination of these two design styles. Additionally, Conformal reads the associated libraries.

Note: Conformal Extended Checks does not support strength specification for gate instances.

Initializing the Design

During the second phase of the verification flow, you constrain and initialize the design. Conformal Extended Checks supports two types of initialization:

- VCD file, which was generated during simulation of the initialization sequence for the storage elements
- Initialization sequence (usually a reset sequence) used to bring the storage elements to a desired state

Additionally, to avoid false negatives it is recommended that users provide:

- Clock waveform definition for all clocks in the design
- Design constraints for verification (for example, pin constraints and instance constraints for storage elements)

You can also specify the kind of assertion instances to use as design constraints.

Note: Clocks are defined in terms of Conformal Extended Checks time units. See Appendix E, “Defining Clocks” for additional information.

Verifying the Design

In the next phase of the verification flow, Conformal Extended Checks analyzes the design and extracts predefined properties. Then, you select properties for verification. (During this phase, Conformal Extended Checks detects any asynchronous clock domain crossings and structural feedback loops.)

Diagnosing Property Violations

During the final phase of the verification flow, Conformal Extended Checks aids in the diagnosis of property violations. You can diagnose a false property using a combination of the

Schematic Viewer, Source Code Viewer (to cross reference the problem), and Waveform Viewer. After you remedy the violation, a new verification session begins.

Extended Checks: Functional Checks

Conformal Extended Checks performs several types of Functional checks in a design. The checks fall into the following main categories:

- HDL (RTL) Rules
- Modeling Rules
- Predefined Property Checks
- Clock Domain Crossing Checks

HDL Rules are covered in the *Conformal HDL Rule Check Reference.* Modeling Rules are covered in the *Conformal Extended Checks Reference Manual*. Predefined Checks and Clock Domain Crossing Checks are introduced in the following sections.

Automatic Predefined Checks

Semantic inconsistencies frequently cause silicon re-spins because they produce different simulation results between RTL and synthesized gates. Conformal Extended Checks uses advanced formal algorithms to automatically catch these problems at the early RTL stage, thereby reducing the dependency on expensive gate level simulation.

Conformal Extended Checks also automatically detects another common source of design re-spins: structural inconsistency. It does this by automatically extracting design properties for formal verification. The following is a list of design properties (static properties) included in the predefined checks:

- Bus Contention and Bus Floating Properties:

  Conformal Extended Checks detects buses that are driven by multiple conflicting inputs, or are in a high-impedance state. Additionally, Conformal Extended Checks performs a sequential traversal to examine the bus control logic and determine whether a conflict, multiple-driven, or floating condition can occur in the design.

- Tristate Properties:

  Conformal Extended Checks examines the entire design for stuck-at-zero or stuck-at-one tristate enable conditions.

- Don't Care Properties:

  Conformal Extended Checks seeks out don't care conditions that might be exercised in the design, causing functional behavior differences between RTL and gate-level models. Conformal Extended Checks looks for:

  - ❑ `//synopsys full_case` logic that accesses unspecified cases, which can cause functional differences between RTL and gate level behavior
  - ❑ `//synopsys parallel_case` logic where overlapping cases might occur
  - ❑ Range-overflow conditions, where an array index exceeds the declared range
  - ❑ X-assignment, where a conditional assignment to a variable generates an "X" value

- Set-Reset Conflict Properties:

  Conformal Extended Checks examines the structure of the design, identifying each sequential element (flip-flop or latch) with set and reset controls. It then proves whether set and reset can ever be enabled simultaneously, leading to undesirable behavior.

- Multi-port Latch Properties:

  Conformal Extended Checks identifies all multi-port latches in a design and proves whether they can be loaded with conflicting values.

- No-Divide-By-Zero Properties

  Conformal Extended Checks ensures that divisors in a division operator are never zero.

- Encoding-Completeness Properties

  Ensures that any reachable state is listed under “from states” in the FSM encoding file.

See Chapter 8, “Predefined Checks” to learn more about this category of checks.

Clock Domain Crossing Checks

As companies integrate more functionality into each new SoC and ASIC design, different parts of chips may run at different clock rates. This level of complexity increases the verification challenge. To address these challenges, Conformal Extended Checks provides structural and Functional checks on clock domain crossings.

- Structural CDC Checks

  Conformal Extended Checks 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 Extended Checks supports the following synchronizers, which promote proper synchronization, to prevent meta-stability problems: D flip-flop synchronizer, Multiplexer synchronizer, and Module synchronizer.

- Functional CDC Checks

  The Functional CDC Checks verify that the data from a source clock domain remains stable until the destination clock domain captures it. A violation indicates instability, which results in ignored or lost data.

See Chapter 9, "Clock Domain Crossing Checks" to learn more about this category of checks.

Extended Checks Operation

This section provides an overview of Conformal Extended Checks operation. It includes the major features of a session and introduces the integrated debugging environment with a brief description of each of the integrated tools.

System Modes

Conformal Extended Checks has two system operating modes: *Setup* and *Verify*. A typical session may include multiple changes between these two modes as you make changes to the design or design constraints, then verify the results of these changes. Conformal Extended Checks signals the present mode with the Setup and Verify command entry prompts:

```
SETUP>
VERIFY>
```

Setup Mode

In the Setup mode, you can read in a design, specify various constraints for the design, and specify an initialization condition in terms of register values, an initialization sequence, or a VCD dump file of a previously simulated initialization.

Verify Mode

In the transition from the Setup mode to the Verify mode, Conformal Extended Checks performs predefined design checks and extracts property instances from the design. In Verify mode, you can select the desired static property instances, invoke the proof engine to prove them, diagnose the error candidates, and examine the counterexamples.

The Verify mode establishes an integrated diagnosis environment that provides convenient access to the proof and diagnosis processes.

Proof

Invoke the proof engine using the `PROVE` command in the Verify mode. Conformal Extended Checks attempts to prove all the property instances you selected and reports the results. Use the `REPORT PROVED DATA` command to learn more about the results of the proof. The Conformal Extended Checks proof engine reports the error candidates as false static property instances.

Diagnosis

Diagnosis is the process of generating counterexamples for the error candidates. It is an essential step in debugging the origin of false properties reported in the proof results. In the Verify mode, the `DIAGNOSE STATIC PROPERTY` command automatically generates counterexamples. You examine counterexamples in the Conformal Extended Checks integrated debugging environment, using various representations (for example, Waveform Viewer, Schematic Viewer, and Source Code Browser). This integrated debugging environment set helps you debug the design by visualizing the origin of the false static properties. Explore the problem with the integrated debugging tools and fix it by either revising the design or changing the constraints or assumptions of the design.

Error Candidates

After the Conformal Extended Checks proof engine completes the proof for the selected properties, the failing properties become error candidates. Conformal reports the error candidates in the transcript with a FAIL proof status and red circles in the Static Property Manager GUI window. (In the non-GUI mode, you can use the `report proved data -fail` command.) Diagnose the error candidates using the `DIAGNOSE STATIC PROPERTY` command, which helps locate the origin of the failure.

Counter-Examples

The counter-example of an error candidate shows an allowed state of the circuit that violates the property. Conformal Extended Checks generates a counter-example of the error candidate when you execute the `DIAGNOSE STATIC PROPERTY` command.

- For a *sequential property*, the counterexample is a consecutive sequence of allowed state transitions. The `DIAGNOSE STATIC PROPERTY` command generates the counter-example with valuation on the fan-in cone of the property.
- For a *sequential counter-example*, the valuation is generated up to the time frame at which the static property is violated.

The counter-example is a helpful guide for locating the cause of a FAIL status. Explore the counter-example using the Conformal Extended Checks integrated graphical debugging environment.

In some cases, you might want to disregard counter-examples with certain properties, such as counter-examples involving pins from black boxed modules, or counter-examples depending on don't care assignments or combination loop assignments. In these cases, run the `SET X-HANDLING` command.

Integrated Debugging Environment

Conformal Extended Checks features advanced diagnosis capabilities with source code cross referencing, waveform display, and gate level schematics.

- Source Code Browser—Displays the source code of the design. Depending on the context, it also locates the relevant source code segment responsible for the false static property. It helps you quickly locate the possible error in the source code.
- Waveform Browser—Displays the waveform of the counterexample for selected signals. It is especially useful for visualizing sequential counterexamples. This feature allows you to save the waveform of the counterexample in a standard waveform file format for later inspection.
- Schematic Viewing—Offers the capacity of a schematic viewing tool to assist the debugging process.

  The multi-timeframe schematic is helpful in debugging sequential circuits. It displays the fan-in cone of the selected gate or the diagnosed property. Additionally, Conformal Extended Checks annotates the counterexample generated by the `DIAGNOSE STATIC PROPERTY` command for convenient visualization. The multi-timeframe schematic offers a convenient interface to visualize the origin of the false static property instance without distraction from the irrelevant design segment.

Overview of Conformal Tcl Commands

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 Tcl Command Entry Mode Support chapter of the *Conformal Extended Checks 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.

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`

Specifying the Command Entry Mode

In Conformal, there are two modes: the default Conformal command entry mode (VPXMODE) and the Tcl command entry mode (TCLMODE). Use the `TCLMODE` command to switch Conformal to the Tcl command entry mode.

To change to Tcl command entry mode, run the following command:

```
tclmode
```

To return to the default Conformal command entry mode, run the following command:

```
vpxmode
```

Using Native Conformal Commands

When Conformal is in Tcl command entry mode, typically you will run native Tcl and Conformal Tcl commands. However, you can also run native Conformal commands as shown in the examples below.

To run native Conformal commands:

- Example one: Preface the native Conformal command with the `vpx` keyword.

  ```
  vpx read design counter.v
  ```

  Partial entry matching is allowed:

  ```
  vpx rea de counter.v
  ```

- Example two: 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 in Tcl command entry mode unless you preface the command with the `vpx` keyword (as shown in Example one, above).

  ```
  read_design counter.v
  ```

To get quick help for native Conformal command names:

To Get Quick Help for Native Conformal Command Names

If you type a native Conformal command incorrectly using the underscore method in Example two (above), Conformal echos 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
```

Duplicate Commands

The following native Tcl commands are also defined as native Conformal commands:

```
break
continue
exit
```

Important

The behaviors of these commands are not the same in Tcl command entry mode as they are in Conformal command entry mode. Use these commands with caution.

Refer to the *Conformal Extended Checks Reference Manual* for detailed descriptions of the native Conformal commands.

2

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 Encounter® Conformal® software.
- At the top level of your installation hierarchy.

Displaying the Version Number

The `VERSION` command displays the Conformal software version number. This is useful when starting a transcript log file to ensure that the file contains a reference to the Conformal version that created the results.

Viewing the License Status

The `LICENSE` command displays the current license status. The current status of the Conformal software license appears in the transcript output.

Starting the Conformal Extended Checks Software

You can start the Conformal software in either the graphical GUI or non-graphical (non-GUI) mode. The GUI mode is the default when you enter `verify` at a UNIX system prompt:

```
UNIX% lec -verify
```

To start a session in the non-GUI mode, enter the following at the UNIX prompt:

```
UNIX% lec -verify -nogui
```

To start a session in 64-bit mode, enter the following at the UNIX prompt:

```
UNIX% lec -verify -64
```

After startup, you can toggle between GUI and non-GUI modes using the `SET GUI` command.

Tip

To determine your current version, use `lec -verify -version` at the UNIX system mode prompt.

In the non-GUI mode, you can start the Conformal software with the `-color` option to specify that you want all of the messages to be color-coded. (For example, error messages appear in red text.) By default, color-coding is off in non-GUI mode.

Initial Command Files

When you start the Conformal software, it searches for and executes initial command files (`.verify.rc`). The software checks for the `VERIFY_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 `VERIFY_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 `.verify.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 the initialization file to run a complete batch file. Use dofiles, as explained in the following section, for this purpose.

Dofile Command Files

The Conformal Extended Checks 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.

  For more information, see Comments in Dofiles on page 35.
- 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% lec -verify -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`:

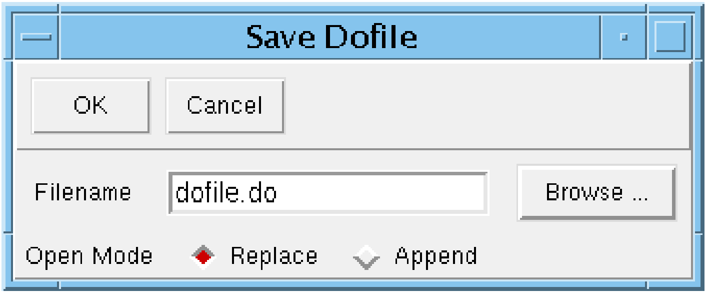
```
    UNIX% lec -verify -nogui -dofile my_dofile
```

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 Extended Checks session.



Filename dofile.do

Open Mode ● Replace ○ Append

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.

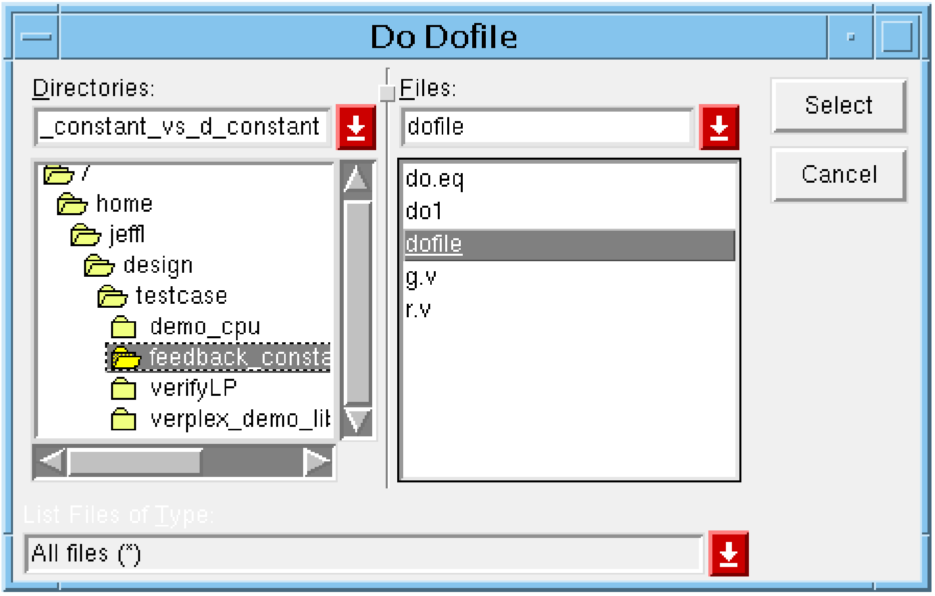
Open Mode

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.


Directories: _constant_vs_d_constant

Files: dofile

List Files of Type: All files (*)

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 \
   ```

-libertyCheckpoint and Restart Facility

The checkpoint and restart facility saves all the data from a session (`SAVE SESSION -checkpoint command`) as a *checkpoint* such that it can be restored 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 text 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).

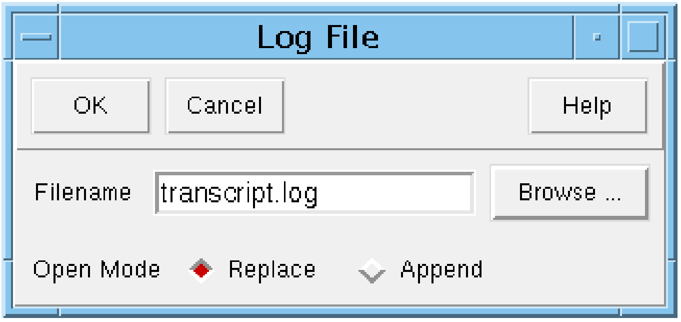
Transcript Messages

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 Extended Checks retains the transcript in a file. To save the transcript output in a log file, see Recording Transcript Log Files on page 39.

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*).



Filename transcript.log

Open Mode ● Replace ○ Append

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 39 to capture a transcript of the beginning of the session.

Log File Form Fields and Options

Filename

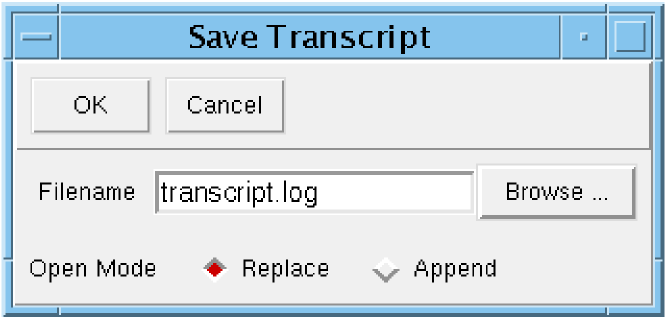
Specifies a transcript name. Type a path, or click *Browse* to choose an existing file from the Log File browser window.

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.



Filename transcript.log

Open Mode ● Replace ○ Append

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:

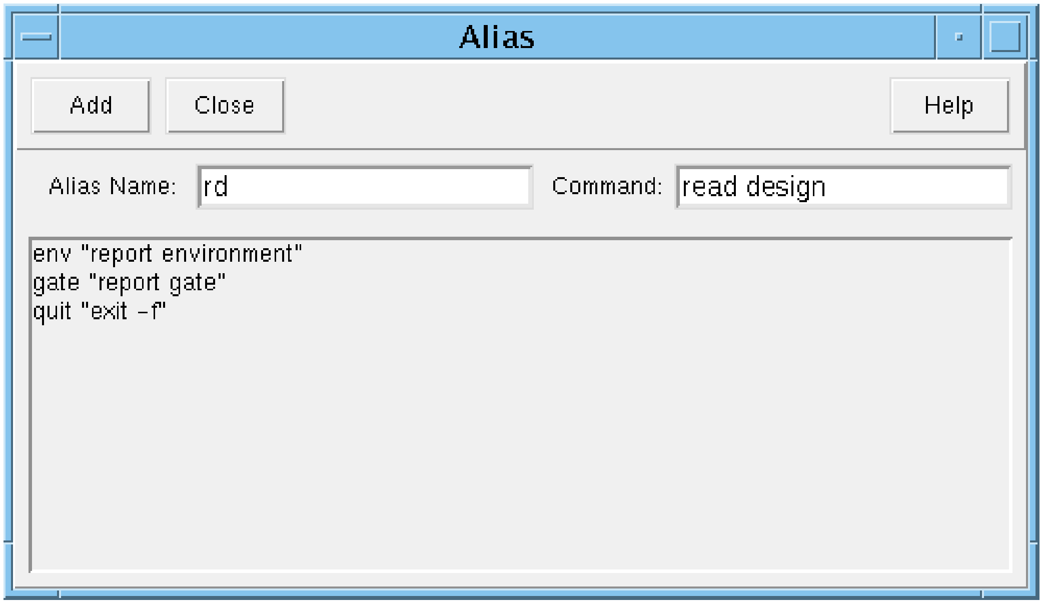
Example command:

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



Alias Name: rd

Command: read design

```
env "report environment"
gate "report gate"
quit "exit -f"
```

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*.

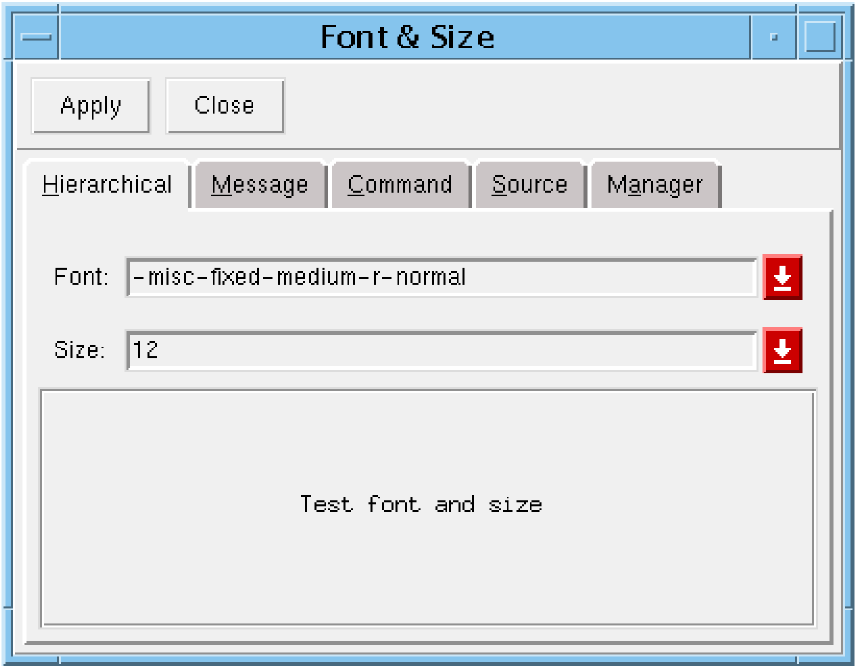
Setting Preferences

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*.



Font: -misc-fixed-medium-r-normal

Size: 12

Test font and size

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*.

Module Browser On

Displays or hides the Hierarchical Browser in the main window.

Icon Bar

Displays or hides the Icon Bar in the main window.

Pass/Fail Icon Style

Use the *Preferences – Pass/Fail Icon Style* pull-down menu to specify the icon style for pass/fail indicators that are displayed in the Manager windows.

By default, the Conformal software displays a *Color Circle* (red circle to denote fail or non-equivalent, and a green circle to denote pass or equivalent). Alternatively, you can change this preference to a *Check Mark* (red X or green check) for these indicators.

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.

Accessing Online Help and Documentation

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
  ```

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.

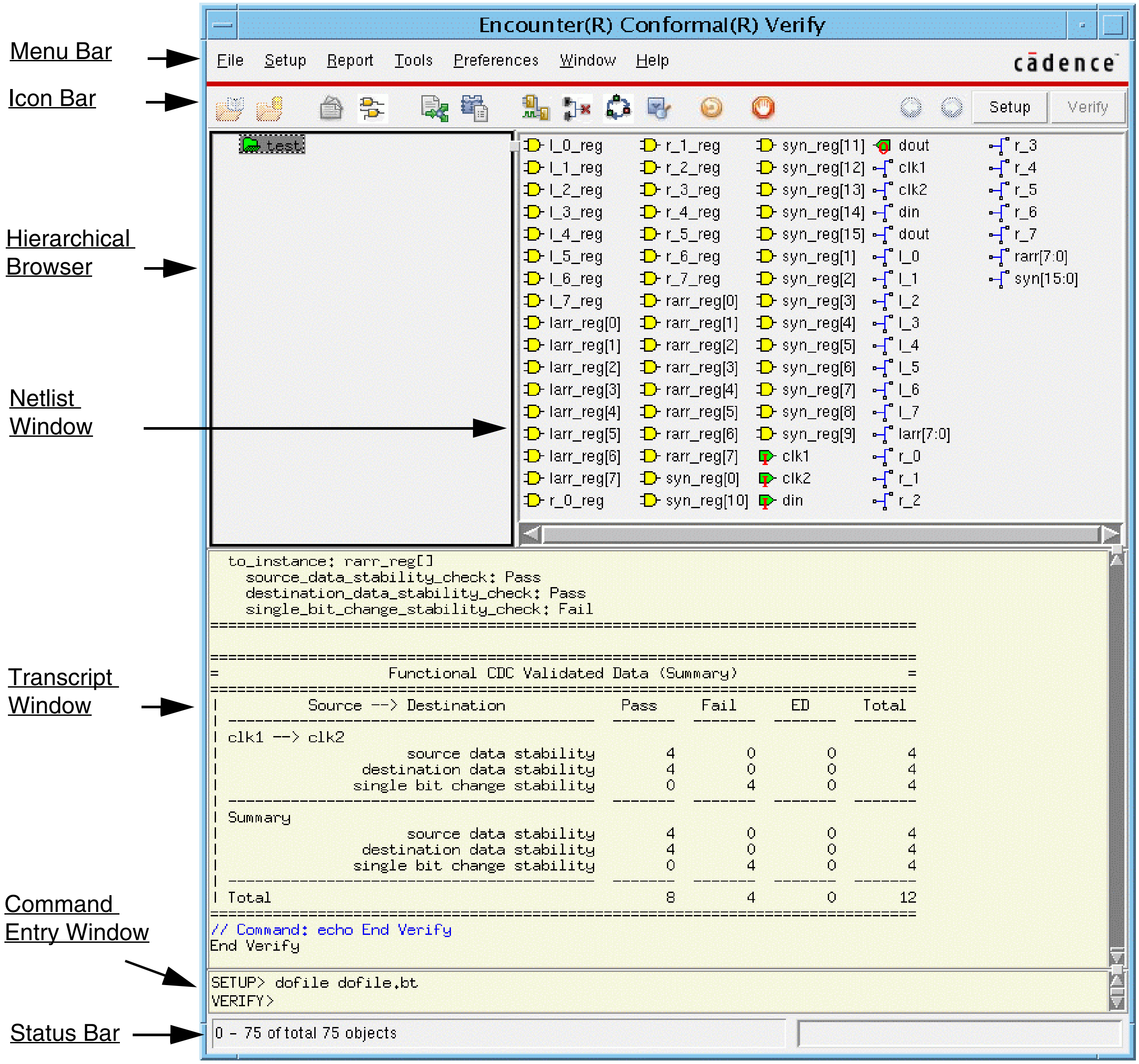
You can also use the `LICENSE` command to review the current license status. The current status of the license appears in the transcript output.

3

Using the Graphical User Interface

Main Window

This section describes some of the basic features of the main window.



```
  to_instance: rarr_reg[]
    source_data_stability_check: Pass
    destination_data_stability_check: Pass
    single_bit_change_stability_check: Fail
===============================================================================

===============================================================================
=                   Functional CDC Validated Data (Summary)                   =
===============================================================================
|          Source --> Destination            Pass     Fail     ED      Total
| ----------------------------------------  -------  -------  -------  -------
| clk1 --> clk2
|                    source data stability        4        0        0        4
|               destination data stability        4        0        0        4
|              single bit change stability        0        4        0        4
| ----------------------------------------  -------  -------  -------  -------
| Summary
|                    source data stability        4        0        0        4
|               destination data stability        4        0        0        4
|              single bit change stability        0        4        0        4
| ----------------------------------------  -------  -------  -------  -------
| Total                                           8        4        0       12
===============================================================================
// Command: echo End Verify
End Verify
```

```
SETUP> dofile dofile.bt
VERIFY>
```

0 - 75 of total 75 objects

Selecting Multiple Items

From the various windows, you can 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

You can use 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 name of the selected object is displayed in an ivory text box. As you move the box to another window, the background of the text box changes to black if you have reached a window where you can drop the object.

Menu Bar

The menu bar represents categories of commands. Each of the headings supports a pull-down menu of related commands. Click a menu bar category 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 on an enabled command to run it.

The drop-down menus support meta-key invocation for menu commands using mnemonics. The mnemonic for each command name is shown with an underscore. For example, run 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.

Window Menu

Note: The following information also applies to the Manager windows.

The *Window* drop-down menu is a dynamic menu that changes as you open and close windows. All active windows are listed in the *Window* drop-down menu. Clicking on a window name brings it to the front of your screen.

Use the *Window – Cascade* menu command to refresh your desktop and display the main window on top with all other open windows in a cascading view to the left of the main window.

Icon Bar

The main Conformal Extended Checks 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 not highlighted, 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 not highlighted, since it is available in the Verify operating mode only.

The following table lists the various icons and their descriptions.

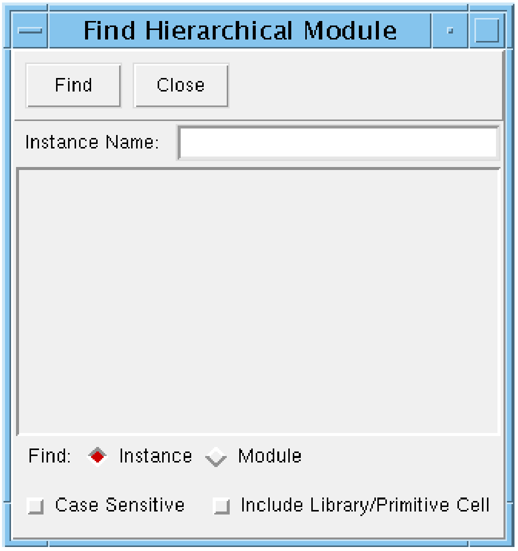
Icon	Icon Name	Description
📂	*Read Library*	Opens the Read Library window.
📂	*Read Design*	Opens the Read Design window.
📄	*Source Code Manager*	Opens the Source Code Manager.
⬇	*Flattened Schematics*	Opens the Flattened Schematics window.
RULE ✓	*HDL Rule Manager*	Opens the HDL Rule Manager.

Icon	Icon Name	Description
	Modeling Rule	Opens the Modeling Rule Manager.
	Power Rule Manager	Opens the Power Rule Manager.
	Clock Domain Crossing	Opens the Clock Domain Crossing Manager. For more information, see Chapter 9, "Clock Domain Crossing Checks."
	Static Property Manager	Opens the Static Property Manager. For more information, see Static Property Manager on page 150.
	FSM Manager	Opens the FSM Manager. For more information, see FSM Manager on page 164.
	Find	Opens the Find Hierarchical Module window. Refer to "Find Hierarchical Module" on page 50.
	Refresh Hierarchy	Refreshes the main window display. Module expansions are minimized and the netlist window is cleared.
	Stop	Interrupts processing.
	Previous Page	Switches Netlist window 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 window 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.
cādence™	*About*	Opens the Company/Product Information window.

Find Hierarchical Module

Use the Find Hierarchical Module form to locate an instance in the Hierarchical Browser. You can open this form by clicking the *Find* icon located on the menu bar, or pressing `Ctrl-f` in the Hierarchical Browser.



Find: ● Instance ○ Module

□ Case Sensitive □ Include Library/Primitive Cell

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, located in the main window, displays the hierarchical modules of the Golden and Revised designs. The root module is displayed along with its hierarchical contents. Clicking the *+* and - icons expand and compress the hierarchical display. Click the *Refresh* icon, located on the icon bar near the top of the main window, compresses all hierarchical modules back to the root module. The module name is displayed first, and instance names are enclosed in parentheses ( ).

To display or hide the Hierarchical Browser in the main window, choose the *Preferences – Hierarchical Browser On* check box.

Running Commands on Selected Modules

Run certain commands when you select a module or instance in the hierarchical display. You cannot run 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.

Running Commands on the Root Module

In the Setup system mode, you can execute certain commands from within the Hierarchical Browser window. (Commands cannot be executed on selected library cells.) The following table of commands relates to the root module.

Command	Description
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.

Running Commands on a Module or Instance Other Than Root

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

Command	Description
Root Module	Executes the Set Root Module command.
Add Black Box	Executes the Black Box command for the selected module and all its instances.
Tied Signals	Opens the Tied Signals form.
Source Code	Opens the Source Code Manager and highlights the selected location.
Schematics	Opens the schematic view of the selected module.

Hierarchy Browser Window Icons

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

Module

🚩 Assertion Monitor

Assertion Constraint

Netlist Window

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

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

Module

	Circuit Element
I	Input Pin
O	Output Pin
I/O	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 Extended Checks 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.

Clearing 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 window. Use it to execute commands from the keyboard as an alternative to using the menus and icons.

Commands you execute using the menus or icons are transcribed to the Command Entry window. If you use the Save Dofile feature, Conformal writes all of the commands that are listed in the Command Entry window to the file.

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 window. It shows the status of certain processing commands. 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 GUI mode to the non-GUI command line mode, choose *File – Exit GUI* from the main window.

To return to GUI mode, use the `SET GUI ON` command.

Exiting the Conformal Software

To exit completely, use the `EXIT` command, or choose *File - Exit* from the main window.

Note: All design and diagnosis information is lost when you terminate the session.

Saving GUI Settings

Choosing *File - Exit* opens a confirmation window. By default, the Conformal software does not automatically save GUI settings for future sessions. To save your preferred settings, 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

File Menu

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

- *Read Library*—Specify library filenames you will include with a design.

  See Read Library Form on page 99.

- *Read Design*—Specify the filename the Conformal software reads in as the design.

  See Read Design Form on page 103.

- *Save Dofile*—Save commands to a dofile to be used later as a batch file to repeat the Conformal Extended Checks session.

  See Saving a Dofile on page 37.

- *Do Dofile*—Execute a batch file of commands, or a Dofile set of commands from a previous session.

  See Executing Commands in a File on page 38.

- *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.

  See Transcript Messages on page 41.

- *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.

  See Creating a Transcript File on page 42.

- *Alias*—Create, view, or remove an alias.

  See Alias Form on page 44.

- *Search Path*—Create, modify, or delete directory search paths.

  See Adding Search Paths on page 92.

- *Environment*—Set global options for the design.

  See Setting Global Options on page 95.

- *Clock*—Add and delete waveforms for clock input signals using a graphical waveform display.

  See Defining Clocks on page 108.

- *Clock Domain Rule*—Specifies clock domain assignment rules.

  You can also open this form from the Clock Domain Crossing Manager's *Setup* menu. See "Clock Domain Rule" on page 224 of Chapter 9, "Clock Domain Crossing Checks" for information related to this feature.

- *Pin Constraints*—Add and delete pin constraints to primary input pins.

  See Pin Constraints on page 112.

- *Pin Equivalences*—Add and delete pin equivalences.

  See Pin Equivalences on page 115.

- *Tied Signals*—Add and delete tied signals to floating nets and pins.

  See Tied Signals on page 117.

- *Root Module*—Specify a new root module.

  See Changing the Root Module on page 92.

- *Renaming Rule*—Add, delete, and test renaming rules.

  See Renaming Rules on page 127.

- *Notranslate Modules*—Add and delete design or library modules that will not be translated.

  See Adding Notranslate Modules on page 94.

- *Ignore Reset Constraint*—Ignore reset constraints.

  See Ignore Reset Constraints on page 119.

- *Initial State*—Specify an initialization sequence for a circuit through a VCD dump file or an initial sequence file, and add individual initial states.

  See Applying an Initialization Sequence on page 120.

- *Assertion Constraint*—Turn assertion library instances into proof assumptions or proof obligations.

  See Assertion Constraints on page 123.

Using the Report Menu

Use the Report form to display extensive design information in the Transcript window of the main window. For information on saving the reports to a file, see Transcript Messages on page 41.

For a description of the categories that you can select from the *Report* menu and the Report form, see Running Reports on page 137.

Using the Tools Menu

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

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

  See HDL Rule Manager on page 72.
- *Modeling Rule*

  See Modeling Rule Manager on page 75.
- *Power Rule Manager*

  See the *Conformal Low Power User Guide*.
- *Clock Domain Crossing*

  See Clock Domain Crossing Manager on page 218
- *Static Property Manager*

  See Static Property Manager on page 150.
- *FSM Manager*

  See FSM Manager on page 164.
- *Module Schematics*
- *Clock Tree Schematics*

4

Managing Rule Checks

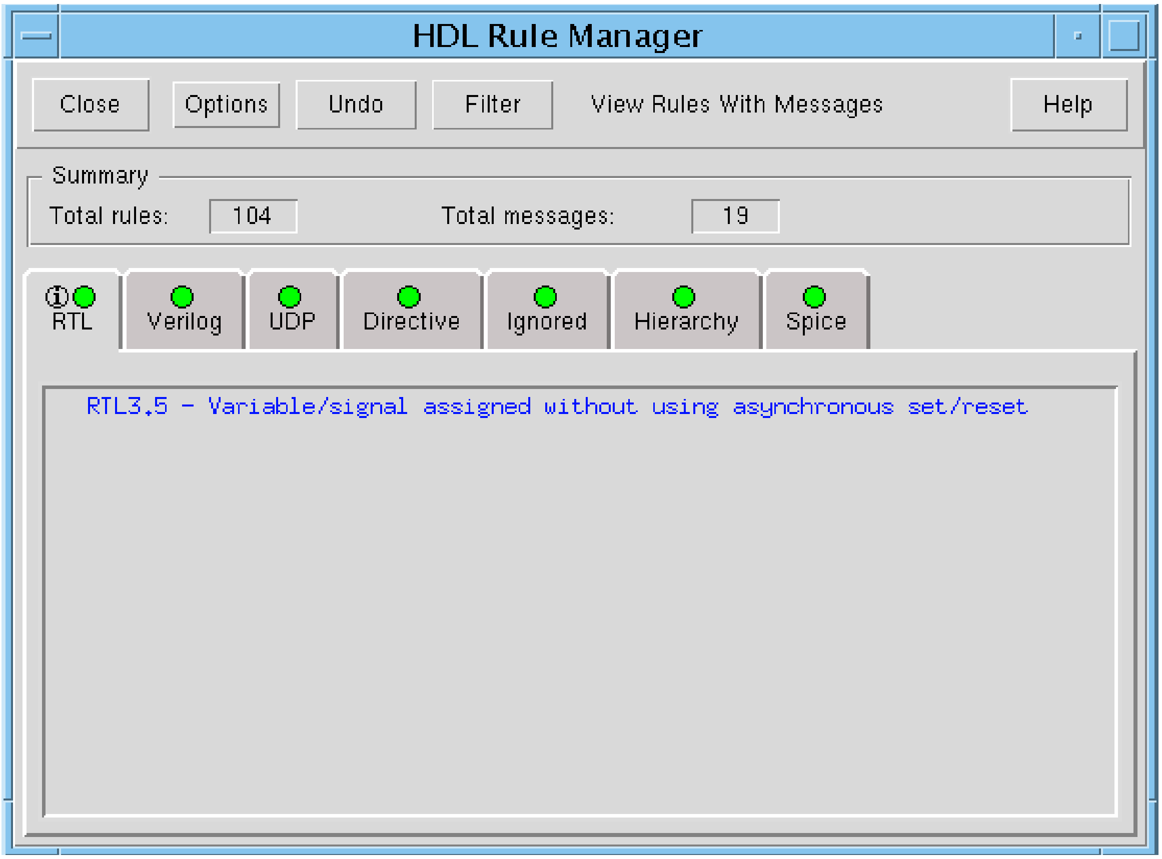
HDL Rule Manager

HDL rules consist of a group of desirable 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.

You can view all the HDL rule check messages and their details in the *Reference Guide*, or type '`help <rule>`' at the command line.

You can use the HDL Rule Manager to manipulate the HDL Rule Checks that are done when reading in libraries and designs. There are two ways to open the HDL Rule Manager from the Main window:

- ➤ Choose *Tools – HDL Rule*.
- ➤ Click the *HDL Rule Manager* toolbar widget.



Total rules: 104

Total messages: 19

RTL3.5 - Variable/signal assigned without using asynchronous set/reset

For the HDL Rule Manager, see the following for more information:

- HDL Rule Manager Fields and Options on page 63
- Diagnosing HDL Rule Violations on page 64
- Status Icons on page 69
- Changing the Severity of Rule Checks on page 69
- Reporting Messages for Rule Checks on page 71.
- Marking Rule Occurrences as Waived on page 71
- Viewing a Specific Message on page 72
- Filtering Rules on page 73

The HDL Rule Manager includes the following tabs, corresponding to the rule checking categories, and a display area.

- *RTL*—For designs that are written in the register transfer level of abstraction.
- *Verilog*—For designs that are written in Verilog.
- *UDP*—For designs that contain user-defined primitives.
- *Directive*—For designs that include directives or pragmas.
- *Ignored*—For designs that include unsupported or redundant constructs, which are ignored by the checker.
- *Hierarchy*—For designs that contain hierarchical components.
- *Spice*—For designs that contain SPICE netlists.

HDL Rule Manager Fields and Options

Options	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 to open the Page Size form to specify the page limits to control the number of rules that are displayed. The default is 25.

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 *Options – Hidden Rules* 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 73.

Diagnosing HDL Rule Violations

You can use the integrated debugging environment, specifically the Source Code Manager, to investigate the cause of HDL Rule violations:

1. Locate a highlighted rule check and click the *+* symbol to expand the entry.
2. Click to select an individual occurrence.
3. Right-click and choose one of the following:
    - *Waive Occurrence*—Marks the specified occurrences as waived for that particular entry. This runs the `ADD RULE WAIVER` command, which causes the selected occurrence to be hidden.

      For more information, see Marking Rule Occurrences as Waived on page 71.
    - *Source Code*—Opens the source code browser for the entry.

Modeling Rule 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.

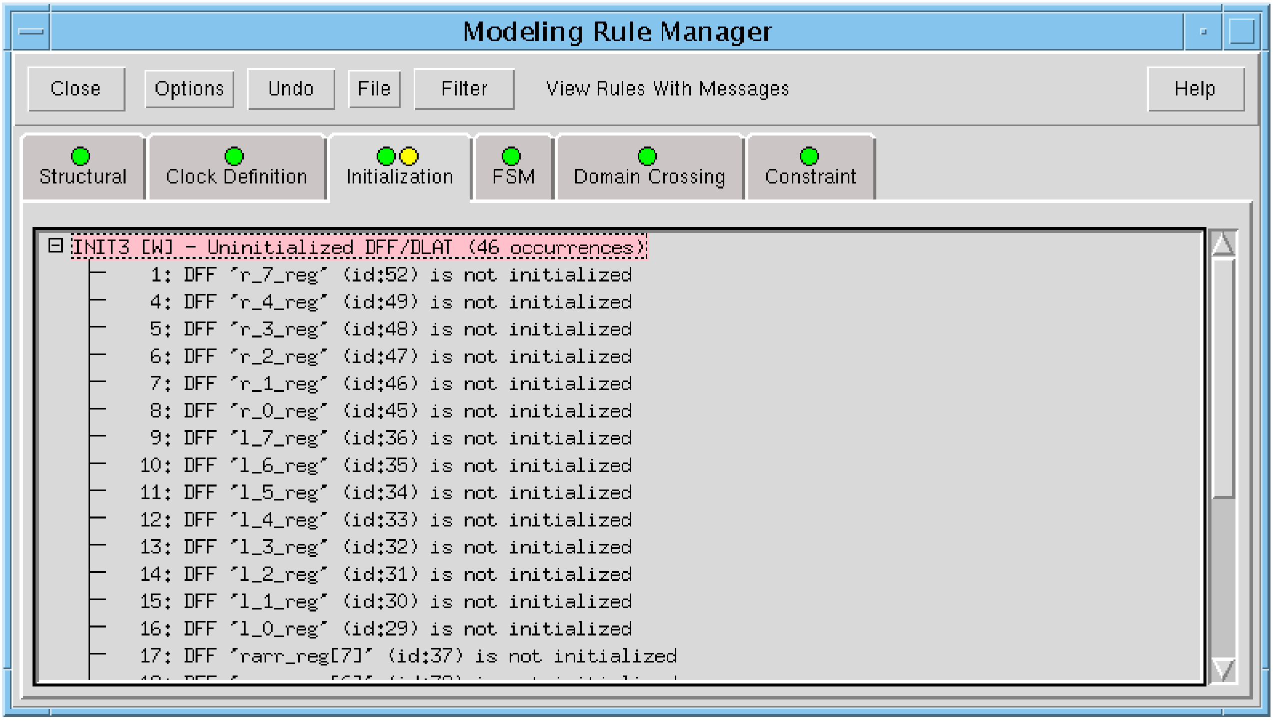
Modeling messages indicate any modeling warnings encountered during the analysis and modeling of the design.

Use the Modeling Rule Manager to to manipulate the modeling rule checks, which are active as the system mode changes from Setup to Verify. In the Setup mode, use this window to change the severity level of rule violations. In the Verify mode, you can view checks performed during design elaboration.

There are two ways to open the Modeling Rule Manager from the Main window:

➤ Choose *Tools – Modeling Rule*.

➤ Click the *Modeling Rule Manager* toolbar widget.



```
⊟ INIT3 [W] - Uninitialized DFF/DLAT (46 occurrences)
    ├─  1: DFF 'r_7_reg' (id:52) is not initialized
    ├─  4: DFF 'r_4_reg' (id:49) is not initialized
    ├─  5: DFF 'r_3_reg' (id:48) is not initialized
    ├─  6: DFF 'r_2_reg' (id:47) is not initialized
    ├─  7: DFF 'r_1_reg' (id:46) is not initialized
    ├─  8: DFF 'r_0_reg' (id:45) is not initialized
    ├─  9: DFF 'l_7_reg' (id:36) is not initialized
    ├─ 10: DFF 'l_6_reg' (id:35) is not initialized
    ├─ 11: DFF 'l_5_reg' (id:34) is not initialized
    ├─ 12: DFF 'l_4_reg' (id:33) is not initialized
    ├─ 13: DFF 'l_3_reg' (id:32) is not initialized
    ├─ 14: DFF 'l_2_reg' (id:31) is not initialized
    ├─ 15: DFF 'l_1_reg' (id:30) is not initialized
    ├─ 16: DFF 'l_0_reg' (id:29) is not initialized
    ├─ 17: DFF 'rarr_reg[7]' (id:37) is not initialized
```

For the Modeling Rule Manager, see the following for more information:

- Diagnosing Modeling Rule Violations on page 67
- Writing Rule Violations to a File on page 68
- Status Icons on page 69
- Changing the Severity of Rule Checks on page 69
- Reporting Messages for Rule Checks on page 71.
- Marking Rule Occurrences as Waived on page 71
- Viewing a Specific Message on page 72
- Filtering Rules on page 73

The Modeling Rule Manager includes the following tabs, corresponding to the rule checking categories, and a display area.

- *Structural*—Checks for structural errors.
- *Clock Definition*—In a design with ill-defined clocks, Conformal Extended Checks detects when sequential parts of the circuit might not trigger or might trigger improperly.
- *Initialization*—Checks for conditions that pertain to initializing a design.

  Note: The `INIT2` rule check has special handling instructions. `INIT2` is usually caused by a combination of conflicting initialization commands and/or constraints. Thus, during diagnosis with the GUI, the `INIT2` schematic contains all the gates directly involved in the warning message. The schematic also displays the fan-ins of those gates (two to three levels deep).
- *FSM*—Checks standard Finite State Machines (FSMs) to ensure they are coded correctly and efficiently.
- *Domain Crossing*—Checks clock domain assignments for missing or conflicting information.
- *Constraint*—Checks for over-constrained errors.

Modeling Rule Manager Fields and Options

Options

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 to open the Page Size form to specify the page limits to control the number of rules that are displayed. The default is 25.

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 *Options – Hidden Rules* and select *Unwaive Occurrence*.

File

Opens the Write Rule Check form where you write the low power violations into a file. For more information, see Writing Rule Violations to a File on page 68.

Filter

Opens the Filter Rule form where you can add or delete rule filters. For more information, see Filtering Rules on page 73.

Diagnosing Modeling Rule Violations

You can use the integrated debugging environment, specifically the Source Code Manager and Flattened Schematics, to investigate the cause of Modeling Rule violations:

1. Locate a highlighted rule check and click the *+* symbol to expand the entry.
2. Click to select an individual occurrence.
3. Right-click and choose one of the following:
   - *Waive Occurrence*—Marks the specified occurrences as waived for that particular entry. This runs the `ADD RULE WAIVER` command, which causes the selected occurrence to be hidden.

     For more information, see Marking Rule Occurrences as Waived on page 71.

- *Source Code*—Opens the source code browser for the entry.
- *Report Gate*—Returns a gate-list summary for the entry.
- *Schematics*—Opens a schematic viewer for the entry.

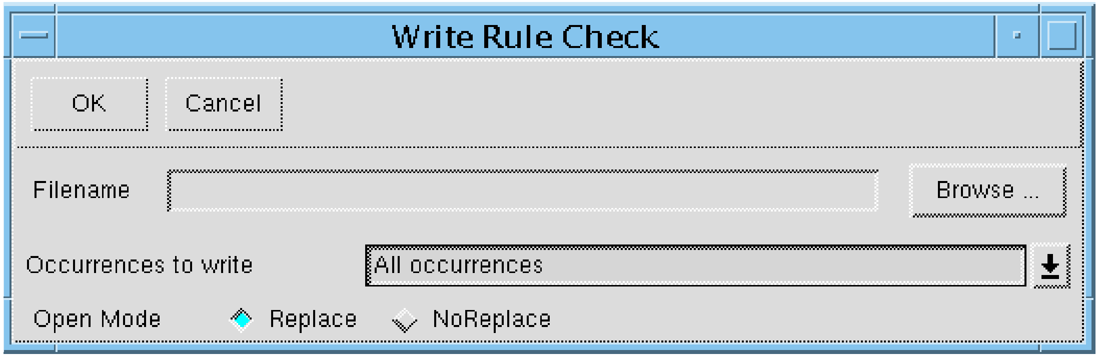
Note: Each rule also has specific commands that help you diagnose the problem (for example, executing a `REPORT PATH` command).

Writing Rule Violations to a File

To write the rule violations into a rule file, use the following procedure:

1. Click *File – Write Rule Check*.

   This opens the Write Rule Check form.



Filename

Occurrences to write: All occurrences

Open Mode ● Replace ○ NoReplace

Write Rule Check Form Fields and Options

Filename

Type the file of the rule file, or click *Browse* to choose a file from the Write Rule Check browser window.

Occurrence Type

Complete (the default) reports all occurrences regardless whether they are hidden or not. *Hidden* reports only occurrences that are hidden. *Waived* reports only occurrences that are marked as waived.

Open Mode

Replace (the default) overwrites the contents of an existing file. *Append* appends the contents to an existing file.

Status Icons

In the Rule Managers, each of the tabs displays a category name and an icon that indicates one of the following conditions for the category of rules:

Icons in Setup Mode			
❓			Indicates that the checks have not yet been executed.
ⓘ			Indicates that the checks are in a severity level of Ignore.
Icons in Verify Mode			
✓	or	🟢	Indicates that the rules in this category have passed.
✖	or	🔴	Indicates that the design triggered one or more rule failures in the applicable category.
⚠	or	🟡	Indicates that the checks are in a severity level of Warning.

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.

Changing the Severity of Rule 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 check hrc7 -help
```

The output shows the rule name, default severity, and description:

```
HRC7  WARN  Module specified by ‘add notranslate modules’ command cannot be found
```

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 -setting
```

```
===============================================================================
=                                RTL Rules                                     =
===============================================================================
HRC7: Module specified by ‘add notranslate modules’ command cannot be found
    Type: Design        Severity: Warning
    Type: Library       Severity: Warning
```

To change the default severity level to an error, you would run the following command (in Setup mode):

```
    set rule handling HRC7 -error
```

To show the new severity level for `HRC7`, you would run the following command:

```
    report rule check HRC7 -setting
```

```
===============================================================================
=                                RTL Rules                                     =
===============================================================================
HRC7: Module specified by ‘add notranslate modules’ command cannot be found
    Type: Design        Severity: Error
    Type: Library       Severity: Error
```

Note: However, if after changing `HRC7` rule’s severity to an error, you run the `report rule check HRC7 -help` command, you will still get the (default) severity of `Warning`.

Changing Severity in the Rule Managers

To change the severity of the rule handling, use the following procedure in Setup mode and *before* you read in the library and design files:

1. Click *Option – View* and select *All* from the pop-up menu.
2. Click to select a rule check number.
3. Right-click and choose *Severity* to open the pop-up menu and select *Warning*, *Error*, *Note*, or *Ignore*.

Note: Conformal does not report rules with a severity of *Ignore* as violations.

Enabling and Disabling Rule Checks

Use the `SET RULE HANDLING` command to exclude specified entities (for example, a specified module) from rule checking.

Use the `ADD IGNORE RTLCHECK` command to ignore HDL (RTL) rule checking for all or specified modules. By default, rule checking is enabled. Thus, you will only use the `DELETE IGNORE RTLCHECK` command to reverse the effects of the `ADD IGNORE RTLCHECK` command.

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 Messages for Rule Checks

Use the `REPORT RULE CHECK` command to view a summary or verbose report of messages. Report information displays in the transcript section of the main window.

Note: Conformal does not report rules with a severity of *Ignore* except with the `REPORT RULE CHECK` command. (Use the `rule_name` or `-ignore` option.)

Alternatively, you can use the Rule Manager to report individual rules and violations. To view a report for a particular rule check message, use the following procedure:

1. Click to select a rule check number.
2. Right-click and choose *Report* and one of the following from the pop-up menu:
   - ❑ *Summary*— Displays total number of occurrences for the selected rule check.
   - ❑ *Verbose*— Displays the rule check message, the total number of occurrences, and the severity level for the selected rule check.

Marking Rule Occurrences as Waived

To mark the specified occurrences as waived, you can use the `ADD RULE WAIVER` command or the Rule Manager:

1. From a Rule Manager, expand a rule to view the list of occurrences.

2. Right-click on an occurrence and choose *Waive Occurrence* from the pop-up menu.

Note: This mark is lost when the rule is re-checked.

After marking an occurrence as waived, you can use the `DELETE RULE WAIVER` command to unmark the waived occurrence. From the Rule Manager, or you can by selec the *Undo* button. This will only undo the last waiver mark. To undo previous waiver marks, select *Options – Hidden Rules* and select *Unwaive Occurrence*.

Tip

You can report waived rule occurrences with the `REPORT RULE CHECK -waived` command.

Viewing a Specific Message

To view the verbose listing of a specific rule messages in the Rule Managers, do the following:

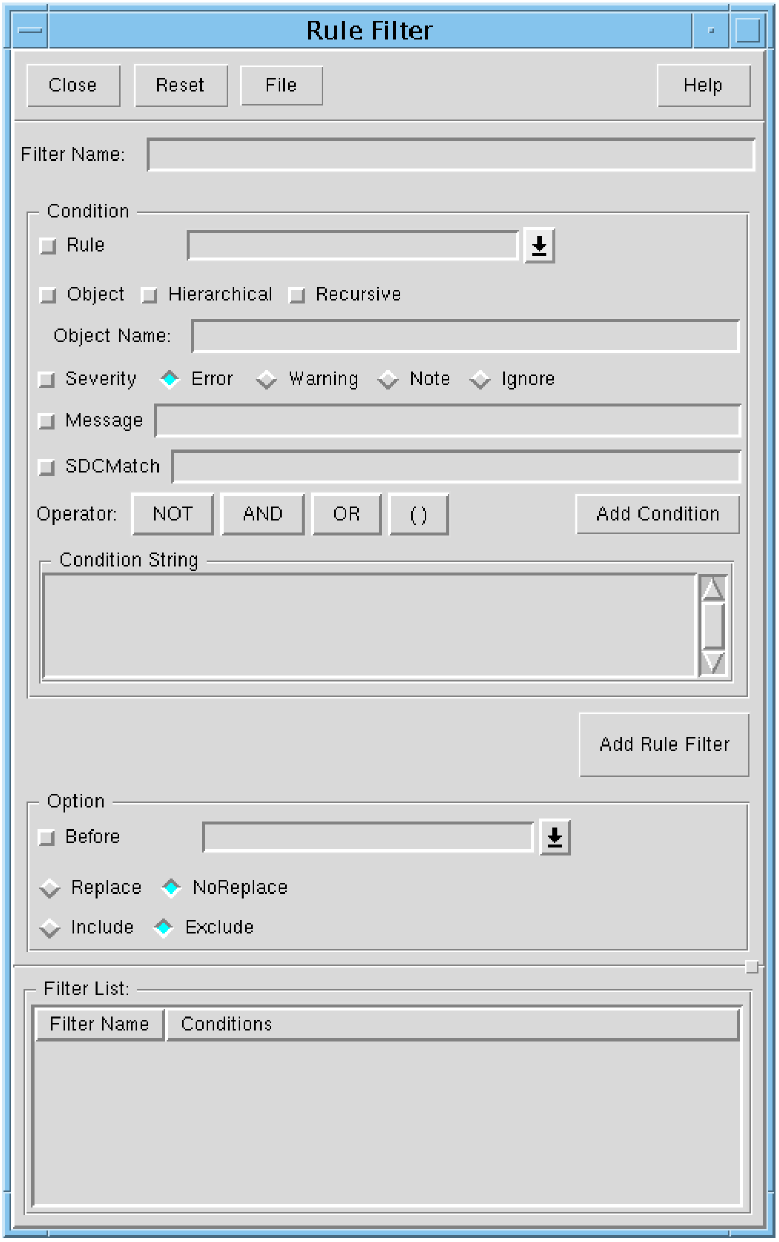
1. Locate a highlighted rule check number.
2. Click the *+* symbol preceding a highlighted rule to expand the entry.
3. Click the *+* preceding the location of the occurrence.

Some messages can be further expanded to show where they are located in the library or design.

Filtering Rules

You can add or delete rule filters with the Rule Filter form.

➤ From the HDL or Modeling Rule Manager, click the *Filter* button.



□ Rule

□ Object □ Hierarchical □ Recursive

□ Severity ● Error ○ Warning ○ Note ○ Ignore

□ Message

□ SDCMatch

□ Before

○ Replace ● NoReplace

○ Include ● Exclude

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.

   ❑ *Before* — Specifies that the new or replaced filter is inserted before the first filter whose name matches the provided pattern. This can be the same name as the one being replaced. In this case, the replaced filter stays in the same position in the list of filters. Type the name of the filter or use the pull-down menu.

- *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.
- *Include* — Specifies that the filter will cause any matching occurrence not to be filtered out, unless this is reversed by another filter down the list.
- Exclude — Specifies that the filter will cause any matching occurrence to be filtered out, unless this is reversed by another filter down the list.

4. Click the *Add Rule Filter* button.

Deleting Rule Filters

To delete a rule filter:

1. Select a filter in the *Filter List*.
2. Right-click and choose *Delete* or *Delete All* from the pop-up menu.

5

Using the Setup Mode

Overview

Conformal Extended Checks has two operating modes, Setup and Verify. After startup, the Conformal software begins operation in the Setup system mode, indicated by the `SETUP>` prompt in the command entry window.

Setting Options

The following sections describe the settings you can apply. If you decide to use any of these options, you must choose them *before* reading in the library and design files.

- Change the Severity of Rule Checks

  See Chapter 4, "Managing Rule Checks" for more information.

- Specify Case Sensitivity

  Use the `SET CASE SENSITIVITY` command to specify whether names you enter are case sensitive. The system default is no case sensitivity.

- Specify Directives Handling

  Use the `SET DIRECTIVE` command to specify whether to enable or disable the effects of directives when reading in Verilog or VHDL files.

  To see a list of the supported vendor names and directives, and information on enabling and disabling these directives, see the `SET DIRECTIVE` command in the *Conformal Extended Checks Reference Manual*, or type `help set directive -verbose` in the command line.

- Specify Text Handling Rules

  The `SET NAMING RULE` command specifies special text handling rules such as hierarchical separators, tristate naming rules, register naming rules, array delimiters, instance names, or variable names. This command has no effect unless you use it before reading in a Verilog or VHDL design file.

  Note: This has no effect on the way key points are reported (for example, the `REPORT GATE` command).

- Set Handling for Undefined Cells

  By default, when Conformal Extended Checks finds an undefined cell, it reports an error. The `SET UNDEFINED CELL` command prevents the tool from reporting an error. Instead, Conformal Extended Checks handles these undefined cells as blackboxes.

Additionally, the `-auto_assign` option sets ports of an undefined cell with drivers to `input`. However, the `-noauto_assign` option sets all ports of an undefined cell to `inout`.

Note: For information on replacing blackboxed modules with synthesized modules, see the `WRITE BLACKBOX WRAPPER` or `SUBSTITUTE BLACKBOX WRAPPER` command in the *Conformal Extended Checks Reference Manual*, or type `help <command> -verbose` in the command line.

- Set Handling for Undefined Ports

  Undefined ports in the design or library cause an error message unless you ignore them using the `SET UNDEFINED PORT` command.

- Specify a Value for Undriven Signals

  Globally set all undriven signals in the design to `0`, `1`, `X`, or `Z` using the `SET UNDRIVEN SIGNAL` command.

- Set a Design Root Level

  After you read in a design, Conformal Extended Checks treats the top module as the root module by default. However, you can specify a different root module with the `SET ROOT MODULE` command. Use this command to focus on specific parts of a design for verification and debugging.

  For more information, see Changing the Root Module on page 80.

- Specifying Search Paths

  Use the `ADD SEARCH PATH` command to specify the location of HDL files or libraries that must be included in the session, but are not in the current working directory. The `READ LIBRARY` and `READ DESIGN` commands use the search path set by this command. Conformal searches for design and library files in the order of the paths listed from left to right in the command string.

  For more information, see Adding Search Paths on page 80.

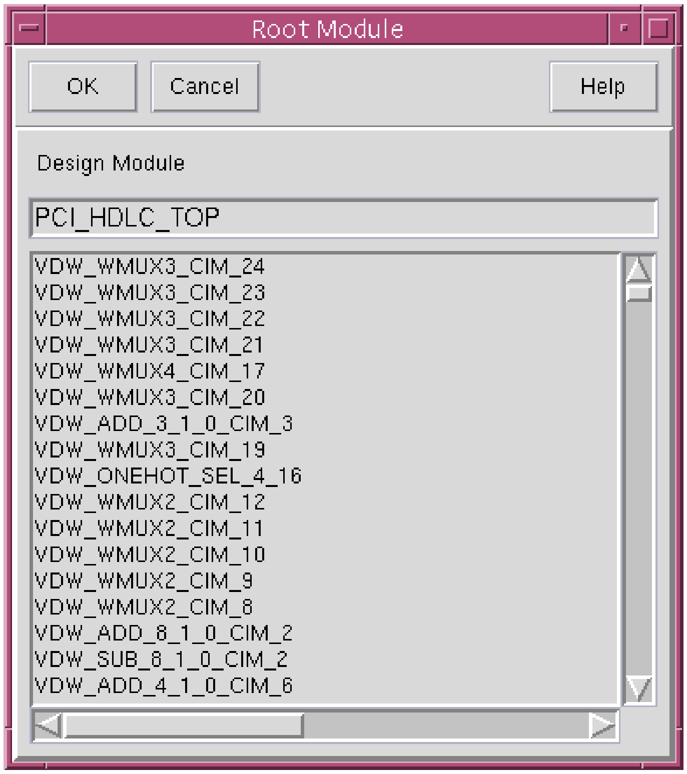
- Specify Non-Compiled Modules

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

  For more information, see Adding Notranslate Modules on page 82.

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.



Design Module

PCI_HDLC_TOP

```
VDW_WMUX3_CIM_24
VDW_WMUX3_CIM_23
VDW_WMUX3_CIM_22
VDW_WMUX3_CIM_21
VDW_WMUX4_CIM_17
VDW_WMUX3_CIM_20
VDW_ADD_3_1_0_CIM_3
VDW_WMUX3_CIM_19
VDW_ONEHOT_SEL_4_16
VDW_WMUX2_CIM_12
VDW_WMUX2_CIM_11
VDW_WMUX2_CIM_10
VDW_WMUX2_CIM_9
VDW_WMUX2_CIM_8
VDW_ADD_8_1_0_CIM_2
VDW_SUB_8_1_0_CIM_2
VDW_ADD_4_1_0_CIM_6
```

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

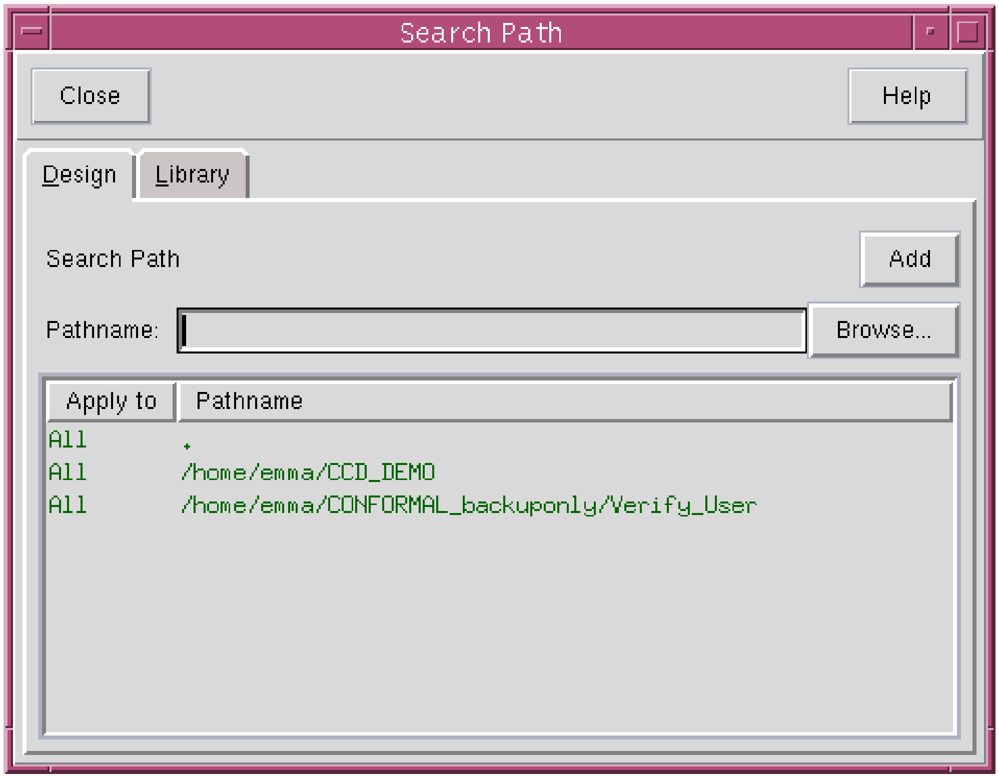
To sort the list alphabetically, right-click in the column display area and choose *Sort* from the pop-up menu.

Adding Search Paths

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

Note: If you do not add search paths, the software 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.



```
All       .
All       /home/emma/CCD_DEMO
All       /home/emma/CONFORMAL_backuponly/Verify_User
```

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.

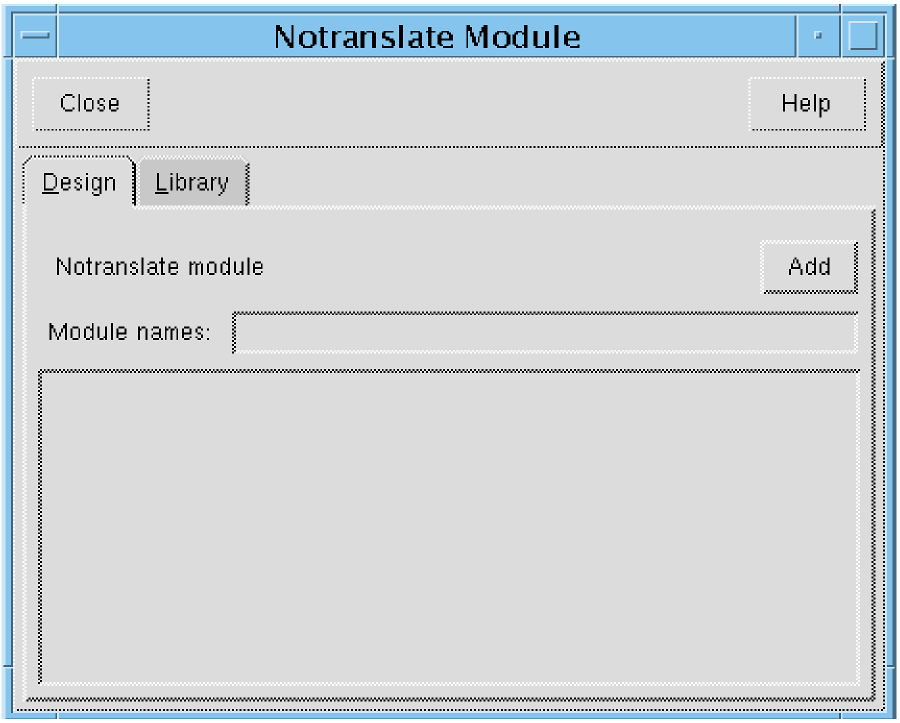
Pathname list box	Lists the directory search paths. To delete directory search paths, right-click on a path to bring up the pull-down menu and select either a *Delete Search Path* or *Delete All Search Paths*.

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

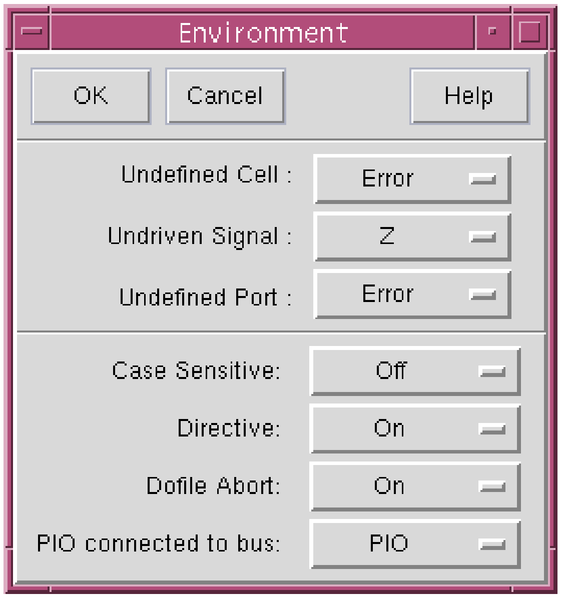
Add the notranslate modules, and adds the notranslate module names to the list box.

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.



Undefined Cell : Error
Undriven Signal : Z
Undefined Port : Error
Case Sensitive: Off
Directive: On
Dofile Abort: On
PIO connected to bus: PIO

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 Conformal 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 Conformal Extended Checks handles any PIO that is connected to a bus. Choose one of the following:

- *PO*—Conformal Extended Checks treats them as primary outputs.
- *PIO*—Conformal Extended Checks treats them as primary inputs/outputs.
- *Exit*—Conformal exits the session and returns to the system prompt if an error message occurs.

Reading in Libraries and Designs

The procedures described in this section apply to reading and writing library and design files.

Reading in Library Files

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, Conformal uses the first module found and ignores all others. However, you can use the `-lastmod` option to specify that Conformal use the last module and ignore earlier ones.

Note: The library can also be in the Synopsys Liberty format.

Note: While reading Liberty files, the Conformal software records the timing relations, if available, between clock and data pins for each black box module. These relations then will be translated into commands `add data association <datapins> ... -domain <clock domain of the clock pin>`, which are applied automatically in Verify mode.

Reading in Multiple Library Files

Cadence recommends that you use one of the following methods to read in multiple library files.

Method 1:

List all the library files in the `READ LIBRARY` command explicitly or using 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.v file2.v file3.v… \
-verilog
```

or

```
read library lib/*.v -verilog
```

Method 2:

1. Create a file containing all the necessary library files. For example, a file called `verilog_all.v` might contain the following:

   ```
   `include “file1.v”
   `include “file2.v”
   `include “file3.v”
   …
   ```

2. Then, append the name of this newly-created file to the `READ LIBRARY` command:

   ```
   read library verilog_all.v -verilog
   ```

Method 3:

Read multiple library files of different languages as follows:

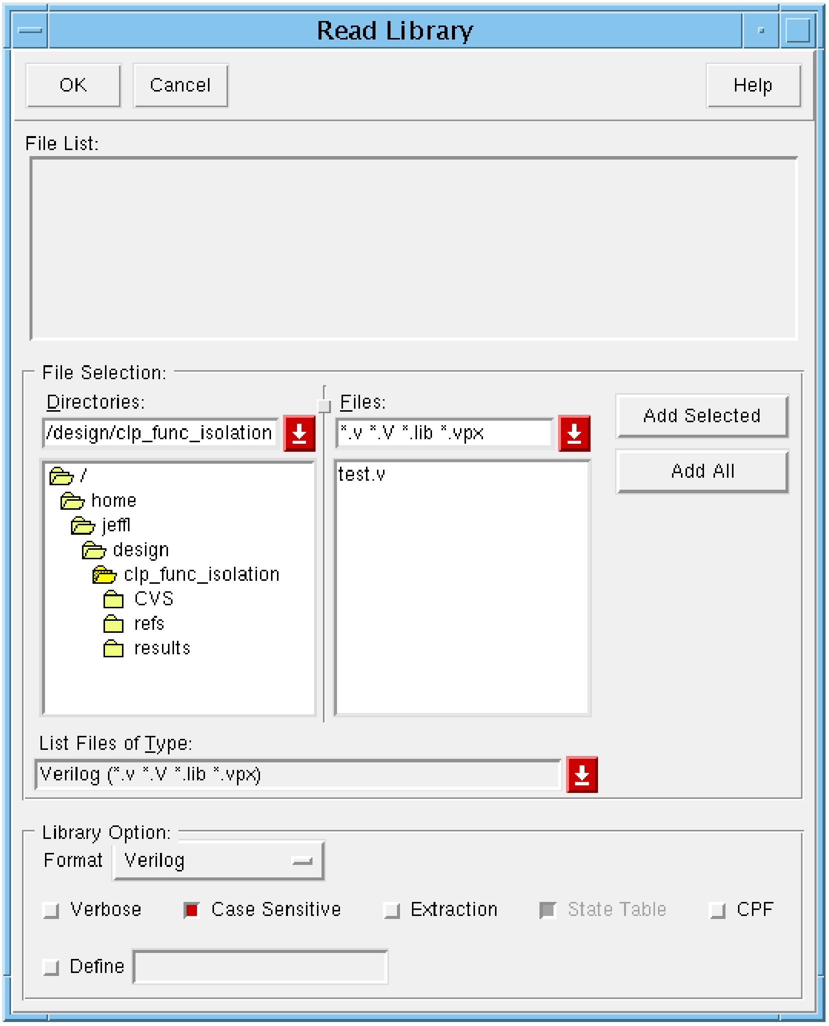
```
read library file1.v -verilog
read library file2.vhd -vhdl -append
```

Writing Libraries

After you read in a library, write it out in Verilog format to learn how Conformal parses User-Defined Primitive (UDP) library models. To write out the library, use the `WRITE LIBRARY` command.

Read Library Form

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



Directories: /design/clp_func_isolation

- /
  - home
    - jeffl
      - design
        - clp_func_isolation
          - CVS
          - refs
          - results

Files: *.v *.V *.lib *.vpx

test.v

List Files of Type: Verilog (*.v *.V *.lib *.vpx)

Format: Verilog

☐ Verbose ☒ Case Sensitive ☐ Extraction ☐ State Table ☐ CPF

☐ Define

Read Library Fields and Options

File List

Lists the library files that the Conformal Extended Checks reads in for this session. As you build the list of files, the Conformal Extended Checks 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 a format.

Verbose

Displays the verbose messages for parsing and translating each library module.

Case Sensitive

Specifies that the Conformal Extended Checks should handle the library as case sensitive.

Note: This option is not available for VHDL.

Extraction

Specifies that the Conformal Extended Checks 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.

CPF

Specifies to read the libraries included in the `library_set` commands of the input CPF file(s). You can use this option to read in the library files directly from CPF files, instead of listing them in the dofile.

Define Specifies the text macro name you want to 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 Conformal Setup mode. The design formats currently supported are Verilog and VHDL. If you must overwrite the design, use the `-replace` option. If Conformal 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 to specify that Conformal 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
```

Reading in Multiple Design Files

Cadence recommends that you use one of the following methods to read multiple design files of the *same* language.

Method 1:

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

```
read design file1.v file2.v file3.v... \
-verilog
```

Or

```
read design src/*.v -verilog
```

Method 2:

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

```
`include “file1.v”
`include “file2.v”
`include “file3.v”
…
```

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

   ```
   read design foo.v -verilog
   ```

Writing the Design

Use the WRITE DESIGN comand after you read in a design. Conformal writes it out in Verilog format. This is useful in learning how Conformal parses RTL designs.

Using Verilog Command Filelists

The following describes a time-saving device that allows you to create a single Verilog filelist rather than use the READ LIBRARY and READ DESIGN commands separately. Read in this filelist using the READ DESIGN command with the -file option.

For example, assume a directory named /user/library/verilog contains the following library files:

```
and.v or.v
dff.v dlat.v
```

And assume in the current working directory, there is an RTL directory containing the Verilog code:

```
$CWD/rtl/foo.v
```

First, create a Verilog filelist with the above contents. In this example, the file name is `bt.vc`:

```
-y /user/library/verilog
rtl/foo.v
```

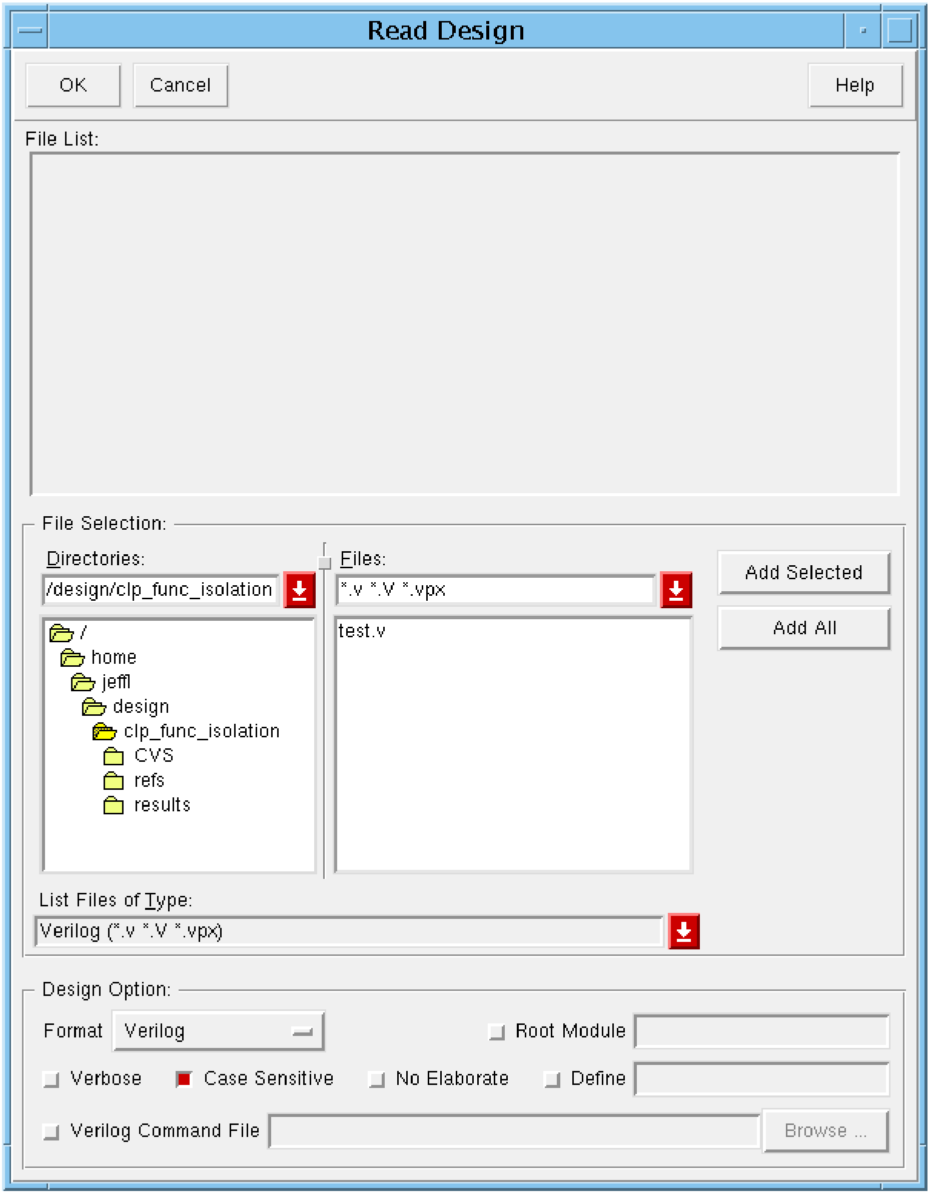
Then, use the READ DESIGN command with the -file option to read in the design and libraries without using the READ LIBRARY command. (See the following example.)

```
read design -file bt.vc -verilog
```

Conformal uses the specified library directory to locate the modules needed in the design.

Read Design Form

Use the Read Design form (*File – Read Design*) to specify the design filenames.



Read Design

File List:

File Selection:

Directories: /design/clp_func_isolation

Files: *.v *.V *.vpx

- /
  - home
    - jeffl
      - design
        - clp_func_isolation
          - CVS
          - refs
          - results

test.v

List Files of Type: Verilog (*.v *.V *.vpx)

Design Option:

Format: Verilog

□ Root Module

□ Verbose ☒ Case Sensitive □ No Elaborate □ Define

□ Verilog Command File

Read Library Fields and Options

File List

Lists the design files that the Conformal Extended Checks reads in for this session. As you build the list of files, the Conformal Extended Checks 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 library 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 93 for more information.

Root Module

Designates a root module other than the top module. Click the check box and type the name of the intended top root module in the field.

Note: If a single top-level module exists, by default, Conformal uses it. However, if multiple top-level modules exist, Conformal specifies one. Use this option to change that specification.

Verbose

Displays the verbose messages for parsing and translating each module in the design.

Case Sensitive

Specifies that the Conformal Extended Checks must handle the design as case sensitive.

Note: This option is not available for VHDL.

No Elaborate

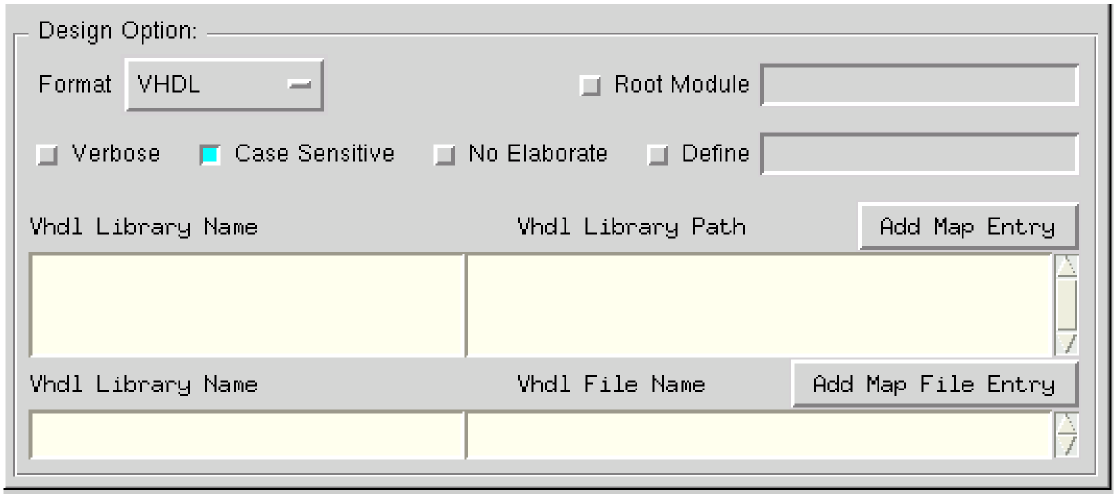
Specifies that you intend to read in multiple files of different languages.

Define
Specifies the text macro name you want to define.

Verilog Command File
If you are using Verilog command file lists, click this check box and type the name of the Verilog command file, or click *Browse* to open the Verilog Command File window to choose a file.

Specifying Design Options for VHDL Designs

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



Format: VHDL

☑ Case Sensitive

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.

Placing Design Constraints

After Conformal successfully reads the design and libraries, place constraints on the design. 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

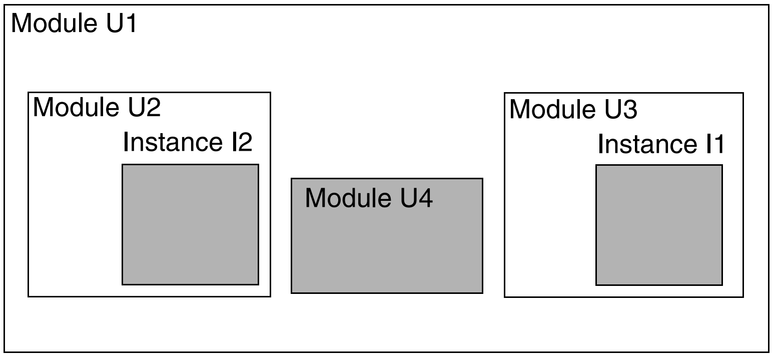
Conformal Extended Checks verifies automatic checks. A missing constraint often appears as one or more false properties. Generally, these are easily debugged since Conformal Extended Checks shows a counterexample having behavior you do not expect. By adding the proper constraints, you avoid these false negatives.

The following sections show procedures that are commonly used to set constraints.

Black Boxes

To treat any module or instance as a blackbox, use the `ADD BLACK BOX` command.

Note: Conformal Extended Checks 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

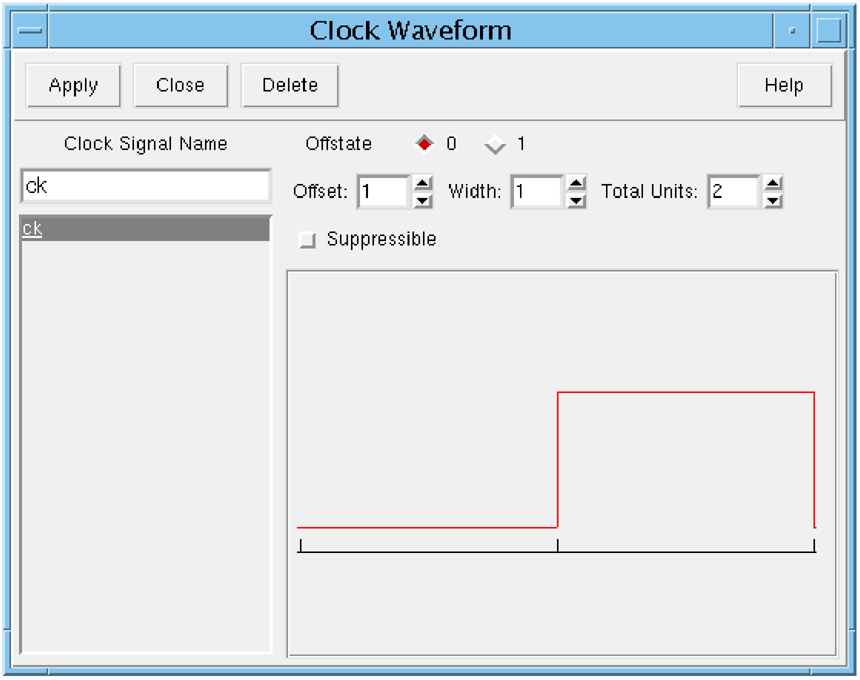
Use the `DELETE BLACK BOX` command, or use the following procedure in the Hierarchical Browser window:

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.

Defining Clocks

Use the ADD CLOCK command, or the Clock Waveform form (*Setup – Clock*), to add clock waveform information to the design's clock pins.



Clock Signal Name: ck

ck

Offstate ● 0 ○ 1

Offset: 1 Width: 1 Total Units: 2

□ Suppressible

Use the Clock Waveform form to add and delete waveforms for clock input signals using a graphical waveform display.

Clock Waveform Fields and Options

Clock Signal Name
Specifies the name of the clock signal you intend to define.

Offstate
Specifies an initial value for the clock waveform. Choose *0* or *1*.

Offset
Specifies the length of the initial value of the clock waveform.

Width
Specifies the total length of the new value after the clock waveform changes its initial value.

Total Units
Specifies the total length of the clock cycle.

Suppressible

Arbitrarily suppresses the pulse of the selected clock. By default, the specified clock is enabled. Use this option to override the default and arbitrarily suppress the pulse of the defined clock.

For more information on deriving waveforms from clock frequencies, see Appendix E, “Defining Clocks”.

Note:

Tip

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

Redefining Clocks

The defined clocks are displayed in the *Clock Signal Name* list, so you can double-click a clock to select it and display it in the *Clock Signal Name* field. When you redefine clocks, Conformal Extended Checks deletes the selected clock, closes any open CDC windows, and adds the newly defined clock.

Deleting Clock Waveforms

Use the `DELETE CLOCK` command, or the Clock Waveform form (*Setup – Clock*), to delete waveforms for clock input signals.

In the Clock Waveform form, double-click the appropriate signal name in the list so it is displayed in the *Clock Signal Name* field and click *Delete*.

Examples of Clock Waveforms

In the following figure, the `ADD CLOCK` command adds `CLK` and `CLK1` as described below.

- `CLK`:

  The default values are added as follows.

  - Start at value 1 at time unit 0
  - Go down to a value of 0 at time unit 1
  - Return to a value of 1at time unit 2, and so on

- `CLK1`:

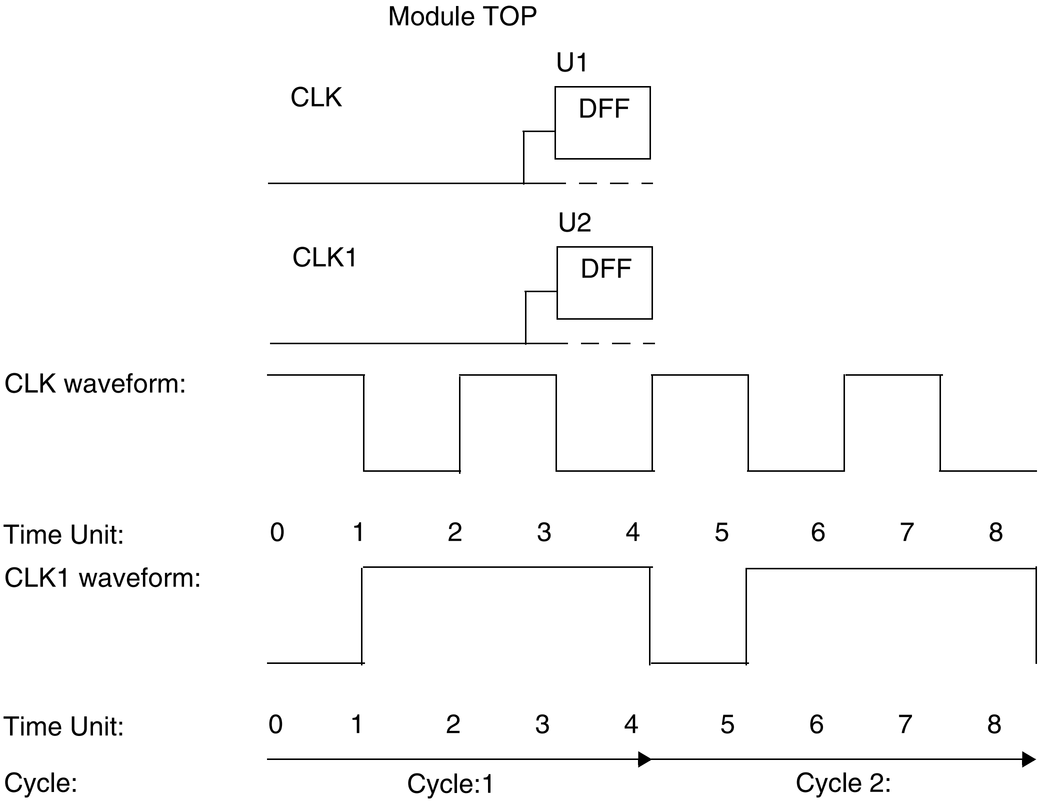
  The `-waveform` option specifies the values as follows.

  Cycle One:

  - ❑ Start at value 0 at time unit 0
  - ❑ Go up to a value of 1 at time unit 1
  - ❑ Remain at a value of 1 for time units 2 and 3
  - ❑ Return to a value of 0 at time unit 4

  Cycle Two:

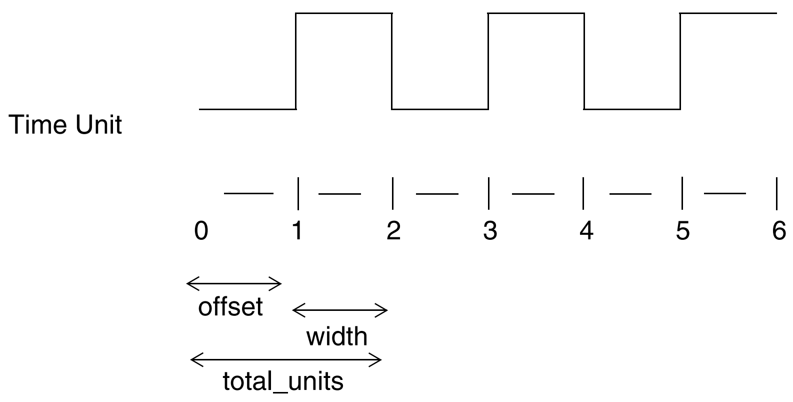
  - ❑ Start at time unit 4 with a value of 0
  - ❑ Go up to a value of 1 at time unit 5
  - ❑ Remain at a value of 1 for time units 6 and 7
  - ❑ Return to a value of 0 at time unit 8



```
add clock 0 CLK
```

This clock will default to waveform 0 1 1 2

CLK Waveform

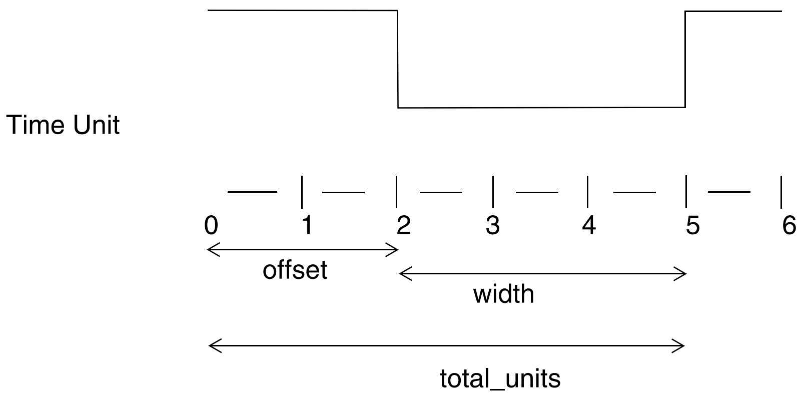


```
add clock 1 -waveform 2 3 5 CLK*
```

This command specifies a waveform for all clock signals that begin with the characters `CLK`. The waveform is as follows:

- Starts with value 1
- Offset = 2
- Width = 3
- Total_units = 5

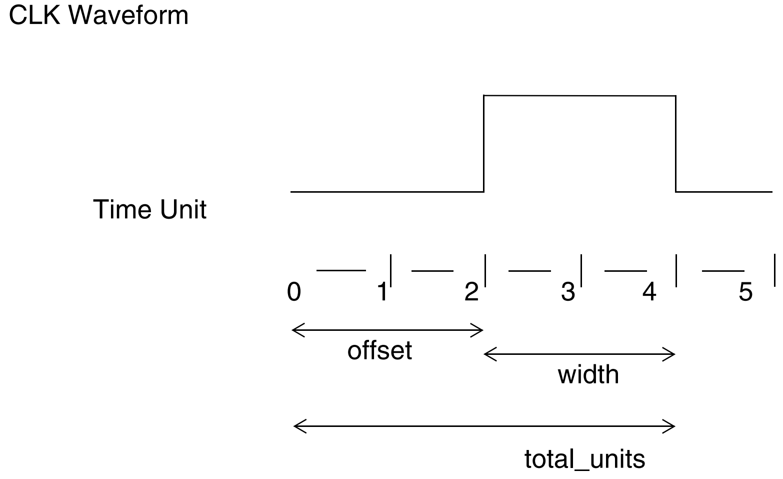
CLK Waveform



```
add clock 0 -waveform 2 2 4 CLK*
```

This command specifies a waveform for all clock signals that begin with the characters `CLK`. The waveform is as follows:

- Starts with value 0
- Offset = 2
- Width = 2
- Total_units = 4

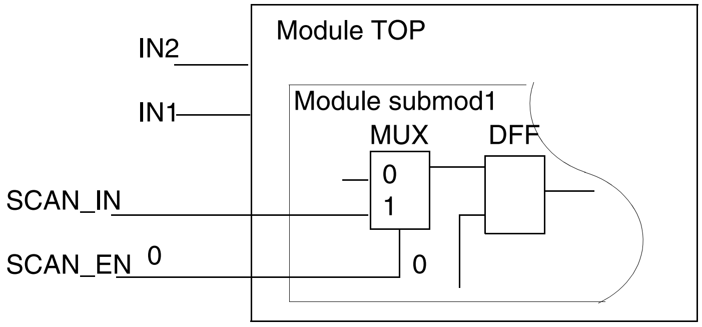


Clock Domain Rules

See “Clock Domain Rule” on page 224 of Chapter 9, “Clock Domain Crossing Checks” for information related to this feature. This feature is also available in the Verify system mode.

Pin Constraints

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

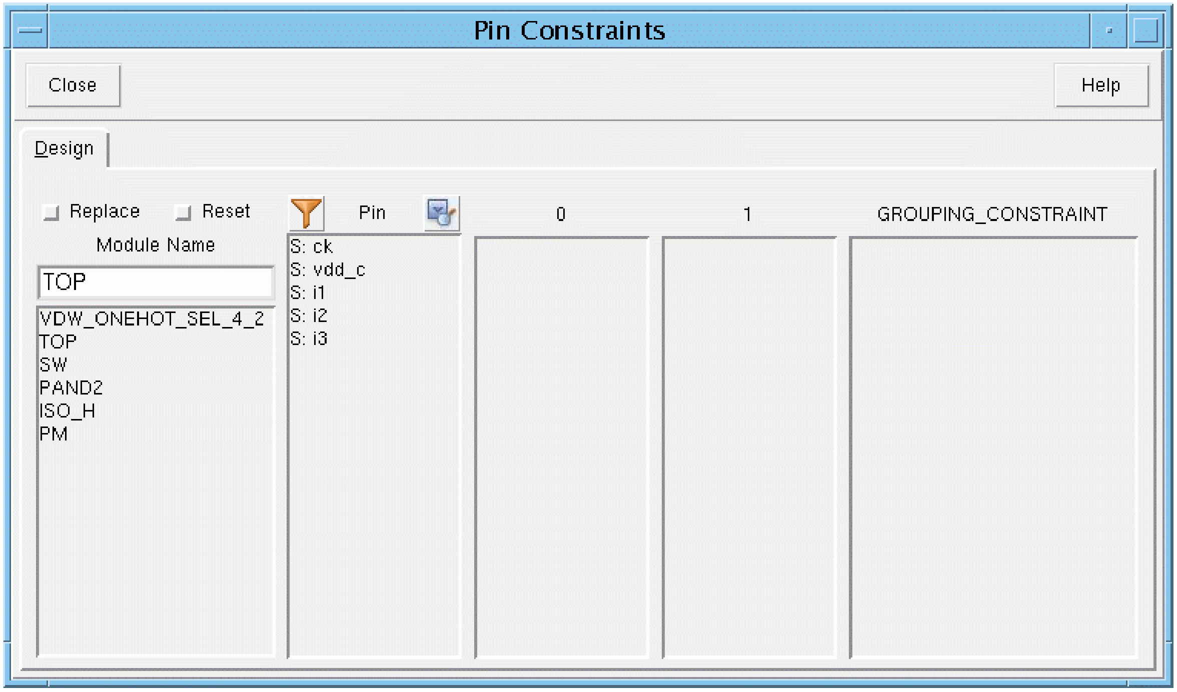


```
SETUP> add pin constraints 0 SCAN_EN
```

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

Alternatively, you can use the Pin Constraints form from the main window to add and delete primary input pin constraints.

➤ Choose *Setup – 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 Constraint 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.

   Conformal Extended Checks adds the selected primary input to the list in the appropriate column.

   The *GROUPING_CONSTRAINT* column lists *One Hot*, *One Cold*, *Zero One Hot*, and *Zero One Cold* pin groups with an identifying heading (for example, `---ONE_HOT---`).

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.

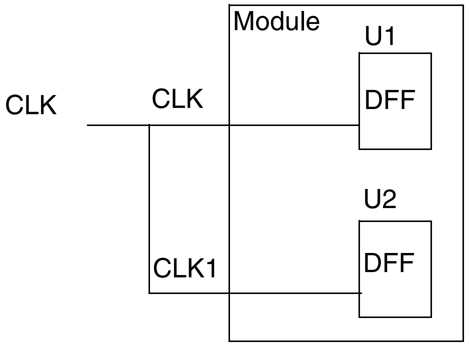
   Conformal Extended Checks removes the pin constraint. In the case of *GROUPING_CONSTRAINT*, Conformal Extended Checks deletes the entire group.

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.

Pin Equivalences

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

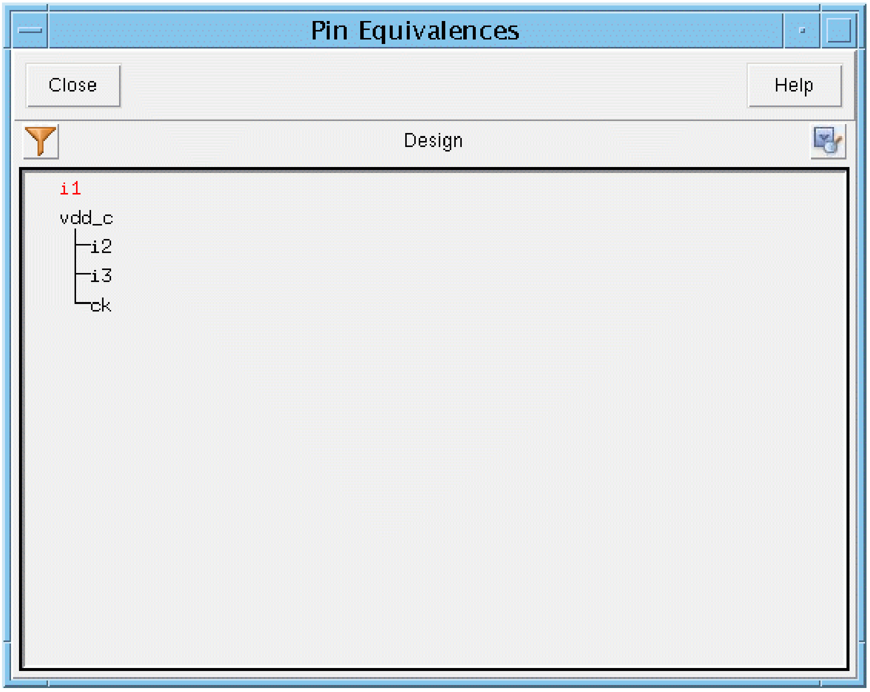


```
SETUP> add pin equivalence CLK CLK1
```

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 from the main window to add and delete pin equivalences.

1. Choose *Setup – 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, as shown in figure above. 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*.

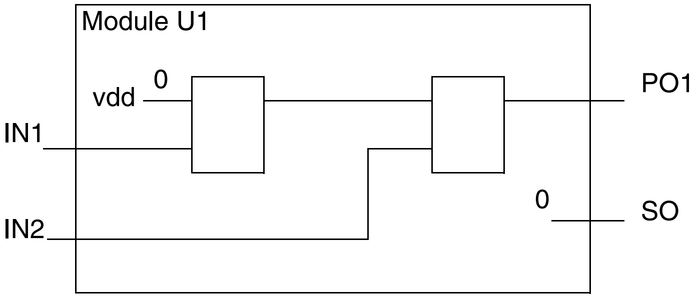
   Conformal Extended Checks displays the added pin equivalences below the target primary input with a connecting line. Conformal Extended Checks denotes inverted pin equivalences with (-) following the primary input name.

Deleting Pin Equivalences

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

Tied Signals

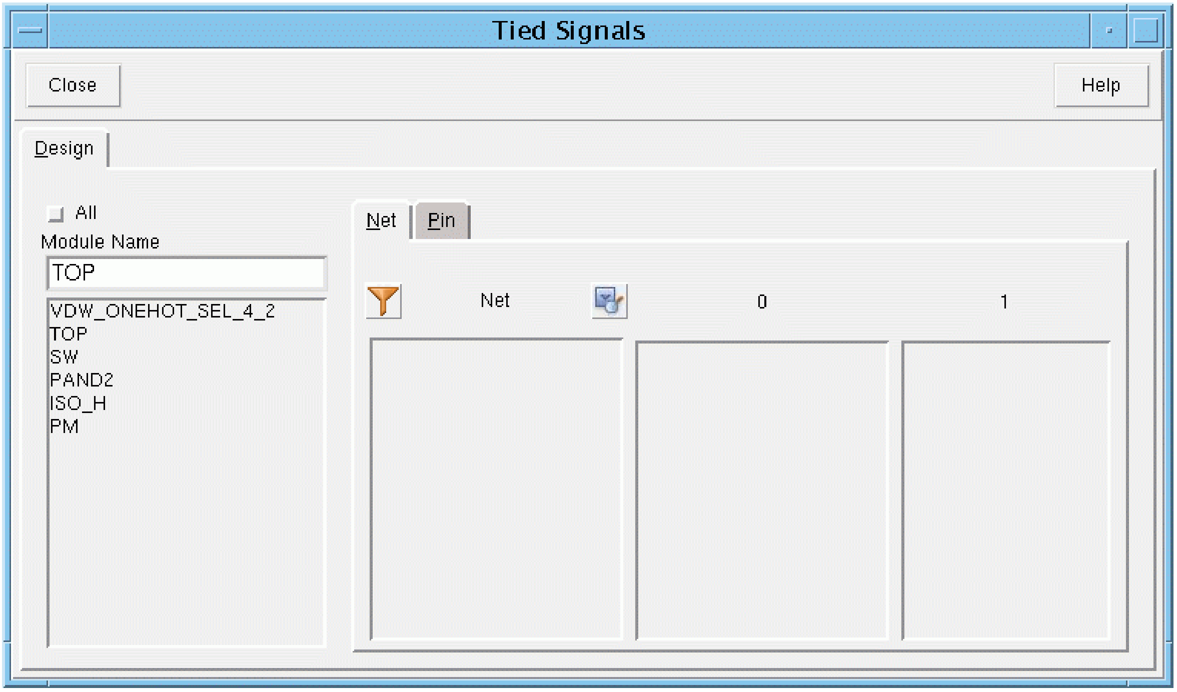
To tie any floating nets or pins to Logic-0 or Logic-1, use the ADD TIED SIGNALS command.



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

To delete specified tied signals from the design, 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.



Module Name: TOP

VDW_ONEHOT_SEL_4_2
TOP
SW
PAND2
ISO_H
PM

There are four columns in the design section of the form. 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 Tied Signals 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.

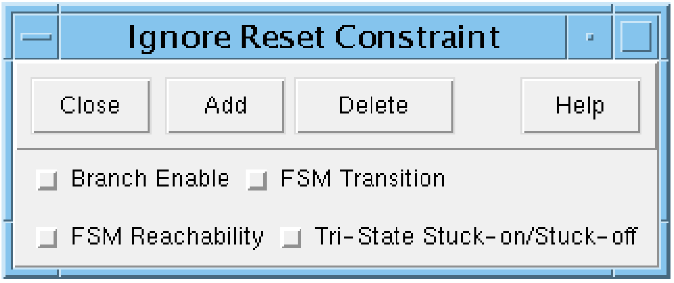
   The name appears in the *Module Name* field.
2. Click the *Net* or *Pin* tab to show the appropriate list of tied signals.
3. Click a net or pin name under the *0* or *1* column to select it.
4. Right-click and choose *Delete Tied Signal* from the pop-up menu.

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.

Ignore Reset Constraints

Use the ADD_IGNORE_RESET_CONSTRAINT command, or the Ignore Reset Constraint form (*Setup – Ignore Reset Constraint*), to ignore reset constraints for the Branch Enable, Tristate Stuck-on/Stuck off, FSM Transition, and FSM Reachability checks or to reinstate them. Other checks such as bus and dont_care are not affected by this command.



Ignore Reset Constraint Fields and Options

Add	Ignores the reset constraints for the selected checks.
Delete	Reinstates the reset constraints for the selected checks.
Branch Enable	Ignores reset constraints for Branch Enable checks.
FSM Transition	Ignores reset constraints for FSM Transition checks.
FSM Reachability	Ignores reset constraints for FSM Reachability checks.
Tri-State Stuck-On/Stuck-Off	Ignores reset constraints for Tristate Stuck-on/Stuck off checks.

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
  ```

  For an initialization sequence file, the following syntax must appear on each line of the initialization sequence file:

  ```
  <time> <value> < <cell_name*>…
                  |< <type_specifier> < |cell_name*>… >…
                   >
  ```

  The following are examples of initialization sequence file lines:

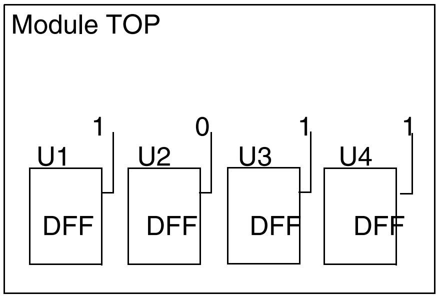
  ```
  0   8’b11110000   in7[5:2]   // assign in7[5:2] = 4’b0000 (truncated)
  0   0   -PI                  // all PIs
  0   0   -DFF                 // all DFFs
  0   $random       -PI *      // assign random number to PIs
  ```

  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.

  For detailed information about writing an initialization sequence file, refer to Appendix F, “Initialization Sequence File.”

- Add initial states to individual flip-flops using the ADD INITIAL STATE command, as depicted in the following figure:



```
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
```

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, Conformal Extended Checks issues a warning and *does not* overwrite the assignment.

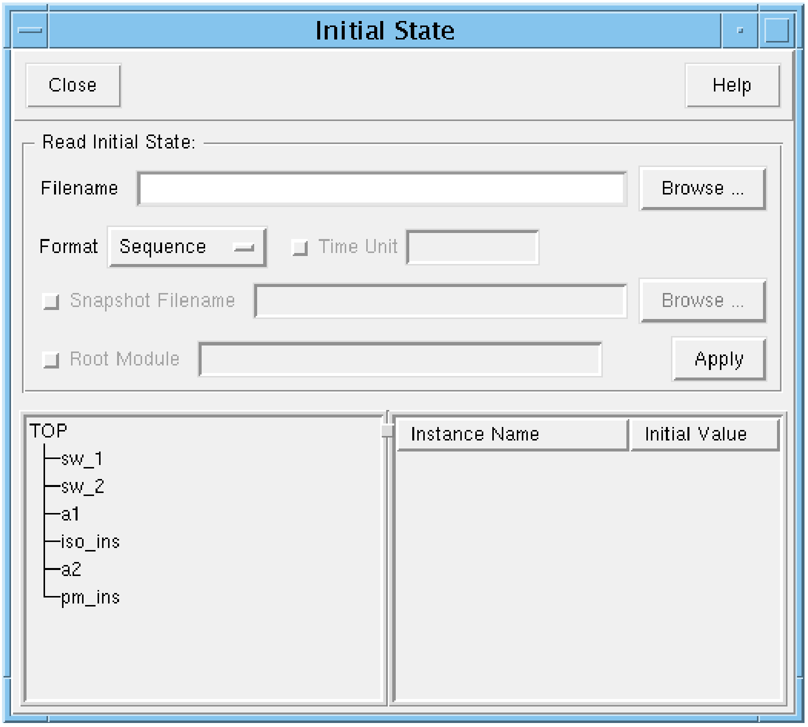
The following is an example of a conflict and the warning:

```
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
```

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.



Format Sequence

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 an Initialization Sequence File

Use the following procedure 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 an initial state from the pop-up menu:

Deleting Previously Specified Initial States

1. Click an instance name in the *Instance Name* list.
2. Right-click to open the pop-up menu and choose *Delete Initial State* or *Delete All Initial States*.

Assertion Constraints

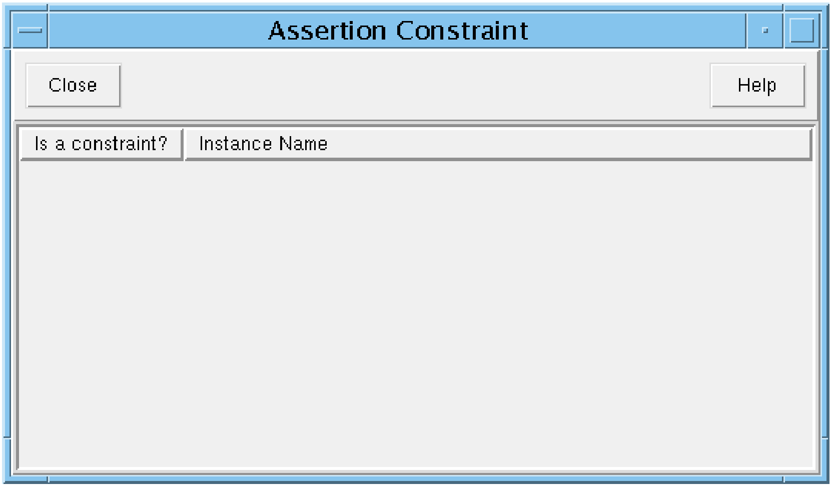
Use the Assertion Constraint form to turn assertion library instances into proof assumptions (constraints) or proof obligations (properties to be proved). By default, assertion library instances are assumed to be proof obligations.

In the main Conformal Extended Checks window, assertion monitors and assertion constraints are depicted as follows:

🚩 Assertion Monitor

❓ Assertion Constraint

To convert an instance from a property into a constraint (or to reverse this action) access the Assertion Constraint form from the *Setup* drop-down menu.



Changing Instances from Proof Obligations to Constraints

Use the following procedure to change one or all instances from proof obligations to 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 Conformal Extended Checks will treat the instance as a constraint.

3. Click *Close*.

Changing Instances from Constraints to Proof Obligations

Use the following procedure to change one or all instances from constraints to proof obligations.

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*. Right click and choose one of the following from the pop-up menu.

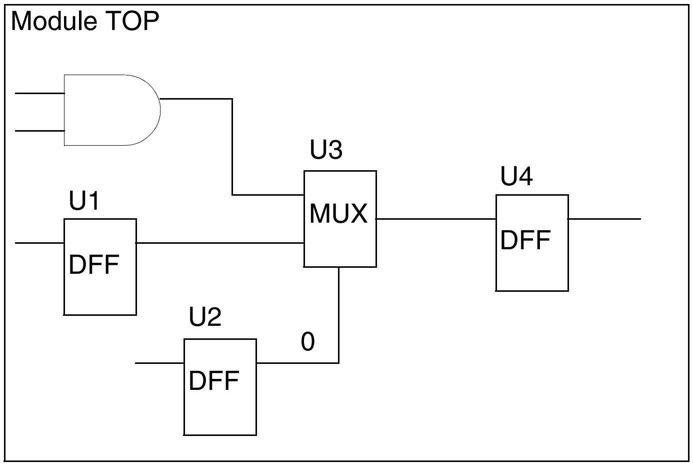
   No replaces *Yes* in the *Is a constraint?* column to show that Conformal Extended Checks will *not* treat the instance as a constraint.
3. Click *Close*.

Refreshing the Hierarchy Browser

➤ Click the *Refresh Hierarchy* icon.

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 only takes a value of 0 or 1 on the output of the instances.

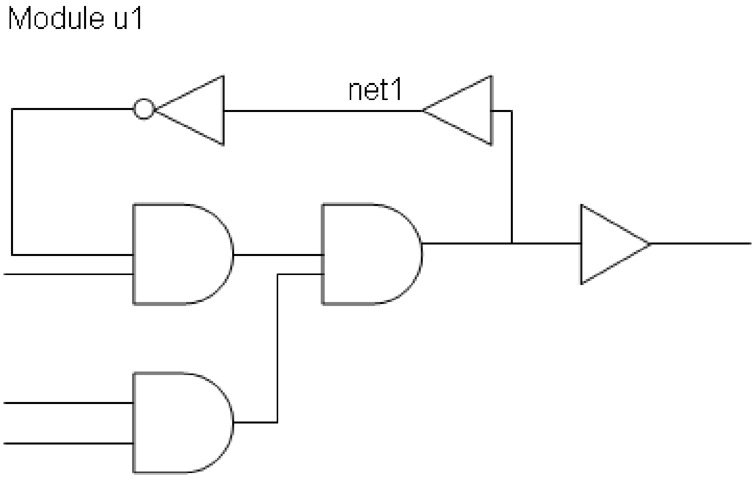


```
SETUP> add instance constraints 0 /TOP/U2
```

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.

Primary Inputs

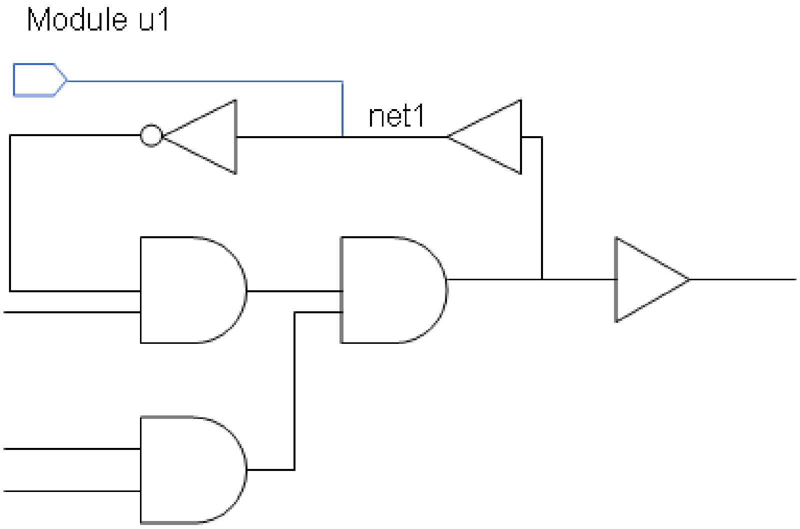
To specify primary inputs, use the ADD PRIMARY INPUT command. Use this command when you want to cut a large logic cone to help ease validation, or constrain a particular net. For example, for the following circuit:



If you use the following command:

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

The Conformal software will interpret the circuit as follows:



Filtering Extraneous Counterexample Values

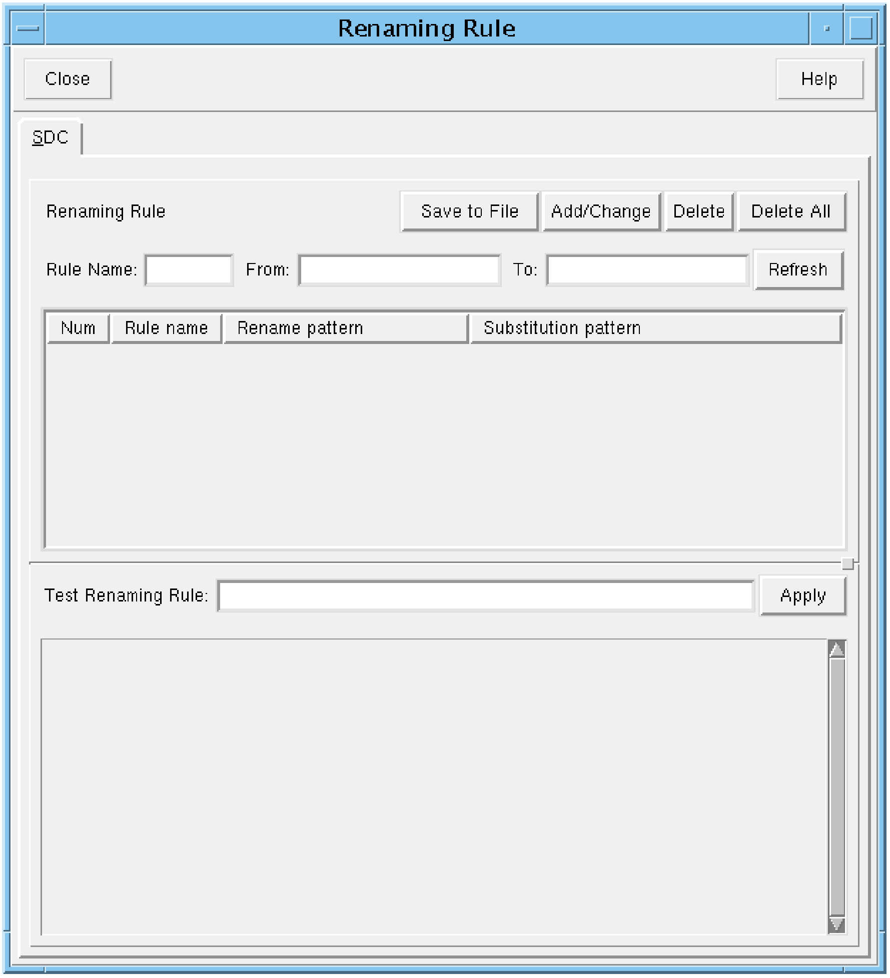
By default, Conformal Extended Checks allows the outputs of black boxes, don’t cares, and combinational loops to participate in the counterexample of a false property. This default setting can result in false negatives. To change the counterexample handling behavior, use the `SET X HANDLING` command.

Renaming Rules

When the naming conventions in the design are not the same, you can use the `ADD RENAMING RULE` command to identify string patterns and define temporary substitute string patterns. Additionally, you can apply renaming rules to module names when you use the `ADD RENAMING RULE` command. Use this command before you use the `READ LIBRARY` and `READ DESIGN` commands.

Alternatively, you can use the Renaming Rule form (*Setup – Renaming Rule*), to add or delete renaming rules.

You can also use the TEST RENAMING RULE command or the Renaming Rule form to test renaming rules for mapping performance based on name mapping.



Renaming Rule Form Fields and Options

Save to File	Opens the Save Renaming Rules window, where you can click a file in the *Files* display or type a name in the *Files* field.
Add/Change	Adds the renaming rule to the list, or updates a renaming rule change in the list.

Delete — Deletes the selected renaming rule.

> *Tip*
>
> You can also right-click on a rule to open the pop-up menu where you can delete the selected or all renaming rules.

Delete All — Deletes all renaming rules.

Rule Name — Specifies a unique rule name.

From — Specifies the renaming pattern.

To — Specifies the substitution pattern.

Refresh — Refreshes the rule list.

Test Renaming Rule — Specifies the renaming rule to be checked. After adding renaming rules, you can use bottom part of the form to check their effectiveness.

Reporting Renaming Rules

Use the `REPORT RENAMING RULE` command or the Renaming Rule Report form (*Report – Renaming Rule*) to display the list of renaming rules. The list displays a rule number along with a renaming rule. If you do not enter options, the Conformal software displays all renaming rules.

Reading an SDC File

In Setup mode, use the `READ SDC` command to read in the constraint file you want to verify. If you must overwrite the existing SDC information, use the `-replace` option.

For information on the supported SDC commands, see the `READ SDC` command in the *Conformal Extended Checks Reference Manual*, or type `help read sdc -verbose` in the command line.

6

Using the Verify Mode

Overview

Use the `SET SYSTEM MODE` command to move to the Verify mode. The Verify mode is indicated by a `VERIFY>` prompt.

Automatic Checks

When entering Verify mode, Conformal Extended Checks automatically performs the following sets of Automatic Checks:

- HDL Rule Checks (also referred to as RTL Rules)
- Modeling Rule Checks
- SDC Rule Checks

Additionally, Conformal Extended Checks automatically extracts all possible predefined properties. After you enter Verify mode, specify the Predefined Checks you would like Conformal Extended Checks to perform. (See “Selecting Predefined Properties for Verification” on page 121.)

Showing Rule Check Messages

Use the `REPORT RULE CHECK` command to report a list of violations of the automatically performed rule checks.

Rules with a severity of “Ignore” are not reported except with the `rule_name` option; and in this case, Conformal reports only the number of occurrences.

Performing Domain Crossing Checks

Refer to Chapter 9, “Clock Domain Crossing Checks.”

Ignoring Predefined Properties During Verification

Use the `ADD IGNORED PROPERTY` command to ignore all or specific predefined properties during a proof. You must use this command before the `ADD STATIC PROPERTY` command.

Note: When you designate specified properties as ignored properties, you cannot add them to the prove list until you make them available (`DELETE IGNORED PROPERTY` command). Likewise, when you add specified properties to the prove list, you cannot designate these same properties as “ignored” until you delete them from the prove list (`DELETE STATIC PROPERTY` command).

Note: Conformal automatically assigns ID numbers. They can differ from one version to another. Always use the full path in dofiles and when you rerun a design with a different Conformal version.

Selecting Predefined Properties for Verification

When you select the Verify mode, Conformal Extended Checks automatically extracts all possible predefined properties. Specify the properties you wish to verify with the `ADD STATIC PROPERTY` command.

Note: When you add specified properties to the prove list, you cannot designate these same properties as “ignored” until you delete them from the prove list (`DELETE STATIC PROPERTY` command). Likewise, when you designate specified properties as ignored properties, you cannot add them to the prove list until you make them available (`DELETE IGNORED PROPERTY` command).

Note: Conformal automatically assigns ID numbers. They can differ from one version to another. Always use the full path in dofiles and any time you rerun a design with a different Conformal version.

Note: If you do not specify parameters, Conformal Extended Checks adds all static properties.

Setting Proof Options

Specify the desired level of effort for Conformal Extended Checks to use to prove the properties with the `SET PROVE EFFORT` command:

Specify whether proofs will be performed with the hierarchical (that is, bottom-up) or flat (that is, top-down) method with the `SET PROVE OPTIONS` command.

Starting the Proof

The `PROVE` command starts the Conformal Extended Checks formal verification of the selected properties.

Reporting Proof Results

When the Conformal Extended Checks proof is finished, display the results using the `REPORT PROVED DATA` command.

There are two classes of options for this command. One class specifies the static properties to be reported. The other class specifies the format of the report. Refer to the *Conformal Extended Checks Reference Manual* for detailed descriptions of all options, or type `help report proved data -verbose` in the command line.

If you do not specify parameters, Conformal Extended Checks reports all the properties added with the `ADD STATIC PROPERTY` command and ignores all properties added with the `ADD IGNORED PROPERTY` command. The `-verbose` option allows you to visualize the counterexample for a given false property.

Running Reports

Use the *Report* menu 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 Verify mode.

The *Report* menu and Report form contains the following categories:

- Assertion Constraints Report on page 124
- Clock Association Report on page 124
- Clock Domain Report on page 124
- Clock Domain Priority Report on page 124
- Data Association Report on page 125
- Design Data Report on page 125
- Environment Report on page 125
- Floating Signals Report on page 125
- Generated Clock Report on page 126
- Instance Constraints Report on page 126
- Modules Report on page 126
- Notranslate Modules Report on page 127
- Pin Constraints Report on page 127
- Pin Equivalences Report on page 128
- Primary Inputs Report on page 128
- Primary Outputs Report on page 128
- Search Paths Report on page 128
- Synchronization Rule on page 129
- FSM Report on page 129

Assertion Constraints Report

Use the `REPORT ASSERTION CONSTRAINTS` command or the Assertion Constraints Report (*Report – Assertion Constraints*) window to specify and run a report of all the previously specified assertion constraints information.

Assertion Constraints Report Form Fields and Options

Instance Name	Specifies the name of a single instance.
All	Reports information on all assertion constraints
Summary	Specifies a summary report.
Verbose	Specifies a more detailed report.

Clock Association Report

Use the `REPORT CLOCK ASSOCIATION` command or the Clock Association Report form (*Report – Clock Association*) to display clock associations.

There are no customized options for the Report Clock Association form. Click *Apply* to view the report in the Transcript window.

Clock Domain Report

Use the `REPORT CLOCK DOMAIN` command or the Clock Domain Report form (*Report – Clock Domain*) to display clock domains that you have defined, determined by the Conformal software, or labeled as error or unknown according to the defined clock domain rule.

There are no customized options for the Report Clock Domain form. Click *Apply* to view the report in the Transcript window.

Clock Domain Priority Report

Use the `REPORT CLOCK_DOMAIN PRIORITY` command or the Data Association Report form (*Report – Clock Domain Priority*) to display clock domain priorities.

There are no customized options for the Report Clock Domain Priority form. Click *Apply* to view the report in the Transcript window.

Data Association Report

Use the `REPORT DATA ASSOCIATION` command or the Data Association Report form (*Report – Data Association*) to display data associations.

There are no customized options for the Report Data Association form. Click *Apply* to view the report in the Transcript window.

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

Design Module Name	Specifies the module name for the design.
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

Generated Clock Report

Use the `REPORT GENERATED CLOCK` command or the Generated Clock Report form (*Report – Generated Clock*) to display clocks that were originally added with the Conformal software.

There are no customized options for the Generated Clock form. Click *Apply* to view the report in the Transcript window.

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

Design Module Name — Specifies the module name for the design.

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

Design Module Name — Specifies the module name for the design.

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

Design Module Name	Specifies the module name for the design.
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.

Synchronization Rule

Use the `REPORT SYNCHRONIZATION RULE` command or the Synchronization Rule Report form (*Report – Synchronization Rule*) to display synchronization rules.

There are no customized options for the Synchronization Rule 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.

FSM Report

Use the `REPORT FSM` command or the FSM Report form (*Report – FSM*) to display automatically extracted finite state machines.

FSM Report Form Fields and Options

Design Module Name	Specifies the module name for the extracted finite state machines.

Format	*Summary* returns an FSM list with columns for the module name, FSM name, and size. It concludes with the total number of FSMs reported. *Verbose* lists each FSM by name with expanded information.

Debugging

Debugging is the fourth phase of the Conformal Extended Checks session. It takes place in the Verify mode. For this phase, you can rely solely on the Conformal Extended Checks integrated debugging environment, or use the Test Vector Interface (TVI) to produce test vectors for use with a Verilog simulator.

Integrated Tools

To debug your design using the Conformal Extended Checks integrated debugging environment, use the `DIAGNOSE STATIC PROPERTY` and `REPORT GATE` commands described below. The `DIAGNOSE STATIC PROPERTY` command produces counterexamples of the false properties. You can then use the schematic viewer in the GUI mode to visualize errors. The `REPORT GATE` comand reports specified information about circuit structure.

Diagnosing False Properties

To diagnose false properties, use the `DIAGNOSE STATIC PROPERTY` command.

Note: Conformal automatically assigns ID numbers. They can differ from one version to another. Always use the full path in dofiles and any time you rerun a design with a different Conformal version.

Conformal Extended Checks generates a counterexample for each false property when you use this command. You can find the schematic viewer, which is available in the GUI mode, helpful in identifying probable causes of false properties. If the current session is in non-GUI mode, switch to the GUI mode using the `SET GUI` command.

Generating Gate Reporting

The `REPORT GATE` command is useful in debugging. This command reports the total circuit structure or a selected fan-in cone in a variety of ways; including, gate counts, fan-in/fan-out cones, statistics on a variety of objects, simulation value, and source code correspondence.

Note: Conformal automatically assigns ID numbers. They can differ from one version to another. Always use the full path in dofiles and any time you rerun a design with a different Conformal version.

You can use the `REPORT GATE` command's `-simvalue` option to report the results of the `DIAGNOSE STATIC PROPERTY` command. With this option, the command displays the counterexample valuation being diagnosed.

TVI

As an alternative to the Conformal Extended Checks integrated debugging environment, you can use the TVI feature to interface Conformal Extended Checks with a commercial Verilog simulator to debug each false property. The TVI feature generates a Verilog testbench file and a Verilog simulation script file for each Conformal Extended Checks `WRITE DIAGNOSIS DATA` command with the `-verilog` option. It is compatible with simulators that allow VCD waveform viewing and source-level debugging.

The general flow for this process is:

- Read libraries and designs
- Specify the location of the simulation library (optional)
- Specify the simulator you intend to use (optional)
- Diagnose a target property for counterexample vectors
- Convert the counterexample vectors into a Verilog testbench

To Specify Simulator Information

After you read in libraries and designs, you can specify the location of the assert.vlib simulation library and which simulator you intend to use. These are optional commands.

```
setenv OVL_LIB <simulation_library_directory>
setenv VPX_SIM <Verilog_simulator>
```

At this point, continue with the command flow to prepare the design for verification, select verification options, and perform the verification (as described earlier in this chapter).

Generating Counterexample Vectors

After you issue the PROVE command, you can use the DIAGNOSE STATIC PROPERTY command to generate counterexample vectors for targeted properties with the following command.

Note: Conformal automatically assigns ID numbers. They can differ from one version to another. Always use the full path in dofiles and any time you rerun a design with a different Conformal version.

Generating a Verilog Testbench

To convert counterexample vectors to a Verilog testbench, use the WRITE DIAGNOSIS DATA command. Then, use a commercial simulator to simulate the Verilog testbench file along with the Verilog design files. If you do not specify a filename for the WRITE DIAGNOSIS DATA command, Conformal Extended Checks assigns a default filename as follows:

```
Vpx_<root_modulename>_<property_name>.tv
```

Conformal Extended Checks also generates a Verilog-XL compatible simulation script. If you do not specify a filename, Conformal Extended Checks assigns a default filename as follows:

```
Vpx_<root_modulename>_<property_name>.tv.runvlg
```

Note: If you define both OVL_LIB and VPX_SIM, TVI automatically generates simulation script file and invokes Verilog simulation.

The Verilog testbench file and Verilog simulation script file are both in source form. This allows you to customize the files when necessary. The testbench file can require modification to address differences in timing delay models between the Conformal Extended Checks cycle-based models and Verilog event-based simulation models. Additionally, you can need to modify the script file to accommodate the particular simulator you are using.

See Appendix D, “TVI Testbench and Simulation Script Files” for a general outline of a TVI-generated testbench file and a description of a simulation script file.

Property Investigation

Dynamic constraints can also aid debugging. Dynamic constraints provide a way for you to explore and analyze your design using temporary assumptions. You can use the ADD DYNAMIC CONSTRAINT command to specify the constraints you want to apply, then use the EXPLORE command to explore a property within the dynamic constraint environment.

Adding Dynamic Constraints

The ADD DYNAMIC CONSTRAINTS command, used in conjunction with the EXPLORE command, helps explore properties further. This command gives you a time-efficient method of debugging that does not require you to exit the Verify system mode to apply constraints, and then rerun the comparison to see how the constraints affect the comparison results. However, if you do exit the Verify system mode and rerun comparison, dynamic constraints no longer have any effect.

You can use the REPORT DYNAMIC CONSTRAINT command to display all the constraints that you applied.

Exploring Properties

Once you have applied dynamic constraints using the ADD DYNAMIC CONSTRAINT command, you can explore either properties or CDC Functional checks within the dynamic constraint environment using the EXPLORE command. This command provides an efficient way of quickly obtaining explored results.

Concluding the Session

The following two commands provide closure for the Conformal Extended Checks session.

Displaying a Session Runtime Summary

The USAGE command displays the total CPU runtime and peak memory use during the current Conformal Extended Checks session.

Exiting Conformal

To conclude a Conformal Extended Checks session, use the EXIT command. This closes Conformal and returns to the operating system prompt. By default, Conformal *does not* automatically save GUI settings for future sessions. To save your preferred settings, use the GUI Exit window and select the applicable radio button. See Exiting the GUI and Software on page 62 for a list of supported settings.

Exit Status Codes

On exiting, Conformal Extended Checks 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 counterexample 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
```

7

Property Managers

Static Property Manager

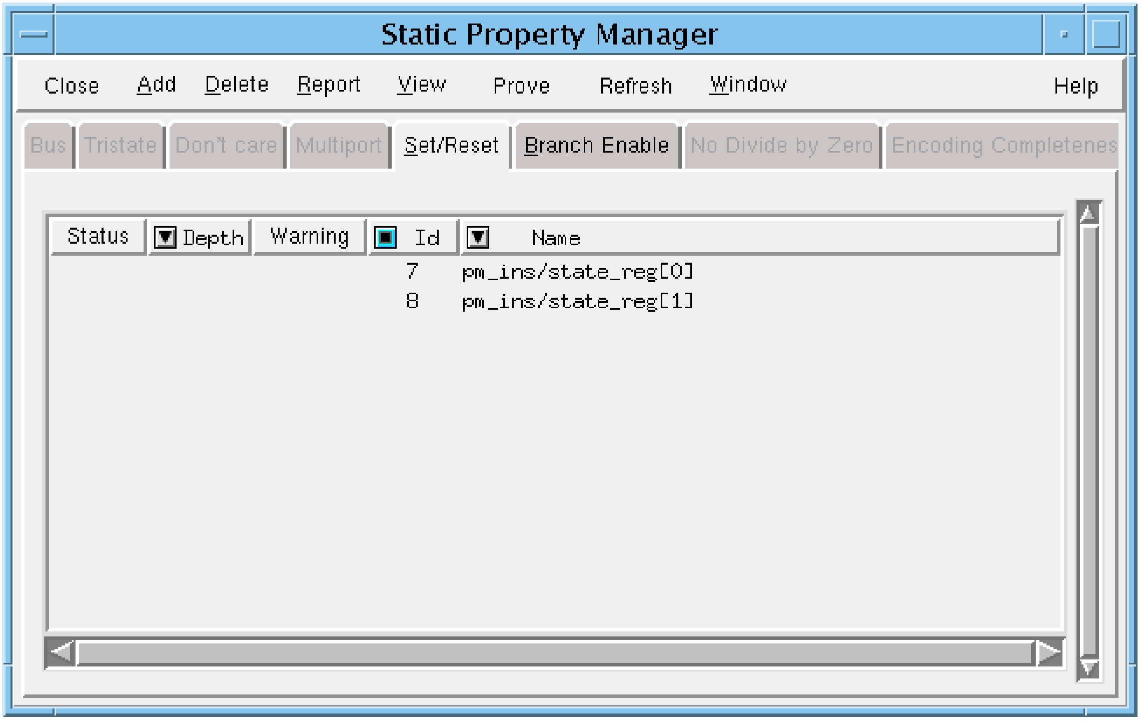
The Static Property Manager displays information about the automatically extracted predefined properties. Additionally, it is an integrated environment that allows you to select properties to include or exclude them from the proof, specify proof depth (`-combinational` or `-time_unit`), invoke the Conformal Extended Checks proof engine, report property data, and access integrated diagnosis tools (for example, the Waveform Viewer and the Flattened Schematics window).

For more information on automatically extracted predefined checks, refer to Chapter 8, “Predefined Checks.”

There are two ways to open the Static Property Manager from the Main window:

➤ Choose *Tools – Static Property*.

➤ Click the *Static Property Manager* toolbar widget.



Status	Depth	Warning	Id	Name
			7	pm_ins/state_reg[0]
			8	pm_ins/state_reg[1]

The Static Property Manager has an extensive drop-down menu system that groups commands in an easily accessible form. Access all of the important phases of the proof session from this main menu. The display area includes tabs and lists of properties that you can include in a proof.

The following table describes the different uses for each menu item.

Menu	Description
Add, *Delete*, and *Report*	The *Add*, *Delete*, and *Report* menus include the common objects listed in the following table. These are the objects on which the `ADD`, `DELETE`, and `REPORT` commands act.
Refresh	Use the *Refresh!* option to refresh the Static Property Manager. **Note:** When you change Pass/Fail Icon preferences, Conformal prompts you to refresh the window.

The following table explains the drop-down menu options available in the *Add*, *Delete*, and *Report* menus.

Table 7-1 Table of Common Objects

Object	Definition
All	Includes the six properties listed in the remainder of this table.
All Bus Property	Includes all extracted Bus Property instances.
All Tristate Property	Includes all extracted Tristate Property instances.
All Tristate Connected to PO	Includes all Tristate Property instances directly connected to Primary Outputs.
All Don’t Care Gate Property	Includes all Don’t Care Gate Property instances.
All Multiport Property	Includes all Multiport Property instances.
All Set/Reset Property	Includes all Set/Reset Property instances.
All Branch Enable	Includes all Branch Enable Property instances.

Working in the Display Area

The tab headings in the display area group the properties according to property categories (for example, *Bus* and *Tristate*). Click a tab to bring the desired category to the front. If the design does not include one or more of the predefined static properties, the applicable tab is grayed out. By default, the Static Property Manager displays all properties. See Static Property View Options on page 139 to display specific properties.

Each property category includes column headings for:

- *Proof Status*

  Status is indicated by an icon that identifies whether the instance has been added to the "Prove" list or (after you invoke the `PROVE` command) whether it passed or failed the proof. Table 7-2 on page 139 shows the status indicators.

  Note: Separate columns reflect properties relevant to the particular category. For example, the Bus sub-level headings include: Floating, Contention, and Multiple Driven.

- *Status*

  Status may be shown in a "Status" column or in a property-specific column. (For example; Floating, Contention, and Multiple Driven.)

- *Depth*

  Displays the proof depth integer, infinity, or combinational. (This applies to instances with a status other than "Pass".)

- *Warning*
- *Id*
- *Name*

For additional information about the relevant properties and conditions appearing in these sub-category columns, see Chapter 8, "Predefined Checks."

Each of the tabs displays a category name and not run or pass/fail icon. After proof, if a category includes both pass and fail instances, both the pass and fail icons are displayed on the tab. By default, Conformal displays a green-filled circle to indicate pass or a red circle to indicate fail. If you prefer, Conformal displays check marks (red X or green check).

Table 7-2 Status Indicators

✓ 🟢	Pass and Pass – Depth M
✗ 🔴	Fail and Fail – Depth M
🟡	Explored – Depth N
?	Not run
ⓘ	Ignored

Sorting Instances

▼■ Click the triangular sort icon in the *Depth*, *ID*, or *Name* column headings to sort instances numerically (by depth or ID number) or alphabetically by name. The square icon indicates the column that was sorted. (The complementary columns are rearranged accordingly.)

Changing the Name Format

To changing the name format, choose the *View – Name Format* and click *Full* (the default) or *Short*.

Sorting the Instances

To view instances alphabetically, by name, or by ID number, choose *View – Sort* and click *by Id* (the default) or *by Name*.

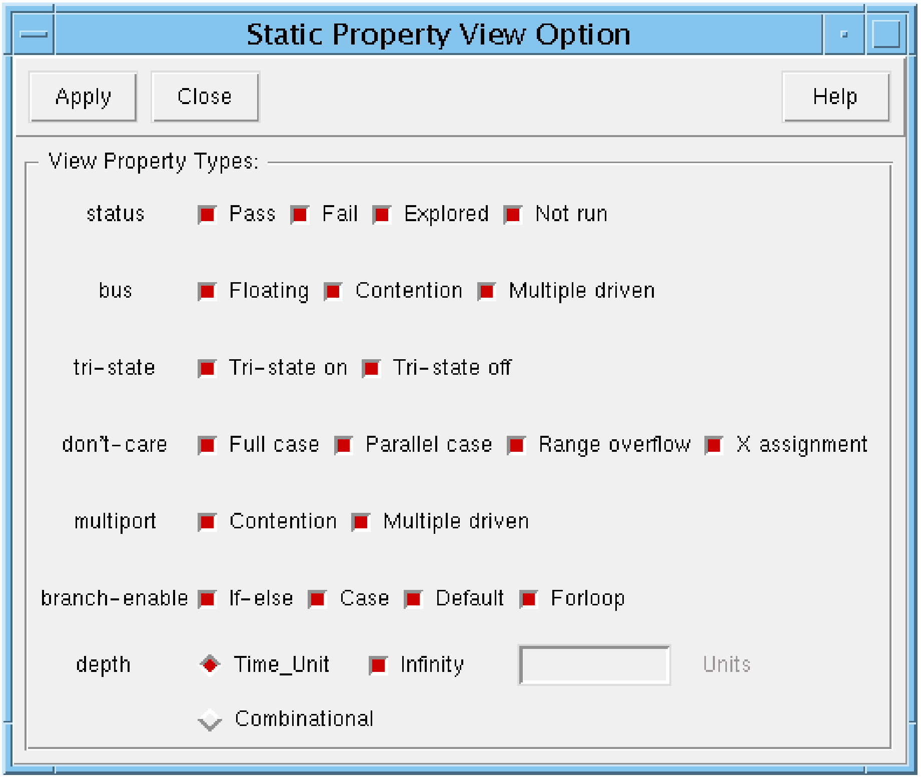
Static Property View Options

By default, the Static Property Manager displays all properties. Use the Static Property View Option form to deselect properties to make the view more manageable, which is especially useful when you encounter a large number of properties.

To open the Static Property View Option form, do the following:

1. Click the *View* button located on the Static Property Manager menu bar.

2. Click *Option* on the drop-down menu.



View Property Types:

status ☒ Pass ☒ Fail ☒ Explored ☒ Not run

bus ☒ Floating ☒ Contention ☒ Multiple driven

tri-state ☒ Tri-state on ☒ Tri-state off

don't-care ☒ Full case ☒ Parallel case ☒ Range overflow ☒ X assignment

multiport ☒ Contention ☒ Multiple driven

branch-enable ☒ If-else ☒ Case ☒ Default ☒ Forloop

depth ◆ Time_Unit ☒ Infinity Units

◇ Combinational

Static Property View Options Form Fields and Options

status	Specifies the status types for display.
bus	Specifies the bus properties for display.
tristate	Specifies the tristate properties for display.
don't-care	Specifies the don't care properties for display.
multiport	Specifies the multiport properties for display.
branch enable	Specifies the branch enable properties for display.

depth

Specifies the proof depth.

- For Sequential Properties, click *Time_Unit* and *Infinity*
- For Sequential Properties, de-select *Infinity* and enter an integer in the *Units* field.
- For Combinational Properties, select *Combinational*.

Working in the Static Property Manager

This section describes how to use the Static Property Manager:

- Selecting Properties to Prove on page 141
- Deleting Properties From the Prove List on page 142
- Executing a Proof on page 143
- Configuring and Executing Reports on page 144
- Diagnosing Properties on page 147
- Accessing Debugging Tools on page 148

Important

> When you add specified properties to the prove list, you cannot designate these same properties as “ignored” until you delete them from the prove list. Likewise, when you designate specified properties as ignored properties, you cannot add them to the prove list until you make them available. Use the *Delete* drop-down menu to delete ignored and static properties.

The correct sequence of commands is as follows:

1. *Add – Ignored Property* to exclude selected properties from the “Prove” list.
2. *Add – Static Property* to add properties to the “Prove” list.

Selecting Properties to Prove

This section describes how to add and ignore properties in the prove list.

1. Click *Add* on the menu bar.

2. Click *Static Property* or *Ignored Property*.
3. Click any of the common objects (listed in the Table 7-1 on page 137) to add the corresponding category of predefined properties to the “Prove” list or to ignore them.
4. Repeat steps 1-3 to add multiple common objects.

If you choose *Static Property* in step 2, Conformal Extended Checks displays the properties with a circled red question mark to represent their not run status.

If you choose *Ignored Property* in step 2, Conformal Extended Checks displays the properties with a circled (i) to represent their ignored status.

You can use the pop-up menu to add/ignore properties:

1. Click a property to select it.
2. Right-click and choose *Add* from the pop-up menu.
3. Click *Static Property* or *Ignored Property*.

Deleting Properties From the Prove List

In the Static Property Manager, use the *Delete* drop-down menu to reinstate ignored properties, or remove properties from the prove list.

To delete instances from the prove list or remove instances from the ignored status, complete the following steps:

1. Click *Delete* on the menu bar.
2. Click *Static Property* or *Ignored Property*.
3. Click any of the common objects (listed in Table 7-1 on page 137) to delete the corresponding category of predefined properties from the applicable list: Prove or Ignore.

   Conformal removes the corresponding status symbol on the selected properties. However, the property identification numbers and names remain.

You can also delete or reinstate properties using the pop-up menu:

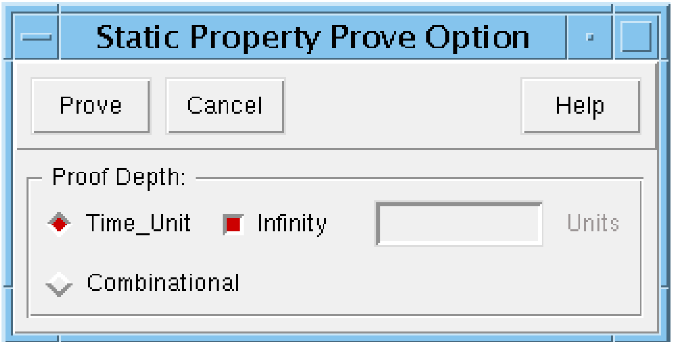
1. Click a property to select it.
2. Right-click and choose *Delete* from the pop-up menu.
3. Click *Static Property* or *Ignored Property*.

Executing a Proof

After you add properties using the *Add* drop-down menu or the pop-up menu, use the following procedure to access the Prove Option form. In this form, you will specify and execute a proof.

To execute a proof, do the following after you add properties to the prove list, as described in Selecting Properties to Prove on page 141:

1. Click the *Prove!* button located on the menu bar of the Static Property Manager to open the Static Property Prove Option form.



Proof Depth:
◆ Time_Unit ☒ Infinity [ ] Units
◇ Combinational

2. Select the *Proof Depth*:
   - ❑ For a sequential proof:
     - ❍ To prove properties with time unit infinity, accept the default settings.
     - ❍ To prove properties through a specified time unit, de-select *Infinity* and enter an integer in the *Units* field.
   - ❑ For a combinational proof, click *Combinational*.
3. Click *Prove*.

When Conformal Extended Checks completes the proof, the status of each instance on the prove list is identified with one of the following icons.

✓ 🟢	Pass and Pass – Depth M
✗ 🔴	Fail and Fail – Depth M
🟡	Explored – Depth N
?	Not run

ⓘ Ignored

Configuring and Executing Reports

After you add one or more properties, use the *Report* drop-down menu to configure and execute reports.

Chapter 8, “Predefined Checks” includes descriptions of available report formats. Refer to “Property Reports” on page 169.

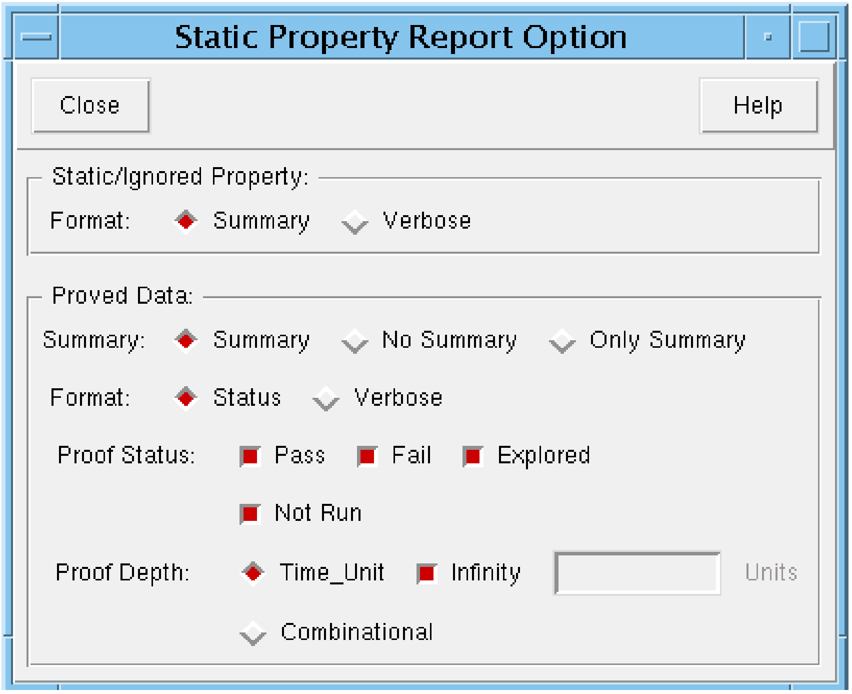
1. Click a property to select it.
2. Right-click and choose *Report* from the pop-up menu.
3. Click *Static Property*, *Ignored Property*, or *Proved Data* to specify the type of property Conformal Extended Checks should report.

Viewing a Static Property or Ignored Property Report

After you add properties (*Add – Static Property* or *Add – Ignored Property*) specify and view a report with the following procedure. The Static Property report displays data on all properties currently included on the “Prove” list. The Ignored Property report displays data on all properties that have been excluded from the “Prove” list.

Note: This window includes a *Static/Ignored Property* section and a *Proved Data* section. For the following procedure, use the *Static/Ignored Property* portion of the window.

1. Choose *Report – Option* to open the Static Property Report Option form.



Static/Ignored Property:

Format: ● Summary ○ Verbose

Proved Data:

Summary: ● Summary ○ No Summary ○ Only Summary

Format: ● Status ○ Verbose

Proof Status: ☒ Pass ☒ Fail ☒ Explored ☒ Not Run

Proof Depth: ● Time_Unit ☒ Infinity ▭ Units ○ Combinational

2. Click one of the *Format* radio buttons.
3. Click *Close*.
4. Execute the report as follows:
   a. From the *Report* drop-down menu, click *Static Property* or *Ignored Property*.
   b. Choose any of the common objects listed in Table 7-1 on page 137.

      The specified report appears in the Transcript window of the main Conformal Extended Checks window.

You can also use the pop-up menu to view reports:

1. Click a property to select it.
2. Right-click and choose *Report* from the pop-up menu.
3. Choose *Static Property* or *Ignored Property.*

Viewing a Proved Data Report

Note that this window includes a *Static/Ignored Property* section and a *Proved Data* section. For the following procedure, use the *Proved Data* portion of the window.

After you add properties to the “Prove” list (*Add – Static Property*) and run the `PROVE` command, use the following procedure to specify and view Proved Data reports.

➤ Choose *Report – Option* to open the Report Option form.

Perform the following steps in the Proved Data section.

1. Specify a *Summary* preference.
2. Specify a *Format* type:
3. Click one or more check boxes to specify a *Proof Status* selection.

   By default, Conformal Extended Checks includes all properties in the Proved Data reports. Click a check box to de-select it and exclude the specified status type from the report. (For example, de-select the Pass check box if you do not want to include properties that have a Pass proof status.)
4. To specify *Proof Depth* options, do one of the following:
   - ❑ For Sequential Properties, do one of the following:
     - ❍ Click *Time_Unit* and *Infinity*
     - ❍ De-select *Infinity* and enter an integer in the *Units* field.
   - ❑ For Combinational Properties, select *Combinational*.
5. Click *Close*.
6. Execute the report as follows:

   Choose *Report – Proved Data* and choose any of the common objects listed in Table 7-1 on page 137.

   The specified report appears in the Transcript window of the main Conformal Extended Checks window.

You can also use the pop-up menu to view reports:

1. Click a property to select it.
2. Right-click and choose *Report* from the pop-up menu.
3. Choose *Proved Data.*

Diagnosing Properties

This section describes how you can diagnose properties in the Static Property Manager.

Choosing a Diagnosis Tool

Use the *View – Diagnose Cone* choice to specify which tool opens when you diagnose a property*.*

1. Choose *View – Diagnose Cone*.
2. Choose *Multi-Timeframe Schematics* or *Schematics* to designate a diagnosis tool:

   This controls the diagnosis tool that Conformal uses when you diagnose a property using the *Diagnose* pop-up menu command.

In addition to accessing the Schematics (Multi-Timeframe) window through the *Diagnose* option on the pop-up menu, you can use the following procedure to open the Gate manager and show the fan-in cone of the selected instance.

The valuation of the counterexample is annotated on the circuit.

1. Click a property to select it.
2. Right-click and choose *Multi-Timeframe Schematics* from the pop-up menu.

Diagnosing an Instance

Use the following procedure to start active diagnosis, where Conformal generates a counterexample for an instance you select.

In the Schematics (Multi-Timeframe) window, the valuation of the counterexample is annotated on the fan-in cone of the predefined property instance.

1. Click a false property to select it.
2. Right-click and choose *Diagnose* from the pop-up menu.
3. Click the applicable choice.

   Choices are category-specific to pinpoint your diagnosis point. Your choice may be *Static Property* (as with Don’t Care property instances); but, some property types include additional choices. For example, when you choose *Diagnose* on a bus property instance, click *Floating*, *Contention*, or *Multiple Driven* to specify your diagnosis point.

Conformal Extended Checks opens the Waveform Display (when applicable); and the Flattened Schematics.

4. Continue the diagnosis by opening the Source Code Manager, as explained in “Opening the Source Code Manager” on page 148.

Important

To learn about the debugging process for the Branch Enable property, refer to “Branch Enable” on page 175.

Saving Diagnosis Data

After diagnosing an instance (see “Diagnosing an Instance” on page 147), you can save the diagnosis data to a file.

1. Click a property to select it.
2. Right-click and choose *Write Diagnosis Data* from the pop-up menu to open the Write Diagnosis Data form.
3. Choose the *VCD* or *Verilog File Type* radio buttons:
4. Enter a name in the *File Name* field, or use the *Browse* button to locate the appropriate file through the Save Diagnosis Data File window.
5. Click *OK*.

Accessing Debugging Tools

This section discusses how you can open the various debugging tools from the Static Property Manager.

Opening the Source Code Manager

The Source Code Manager automatically opens when you double-click an instance in the Flattened Schematics window.

As you diagnose instances with the integrated debugging tools, you may move from one diagnostic tool to another. The following procedure opens the Source Code Manager in a context-dependent mode.

1. Click a property in the Static Property, Waveform Display, Flattened Schematics (Multi-Timeframe), or Flattened Schematics window to select it.

2. Right-click and choose *Source Code* from the pop-up menu.

   The Source Code Manager opens, scrolled to the relevant line of code. Conformal highlights the appropriate location.

Opening the Waveform Viewer

For sequential properties with a depth greater than 0, the Waveform Display opens automatically when you choose *Diagnose* from the pop-up menu.

In addition to accessing the Waveform Viewer through the *Diagnose* option on the pop-up menu, you may use the following procedure.

1. Click a property to select it.
2. Right-click and choose *Debug Waveform* from the pop-up menu.

   The Waveform Viewer opens, showing the waveform of the counterexample for the diagnosed selected instance.

Opening The Schematic Viewer

Use the following procedure to open the Schematic Viewer window and show a selected instance.

1. Click a property to select it.
2. Right-click and choose *Schematics* from the pop-up menu.

Opening the ED/Seq Exploration Manager

Use the following procedure to open the ED/Seq Exploration Manager, which you can use to sequentially explore designs.

1. Click a property to select it.
2. Right-click and choose *ED/Seq Explorer* from the pop-up menu.

FSM Manager

The FSM Manager displays finite state machine checks information. Additionally, it is an integrated environment that lets you:

- Select all or specified categories (Transition, Reachability, or Deadlock) to include or exclude them from the proof process
- Select specified properties to include or exclude them from the proof process
- Report FSM property data
- Invoke the Conformal Extended Checks proof engine
- Access integrated diagnosis tools (for example, the Waveform Viewer and the Schematic Viewer)

To access the FSM Manager, click on *Tools* on the menu bar in the main Conformal – Verify window and choose *FSM Property*.

For more information on the FSM Manager features and operation, refer to Chapter 10, “FSM Checks.”

8

Predefined Checks

The Predefined Checks (also referred to as Static Property Checks) are a large collection of checks that validate the design intent for automatically extracted properties (for example, no bus contention). Informally, a property is a general behavioral attribute (that is, collection of logical and timing relationships) used to characterize a design. A property can either be asserted, which means it must be proved, or assumed, which means it is a constraint.

This chapter includes the following sections:

Definitions for Predefined Checks are explained in the following sections with simple demonstration examples.

Property Proofs

Generally, you will add properties for proof, and then use the default `PROVE` command. *The default settings are appropriate for most cases.*

The default setting for proof depth is `-time_unit infinity`. This specifies that a property must hold true for all time units after initialization.

Note: Undefined clocks connected to the property must be resolved before you enter the `PROVE` command.

Note: Conformal Extended Checks does not support strength specification for gate instances.

To Select All Properties and Prove

Use the example commands shown below.

```
VERIFY> add static property -all
VERIFY> prove
```

On rare occasions, you may decide to override the `-time_unit infinity` default setting. Conformal Extended Checks supports the following depth options for those *rare* cases:

- Proof Depth: Combinational
  The `PROVE` command's `-combinational` option specifies that the property must hold true regardless of the values of flip-flops and latches. This type of proof is useful in designs with scan-in storage elements since one cannot assume a specific value for the registers on a scan-in path.

- Proof Depth: Time Unit
  The `-time_unit unit` option specifies that a property must hold true for at least `unit` time units after initialization. Note that undefined clocks connected to the property must be resolved prior to proving a property with `-time_unit unit`.

  A special form of this command occurs when `unit` is set to 0. The `-time_unit 0` option is not equivalent to `-combinational`. The former assumes the values defined during the initialization for the flip-flops and latches are used during the verification. In addition, any latches that are enabled after initialization are considered to be transparent during a proof with `-time_unit 0`.

Tip

For properties that return a proof status of ED (refer to "Proof Status" on page 153), re-run the proof with greater effort. Conformal Extended Checks remembers the proof results from previous proofs, and when it resumes the proof, it proves only the unproven properties and moves forward from the depth that was previously proven.

Example:

```
add static property -all // 10 properties
prove // 5 passed, 5 ED depth 12

set prove effort high
prove
// The 2nd proof begins where the previous proof
// ended, that is, the 5 EDs at depth 12.

// The result could now be 5 EDs at depth 20.
// Any subsequent proofs will start on the
// unproven properties from the depth already
// proven.
```

Proof Status

After Conformal Extended Checks proves the added properties, it returns a summary of the outcome of the proof. The following table lists the possible status indicators with short definitions.

Proof Status	Definition
Pass	Indicates properties that hold true from initialization to infinity.
Pass – Depth M	Indicates properties that passed to the extent of the target you specified. For example, If you specify `-time_unit 100` for a particular proof, and Conformal Extended Checks returns a "Pass" and notes a depth of 100, this means that the specified properties passed up to and including time 100. Conformal Extended Checks did not attempt to prove the specified property beyond time 100.

Proof Status	Definition
Fail	Indicates properties that did not hold true at some point during the proof. Conformal Extended Checks indicates the time, counting from zero (immediately after initialization), at which it proved to be a failed property.
Fail – Depth M	Indicates properties that failed within the target you specified, which is noted on the report as the depth. Example: `prove -time_unit 5` This indicates that the user requested a check for this property through time 5. Conformal Extended Checks reports Fail with a depth of 5, which indicates that up through depth 5, the desired behavior was not present. This status applies to FSM Transition, FSM Reachability, Branch Enable, and Tristate properties.
Explored – Depth N	Indicates inconclusive results. Conformal Extended Checks explored this property up to and including the depth noted on the report. This property requires further investigation and debugging. For example, if you specify `-time_unit 100` for a particular proof, and Conformal Extended Checks is able to prove a property up to and including time 80, Conformal Extended Checks reports the applicable property as “Explored” and notes the depth as 80, which indicates that the status is unresolved and this property requires further investigation. Conformal Extended Checks did not explore the specified property beyond time 80.
Not Run (NR)	Indicates properties that were not processed.

Property Reports

This section describes property reports, which include the following:

- Static Property and Ignored Property Reports on page 155
- Proved Data Reports on page 155

For more information on the commands that produce these reports, refer to the *Conformal Extended Checks Reference Manual.*

Note: The gate ID numbers specified in property reports are assigned by Conformal. These identifiers may be especially useful to you during diagnosis. For example, use gate IDs in commands such as `REPORT GATE` and in the schematic viewer. However, since Conformal automatically assigns ID numbers, they can differ from one version to another.

Static Property and Ignored Property Reports

Static Property and Ignored Property reports are available in the following two formats:

- Summary—lists the number of properties by type. This is the default format for Static Property and Ignored Property reports.
- Verbose—contains expanded information. It is organized by property type and it lists each property ID number and name. It concludes with a summary of the reported properties.

Proved Data Reports

The Proved Data report is available in two formats: Status and Verbose. By default, Proved Data reports conclude with a summary of proof results for all proved properties. However, you may specify an alternative to this format (see Status and Verbose Report Summary Options on page 156).

Use the following legend to decipher Proved Data reports:

```
Legend:
F-Case     full-case
P-Case     parallel-case
ED         explored depth
NR         not run (not processed)
```

Status

The Status format is the default. It features the following:

- Groups properties by type (for example, bus)
- Itemizes properties (gates) by identification number and name
- Lists the proof status of each property
- Lists the depth at which the proof status was generated, when applicable (see Property Reports on page 155).

Verbose

The Verbose format lists each property and its proof status. The report includes the property ID number and gate name, and lists its characteristics and counterexample (witness).

Status and Verbose Report Summary Options

Specify your summary preferences for the Proved Data Status and Verbose reports with one of the following options:

- `-summary`—organized by property types, this lists the number of properties of each type by status and the total of each property type. It ends with the total number of properties with a breakdown according to proof status. The default for Status and Verbose reports is `-summary`.
- `-nosummary`—omits the summary table from the specified report.
- `-onlysummary`—returns the summary table with no additional information.

  Note: Regardless of whether you specify a Status or Verbose format, the software only generates a summary table when you use this option.

Bus Check

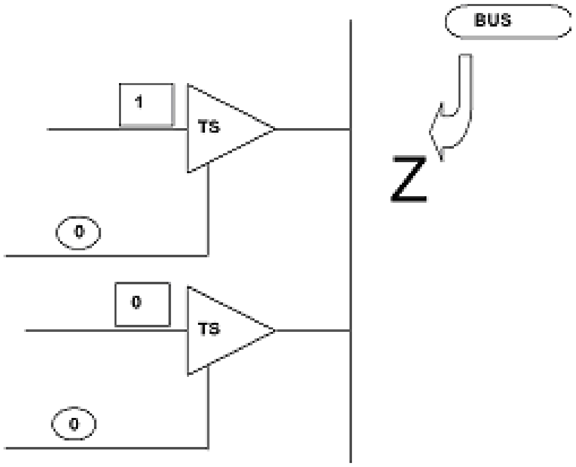
Conformal Extended Checks identifies any net that is driven by more than one tristate as a bus. This section discusses the following three bus checks, which are verified by Conformal Extended Checks:

- Bus Check: Floating on page 157
- Bus Check: Multiple Driven on page 157

- Bus Check: Contention on page 158

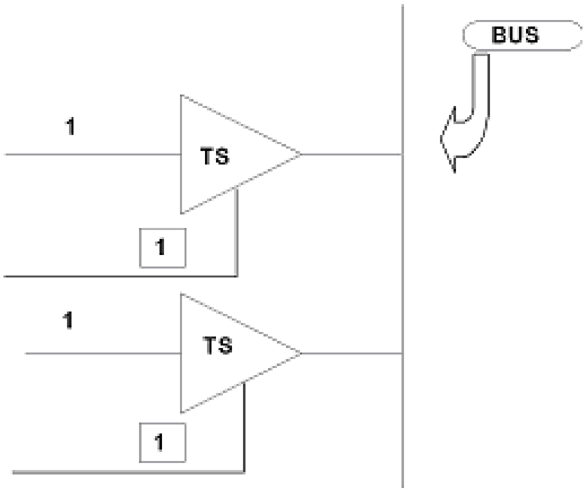
Bus Check: Floating

Bus Check: Floating is a predefined check that determines if the design includes buses that are not actively driven by any tristate. This predefined check determines only *internal* buses. For buses directly connected to PI/PO/PIO, this predefined check is not verified, because the bus could be driven by other circuits outside of this design.



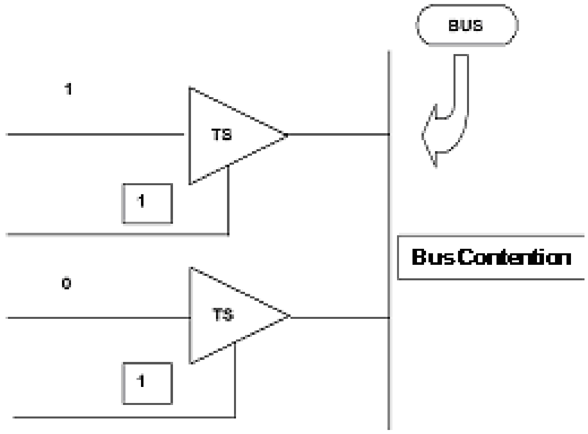
Bus Check: Multiple Driven

The Bus Check: Multiple Driven is a predefined check that determines if the design includes more than one activated tristate driving the bus.



Bus Check: Contention

The Bus Check: Contention is a predefined check that determines if the design includes more than one activated tristate connected to the bus with conflicting data. This is a safety property, since bus contention results in short circuit.



Black Boxes Connected to Buses

Conformal Extended Checks treats a black box module pin connected to a bus as an input pin by default if the direction of this pin cannot be determined at compile time (for example, when using the `SET UNDEFINED CELL` command). To check the implications of undefined modules on buses, set the default behavior of pins connected to buses to PIOs by issuing the command

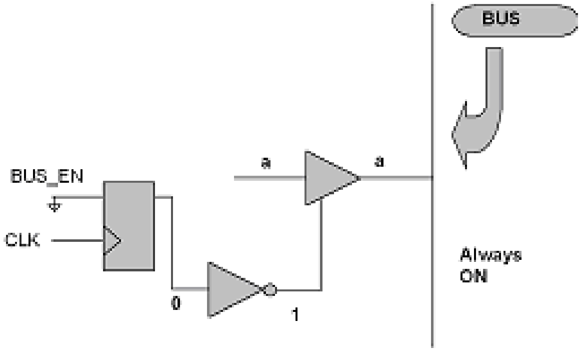
```
set undefined cell black_box -noauto_assign
```

Tristate Check

The Tristate Check detects if the tristates in a design are functioning as tri-states; that is, if both the on state and off state of the tristate are exercised in the circuit. The Tristate Check looks for the following two conditions:

- Tristate On:
  This predefined check determines if the tristate is always on.

- Tristate Off:
  This predefined check determines if the tristate is always off.



Although many times the design intent of the tristates connected to the output assumes the tristate property functions correctly, it is prudent to distinguish tristates connected to primary outputs. This is especially important because the internal tristates are often tristates between hierarchical modules, so they do not *have* to exercise all possible states in the specific design instance. The Conformal Extended Checks `ADD STATIC PROPERTY` command includes convenient options to add either of the following:

- Only those tristates connected to the primary output
- All tristates

Refer to the *Conformal Extended Checks Reference Manual* for information abut the `ADD STATIC PROPERTY` command and its options.

Don't Care Check

Don’t care conditions in a circuit design are often exercised by the synthesis tool for optimization. The actual occurrence of the don’t care condition in a working circuit will thus result in unpredictable behavior and may often lead to a design error. This section discusses the following don’t care conditions:

- Full Case on page 160
- Parallel Case on page 160
- Range Overflow on page 160
- X-Assignment on page 161

Full Case

The Don’t Care Full Case Check determines if the Synopsys full case directive covers all possible cases. For example, the following Verilog code segment has a false full case property when `reg[1:0]` has a value `2’b10`, unless `reg[1:0]` is never assigned the value `2’b10` during the proof:

```
case (reg[1:0]) // synopsys full_case
      2’b00: ..
    2’b01: ..
  2’b11: ..
endcase
```

Parallel Case

The Don’t Care Parallel Case Check determines if the Synopsys parallel case directive has cases that are not mutually exclusive. For example, the following Verilog code segments have false parallel case properties when `a` equals `2’b11` and `b` equals `2’b11` at the same time.

```
module encoder (y, z, a, b);
output y, z;
input [1:0] a, b;
reg y, z;

always @(a or b) begin
{y, z} = 2'b00;
casez ({a, b}) file://synopsys parallel_case
4'b11??: z = 1'b1;
4'b??11: y = 1'b1;
endcase
end
endmodule
```

Range Overflow

The Don’t Care Range Overflow Check determines if the index of the multiple bits register is out of the range of the defined index. For example, the following Verilog code segment has a false range overflow property:

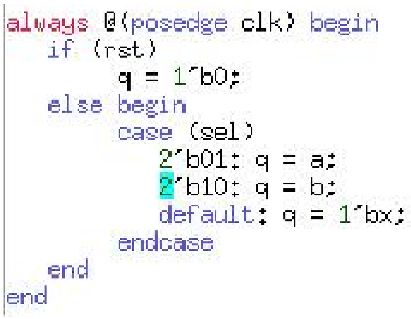
```
reg [3:0] y;
…
ind=4;
y[ind]=0;
```

X-Assignment

Because simulation tools view X-assignment as an unknown, while synthesis tools view X-assignment as a don't care, using the X-assignment can produce mismatches between pre- and post-synthesis designs. That is, designs that are *logically* consistent might not be *semantically* consistent. This could result in a *functional bug* that cannot be found during RTL simulation.

Given ample resources and unlimited time, this bug might be exposed during gate-level simulation; however, since gate-level simulation is not a viable alternative, the X-assignment Check determines if the X-assignment path is actually taken. Thus, Conformal Extended Checks looks for X-assignments that otherwise, could be responsible for missed bugs. (Note that the schematic viewer highlights the X-assignment path during diagnosis.)

In the following example, a violation occurs when the X-assignment path in the "default" statement is taken. There are two possible unintended behavior cases: when `sel` is either `2'b00` or `2'b11.`



```
always @(posedge clk) begin
   if (rst)
        q = 1'b0;
   else begin
        case (sel)
           2'b01: q = a;
           2'b10: q = b;
           default: q = 1'bx;
        endcase
   end
end
```

Branch Enable

Conformal Extended Checks identifies all conditional case-default and if-else statements in a design and verifies whether each conditional branch can be enabled. Conditional statements that cannot be enabled lead to sections of HDL code that are not accessible. Proof results are as follows:

- Pass
  The conditional statement can be enabled or is permanently enabled.
- Fail
  The conditional statement cannot be enabled or cannot be reached because of other conditional statements.

Important

> Diagnosis for Branch Enable properties differs from other properties. Since a fail means that something did *not* happen (that is, Conformal could find no vector pattern that would enable the conditional branch), Conformal does not display any vectors or waveforms.
>
> Additionally, Conformal returns an Explored proof status when a branch cannot be fully verified using the specified effort level. This means that for the level of effort that was expended, an enabling condition could not be conclusively proven. This indicates that a PASS could not be conclusively proven.

For specific information about the diagnosis process, see Debugging on page 166.

Verilog Examples

During the proof process, Conformal considers whether the following conditional statements can be enabled. When a violation occurs, Conformal identifies the property by its location in the code. For example, if Conformal indicates that a failed property's name is `line:2`, the violation is located on line 2 of the code. The following examples are written in Verilog. Also see VHDL Examples on page 164.

If-else

In the following example, the `if` branch will not be enabled if `sel` is never active (see line 2). And the `else` branch will not be enabled if `sel` is *always* active (see line 4).

```
1. if (sel)
2.      out <= a; //dead if sel never active
3. else
4.      out <= b; //dead if sel always active
```

The following example shows an `if` expression on line 12, an `else if` expression on line 15, and the default `else` statement on line 18. Conformal checks whether the `if` or `else if` statements can ever be true. A violation occurs if neither can ever be true.

In the following example, the conditional `else if (trgb)` (see line 15) will never be triggered because of logical masking by `trga`. This example also shows the complexity of debugging. Conformal would report that `else if (trgb)` is not reachable, because logic dependencies elsewhere in the design prevent the branch. The cause is not obvious without further diagnosis.

```
 1. always @ (posedge clk) begin
 2.      if (reset) begin
 3.                 trga <= 1'b0;
 4.                 trgb <= 1'b0;
 5.      end
 6.      else begin
 7.                 trga <= (i || j);
 8.                 trgb <= (i && j);
 9.      end
10. end
11. ...
12. if (trga) begin
13.      ...;
14.      end
15. else if (trgb) begin // dead branch
16.      ...;
17.      end
18. else begin
19.      ...;
20.      end
```

Case, Casex, and Casez

In the following example, a violation occurs when:

- On line 2: If `sel` is never `01`.

- On line 3: If `sel` is always `01` or never `10`.

- On line 4: If `sel` is always `01` or `10`. This is the desired (good) behavior. See “Special Considerations” on page 165.

```
1. case (sel)
2.     2’b01 : out = a; //dead if sel never 01
3.     2’b10 : out = b; //dead if sel always 01 or never 10
4.     default : out = 1’bx; //dead if sel always 01 or 10
5. endcase
```

VHDL Examples

During the proof process, Conformal considers whether the following conditional statements can be enabled. When a violation occurs, Conformal identifies the property by its location in the code. For example, if Conformal indicates that a failed property’s name is `line:2`, the violation is located on line 2 of the code. These examples are written in VHDL. Also see Verilog Examples on page 162.

If-elsif-else-end

In this example, the `if` and `elsif` conditions are spelled out in the code and the `else` is default. A violation occurs when:

- On line 2: `GETA` is never true

- On line 4: `GETA` is always true or `GETB` is never true

- On line 6: `GETA` or `GETB` is always true

```
1. if GETA then
2.    OUT <= A; -- dead if GETA never true
3. elsif GETB then
4.    OUT <= B; -- dead if GETA always true or GETB never true
5. else
6.    OUT <= “0000”; -- dead if GETA or GETB always true
7. end if;
```

Case

In this example, the `when others` structure is the default, and the other expressions are spelled out in the code. A violation occurs when:

- On line 2: If `SEL` is never `01`.

- On line 3: If `SEL` is always `01` or never `10`.

- On line 4: If `SEL` is always `01` or `10`. This is the desired (good) behavior. See Special Considerations on page 165.

```
1. case SEL is
2.      when “01” => OUT := A; -- dead if SEL never 01
3.      when “10” => OUT := B; -- dead if SEL always 01 or
                                  never 10
4.      when others => OUT := 0; -- dead if SEL always 01
                                    or 10
5. end case
```

Special Considerations

This section alerts you to situations that require special considerations with regard to the Branch Enable Check. For example, conditional branches that are intentionally unreachable (such as guarded X-assignment) fail the Branch Enable Check, yet in those cases failing is the desired (good) behavior.

X-Assignment Handling

An X-assignment that propagates will fail the Don’t Care X-Assignment Check (see X-Assignment on page 161). And assignments, including unreachable X-assignments that are guarded by conditional branches, fail the Branch Enable Check. These checks complement each other, and although there is redundancy, no special treatment is needed. Since in most cases, a guarded X-assignment *should* be unreachable, the fail status for the Branch Enable Check confirms the desirable behavior.

Z (Tristate) Handling

The Conformal Tristate Check (see Tristate Check on page 158) finds all tristate drivers in a design and verifies whether they can become stuck on or stuck off. Additionally, the Branch

Enable Check finds unreachable branches, including the branches controlling these tristate drivers. Thus, unreachable guarded Z cases fail the Branch Enable Check, which is redundant with the Tri-State Off Check.

Note: In the case of conditional operators (for example, `assign out = sel ? a : 1'bz ;`) the tristate Check shows if the tristate is stuck on or off. For these cases, the Branch Enable Check is not needed.

Constraint Handling

It is not uncommon to use constraints to provide accurate proof results and missing constraints may affect reachability. However, fixed value constraints that are used to disable logic cause *undesirable* Branch Enable Check fails. A fixed value constraint disables sections of logic (for example, reset or test logic).

Debugging

As noted in the introduction to "Branch Enable" on page 161, the debugging for Branch Enable properties is unique. For Branch Enable, a *fail* means that Conformal could find no vector pattern that would enable the conditional branch. For Case and If-Else Branch Enable Checks, Conformal displays the conditional statement in the source code and the cone of logic *to* that condition. For Case Default Branch Checks, Conformal displays the conditional statement in the source code and the cone of logic to that condition, *if possible*.

Recall that the general GUI procedures for diagnosing properties (such as Bus and Don't Care) is as follows: select a failed property, choose Diagnose from the pop-up menu, and when the Waveform Display and Flattened Schematics open, examine the counter-example.

However, for Branch Enable properties, there are no vectors or waveforms to diagnose. Hence, you must use a different debugging procedure for these properties. To debug a failed Branch Enable property, the appropriate procedure is as follows:

1. Select a failed property.
2. Right-click and choose *Diagnose* from the pop-up menu to open the Source Code Manager.
3. Examine the source code for the origin of Branch Enable failures.
4. Right-click and choose *Schematics* or *Multi-Timeframe Schematics* to determine the origin of failures.

Multiport Check

Conformal Extended Checks identifies all multiport latches in a design, and proves whether multiport latches can be loaded with conflicting values. Multiport latches are the special latches with multiple inputs controlled by enable signals. They are often implemented in the circuit as special elements in the cell library. This check is verified only if there is a multiport latch cell in the design.

Set-Reset Check

The Set-Reset check detects whether the SET and RESET pins of flip-flops or latches are never asserted at the same time. The violation of this predefined check may result in undefined behavior in simulations and in real circuits. It should be noted that during other proofs, Conformal Extended Checks assumes a RESET dominant behavior (that is, when both SET and RESET are activated, Conformal Extended Checks assumes the RESET activated result).

FSM Checks

For specific information about the FSM Transition, Reachability, and Deadlock properties, see Chapter 10, "FSM Checks."

No-Divide-By-Zero Check

This assertion check ensures that divisors in a division operator are never zero.

Encoding-Completeness Check

By default, Conformal reads binary encoding when building a Finite State Machine (FSM). If your gate netlist uses different encoding (for example, one-hot), you must create an FSM encoding file that specifies the correct encoding. The Encoding-Completeness predefined check ensures that any reachable state is listed under `fromstates` in the FSM encoding file.

For example, the following illustrates the contents of an encoding file. The left-hand statements between `.begin` and `.end` specify the `fromstates`.

```
.name MYENC
.fromstates u1/current_state_reg[1] u1/current_state_reg[0]
.tostates   u1/current_state_reg[3] \
u1/current_state_reg[2] \
u1/current_state_reg[1] \
u1/current_state_reg[0]
.begin
00 0001
01 0010
11 0100
.end
```

See “`READ FSM ENCODING`” in the *Conformal Extended Checks User Guide* for more information on reading in FSM encoding files.

9

Clock Domain Crossing Checks

Overview

Conformal Extended Checks provides Clock Domain Crossing (CDC) checks. A clock domain is defined as any clock signals driving D-latches or D flip-flops. Conformal provides the following types of CDC checks:

- *Structural* Clock Domain Crossing checks (CDC) validate an RTL or gate design's clocking schemes. This check is structural in nature (that is, it is not a static timing check).
- *Functional* CDC checks complement and build on Structural CDC checks. The Functional CDC checks identify asynchronous domains and check for data stability as it moves from one clock domain to another.
- *Set/Reset* CDC checks ensure that asynchronous set and reset pins are consistent with the clock domain.

Functional CDC checks build on the Structural CDC checks, and as such, you must meet the following requirements for successful validation:

- Execute Structural CDC checks before you add Functional CDC checks. Although it is not necessary to perform Structural CDC checks on the entire design, you must perform them on those portions where you would like to perform the Functional CDC checks.
- After adding Functional CDC checks, you cannot add nor delete Structural CDC checks until you delete all Functional CDC checks.

Structural CDC Checks

Conformal Extended Checks 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 Extended Checks 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 Extended Checks supports considerable user control over defining what constitutes a clock domain. Conformal Extended Checks uses the definitions to determine the clock domains for each state element.

By default, Conformal Extended Checks evaluates domains from register to register. To specify any other domain crossing combinations for Structural CDC checks, use the `ADD DATA ASSOCIATION` command. After you initiate CDC checks, use Conformal Extended Checks reporting options and integrated debugging tools to diagnose clock domain crossing issues.

Functional CDC Checks

Functional CDC checks validate the REG-REG clock domain category for paths with unstable data. Use Functional CDC checks to verify that the data from a source clock domain remains stable until the destination clock domain captures it. A violation indicates instability, which results in ignored or lost data. The following is a list of Functional CDC checks:

- Source Data Stability
- Destination Data Stability
- Multiplexer Enable Stability
- Single Bit Change (gray encoded)

Set/Reset CDC Checks

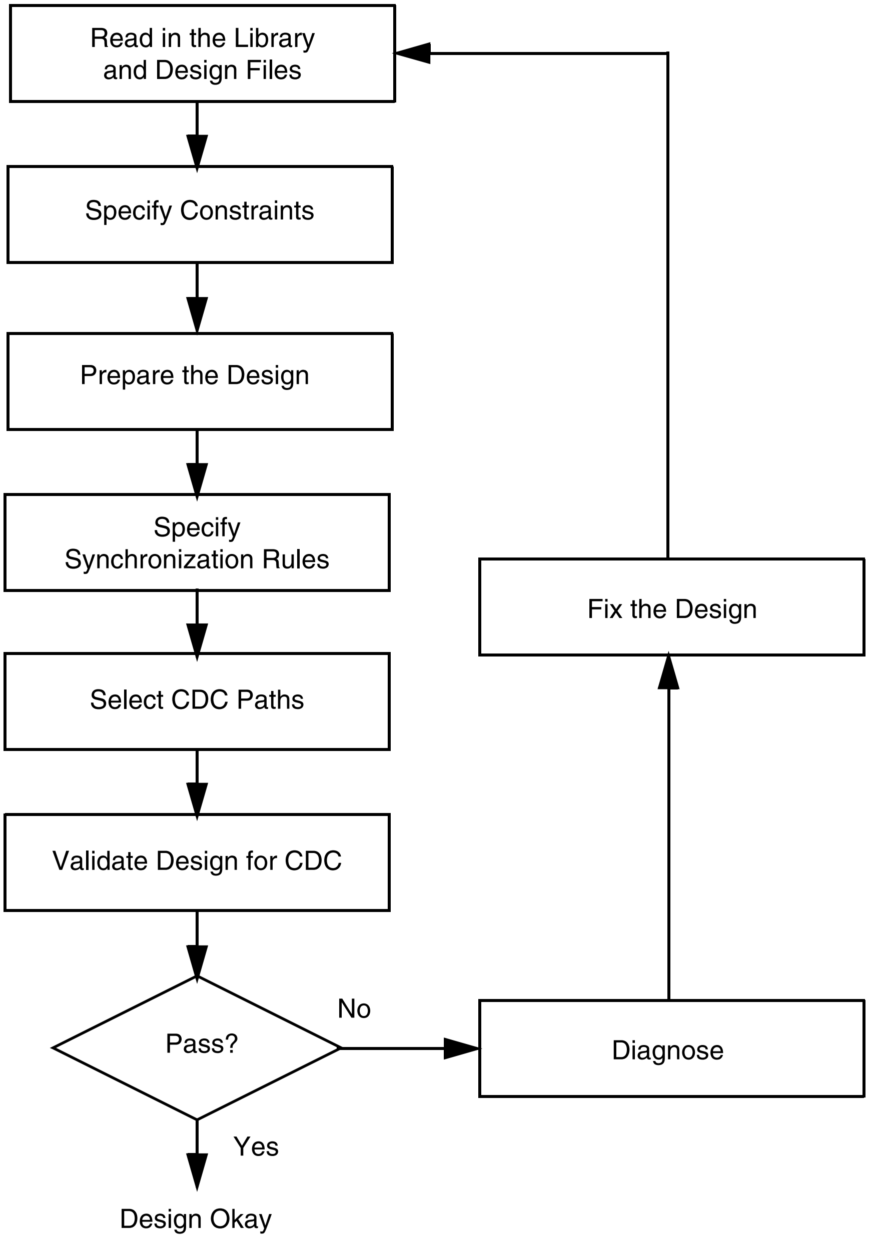
Set/Reset CDC checks ensure that asynchronous set and reset pins are consistent with the clock domain. You cannot add synchronization rules to set/reset CDC checks.

CDC Checks – General Flow

This section describes the general process flow for specifying clocking schemes, checking for violations, and diagnosing CDC issues.

The process flow presented in this section applies to both the command and GUI modes. However, for additional guidance specific to the GUI mode, also see Clock Domain Crossing Manager on page 200. For additional information on commands and their options, see the

Conformal Extended Checks Reference Manual. Also, note that Conformal Extended Checks supports wildcards for string matching.



This flow illustrates the following steps for handling clocking schemes and CDC issues:

1. Read in the library and design files.
2. Place constraints on the design.

3. Prepare the design by specifying clocking schemes that reflect design intent.

   See Preparing the Design on page 175.

4. In Verify mode, specify synchronization rules for CDC checks.

   See Specifying Synchronization Rules on page 182.

5. Select CDC paths.

   See Selecting CDC Paths on page 188.

6. Validate CDC paths.

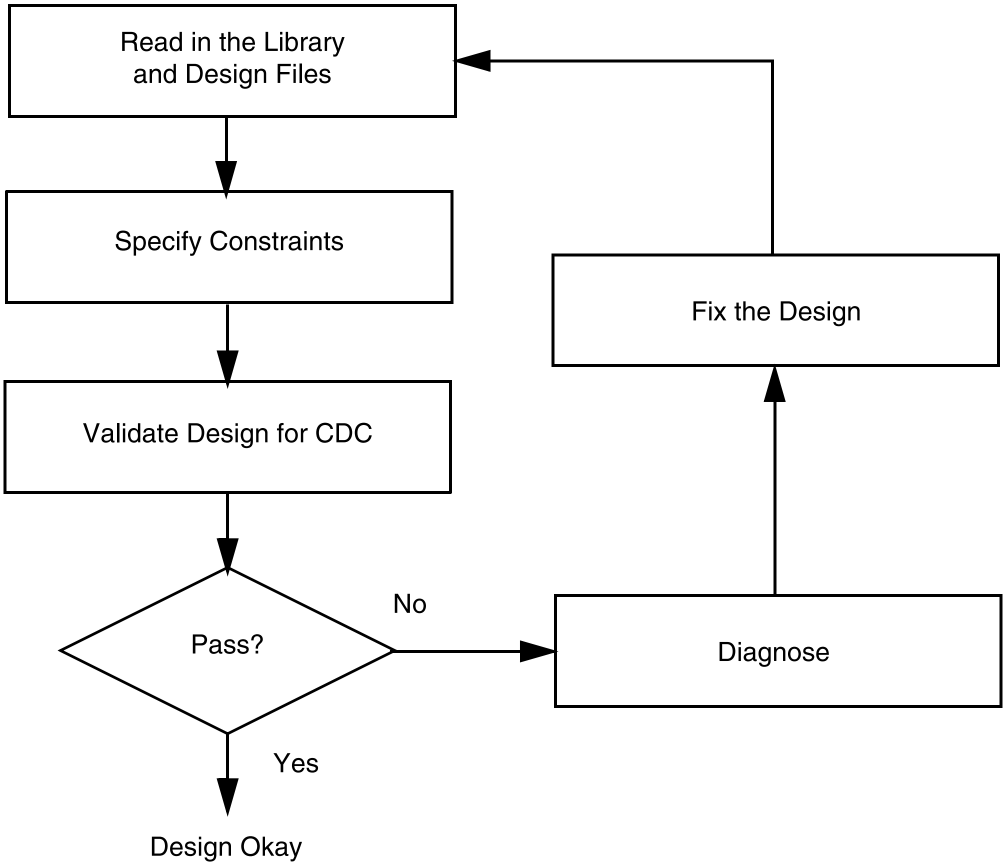
   See Validating CDC Paths on page 189.

7. Diagnose and debug CDC violations.

   See Diagnosis and Debugging on page 190.

CDC Checks – RTL Structural Flow

This section describes the general flow for RTL structural checks.



The RTL structural flow illustrates the following steps:

1. Read in the library and design files.
2. Place constraints on the design.
3. Select the clocking schemes and validate crossings from register to register (using the `VALIDATE -RTL` command.
4. Diagnose and debug CDC violations.

   See Diagnosis and Debugging on page 190.

Sample Dofile

```
read library ...
read design ...
add pin constraint
validate -rtl
```

Preparing the Design

Begin with the typical Conformal Extended Checks session flow (that is, read in the library and design, add constraints, and so forth). Continue with the CDC flow by specifying clocking schemes that accurately reflect design intent. The following commands allow considerable control over clocking scheme specification.

Defining CDC Clocks

In the Setup system mode, use the `ADD CLOCK` command to define clocks. See Appendix E, "Defining Clocks" to learn how to derive waveforms from clock frequencies.

Specifying Domain Assignment Rules

Clock domain rules help Conformal Extended Checks determine the domain of a signal during clock domain analysis. To specify domain assignment rules, use the `SET_CLOCK_DOMAIN_RULE` command.

A clock domain rule consists of two parts: forward domain propagation and backward domain extraction.

Forward Domain Propagation

Forward domain propagation has two parts:

- Propagation Through Sequential Elements
- Propagation Through Combinational Logic Gates.

Propagation Through Sequential Elements

If an output of a sequential element is used as a clock, then the clock domain for that output is dependent on the input of the clock and the options used with the `SET_CLOCK_DOMAIN_`

RULE command. Refer to the following table to better understand how Conformal Extended Checks interprets the options `-unknown`, `-derived`, and `-noderived`.

The following table shows the resulting clock domain type for the output of the sequential element based on the clock domain input and the option used with the `SET CLOCK_DOMAIN RULE` command. The resulting clock domain will be: unknown, known, new, or error. The following examples illustrate how to read the information presented in this table. The general procedure is to find the “Results of Propagation Through Sequential Elements” at the intersection of an “Input Clock Domain” row and a “SET CLOCK_DOMAIN RULE Option” column.

- Example one:

  If the input clock domain is known and the `SET CLOCK_DOMAIN RULE` option is `-unknown`, the output clock domain is “unknown”.

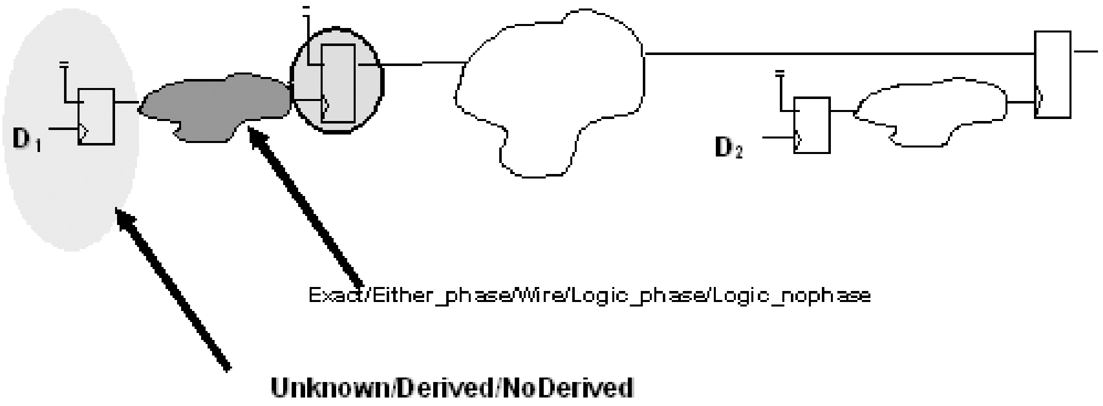
- Example two:

  If the input clock domain is known and the `SET CLOCK_DOMAIN RULE` option is `-derived`, then the output clock domain is the same as the input clock domain (known).

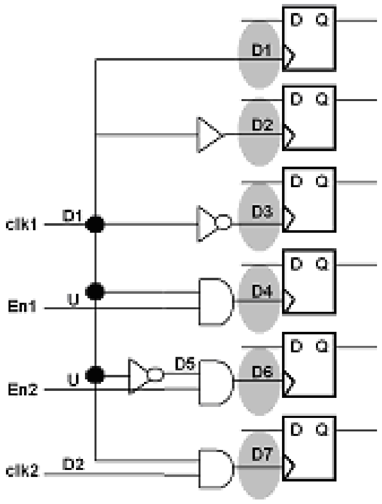
	SET CLOCK_DOMAIN RULE Options:		
Input Clock Domain	**-unknown**	**-derived**	**-noderived**
Known	unknown	known	known (new)
Unknown	unknown	unknown	unknown
Error	unknown	error	error

Propagation Through Combinational Logic Gates.

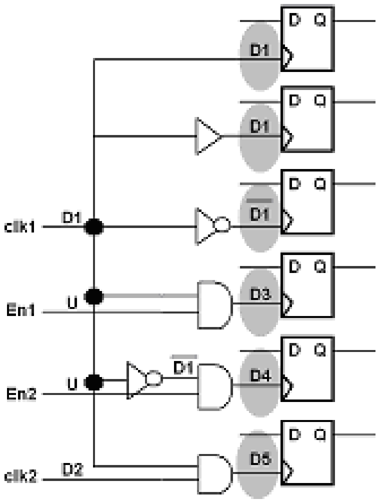
The following figures show how Conformal Extended Checks interprets the options of the `SET CLOCK_DOMAIN RULE` command.


D1
D2
Exact/Either_phase/Wire/Logic_phase/Logic_nophase
Unknown/Derived/NoDerived

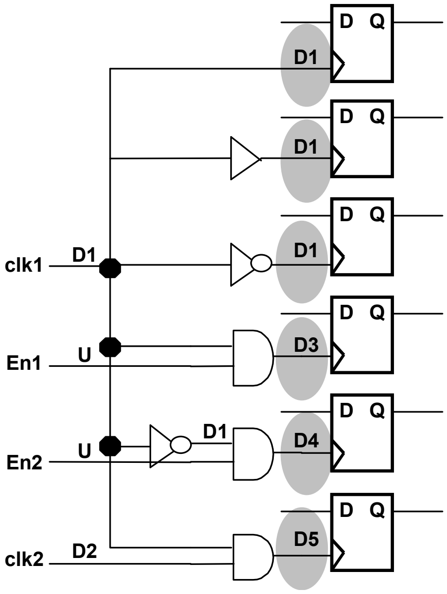
The following figure depicts the Conformal Extended Checks clock domain analysis related to the `-wire` option.



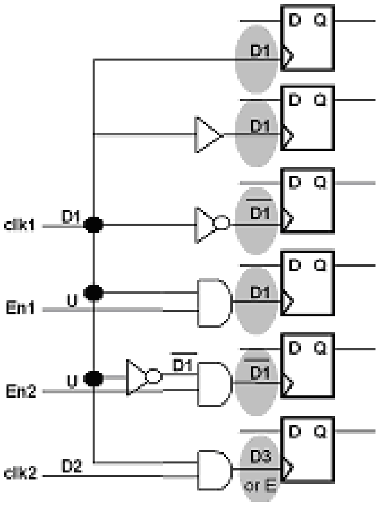
The following figure depicts the Conformal Extended Checks clock domain analysis related to the `-exact_phase` option.



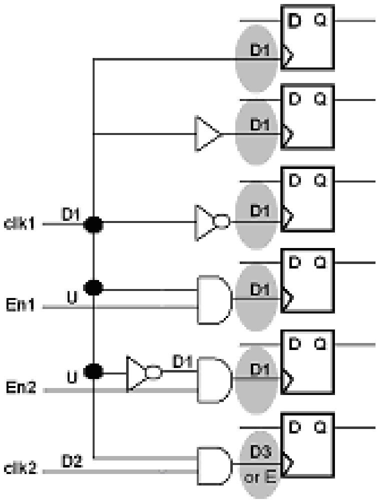
The following figure depicts the Conformal Extended Checks clock domain analysis related to the `-either_phase` option.



The following figure depicts the Conformal Extended Checks clock domain analysis related to the `-logic_phase` option.



The following figure depicts the Conformal Extended Checks clock domain analysis related to the `-logic_nophase` option.



Backward Domain Extraction

Use the `SET CLOCK_DOMAIN RULE` command's `-noextract` or `-extract` options to specify how you want to handle domain extraction.

- `-noextract`
  Tells Conformal Extended Checks not to automatically extract domains.

- `-extract buffer`
  Automatically extracts domains starting from the unknown domain clock ports of D flip-flops and D-latches, and then backward trace through buffers and inverters. In this case, Conformal Extended Checks will recognize the gate that stops the backward extraction as an extracted clock domain.
- `-extract wire`
  Automatically extracts domains such that all unknown domain clock ports of D flip-flops and D-latches are treated as extracted clock domains.

Specifying CDC Options

Use the `SET CDC OPTION` command to specify characteristics of CDC domains.

Associating Clock Domains

By default, even though you can define multiple clocks with the same frequency using the `ADD CLOCK` command, Conformal Extended Checks considers them asynchronous to each other. In multi-asynchronous clock designs, synchronization failures can occur when signals are generated in one domain and sampled in another domain.

You can use the `ADD CLOCK ASSOCIATION` command to associate a group of synchronous clocks. Once the clocks are associated, signals belonging to these associated clocks are considered to be in a single clock domain. It is also possible to add clocks to an existing association by issuing the command again with the same representative domain (the first argument).

Note: In the schematic viewer, when you move the cursor over a primary input, output, or blackbox pin that you have associated with a clock domain, the popup info box displays the domain name/ID.

Important

Some input pins can not be clock inputs, but they can be sampled by D flip-flops, D-latches, and output pins. Therefore, input pins can be considered separate domains so that Conformal Extended Checks can identify potential synchronization failures from input pins. Similarly, output pins, which can be sampled by another chip not in the design, can also be considered as separate clocks so that potential synchronization failures at the output pins can be identified. Assign domains to primary inputs, primary outputs, and blackboxes with the `ADD DATA ASSOCIATION` command, not with the `ADD CLOCK ASSOCIATION` command.

Adding Data Associations

Use the `ADD DATA ASSOCIATION` command to:

- Assign specified data signals to an existing clock domain
- Declare a new domain and assign data signals
- Assign data signals to the universal domain

Note: A signal belonging to the universal domain is considered to be synchronous to any clock domain. Hence, Conformal Extended Checks does not mark any violations for signal crossings involving the universal domain.

Since a domain crossing path is defined with respect to a pair of registers; by default, Conformal Extended Checks does not check paths that begin or end with primary input, primary output, blackbox input, or blackbox output ports. However, when you use this command to create associations, Conformal Extended Checks can determine if there is a CDC Check violation for a path that either starts from or ends at the associated pins. This also applies if a path starts *and* ends at the associated pins.

In addition to assigning data signals to a specified `clock_domain`, use the `ADD DATA ASSOCIATION` command with the following options to create a new domain or associate data signals with the universal domain.

- `-new`
  This option creates one new domain. All the specified pins belong to that domain.
- `-universal pathname*…`
  This option specifies that one or more paths belong to the universal clock domain. A signal that belongs to the universal clock domain is considered synchronous to any clock domain. (Conformal Extended Checks does not mark any violations for signal crossings involving the universal domain.)

Creating New Domains for Generated Clocks

Use the `ADD GENERATED CLOCK` command to create a new clock domain for the specified ID or path of an internally generated clock.

Defining Domain Priority Levels

The `SET CLOCK_DOMAIN PRIORITY` command enables Conformal Extended Checks to resolve clock domain assignment for an output of a logic gate in cases where the two inputs have different known domains. Alternatively, this command resets all clock domain priorities.

Specify a prioritized list of defined CDC clocks either on the command line or by reading through a specified file:

- To specify priority by file, list one clock per line in descending priority order
- To specify priority by command line, list arguments in descending priority order

If you do not specify all defined CDC clocks in the command, all unspecified CDC clocks have the same priority as the lowest specified CDC clock.

The `SET CLOCK_DOMAIN PROPERTY` command works in conjunction with `SET CLOCK_DOMAIN RULE -conflict_error`.

Recalculating the Clock Domain Information

When it is necessary to manually update clock domain information, use the `PROPAGATE CLOCK DOMAIN` command.

Specifying Synchronization Rules

In this step, you specify synchronization rules for CDC checks. You can define your own synchronization rules, or you can use the predefined synchronization rules that Conformal provides.

- Specifying User-Defined Synchronization Rules on page 182
- Using Predefined Synchronization Rules on page 183

When you use the predefined synchronization rules alone—without user-defined synchronization rules—Conformal automatically enters the *Categorization* flow when you issue the `VALIDATE` command. In this flow, Conformal attempts to categorize the CDC paths in your design into one of the predefined synchronization rules. However, if you have user-defined synchronization rules, Conformal enters the normal *Validation* flow. The flow you are in affects the available options for the `REPORT VALIDATED DATA` command and the tabs available in the Clock Domain Crossing Manager.

Specifying User-Defined Synchronization Rules

To define the synchronizers used in specified portions of the design, use the `ADD SYNCHRONIZATION RULE` command. CDC checks support three types of synchronizers: synchronizer modules, registers, and multiplexers.

Note: You cannot specify synchronization rules for Set/Reset CDC checks.

See Synchronization Rule on page 218 for more information on defining synchronizers from the Clock Domain Crossing Manager.

Using Predefined Synchronization Rules

As an alternative to adding your own synchronization rules, Conformal offers predefined synchronization rules.

When you use the predefined synchronization rules alone—without user-defined synchronization rules—Conformal automatically enters the *Categorization* flow when you issue the `VALIDATE` command. In this flow, Conformal classifies the CDC paths into one of the predefined synchronization rules by validating the CDC path against the predefined synchronization rules, in linear order. In other words, Conformal categorizes a CDC path into the first predefined synchronization rule that it satisfies. Once a path satisfies a rule, Conformal ignores all subsequent rules.

Displaying Predefined Synchronization Rules

To display a list of predefined synchronization rules, use the `REPORT PREDEFINED_SYNCH_RULE` command shown below:

The following shows a portion of a sample output report.

```
==============================================================================
=                    Predefined Synchronization Rule List                    =
==============================================================================
 Order: 1
 Name: PRE_DFF_1
 Type: DFF
 CDC path: wire single_destination
 Sync chain:
   Length (min, max): 2, Inf
   Structural: wire single_chain
   Allowing DLAT: no

 Order: 2
 Name: PRE_MUX0_1
 Type: Mux
 CDC path: wire single_destination
 DFF before MUX (min, max): 0, 0
 Data hold check: yes
```

Where:

- `Order`—specifies the category of the CDC paths into predefined synchronization rules, in linear order. This number controls the order for which each rule is checked.
- `Type`—denotes the type of synchronizer (DFF or MUX)
- `CDC path`—specifies what is allowed for the CDC path.

    - ❑ `logic`, `wire`, or `buffer`—specifies the allowed CDC path logic.
    - ❑ `single_destination`—does not allow fanouts from the CDC path.
    - ❑ `multiple_destination`—allows fanouts from the CDC path.

    Note: For more information on CDC path, see the description for the `-CDC_PATH` option of the `ADD SYNCHRONIZATION RULE` command.

- `Sync chain` (for DFF types only)—specifies what is allowed between elements of the DFF synchronizer.
    - ❑ `Length (min, max)`—specifies the minimum and maximum number of flip-flops in the DFF synchronizer.
    - ❑ Structural—Specifies the structural properties of the DFF synchronization chain.
        - ❍ `logic`, `wire`, or `buffer`—specifies the allowed logic.
        - ❍ `single_chain`—does not allow fanouts from the DFF synchronization chain.
        - ❍ `multiple_chain`—allows fanouts from the DFF synchronization chain.
        - ❍ `Allowing DLAT`—specifies whether DLATs are allowed.
- `DFF before MUX (min, max)` (for MUX types only)—specifies the minimum and maximum number of DFFs before the MUX.
- `Data hold check` (for MUX types only)—specifies whether the data hold conditions must exist (that is, specifies whether feedback from the destination register to the MUX data input is required).

For information on how to display the predefined synchronization rules in the Clock Domain Crossing Manager, see Working in the Predefined Tab on page 235 for information

Risk Levels

By default, predefined synchronization rules are arranged by their risk level. For example, `PRE_DFF_1`, which has the default `Order` of `1`, is considered the safest synchronization scheme. While, `PRE_MUX1_4` with the default `Order` of `12`, is considered one of the riskiest synchronization schemes.

This section explains the risk level and conditions associated with each predefined synchronization rule.

The following table uses the following terms to describe the risk-level for a synchronization scheme that falls under a particular rule:

- *Unconditional pass*—Safe under most conditions.
- *Conditional pass*—Safe, but certain conditions can cause problems.
- *Potential fail*—Risky under most conditions.

Also, the allowed type of CDC path logic can affect a synchronization scheme's risk level.

- *Wire*—Considered safe.
- *Buffers*—Considered risky because buffer chains can cause unwanted delays.
- *Logic*—Considered risky because there is no limitation to a chain's logic. For example, as the number of gates increases, so does the delay.
- *Multiple_destination*—Considered risky because fanouts can cause delays in arrival time.

Order	Synchronization Rule	Allowed Logic	Description
1	`PRE_DFF_1`	`wire`	Unconditional pass
2	`PRE_MUX0_1`	`wire`	Unconditional pass
3	`PRE_MUX1_1`	`wire`	Unconditional pass
4	`PRE_DFF_2`	`buffer`	Conditional pass—This synchronization rule allows buffers. The buffer depth can cause a significant delay and adversely impact crossing. You can also encounter problems when the predefined convergence check fails.
5	`PRE_MUX0_2`	`buffer`	Conditional pass—Considered risky because buffer chains can cause unwanted delays.
6	`PRE_MUX1_2`	`buffer`	Conditional pass—Considered risky because buffer chains can cause unwanted delays.

Order	Synchronization Rule	Allowed Logic	Description
7	`PRE_DFF_3`	`buffer, Multiple_ destination`	Potential fail—Considered risky because timing differences can cause glitches in both the synchronization chain and the CDC path. You can also encounter problems when the predefined convergence check fails.
8	`PRE_MUX0_3`	`buffer, Multiple_ destination`	Potential fail—Considered risky because timing differences can cause glitches in the CDC path.
9	`PRE_MUX1_3`	`buffer, Multiple_ destination`	Potential fail—Considered risky because timing differences can cause glitches in the CDC path.
10	`PRE_DFF_4`	`logic, Multiple_ destination`	Potential fail—Considered risky because glitches can occur due to logic in the synchronization chain and CDC path, and the interaction between reconverging data and control.
11	`PRE_MUX0_4`	`logic, Multiple_ destination`	Potential fail—Considered risky because logic in the CDC path can cause glitches.
12	`PRE_MUX1_4`	`logic, Multiple_ destination`	Potential fail—Considered risky because logic in the CDC path can cause glitches.

Customizing Predefined Synchronization Rules

Use the `SET PREDEFINED SYNCH_RULE` command to customize the predefined synchronization rules.

See Predefined Synchronization Rule on page 215 for information on how to customize predefined synchronization rules from the Clock Domain Crossing Manager.

Handling Multiplexers

Use the `-mux` option with the `SET PREDEFINED SYNCH_RULE` or `ADD SYNCHRONIZATION RULE` command to identify data paths in a design. The meanings of some typical parameter values are:

- `-mux 0 0`:
  The multiplexer synchronizer is located in the CDC path.
- `-mux 1 1`:
  The multiplexer synchronizer is located in the sync chain, that is, after 1 destination D flip-flop.
- `-mux 0 1`:
  The multiplexer synchronizers are located in the CDC path *and/or* the sync chain.

Handling sync_modules

Use the `-sync_module` option with the `SET PREDEFINED SYNCH_RULE` or `ADD SYNCHRONIZATION RULE` command to identify a module that is used as a synchronizer. This option specifies that Conformal Extended Checks will first check to ensure that a domain crossing path's destination D flip-flop is *inside* the specified module. If this condition is satisfied, Conformal Extended Checks then performs a check on the `cdc_path` to ensure that logic *outside* the module boundary complies with all other parameters of the synchronization rule.

Example:

```
add sync rule r1 -sync_module m1 -cdc_path buffer
```

For this example rule, the CDC path check passes if the CDC path satisfies both of the following conditions:

1. The destination D flip-flop is inside module `m1`.
2. Only the buffer or inverter is in the `cdc_path` outside the module boundary.

Requirements

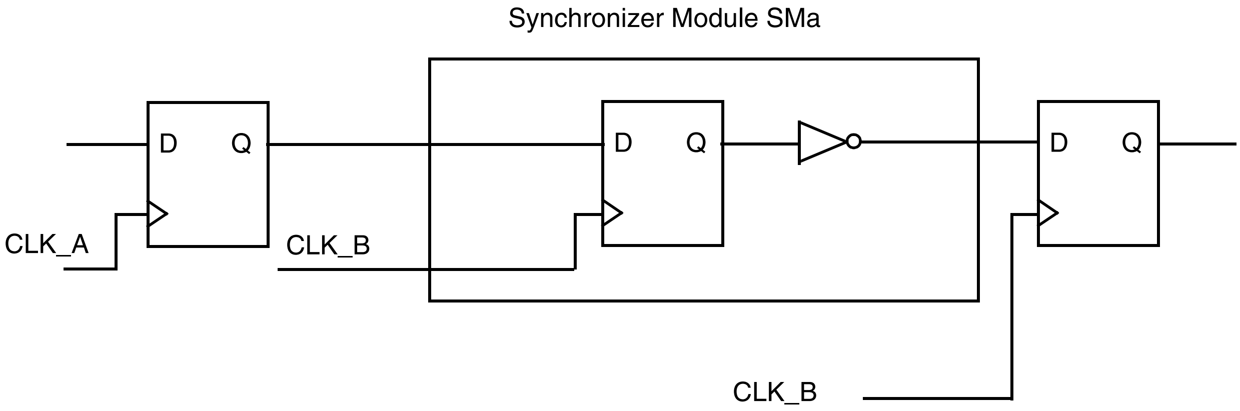
The following lists the synchronizer module scheme requirements:

- Cannot be a black box module
- Must have at least one DFF in the synchronizer module
- All DFFs in this module are driven by the same domain

- The clock for the synchronizer module should be the same as the destination clock domain

The following command specifies the rule SR0 for the synchronizer module SMa to paths with the source clock domain CLK_A and the specified destination domain CLK_B:

```
add synchronization rule SR0 -sync_module SMa -source CLK_A -dest CLK_B
```



Selecting CDC Paths

To select portions of the design for the different kinds of CDC checks, use the ADD CDC CHECK command. Conformal Extended Checks extracts the clock domains from the circuit and performs CDC checks for each synchronization rule. By default, domain crossing checks are turned *off*.

Functional CDC checks build on the Structural CDC checks, and as such, you must meet the following requirements for successful CDC checks validation:

- Execute Structural CDC checks before you add Functional CDC checks. Although it is not necessary to perform Structural CDC checks on the entire design, you must perform them on those portions where you would like to perform the Functional CDC checks.
- Once you add Functional CDC checks, you can neither add nor delete Structural CDC checks until you delete all Functional CDC checks.

By default, Conformal Extended Checks evaluates only register to register paths. To specify other paths for Structural CDC checks, such as primary inputs, primary outputs, and blackboxes, use the ADD DATA ASSOCIATION command (see Adding Data Associations on page 181).

Run the ADD CDC CHECK command in the Verify mode after you prepare the design, which has several options to fine-tune the parameters for checking clock domain crossings. By default, the software adds Structural CDC checks. After completion, you can add Functional CDC checks, with the `-functional` option. To focus on the domains in specified modules, use the `-module` option followed by one or more module names.

- Structural CDC checks—performs numerous Structural CDC checks including `cdc_path_logic_type_check`, `cdc_path_destination_check`, and `sync_chain_dff_number_check`. These checks are based on your synchronization specifications.

  See Reporting Validated Data on page 190 for a list of Structural CDC checks.

- Functional CDC checks—performs the Functional CDC checks on a path. This path must first pass the Structural CDC checks. Thus, you cannot select Functional CDC checks until the software performs Structural CDC checks on the specified portion of the design.

  Note: Single-bit change checks can still be added to the Functional checks, even though the Structural check fails.

  Functional CDC checks include stability checks for source data, destination data, multiplexer enable, and single bit change. By default, if you do not specify the type of Functional CDC check, the software runs all Functional CDC checks.

  For a description of the types of Functional CDC checks, see the `ADD CDC CHECK` command.

  Note: In the case where a source clock period is greater than the destination clock period, the Functional check could pass and the Structural check could fail. The active clock edges of the source and destination clocks can still be too close to satisfy the setup time requirement. This is not able to be verified with a 0-delay model Functional check, but normally synchronizers will be placed and it will be checked structurally. The crossing can be ignored if you have determined that close active clock edges should not be a problem.

- Set/Reset CDC checks—Set/Reset checks ensure that asynchronous set and reset pins of the flip-flops are consistent with the clock domain.

Validating CDC Paths

After specifying clocking schemes and defining which signal paths will be checked, run the `VALIDATE` command to validate whether the structure of the design follows the clocking schemes. Conformal Extended Checks automatically summarizes the results of the checks. For Structural CDC checks, Conformal Extended Checks reports the status (pass and fail) for each domain crossing. For Functional CDC checks, Conformal Extended Checks reports the status (pass, fail, and explored depth) for each domain crossing. The Functional CDC checks report further specifies the results according to stability checks for source data, destination data, and so forth.

Diagnosis and Debugging

During diagnosis, you determine the following:

- Is the violation a result of a specification issue?
- Does an actual design problem exist? (This requires a design modification to insert synchronizers.)

Use the following Conformal integrated tools to diagnose CDC Check issues:

- Clock Domain Crossing Manager
- Schematic Viewer
- Source Code Manager
- Waveform Display (for Functional CDC checks)

Reporting Validated Data

In preparation for diagnosis, use the `REPORT VALIDATED DATA` command or work in the GUI Clock Domain Crossing Manager to report on domains and link to the integrated debugging tools to examine the paths that are involved in violations. (See Display Area on page 203 for related debugging processes.)

The available options for `REPORT VALIDATED DATA` depend on your current flow. When you use the predefined synchronization rules alone (`SET PREDEFINED SYNCH_RULE` command), Conformal automatically enters the *categorization* flow when you issue the `VALIDATE` command. However, if you have user-defined synchronization rules (`ADD SYNCHRONIZATION RULE` command), Conformal enters the normal *validation* flow.

Reviewing Validated Data

The `REPORT VALIDATED DATA` command provides information about validation results and includes debugging information.

Categorization Flow

While in the categorization flow, `REPORT VALIDATED DATA` can output a summarized report similar to the following example.

```
================================================================================
=               Clock Domain Crossing Categorization (Summary)                 =
================================================================================
Category          No. of CDC Paths
---------------   ----------------
PRE_DFF_1                        1
PRE_MUX0_1                       2
PRE_MUX1_1                       1
PRE_DFF_2                        2
PRE_MUX0_2                       2
PRE_MUX1_2                       1
PRE_DFF_3                        1
PRE_MUX0_3                       1
PRE_MUX1_3                       1
PRE_DFF_4                        2
PRE_MUX0_4                       1
PRE_MUX1_4                       1
MIXED                            1
FAIL                             3
NOT_RUN                          0
---------------   ----------------
Total                           20
==================================================================================
```

This report displays the number of CDC paths that fall into each predefined synchronization rule category. Conformal classifies the CDC paths into one of the predefined synchronization rules by validating the CDC path against the predefined synchronization rules, in linear order. In other words, Conformal categorizes a CDC path into the first predefined synchronization rule that it satisfies. Once a path satisfies a rule, Conformal ignores all subsequent rules.

A CDC path can be classified under the following:

- `MIXED`—When the vector has individual bits that are classified into different categories for word-level reporting.
- `FAIL`—When the path does not satisfy any predefined synchronization rule.
- `NOT_RUN`—When you do not run *Validate* after adding a CDC check, or if you stop a validation before it completes.

To report each CDC path, its source/destination clock domains, and the predefined synchronized rule that it is categorized under, use `REPORT VALIDATED DATA -verbose`. A verbose report would look similar to the following example.

```
================================================================================
=                      CDC Categorization Data (Verbose)                       =
================================================================================
source_clock_domain: ck1
destination_clock_domain: ck2
from_instance: F11/out_reg
to_instance: F121/out_reg
predefined_sync_rule: PRE_DFF_1
--------------------------------------------------------------------------------
source_clock_domain: ck1
destination_clock_domain: ck2
from_instance: F21/out_reg
to_instance: F22/out_reg
predefined_sync_rule: PRE_MUX0_1
--------------------------------------------------------------------------------
source_clock_domain: ck1
destination_clock_domain: ck2
from_instance: F31/out_reg

to_instance: F321/out_reg
predefined_sync_rule: PRE_MUX1_1
--------------------------------------------------------------------------------
```

For more information, see Working in the Predefined Tab on page 235, which describes how to display this information in the Clock Domain Crossing Manager.

Validation Flow

While in the normal validation flow, `REPORT VALIDATED DATA` outputs a summarized report that displays pass, fail, and not run (NR) results for structural and functional CDC checks. For functional SDC checks, this report also includes explored depth (ED) data.

Note: For each CDC path, if *any* of its sync rules is PASS its validated result is PASS. For a CDC path, if *all* of its sync rules are FAIL, its validated result is FAIL.

A CDC path can be classified under the following:

- `MIXED`—When the vector has individual bits that are classified into different categories for word-level reporting.
- `FAIL`—When the path does not satisfy any predefined synchronization rule.
- `NR`—When you do not run *Validate* after adding a CDC check, or if you stop a validation before it completes. NR also occurs when there is a crossing between different domains, but there is no synch rule to describe the synchronizations involved.

In the validation flow, each synchronization rule has sub-checks. To view the data for these subchecks, use the `REPORT VALIDATED DATA -verbose` command.

This following is a sample report for CDC validated data:

```
================================================================================
=                   Structural CDC Validated Data (Verbose)                    =
================================================================================
source_clock_domain: clkA
destination_clock_domain: clkB
  from_instance: din_clkA_reg[]
  to_instance: dout_reg[]
  status: Fail
  sync_rule: 2ff_mux : DFF : Fail
     cdc_path_logic_type_check: WIRE : Pass
     cdc_path_destination_check: SINGLE : Pass
     sync_chain_dff_number_check: 1 1 : Fail
     sync_chain_logic_type_check: WIRE : Pass
     sync_chain_number_check: SINGLE : Pass
  sync_rule: r1 : DFF : Fail
    .
    .
    .
================================================================================
=                   Functional CDC Validated Data (Verbose)                    =
================================================================================
source_clock_domain: clkA
destination_clock_domain: clkB
  from_instance: din_clkA_reg[]
  to_instance: dout_reg[]
source_data_stability_check: Fail
destination_data_stability_check: Fail
mux_enable_stability_check: Fail
single_bit_change_stability_check
```

The following describes the attributes and sub-checks for each synchronization rule:

Attributes	**Description**
`source_clock_domain`	Specifies source clock
`destination_clock_domain`	Specifies destination clock
`from_instance`	Specifies the start point of a crossing
`to_instance`	Specifies the end point of a crossing
Sub-checks (structural)	**Description**
`cdc_path_logic_type_check`	Ensures that the content of the CDC path meets the `-CDC_PATH [LOGic | WIRe | BUFfer]` specification.
`cdc_path_destination_check`	Ensures that the number of fanouts from the CDC path meets the `-CDC_PATH [MULtiple_destination | SINgle_destination]` specification.
`sync_chain_dff_number_check`	Ensures that the number of D flip-flops in the synchronizer meets the `-DFF` specification.
`sync_chain_number_check`	Ensures that the number of fanouts from the sync chain meets the `-SYNC_CHAIN [MULtiple_chain | SINgle_chain]` specification.
`mux_select_check`	Ensures that at least one multiplexer select input is driven by the *destination domain* and all others can be driven either by the *destination* or *universal* domain.
`mux_data_hold_check`	Ensures that one of the multiplexer data inputs is driven by the destination register
`mux_data_input_check`	Ensures that all multiplexer data inputs are driven by *source, destination,* or *universal* domain
`mux_to_destination_path_check`	Ensures that all gates on the path from the output of the multiplexer to the destination register are driven by the *destination* domain.

`redundant_sync_flip_flops`	Recognizes DFFs that appear in front of a MUX. This check occurs when you use the following command: `add synchronization rule r0 -mux 0 1` With this command, CDC checks for a MUX with a zero or one DFF in front of the MUX. If it finds one, this check returns *Yes*. Otherwise, this check returns *No*.

Sub-checks (functional)	**Description**
`source_data_stability_check`	Checks that the source data is stable until the destination clock latches the data.
`destination_data_stability_check`	Checks that data going to the destination is stable until destination clock latches data.
`mux_enable_stability_check`	For MUX-based synchronization schemes, checks that the output of the MUX is stable until the destination domain captures the data.
`single_bit_change_stability_check`	Checks that vectors that are flop-synchronized are gray encoded.

The following is a sample report for CDC convergence:

```
===============================================================================
= CDC Convergence Categorization Data (Verbose) =
===============================================================================
source_clock_domain: clka
destination_clock_domain: clkb
from_instance: rctrl/rptr_reg[]
to_instance: wctrl/sync/sdata_q0_reg[]
convergence check: Fail
convergent gate:
wctrl/wfull_reg
-------------------------------------------------------------------------------
source_clock_domain: clkb
destination_clock_domain: clka
from_instance: wctrl/wptr_reg[]
to_instance: rctrl/sync/sdata_q0_reg[]
convergence check: Fail
convergent gate:
rctrl/rempty_reg
==========================================================================
```

Note: For CDC convergence data, the attributes are similar to those discussed in the previous table. There are no subchecks for CDC convergence checks.

The following is a sample report for reset synchronization validated data:

```
================================================================================
= Reset Synchronization Validated Data (Verbose) =
================================================================================
Cell : srst_reg
Id : 19
Clock Domain : clk
Result : PASS (synchronizer not needed)
Cell : srst1_reg
Id : 21
Clock Domain : clk
Result : PASS (synchronizer not needed)
Cell : arst_reg
Id : 14
Clock Domain : clk2
Result : PASS (synchronizer not needed)
Cell : q_reg[3]
Id : 15
Clock Domain : clk
Result : FAIL (synchronizer not present)
Cell : q_reg[2]
Id : 16
Clock Domain : clk
Result : FAIL (synchronizer not present)
Cell : q_reg[1]
Id : 17
Clock Domain : clk
Result : FAIL (synchronizer not present)
Cell : q_reg[0]
Id : 18
Clock Domain : clk
Result : FAIL (synchronizer not present)
Cell : q1_reg
Id : 20
Clock Domain : clk
Result : FAIL (synchronizer not present)
============================================================================
```

Note: There are no subchecks for reset synchronization checks.

Diagnosing CDC Issues

The most efficient way to diagnose CDC Check violations is in the GUI mode. See Clock Domain Crossing Manager on page 200 for information on diagnosing CDC issues using the Clock Domain Crossing Manager. However, you can diagnose Functional checks from the command mode. See Diagnosing Functional CDC Check Failures on page 196.

Diagnosing Functional CDC Check Failures

Diagnose a single failed Functional CDC check by specifying the type of check you are diagnosing (`source_data`, `destination_data`, `mux_enable`, or `single_bit_change`) and the `from-instance` and `to-instance` of the domain in question with the `DIAGNOSE CDC CHECK` command. This command returns a counterexample of the failed check showing the time period during which instability occurred.

The following shows an example transcript.

```
 // Command: diagnose cdc check -source_data in2a_reg U_syn/ out1_reg -source clkA
-destination u_pll/div_by_2

Diagnose Functional CDC:

  Counter-example:

     Time Unit 0:
           1’b0 =>        8: in2

     Time Unit 1:
           1’b1 =>        8: in2


     Time Unit 9:
           1’b0 =>        8: in2

     Time Unit 13:
       <no new assignments needed>

  At Time 9:
    Clock of source REG has a rising edge
    output of source REG changes

    Monitoring point is:
      Gate_id=40,  Name=in2a_reg,  Value 1’b1

  Expecting value at monitoring point to hold stable
    until the next rising edge at u_pll/div_by_2 (id: 4)

At Time 13:
Value at monitoring point changes to:
      Gate_id=40,  Name=in2a_reg,  Value 1’b0

Between Time 9 and 13, there is no rising edge at u_pll/ div_by_2 (id: 4)
```

Writing Functional CDC Check Diagnosis Data

After you execute the `DIAGNOSE CDC CHECK` command, use the `WRITE CDC CHECK` command to generate a file containing the diagnosis data.

Fixing CDC Problems

Diagnosis reveals whether the CDC Check violations are actual violations of design intent or issues involving proper clocking scheme specification. In the case of the latter, review the previous sections and modify clocking scheme specifications accordingly.

For actual CDC structural Check violations of design intent, use the Schematic Viewer and Source Code Manager to locate the cause of the violation. For CDC Functional check violations, also use the Waveform Display to view relationships between clocks. Then, make the appropriate modifications to your design file. You will use a separate text editor for modifications; however, note that you can access an editor from the Source Code Manager. After implementing all the necessary modifications, repeat the entire CDC Check process. That is, manually read in the design and enter commands or save time by executing a dofile.

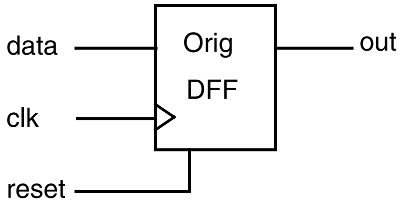
Checking Asynchronous Set and Reset Synchronizers

The CDC asynchronous set/reset synchronizer check verifies that DFFs in the design are synchronized properly during deassertion of set/reset. If a DFF has an asynchronous set or reset, care should be taken to ensure that it is deasserted synchronously. If the deassert of set/reset happens close to the active edge of the clock and the data input changes violating the setup-time window or hold-time window, then the DFF could go into a metastable state.

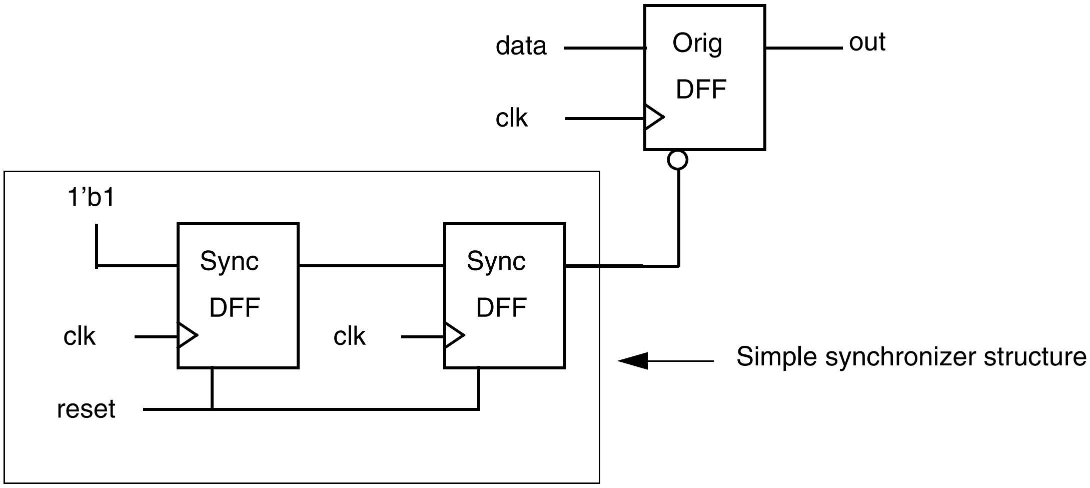
Asynchronous assertion does not cause metastability because the clock is bypassed. A well-known technique to solve this issue is to connect the reset of the DFF to a series of synchronized DFFs that has a tied data input and the same set/reset as the original DFF.

The set/reset synchronizer check verifies that DFFs with asynchronous set/reset ports are synchronized using the following technique.

If the original DFF is as follows:



The reset synchronizer looks like this:



The synchronizer structure has the following properties:

- It is a series of DFFs with the same common clock domain, called *SyncDFFs*.
- Each SyncDFF should the same set/reset source. A SyncDFF cannot have both set and reset.
- The first SyncDFF is tied low/high or is connected to another synchronizer structure whose first SyncDFF is tied high/low. All the DFFs in the synchronizer structures should have the same set/reset source. The first DFF of the reset synchronizer can also have its data and reset with opposing logic.
- The minimum number of SyncDFFs in the synchronizer structure must be 2.
- The sync chain can contain inverters or buffers in the path. Other gate types can also exist provided they effectively act as a buffer/inverter.
- The last gate in the sync chain and the asynchronous reset fanin of the original flip-flop should have opposing logic polarity. This is needed to ensure that set/reset is asserted asynchronously. In the above example, this means that there should be an inverter between the second SyncDFF output and reset input of the Original DFF.

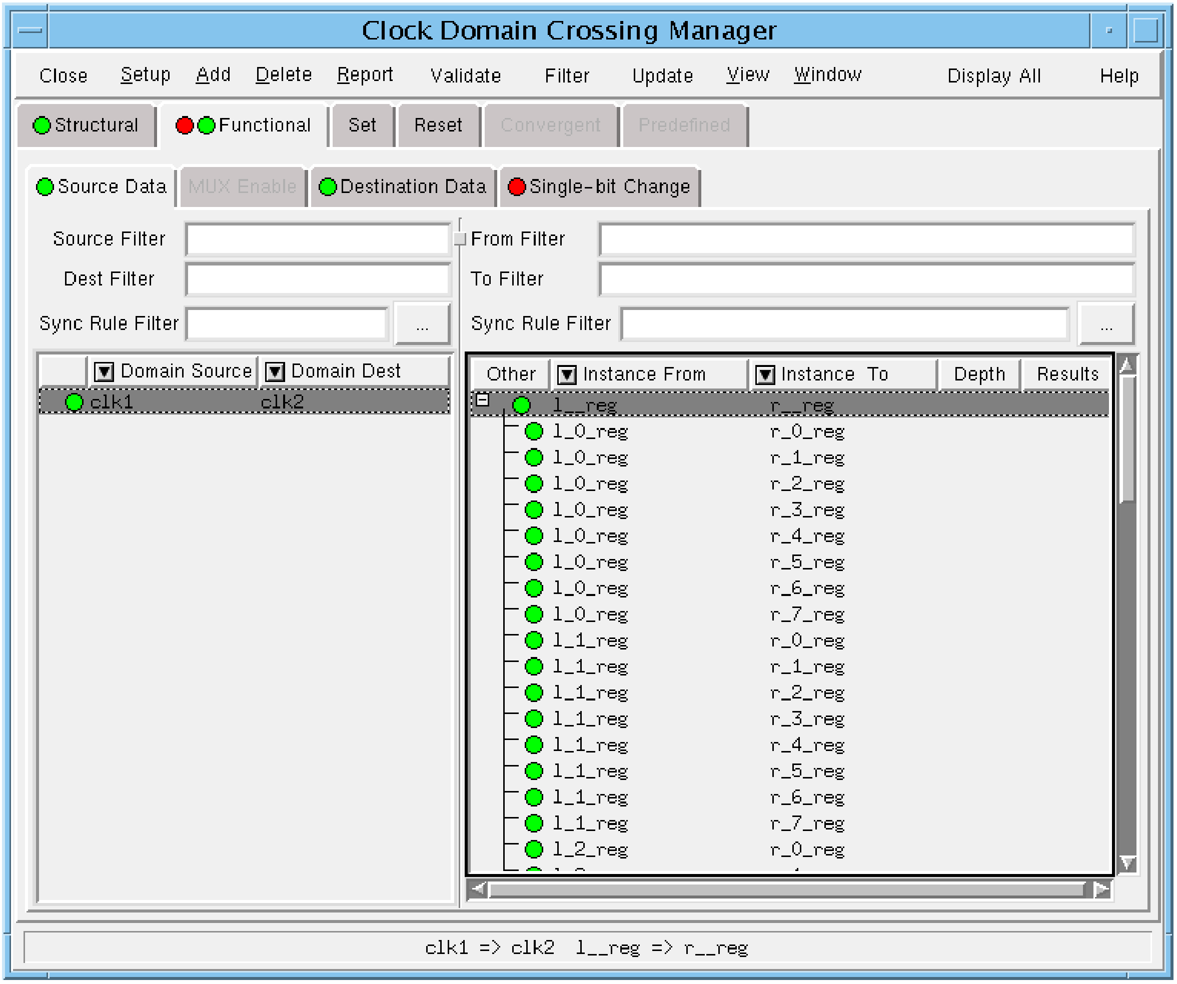
The synchronizer structure removes metastability from the original DFF because deassertion happens synchronously after as many clock cycles as there are sync DFFs. The sync DFF structure appears to violate the synchronizer check because it is connected directly to an asynchronous set/reset. Indeed, the first sync DFF might go metastable when the reset is deasserted. But that value will stabilize before the second sync DFF samples it. The second sync DFF will not go metastable because both the data and the output are low at the time when the reset is deasserted.

Clock Domain Crossing Manager

You can use the Clock Domain Crossing Manager to add, delete, and report clock domain associations, data signal associations, synchronization rules, and clock domains selected for domain crossing checks. You can also use this Manager to initiate validation, report validated data, diagnose failures and write diagnosis data to a file, access integrated diagnosis tools (for example, the Schematic Viewer), and customize predefined synchronization rules.

To open the Clock Domain Crossing Manager (in Verify mode), choose *Tools – Clock Domain Crossing* or click the *Clock Domain Crossing* button from the main GUI.

Also, note that Conformal Extended Checks supports wildcards for string matching.



	Domain Source	Domain Dest
●	clk1	clk2

Other	Instance From	Instance To	Depth	Results
⊟ ●	l__reg	r__reg		
●	l_0_reg	r_0_reg		
●	l_0_reg	r_1_reg		
●	l_0_reg	r_2_reg		
●	l_0_reg	r_3_reg		
●	l_0_reg	r_4_reg		
●	l_0_reg	r_5_reg		
●	l_0_reg	r_6_reg		
●	l_0_reg	r_7_reg		
●	l_1_reg	r_0_reg		
●	l_1_reg	r_1_reg		
●	l_1_reg	r_2_reg		
●	l_1_reg	r_3_reg		
●	l_1_reg	r_4_reg		
●	l_1_reg	r_5_reg		
●	l_1_reg	r_6_reg		
●	l_1_reg	r_7_reg		
●	l_2_reg	r_0_reg		

clk1 => clk2 l__reg => r__reg

For the Clock Domain Crossing Manager and functionality, see the following for more information:

- CDC Checks – General Flow on page 171 to review the order of command entry and process flow.
- Window Layout on page 201 for a description of the menu items and display area fields and options.
- Clock Domain Rule on page 206.
- Clock Domain Priority on page 210
- Clock Domain Assignment on page 211
- Predefined Synchronization Rule on page 215
- Synchronization Rule on page 218
- CDC Checks on page 222 for information on adding, deleting, and reporting CDC checks.
- Diagnosing a Failed Instance on page 234
- Viewing a Schematic of an Instance on page 235
- Diagnosing Set, Reset, and Convergent Domain Crossing Violations on page 235
- Reporting Validated Data for a Single Domain, Instance, or Bit on page 236
- Viewing a Schematic of an Instance on page 237

Window Layout

The Clock Domain Crossing Manager includes a menu bar and two display areas. The tabs allow you to access the display area for *Structural*, *Functional*, *Set*, *Reset*, and *Convergent* CDC check information; the *Predefined* tab allows you to customize the predefined synchronization rules.

Note: The accessible tabs for the Clock Domain Crossing Manager depend on your current flow. When you use the predefined synchronization rules alone, Conformal automatically enters the *Categorization* flow when you issue the `VALIDATE` command. Otherwise, it enters the normal *Validation* flow. The *Predefined* tab of the Clock Domain Crossing Manager is only available in the *Categorization* mode.

Menu Options

The following table lists drop-down menu commands you will use as you work with the Clock Domain Crossing Manager:

Table 9-1 Clock Domain Crossing Manager, Menu Bar

Menu/Option	Description
Close!	Closes the Clock Domain Crossing Manager.
Setup Menu	
Clock Domain Rule	Specifies clock domain rules.See Clock Domain Rule on page 206.
Clock Domain Priority	Specifies clock domain priority, resets priority, or re-orders the priority list. See Clock Domain Priority on page 210
Clock Domain Assignment	Opens the Clock Domain Assignment window. See Clock Domain Assignment on page 211.
Predefined Synchronization Rule	Customizes the predefined synchronization rules. See Predefined Synchronization Rule on page 215.
Synchronization Rule	Defines the synchronizers used in specified portions of the design. See Using Predefined Synchronization Rules on page 183.
Add CDC Check	
CDC Check	Specifies CDC checks. See Add CDC Check on page 222.
Delete CDC Check	
CDC Check	Deletes CDC checks. See Delete CDC Check on page 225.
Report SDC Check	
CDC Check	Specifies and executes the Report CDC Check command. See Report SDC Check on page 226.
Validate!	Initiates validation. See Validate! on page 226.
Update!	Refreshes the display. See Update! on page 226.
View Menu	Configures the Clock Domain Crossing Manager display. See View Menu on page 227.

Window	Moves any of the open windows to the front of the screen or refreshes the desktop and displays the main Conformal Extended Checks window on top with all other open windows in a cascading view to the left of the main window.
Help!	Opens the Verify Command Help window.

While the *Close!*, *Window*, and *Help!* buttons are standard throughout the Conformal GUI windows, procedures related to the remaining buttons and drop-down menus are described below.

Display Area

The display area of the Clock Domain Crossing Manager shows clock domain crossing checks, domain path information, and CDC Check results. This is organized by tabs and lists of domains.

The *Structural* and *Predefined* tab headings group the clock domains according to the following categories (start and end types):

- Register (clk1) to register (clk2)
- Primary input to register (clk)
- Register (clk) to primary output
- Primary input to black box
- Black box to primary output
- Primary input to primary output
- Register (clk) to black box
- Black box to register (clk)
- Black box to black box

The *Functional* tab headings group the clock domains according to the type of check:

- Source Data
- Destination Data
- Multiplexer Enable
- Single-bit Change

The *Set*, *Reset*, and *Convergent* tab headings group the clock domains according to the following categories (start and end types):

- Register (clk1) to register (clk2)
- Primary input to register (clk)
- Black box to register (clk)

Click a tab to it to the front. Then, click a sub-tab to view the desired sub-check. Until you specify domains you would like to add in a particular category (for example, *REG-REG*), the applicable tab is grayed out.

After you add domain crossing checks, note that each of the applicable tabs displays the unprocessed (Not Run) status indicator. And when you click any active tab, you will see a list of domains in the domain display area. Click a domain to display expanded information and show instances (and when applicable, bits) in the instance display area. Rest the cursor over a domain or instance to display an information box that lists pertinent details about the selection.

After validation, if a category includes both pass and fail instances, the Clock Domain Crossing Manager displays both the pass and fail icons on the tab. By default, Conformal displays a green-filled circle to indicate pass and a red-filled circle to indicate fail. If you prefer that Conformal displays check marks, you can set this from the *Preferences* menu. If you change preferences, use the *Update!* button on the menu bar to refresh the Clock Domain Crossing Manager and activate the changes.

Status Indicators

The Clock Domain Crossing Manager displays clock domain crossing status indicators in three places: on tabs and preceding domains and instances. These indicators are defined below:

🔴 ❌	red-filled circle (or red X)	indicates a violated path
🟢 ✓	green-filled circle (or green check)	indicates there is no violated path
?	circled red question mark (?)	indicates the path was added, but not yet validated
🟡	yellow-filled circled	indicates the path was explored to Depth N (This applies to Functional CDC checks.)

Domain and Instance Display Areas

The *Clock Domain Crossing* Manager includes two display areas: domains and instances. The Domain display area lists domains for a selected category (for example, REG-REG) and includes two columns: *Domain Source* and *Domain Destination*. When you select a domain (in the domain display area), the instance display area to the right shows expanded information for the selected domain.

Resizing the Display Areas

To resize the display areas, position the cursor over the vertical bar between the domain and instance display areas. When the double-ended arrow appears, click and drag the bar left or right to the desired position.

Filtering the Display

To show names that match a specified string in the domain and instance display areas, specify a string in one of the *Filter* fields and press *Enter*. To return to the original display, click *Display All*, which is located at the far right end of the filter fields.

Sorting Names

▼■ Click the triangular sort icon in a column heading to sort names alphabetically in ascending order. The square icon indicates the column that was sorted. (The complementary column is rearranged to maintain pairing.)

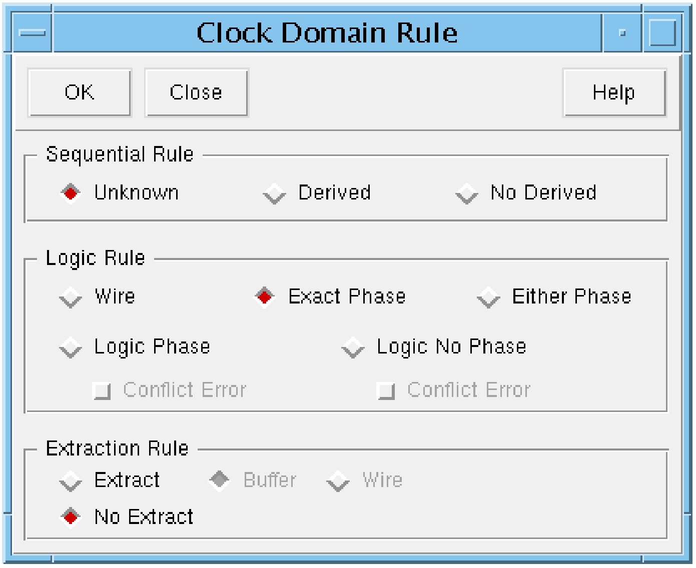
Setup Menu

Use the *Setup* drop-down menu to access windows that allow you to create clock domain rules, specify clock domain priorities, make clock domain assignments, and establish synchronization rules.

Clock Domain Rule

Use the Clock Domain Rule form to specify clock domain assignment rules.

➤ Choose *Setup – Clock Domain Rule*.



Sequential Rule

● Unknown ○ Derived ○ No Derived

Logic Rule

○ Wire ● Exact Phase ○ Either Phase

○ Logic Phase ○ Logic No Phase

□ Conflict Error □ Conflict Error

Extraction Rule

○ Extract ● Buffer ○ Wire

● No Extract

Clock Domain Rule Fields and Options

Unknown
Specifies that the output of the state element has an unknown domain, regardless of the domain of the clock inputs.

Derived
Specifies that the output of the state element has the same domain as the clock input, if all clock inputs have the same known domain.

No Derived
Specifies that the output of the state element has a new known domain (different from its clock inputs) if all clock inputs have the same known domain or different known domains.

Note: For the *Derived* and *No Derived* options, the output of the state element has an unknown domain if all clock inputs have unknown domains, and the output of the state element has an error domain if any clock input has an error domain.

Wire

Specifies that if all inputs have unknown domains, the output has an unknown domain. If none of the logic gate’s inputs has an error domain and at least one input has a known domain, the output has a new known domain.

Exact Phase

Specifies that if all inputs have unknown domains, the output has an unknown domain. If none of the logic gate’s inputs has an error domain and at least one input has a known domain, the output has a new known domain.

Exceptions:

- The output of an inverter has a domain that represents the inverted phase of the input domain. (`Di -> inv -> ~Di`)
- The output of a buffer has the same domain as the input domain. (`Di -> buf -> Di`).

Either Phase

Specifies that if all inputs have unknown domains, the output has an unknown domain. If none of the logic gate’s inputs has an error domain and at least one input has a known domain, the output has a new known domain.

Exceptions:

- The output of a buffer has the same domain as the input domain. (`Di -> buf -> Di`)
- The output of an inverter has the same domain as the input domain. (`Di -> inv -> Di`)

Logic Phase

Specifies the following:

- If all inputs have unknown domains, the output has an unknown domain.
- If any two inputs have different known domains, the output has a new known domain.
- If all inputs have the same known domain, with the exception of zero or more unknown domains, the output has the same known domain as its inputs.

Exception:

The output of an inverter has a domain that represents the inverted phase of the input domain. (`Di -> inv -> ~Di`)

Selecting *Conflict Error* specifies the following:

- If all inputs have unknown domains, the output has an unknown domain.
- If any input has an error domain, the output has an error domain.
- If any two inputs have different known domains, the output has an error domain.
- If all inputs have the same known domain, with the exception of zero or more unknown domains, the output has the same known domain as its inputs.

Exception:

The output of an inverter has a domain that represents the inverted phase of the input domain (`Di -> inv -> ~Di`).

Logic No Phase

Specifies the following:

- If all inputs have unknown domains, the output has an unknown domain.
- If any two inputs have different known domains, the output has a new known domain.
- If all inputs have the same known domain, with the exception of zero or more unknown domains, the output has the same known domain as its inputs.

Exception:

The output of an inverter has the same domain as the input domain (`Di -> inv -> Di`).

Selecting *Conflict Error* specifies the following:

- If all inputs have unknown domains, the output has an unknown domain.
- If any input has an error domain, the output has an error domain.
- If any two inputs have different known domains, the output has an error domain.
- If all inputs have the same known domain, with the exception of zero or more unknown domains, the output has the same known domain as its inputs.

```
Di -> inv -> Di
```

Extract

Selecting *Buffer* automatically extracts domains starting from the unknown domain clock ports of D flip-flops and D-latches, and then backward trace through buffers and inverters. Recognize the gate that stops the backward extraction as an extracted clock domain.

Selecting *Wire* automatically extracts domains so that all unknown domain clock ports of D flip-flops and D-latches are treated as extracted clock domains.

No Extract

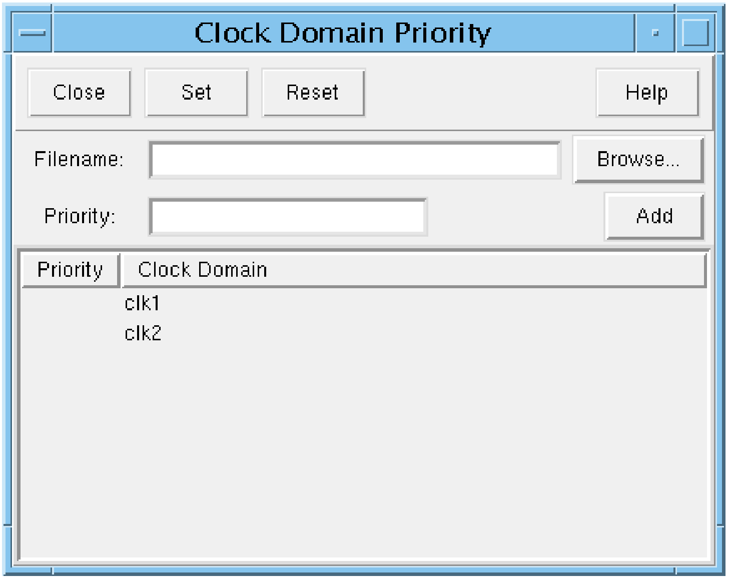
Does not automatically extract domains.

Clock Domain Priority

Use the Clock Domain Priority form to specify clock domain priority, reset priority, or re-order the priority list. This form lists the design's clock domains. If you have already set priorities, the clocks are listed in descending order according to the specified priority.

When you execute actions in the Clock Domain Priority window, Conformal displays the specified clock domain priorities in the Transcript window of the main Conformal Extended Checks window and updates the CDC database.

➤ Choose *Setup – Clock Domain Priority*.



Priority	Clock Domain
	clk1
	clk2

Clock Domain Priority Fields and Options

Set
Specifies domain priority from the displayed clock domain names.

Reset
Removes the priority specification for all clock domains.

Filename
Sets clock domain error resolution priority according to the list in the specified file. Type in the file or click *Browse* to open the Clock Priority File form and choose a file.

Priority

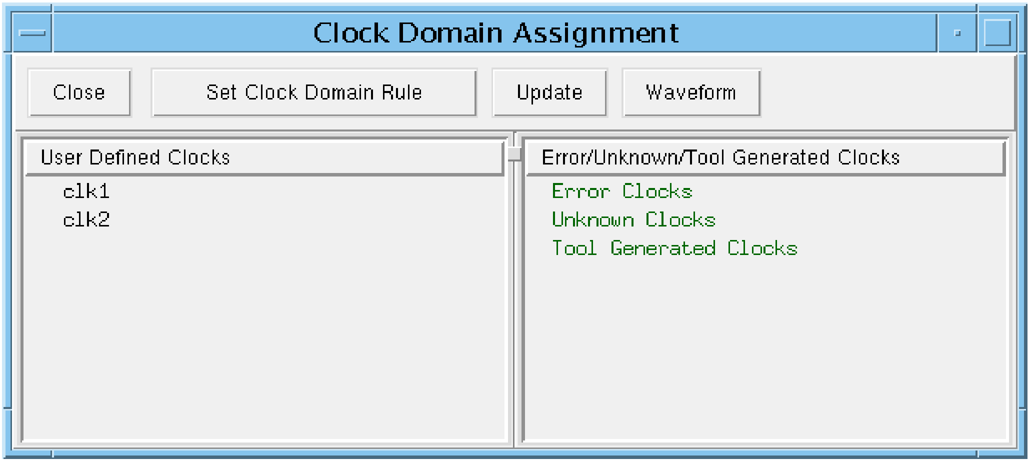
With a name in the *Clock Domain* column selected, enter an integer that represents ordinal position according to priority. Click *Add* to add the priority to the list.

Clock Domain Assignment

Use the Clock Domain Assignment form to add clock associations, access integrated debugging tools, set clock domain rules, and view reports.

When running actions in the Clock Domain Priority window, Conformal displays the specified clock domain priorities in the Transcript window of the main Conformal Extended Checks window and updates the CDC database.

➤ Choose *Setup – Clock Domain Assignment*.



Clock Domain Assignment Fields and Options

Set Clock Domain Rule

Opens the Clock Domain Rule form to specify clock domain assignment rules.

Update

Refreshes the window’s contents and restores default viewing options (for example, collapse all expanded entries).

Waveform

Opens the Waveform Viewer so you can view relationships between clocks.

You can use the pull-down menu in each list by right-clicking over the objects.

Pop-Up Menu for User Defined Clocks

To access the pop-up menu, right-click on a clock name in the *User Defined Clocks* column to select it. To select an option, scroll down and highlight the option. The following table describes these options.

Option	Description
Set Target Clock Association	Specifies the selected clock as the representative clock.
Add Clock Association	Associates a group of synchronous clocks as a single clock domain. For more information, see Associating User-Defined Clocks on page 213 and the `ADD CLOCK ASSOCIATION` command.
Clock Tree Schematics	Opens a clock tree schematic for a selected clock.
Source Code	Opens the Source Code Manager.

Pop-Up Menu for Error/Unknown/Tool Generated Clocks

Click the square-enclosed (+) adjacent to one of the categories show all clocks in that category and click a clock name.

To access the pop-up menu, right-click on a clock name in the *Error/Unknown/Tool Generated Clocks* column to select it. To select an option, scroll down and highlight the option. The following table describes these options.

Option	Description
Report Clock Candidate	For Error Clocks, this displays a Clock Domain Report in the Transcript window of the main Conformal Extended Checks window. By default, this report For Unknown Clocks, this displays a Clock Candidates Report in the Transcript window of the main Conformal Extended Checks window.
Show Error Source Code Only	(For Error Clocks) Displays gates with assignment conflict errors in the Clock Domain Report.
Add Clock Association	Associates a group of synchronous clocks as a single clock domain. For more information, see Associating User-Defined Clocks on page 213 and the `ADD CLOCK ASSOCIATION` command.
Add as Generated Clock	Creates a new clock domain for the specified ID or path of an internally generated clock.
Schematics	Opens a clock schematic for a selected clock.
Source	Source Code Manager

Running Clock Association

Use the following procedures to select clocks for association, remove a single clock from a clock association, and delete a clock association from the Clock Domain Assignment form.

Associating User-Defined Clocks

Select clocks for association from either the User Defined Clocks (step 4) *or Error/ Unknown/Tool Generated Clocks* section of the window (steps 5 and 6).

In step 3, the font color of the selected target clock changes to red. This signifies the representative clock. After you associate clocks, the Clock Domain Assignment window displays the associated clocks below the representative clock with a connecting line.

1. Click a clock name in the *User Defined Clocks* column to select it.
2. Right-click and choose *Set Target Clock Association* from the pop-up menu to specify the clock name as the representative clock.
3. Do one of the following:
   - ❑ Click a clock name in the *User Defined Clocks* section.
   - ❑ In the *Error/Unknown/Tool Generated Clocks* section, click a square-enclosed (+) adjacent to one of the categories show all clocks in that category and click a clock name.
4. Right-click and choose *Add Clock Association* from the pop-up menu.

Repeat steps 3 through 4 to associate additional clocks with the representative clock. Repeat steps 2 through 4 to add a new clock association.

Removing Clock Associations

To remove a single clock from a clock association, or o delete a clock association, use the following procedure:

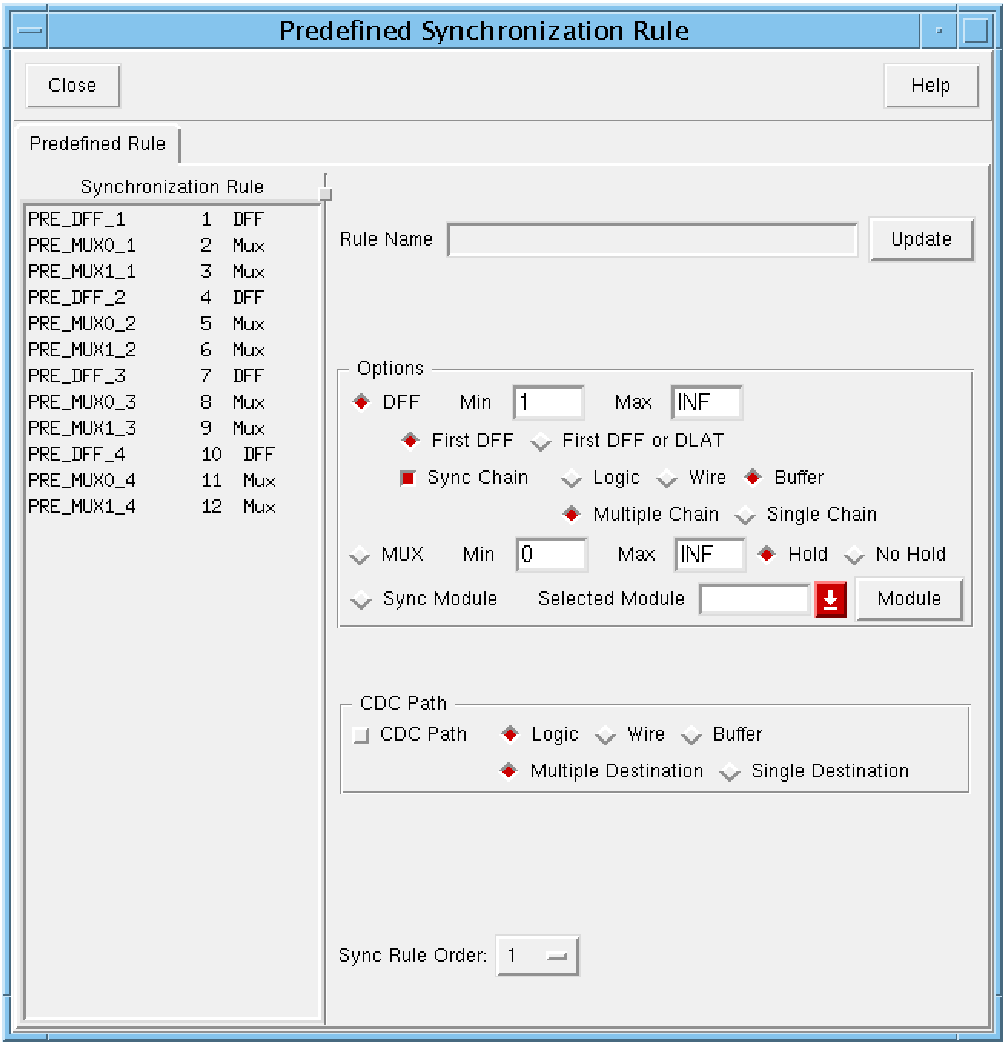
1. Click the square-enclosed (+) adjacent to a representative clock.

   The clock domain association expands to show all associated clocks.
2. Click an associated clock to select it.
3. Right-click and choose *Delete Clock Association* or *Delete All Clock Associations* from the pop-up menu.

Predefined Synchronization Rule

Use the Predefined Synchronization Rule form to customize the predefined synchronization rules provided by the Conformal software.

➤ Choose *Setup – Predefined Synchronization Rule*.



Synchronization Rule

PRE_DFF_1	1	DFF
PRE_MUX0_1	2	Mux
PRE_MUX1_1	3	Mux
PRE_DFF_2	4	DFF
PRE_MUX0_2	5	Mux
PRE_MUX1_2	6	Mux
PRE_DFF_3	7	DFF
PRE_MUX0_3	8	Mux
PRE_MUX1_3	9	Mux
PRE_DFF_4	10	DFF
PRE_MUX0_4	11	Mux
PRE_MUX1_4	12	Mux

Rule Name

Options
● DFF Min 1 Max INF
● First DFF ○ First DFF or DLAT
☑ Sync Chain ○ Logic ○ Wire ● Buffer
● Multiple Chain ○ Single Chain
○ MUX Min 0 Max INF ● Hold ○ No Hold
○ Sync Module Selected Module

CDC Path
□ CDC Path ● Logic ○ Wire ○ Buffer
● Multiple Destination ○ Single Destination

Sync Rule Order: 1

Predefined Synchronization Rule Fields and Options

Rule Name

Specifies one of the predefined synchronization rule names. To view the rule names, use the `REPORT PREDEFINED SYNCH_RULE` command.

Update

Applies the changes made in the form for the predefined synchronization rule in the *Rule Name* field.

DFF

Specifies using D flip-flops as sychronizers.

- *Min*—specifies the minimum number of flip-flops in the DFF synchronizer (an integer value of 1 or greater).
- *Max*—specifies the maximum number of flip-flops in the DFF synchronizer (an integer value that is greater than or equal to *Min*). You can choose `INF` (the default) to specify the maximum number of flip-flops.
- *First DFF*—specifies that the first state element in the DFF synchronizer must be a D flip-flop (that is, a D-latch is not allowed as the first element in the DFF synchronizer).
- *First DFF or DLAT*—specifies that the first state element in the DFF synchronizer can be either a D flip-flop or D-latch.
- *Sync Chain*—identifies what is allowed between the D flip-flops (or D-latch and D flip-flops) of the DFF synchronizer.
  - *Buffer*—allows only wire, buffers, and inverters between the state elements of the DFF synchronizer.
  - *Wire*—allows only wires between the state elements of the DFF synchronizer.
  - *Logic*—allows any logic gate between the state elements of the DFF synchronizer.
  - *Multiple Chain*—allows multiple fanout from within the DFF synchronizer chain.
  - *Single Chain*—does not allow fanout from within the DFF synchronizer chain.

MUX

Specifies using multiplexers as sychronizers.

- *Min*—specifies the minimum number of destination D flip-flops before the multiplexer.
- *Max*—specifies the maximum number of destination D flip-flops before the multiplexer. You can choose `INF` (the default) to specify the maximum number of flip-flops.
- *Hold*—specifies that the data hold condition must exist for the MUX synchronizer.
- *No Hold*—specifies that the data hold condition does not have to exist for the MUX synchronizer.

Synch Module

Specifies using modules as synchronizers. You can type in a name in the *Selected Module* field, or click *Module* to open the Module List form to select a module.

CDC Path

Specifies what is allowed in the CDC path.

- *Logic*—allows any logic in the CDC path.
- *Wire*—allows only wire in the CDC path.
- *Buffer*—allows only wire, buffer, and inverters in the CDC path.
- *Multiple Destination*—allows multiple fanouts from the CDC.
- *Single Destination*—does not allow not allow any fanout from the CDC path.

Sync Rule Order

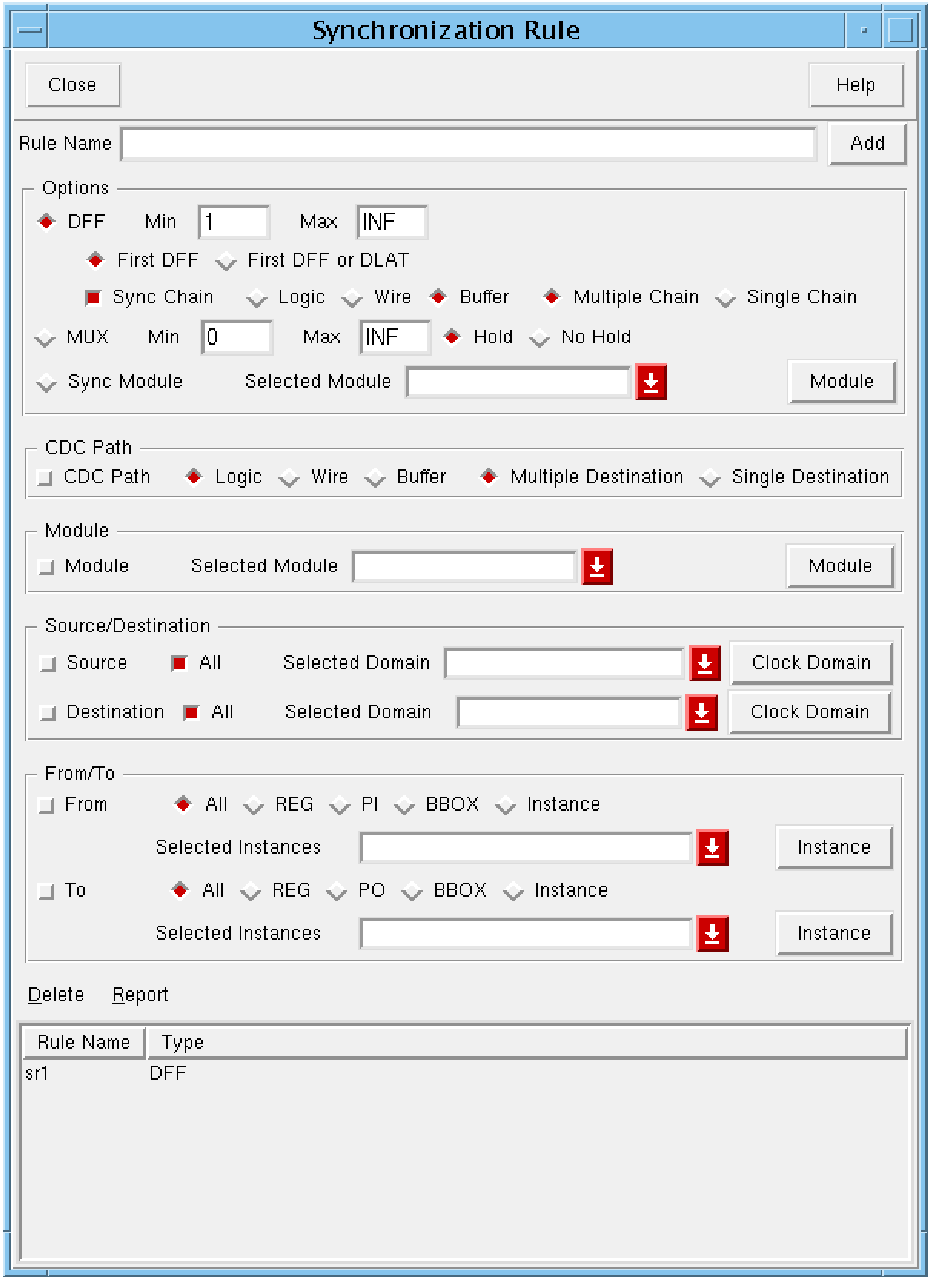
Specifies the linear order for the rule.

Synchronization Rule

Use the Synchronization Rule form to define the synchronizers used in specified portions of the design, use the Synchronization Rule window. For more information, see Specifying Synchronization Rules on page 182.

Note: You cannot apply synchronization rules to Set and Reset checks.

➤ Choose *Setup – Synchronization Rule*.



Rule Name

Options

● DFF Min 1 Max INF

● First DFF ○ First DFF or DLAT

☒ Sync Chain ○ Logic ○ Wire ● Buffer ● Multiple Chain ○ Single Chain

○ MUX Min 0 Max INF ● Hold ○ No Hold

○ Sync Module Selected Module Module

CDC Path

□ CDC Path ● Logic ○ Wire ○ Buffer ● Multiple Destination ○ Single Destination

Module

□ Module Selected Module Module

Source/Destination

□ Source ☒ All Selected Domain Clock Domain

□ Destination ☒ All Selected Domain Clock Domain

From/To

□ From ● All ○ REG ○ PI ○ BBOX ○ Instance

Selected Instances Instance

□ To ● All ○ REG ○ PO ○ BBOX ○ Instance

Selected Instances Instance

Delete Report

Rule Name	Type
sr1	DFF

Synchronization Rule Fields and Options

Rule Name

Specifies an identification name assigned to a specific synchronization rule.

DFF

Specifies using D flip-flops as sychronizers.

- *Min*—specifies the minimum number of flip-flops in the DFF synchronizer (an integer value of 1 or greater).
- *Max*—specifies the maximum number of flip-flops in the DFF synchronizer (an integer value that is greater than or equal to *Min*). You can choose `INF` (the default) to specify the maximum number of flip-flops.
- *First DFF*—specifies that the first state element in the DFF synchronizer must be a D flip-flop (that is, a D-latch is not allowed as the first element in the DFF synchronizer).
- *First DFF or DLAT*—specifies that the first state element in the DFF synchronizer can be either a D flip-flop or D-latch.
- *Sync Chain*—identifies what is allowed between the D flip-flops (or D-latch and D flip-flops) of the DFF synchronizer.

  Click *Logic*, *Wire*, or *Buffer* to further specify the Sync Chain.

  Click *Multiple Chain* or *Single Chain* to specify whether multiple fanouts from within the DFF synchronizer chain are allowed.

MUX

Specifies using multiplexers as synchronizers.

- *Min*—specifies the minimum number of destination D flip-flops before the multiplexer.
- *Max*—specifies the maximum number of destination D flip-flops before the multiplexer. You can choose `INF` (the default) to specify the maximum number of flip-flops.
- *Hold*—specifies that the data hold condition must exist for the MUX synchronizer (that is, whether feedback from the destination register to the mux data input is required).
- *No Hold*—specifies that the data hold condition does not have to exist for the MUX synchronizer.

Synch Module

Specifies using modules as synchronizers. With this option, Conformal Extended Checks recognizes the cell instance and considers all crossings that pass through the specified module as synchronized.

You can type in a name in the *Selected Module* field, or click *Module* to open the Module List form to select a module.

CDC Path

Use this panel to specify what is allowed in the CDC path. The CDC path is the signal path before the synchronizer—that is, the path between the source and destination domains.

Click *Logic*, *Wire*, or *Buffer* to specify the allowable CDC path logic.

Click *Multiple Destination* or *Single Destination* to specify whether fanouts from the CDC path are allowed.

Module

Use this panel to identify the module for which the synchronization rule applies.

Note: This is different from the *Synch Module* option, which identifies a module that is being used as a synchronizer.

You can type in a name in the *Selected Module* field, or click *Module* to open the Module List form to select a module.

Source/Destination

Source—applies the synchronization rule to the paths with the specified source clock domain.

Destination—applies the synchronization rule to the paths with the specified destination domain.

All—allows the synchronization rule to all source or destination clock domains.

Type a clock domain name in the *Selected Domain* field, or click *Clock Domain* to open the Domain List form where you can select one or more clock domain.

From/To

From applies the synchronization rule to the paths that start from the specified key points.

To applies the synchronization rule to the paths that end at the specified key points.

- *All*—applies the synchronization rule to the paths that start from/end at all the allowed key points. The type of key points includes registers, primary inputs, and blackboxes (outputs).
- *REG*—applies the synchronization rule to the paths that start from/end at registers.
- *PI*—applies the synchronization rule to the paths that start from/end at primary inputs.
- *BBOX*—applies the synchronization rule to the paths that start from/end at blackbox output pins.
- *Instance*—applies the synchronization rule to the paths that start from/end at the specified instance(s). The instances can be a register, primary input, or blackbox output.

  Type an instance name in the *Selected Instances* field, or click *Instance* to open the From Instance List form where you can select one or more clock instance names.

Delete

Deletes synchronization rules. Select one or more rule names at the bottom of the form and choose *Delete* or *Delete All* from this pop-up menu.

Report	Reports synchronization rules. Select one or more rule names at the bottom of the form and choose *Report* or *Report All* from this pop-up menu.

CDC Checks

Add CDC Check

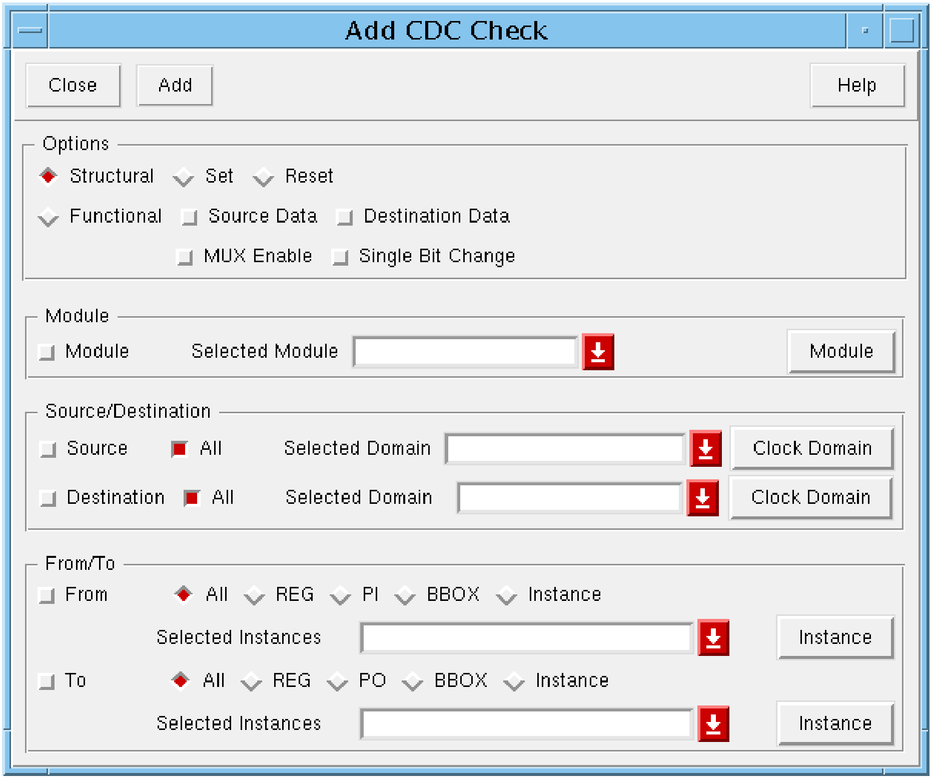
Before you can add Functional CDC checks, you must add and validate Structural CDC checks for the portion of the design where you intend to add Functional CDC checks.

Use the Add CDC Check form to add structural, set, or reset CDC checks. By default, Conformal Extended Checks adds Structural CDC checks. Functional CDC checks build on the Structural CDC checks, and as such, *you must meet the following requirements for successful CDC checks validation:*

- Execute Structural CDC checks before you add Functional CDC checks. Although it is not necessary to perform Structural CDC checks on the entire design, you must perform them on the region where you would like to perform the Functional CDC checks.
- Once you add Functional CDC checks, you can neither add nor delete Structural CDC checks until you delete all Functional CDC checks.

For additional information, see Selecting CDC Paths on page 188.

➤ Choose *Add – CDC Check*.



Add CDC Check Fields and Options

Structural — Adds the Structural CDC check as specified.

Set — Adds a check for asynchronous set pins of the flip-flops to ensure that they are consistent with the clock domain.

Reset — Adds a check for asynchronous reset pins of the flip-flops to ensure that they are consistent with the clock domain.

Functional

Adds the Functional CDC check as specified. Functional CDC Checks can include the following:

- *Source Data* adds the source data Functional CDC check. The source data check determines whether data leaving the source register is held stable for the destination register to latch it.
- *Destination Data*—adds the destination data Functional CDC check. The destination data check determines whether data entering the destination register is held stable for the destination register to latch it.
- *MUX Enable*—adds the MUX enable Functional CDC check. This check determines whether data at the output of the multiplexer synchronizer and the select line of the synchronizer are held stable (after changing at the select input) for the destination register to latch it.
- *Single Bit Change*—adds the single-bit change Functional CDC check. The single-bit change check ensures that a vector of signals crossing different clock domains can only be changed one bit at a time.

If you do not specify the type of Functional CDC check, the software runs all checks by default.

Note: To run the Functional CDC checks on a path, that path must first pass the Structural CDC check. Otherwise, you cannot add the Functional CDC check for that path.

Module

Use this panel to apply this check to the specified modules or module instance path names.

You can type in a name in the *Selected Module* field, or click *Module* to open the Module List form to select a module.

Source/Destination

Source—adds the specified CDC Check to the paths with the specified source clock domain.

Destination—adds the specified CDC Check to the paths with the specified destination clock domain.

All—uses all clock domains as the source or destination clock domains.

Type a clock domain name in the *Selected Domain* field, or click *Clock Domain* to open the Domain List form where you can select one or more clock domain.

From/To

From adds the specified CDC Check to the paths that start from the specified key points.

To adds the specified CDC check to the paths that end at the specified key points.

- *All*—uses all types of key points as the start/end of the paths. The type of key points includes registers, primary inputs, and blackboxes (outputs).
- *REG*—uses registers as the start/end of the paths.
- *PI*—uses primary inputs as the start/end of the paths.
- *BBOX*—uses blackbox (outputs) as the start/end of the paths.
- *Instance*—uses the specified instances as the start/end of the paths. The instances can be registers, primary inputs, or blackbox outputs.

  Type an instance name in the *Selected Instances* field, or click *Instance* to open the From Instance List form where you can select one or more clock instance names.

Delete CDC Check

Use the The Delete CDC Check form to turn off the CDC checks that were originally specified with the `ADD CDC CHECK` command or the Add CDC Check form.

The Delete CDC Check form is similar to Add CDC Check. For a description of the fields and options for this form, see Add CDC Check on page 222 and substitute *Add* with *Delete*.

Report SDC Check

Use the The Report CDC Check form to list the CDC checks that were originally specified with the `ADD CDC CHECK` command or the Add CDC Check form. The Transcript window of the main Conformal Extended Checks window displays the resulting report.

By default, Conformal Extended Checks reports Structural CDC checks.

The Report CDC Check form is similar to Add CDC Check. For a description of the fields and options for this form, see Add CDC Check on page 222 and substitute *Add* with *Report*.

Validate!

Use the *Validate!* button to initiate validation. The Transcript window of the main Conformal Extended Checks window displays the results of the validation.

Before you can initiate validation, you must specify synchronization rules and add domain crossing checks (see Setup Menu on page 205 and Add CDC Check on page 222 for more information.)

Update!

Use the *Update!* button to refresh the window’s contents and to restore default viewing options (for example, name format).

Note: If you change Pass/Fail Icon preferences, Conformal prompts you to refresh windows with pass/fail icons.

View Menu

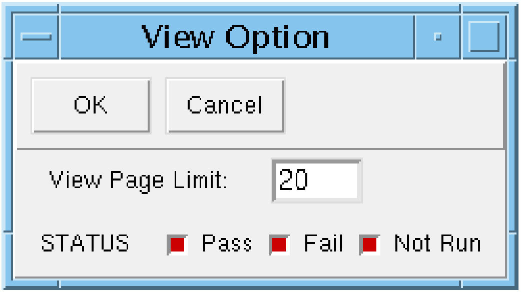
Use the *View* drop-down menu to configure the Clock Domain Crossing Manager display. These functions are especially useful when you encounter a large number of domains.

The changes you make with the View Option window act on the current display. Use the Update! button on the Clock Domain Crossing Manager's menu bar when you select a status that was previously de-selected.

View Option

With the *Structural*, *Functional*, *Set*, or *Reset* tab active, use the View Option form to specify page limits and display paths according to status. By default, Conformal Extended Checks displays all statuses.

➤ Choose *View – Option*.



View Page Limit: 20

STATUS ☒ Pass ☒ Fail ☒ Not Run

View Option Fields and Options

View Page Limit

Specifies an integer for the page limit. The default is 20.

STATUS

Displays domains according to a status of *Pass*, *Fail*, or *Not Run*.

With the *Functional* tab active, you can also select a status of *Explored*.

Tip

When necessary, click *Update!* on the menu bar of the Clock Domain Crossing Manager to refresh the window's contents.

Name Format

By default, Conformal Extended Checks displays full paths. Use the *Name Format* option to shorten names to (...) and the final 29 characters of the name. This process is specific to the type of CDC Check. Thus, if you want to specify name format for all CDC checks, you must perform the process for each tab.

1. Click the *Structural*, *Functional*, *Set*, or *Reset* tab.
2. Choose *View – Name Format* to choose a *Full* or *Short* formats:

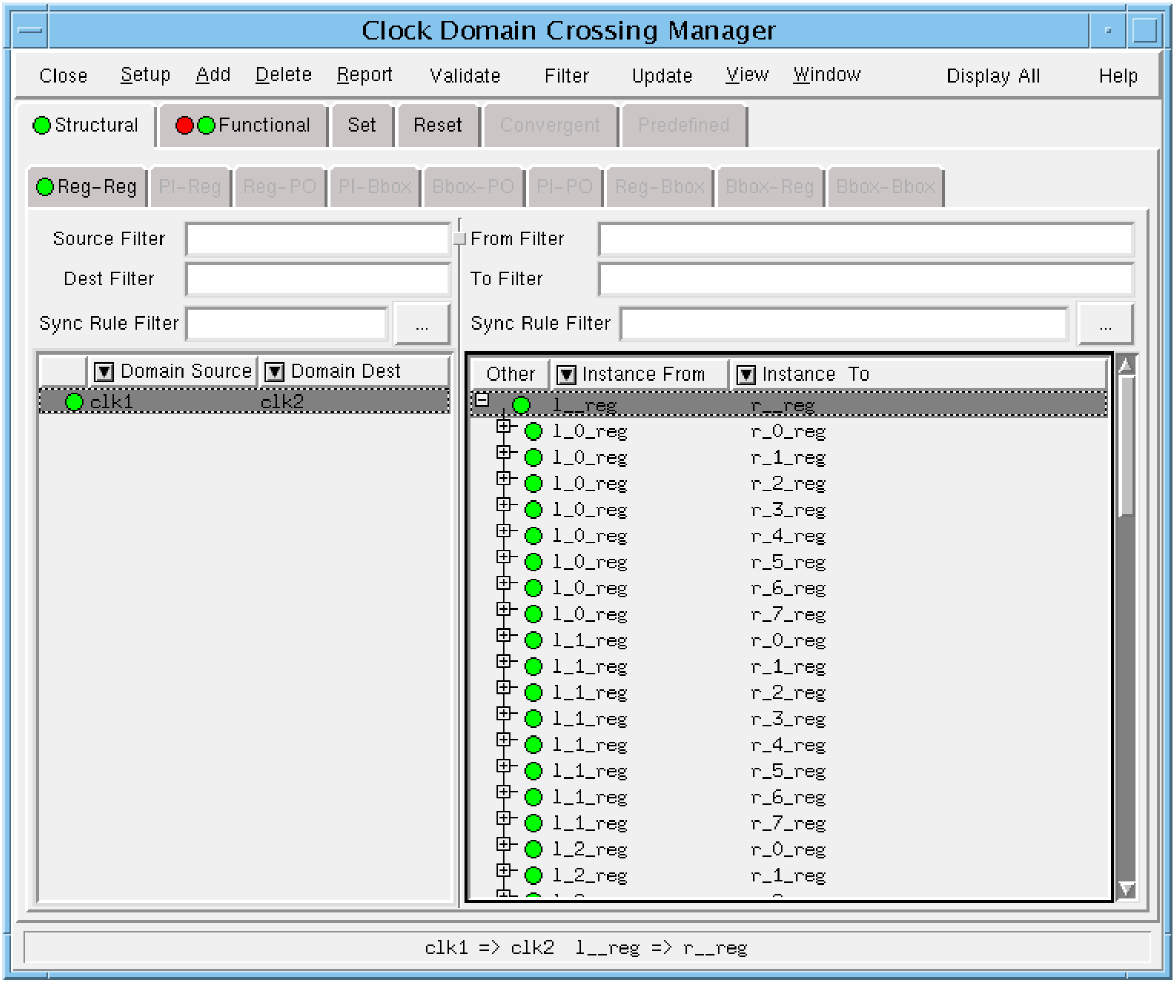
Display Information Box

By default, when you rest the cursor over an entry in the *Domain Source/Destination* or *Instance From/To* section of the window, Conformal Extended Checks displays an information box that identifies the object by name and lists pertinent details about it. You can toggle this feature on or off.

➤ Choose *View – Display Information Box*.

Working in the Structural Tab

Use the *Structural* tab of the Clock Domain Crossing window to access information on Structural CDC checks. For information on the layout of the *Structural* tab, see Display Area on page 203.



	Domain Source	Domain Dest
●	clk1	clk2

Other	Instance From	Instance To
⊟ ●	l__reg	r__reg
⊞ ●	l_0_reg	r_0_reg
⊞ ●	l_0_reg	r_1_reg
⊞ ●	l_0_reg	r_2_reg
⊞ ●	l_0_reg	r_3_reg
⊞ ●	l_0_reg	r_4_reg
⊞ ●	l_0_reg	r_5_reg
⊞ ●	l_0_reg	r_6_reg
⊞ ●	l_0_reg	r_7_reg
⊞ ●	l_1_reg	r_0_reg
⊞ ●	l_1_reg	r_1_reg
⊞ ●	l_1_reg	r_2_reg
⊞ ●	l_1_reg	r_3_reg
⊞ ●	l_1_reg	r_4_reg
⊞ ●	l_1_reg	r_5_reg
⊞ ●	l_1_reg	r_6_reg
⊞ ●	l_1_reg	r_7_reg
⊞ ●	l_2_reg	r_0_reg
⊞ ●	l_2_reg	r_1_reg

```
clk1 => clk2  l__reg => r__reg
```

Note: The Clock Domain Crossing Manager displays a red or green circle to indicate whether a domain crossing generated a violation. If you prefer, Conformal displays check marks (red X or green check).

Adding a Synchronization Rule for a Single Domain or Instance

This process opens the Synchronization Rule window with paths automatically entered in the appropriate fields.

1. Click the *Structural* tab.

2. Click the tab that corresponds to the desired clock domain category (for example, *REG-REG*).
3. Click a domain or instance to select it.
4. Right-click and choose *Add Synchronization Rule* from the pop-up menu.
5. See Synchronization Rule on page 218 to complete the process.

Deleting CDC Checks

From the *Structural* tab of the Clock Domain Crossing Manager, right-click on a domain, instance, or bit to select it, and choose *Delete CDC Check* from the pop-up menu.

Reporting Validated Data for a Single Domain, Instance, or Bit

To view a verbose validated data report, use the following process. The Transcript window of the main GUI window displays the report.

1. Click the *Structural* tab.
2. Click the tab that corresponds to the desired clock domain category (for example, *REG-REG*).
3. Click a domain, instance, or bit to select it.
4. Right-click and choose *Report Validated Data* from the pop-up menu.

Viewing a Schematic of an Instance

This process opens a Flattened Schematic window that displays the selected instance.

1. Click the *Structural* tab.
2. Click the tab that corresponds to the desired clock domain category (for example, *REG-REG*).
3. Click an instance or bit to select it.
4. Right-click and choose *Schematics* from the pop-up menu.

Diagnosing Structural Domain Crossing Violations

To diagnose structural domain crossing violations:

1. Choose *Tools – Clock Domain Crossing* to open the Clock Domain Crossing Manager.
2. Click the *Structural* tab.
3. Click the tab that corresponds to a domain category (for example, *REG-REG*) to view a list of violations.
4. In the domain display area, click the domain involved in the violation.
5. In the instance display area, click an instance to select it. In some cases, you will need to further expand an instance to the bit level.

   Conformal Extended Checks highlights the instance in ivory.
6. Right-click and choose *Diagnose* from the pop-up menu.

   The schematic view of the violation appears. Conformal Extended Checks displays logic elements belonging to different domains with different colors so that it is easier to identify what logic is in which domain. This also helps diagnose mixed-domain errors, where there can be a path from a third domain signal to the CDC path.

   ❑ Source domains are displayed in green.

   ❑ Destination domains are displayed in blue.

   ❑ Error domains are displayed in red.

   ❑ Additional domains are displayed in either purple, yellow, cyan, orange, and gray.

Working in the Functional Tab

Use the *Functional* tab of the Clock Domain Crossing window to access information on the Functional CDC checks. For information on the layout of these tabs, see Display Area on page 203.

Reporting Validated Data for a Single Domain, Instance, or Bit

To view a verbose Validated Data Report, use the following process. The Transcript window of the main GUI window displays the report.

1. Click the *Functional* tab.
2. Click on the desired sub-tab.

3. Click a domain, instance, or bit to select it.
4. Right-click and choose *Report Validated Data* from the pop-up menu.

Deleting CDC Checks

From the *Functional* tab of the Clock Domain Crossing Manager, right-click on a domain, instance, or bit to select it, and choose *Delete CDC Check* from the pop-up menu.

Diagnosing a Failed Instance

This process opens a Flattened Schematic window that displays the selected instance and the Waveform window, which allows you to view relationships between clocks. It also returns a counterexample of the failed check in the Transcript window of the main GUI window. (The counterexample shows the time period during which instability occurred.) You can also access the Source Code Manager during diagnosis.

1. Click the *Functional* tab.
2. Click on the desired sub-tab.
3. In the domain display area, click the domain involved in the violation.
4. In the instance display area, click an instance to select it.

   In some cases, you will need to further expand an instance to the bit-level. Conformal Extended Checks highlights the instance in ivory.
5. Right-click and choose *Diagnose* from the pop-up menu.

Viewing a Schematic of an Instance

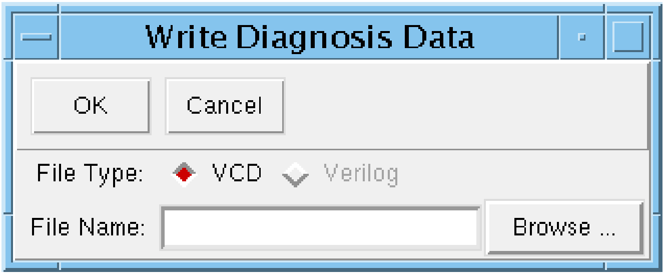
The following procedure opens a Flattened Schematic window that displays the selected instance.

1. Click the *Functional* tab.
2. Click on the desired sub-tab.
3. Click an instance or bit to select it.
4. Right-click and choose *Schematics* from the pop-up menu.

Saving Diagnosis Data

After executing the *Diagnose* command, save the diagnosis data to a VCD file. If you do not specify a file name in step 3, Conformal Extended Checks will assign a filename. For future reference, note the name of the VCD file, which is displayed in the Transcript window of the main GUI window.

1. Click a path in the Instance From/To section of the window to select it.
2. Right-click and choose *Write CDC Check* from the pop-up menu to open the Write Diagnosis Data window.



File Type: ◆ VCD ◇ Verilog

File Name:

3. Do one of the following:

   Enter a filename in the *File Name* field or Click the *Browse* button to select a file.

4. Click *OK*.

Diagnosing Functional Domain Crossing Violations

1. Choose *Tools – Clock Domain Crossing* to open the Clock Domain Crossing Manager.
2. Click the *Functional* tab.
3. Click one of the tabs.
4. In the domain display area, click the domain involved in the violation.
5. In the instance display area, click an instance to select it.

   In some cases, you will need to further expand an instance to the bit-level.

6. Right-click and choose *Diagnose* from the pop-up menu.

   This opens the Waveform window to view relationships between clocks, and a schematic view of the violation. Additionally, diagnosis data is written to the Transcript window.

   Note: In order to analyze the data stability as it moves from one clock domain to the other, data stability points are shown in purple.

Working in the Set, Reset, or Convergent Tabs

Use the *Set*, *Reset*, or *Convergent* tab of the Clock Domain Crossing window to access information on these types of CDC checks. For information on the layout of these tabs, see Display Area on page 203.

Reporting Validated Data for a Single Domain, Instance, or Bit

To view a verbose Validated Data Report, use the following process. The Transcript window of the main GUI window displays the report.

1. Click the *Set*, *Reset*, or *Convergent* tab.
2. Click on the desired sub-tab.
3. Click a domain, instance, or bit to select it.
4. Right-click and choose *Report Validated Data* from the pop-up menu.

Deleting CDC Checks

From the *Set*, *Reset*, or *Convergent* tab of the Clock Domain Crossing Manager:

➤ Right-click on a domain, instance, or bit to select it and choose *Delete CDC Check* from the pop-up menu.

Diagnosing a Failed Instance

This process opens a Flattened Schematic window that displays the selected instance. You can also access the Source Code Manager during diagnosis.

1. Click the *Set*, *Reset*, or *Convergent* tab.
2. Click on the desired sub-tab.
3. In the domain display area, click the domain involved in the violation.
4. In the instance display area, click an instance to select it.

   In some cases, you will need to further expand an instance to the bit-level. Conformal Extended Checks highlights the instance in ivory.
5. Right-click and choose *Diagnose* from the pop-up menu.

Viewing a Schematic of an Instance

The following procedure opens a Flattened Schematic window that displays the selected instance.

1. Click the *Set*, *Reset*, or *Convergent* tab.
2. Click on the desired sub-tab.
3. Click an instance or bit to select it.
4. Right-click and choose *Schematics* from the pop-up menu.

Diagnosing Set, Reset, and Convergent Domain Crossing Violations

Use this procedure to diagnose Set, Reset, and Convergent domain crossing violations.

1. Choose *Tools – Clock Domain Crossing* to open the Clock Domain Crossing Manager.
2. Click the *Set* or *Reset* tab.
3. In the domain display area, click the domain involved in the violation.
4. In the instance display area, click an instance to select it. (In some cases, you will need to further expand an instance to the bit-level.)

   Conformal Extended Checks highlights the instance in ivory.
5. Right-click and choose *Diagnose* from the pop-up menu.

   The schematic view of the violation appears. Conformal Extended Checks displays logic elements belonging to different domains with different colors so that it is easier to identify what logic is in which domain. This also helps diagnose mixed-domain errors, where there can be a path from a third domain signal to the CDC path.

Working in the Predefined Tab

Use the *Predefined* tab of the Clock Domain Crossing Manager to view the predefined synchronization rules.

This tab has three sections:

- Predefined synchronization rule display—The left-most section of this tab displays a list of all the available predefined synchronization rules. As you select a predefined synchronization rule, its specifications appear in the domain and instance display area.

- Domain display—Lists domains for a selected category (for example, REG-REG). The domain display area includes columns that identify *Domain Source* and *Domain Destination*.
- Instance display—When you select a domain (in the domain display), the instance display to the right shows expanded information for the selected domain. The instance display area includes columns that identify *Instance From* and *Instance To*.

This tab is available if you are in the *categorization* flow, which means you do not have any user-defined synchronization rules in your design. For information on synchronization rules and the categorization flow, see Specifying Synchronization Rules on page 182.

For more information:

- See Reviewing Validated Data on page 190 for information on how paths get classified into the different synchronization rules.
- See Displaying Predefined Synchronization Rules on page 183 for information on how to display the predefined synchronization rules using the command line.
- See Predefined Synchronization Rule on page 215 for information on how to customize predefined synchronization rules using the *Setup* menu option, or see Customizing Predefined Synchronization Rules on page 186 for information on how to customize the predefined synchronization rules using the command line.

Reporting Validated Data for a Single Domain, Instance, or Bit

To view a verbose Validated Data Report, use the following process. The Transcript window of the main GUI window displays the report.

1. Click the *Predefined* tab.
2. Click on the desired sub-tab.
3. Click a domain, instance, or bit to select it.
4. Right-click and choose *Report Validated Data* from the pop-up menu.

Deleting CDC Checks

From the *Predefined* tab of the Clock Domain Crossing Manager:

➤ Right-click on a domain, instance, or bit to select it and choose *Delete CDC Check* from the pop-up menu.

Viewing a Schematic of an Instance

The following procedure opens a Flattened Schematic window that displays the selected instance.

1. Click the *Predefined* tab.
2. Click on a sub-tab.
3. Click an instance or bit to select it.
4. Right-click and choose *Schematics* from the pop-up menu.

10

FSM Checks

Overview

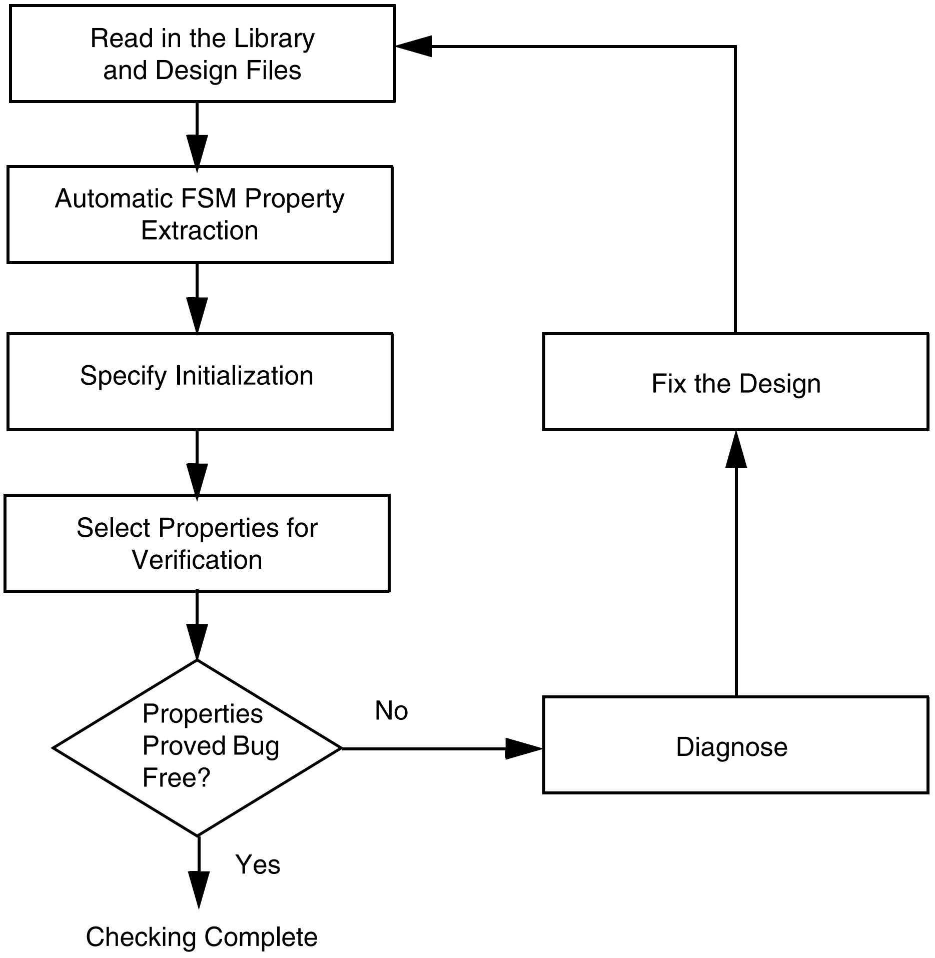
Designers frequently use Finite State Machines (FSMs) in control logic. The Conformal Extended Checks software applies formal techniques for verification of standard FSMs to ensure they are coded correctly and efficiently.

Conformal Extended Checks automatically extracts FSMs when reading in the design. This automatic extraction obtains the state variables, valid states, and possible transitions between states. The software checks the following properties:

- FSM Reachability—the FSM property will pass if the corresponding state can be reached from the initial state.
- FSM Transition—the FSM Transition property will pass if the transition can really happen.
- FSM Deadlock—the FSM Deadlock property will pass if the FSM can always exit the corresponding state.

FSM Implementation

Conformal FSM checks occur within the normal Conformal Extended Checks session flow. The flow relevant to FSM property checking is depicted in the following figure.


Read in the Library and Design Files
Automatic FSM Property Extraction
Specify Initialization
Select Properties for Verification
Properties Proved Bug Free?
No
Yes
Checking Complete
Diagnose
Fix the Design

Special Handling – Reset Constraints

The Conformal software normally performs Functional checks on a design when it is in functional mode, and constraints (called the reset constraints) are used to keep a design in functional mode. However, for certain types of Functional checks, including FSM Transition and FSM Reachability, applying reset constraints to the design actually invalidates some of the results. Ideally, for these types of checks, the software should ignore these reset constraints. Use the `ADD IGNORE RESET_CONSTRAINT` command to ignore reset constraints for the specified checks.

Note: Other checks such as bus, dont_care, and assertion, still require the reset constraints and are not affected by this command.

Proof Status

By default, Conformal proves FSM properties sequentially (`-time_unit infinity`). After Conformal proves the added properties, it returns a summary of the outcome of the proof. The following table lists the possible status indicators with short definitions. For additional information about how Conformal produces verification results, see Chapter 8, "Predefined Checks.".

Proof Status	Definition
Pass	Indicates properties that hold true from initialization to infinity.
Pass – Depth M	Indicates properties that passed to the extent of the target you specified. For example, If you specify the following for a particular proof: `-time_unit 100` and Conformal returns a Pass and notes a depth of 100, which means that the specified properties passed up to and including time 100. The software did not attempt to prove the specified property beyond time 100.
Fail	Indicates properties that did not hold true at some point during the proof. This indicates the time, counting from zero (immediately after initialization), at which it proved to be a failed property.
Fail – Depth M	Indicates properties that failed within the target you specified, which is noted on the report as the depth. For example: `prove -time_unit 5` This checks this property through time 5. The software reports Fail with a depth of 5, which indicates that up through depth 5, the desired behavior was not present. This status applies to FSM Transition and FSM Reachability properties.

Proof Status	Definition
Explored – Depth N	Indicates inconclusive results. The software explored this property up to and including the depth noted on the report. This property requires further investigation and debugging. For example, if you specify the following for a particular proof: `-time_unit 100` and the software is able to prove a property up to and including time 80, Conformal reports the applicable property as Explored and notes the depth as 80. This indicates that the status is unresolved and this property requires further investigation. The software did not explore the specified property beyond time 80.
Not Run (NR)	Indicates properties that were not processed.

Reporting Conformal FSM Checks

For a description of available property report formats, see Property Reports on page 169.

Reporting Extracted FSM Properties

Conformal Extended Checks automatically extracts FSMs when reading in a design. In addition to the general property reports, use the `REPORT FSM` command to list extracted FSM properties during Setup or Verify mode.

Diagnosis

Modeling Rule Violations

When entering Verify mode, the Conformal software automatically performs Modeling Rule Checks. Use the `REPORT RULE CHECK` command to report a list of violations.

Diagnosing and Debugging Properties

Use Conformal Extended Checks to prove that all the FSM states can be reached, transitioned to and from, and that there is no deadlock state. The FSM proof is sequential in nature and covers the entire design space. Use the Conformal Extended Checks commands to diagnose and debug FSM properties just as you would other property types. Use the integrated diagnosis tools, including the FSM integrated tools to explore FSM issues.

For more about these tools and their features, see FSM Manager on page 245 and State Diagram Window on page 261.

Diagnosing FSM Deadlock Properties

To initiate diagnosis for a failed Deadlock property, you can use the pop-up menu diagnosis process. The Diagnose feature does the following:

- Executes the `DIAGNOSE` command (look for the counterexample in the Transcript window of the main window)
- Opens the State Diagram window (see State Diagram Window on page 261).
- For FSM Deadlock properties with a depth greater than 0, opens the Waveform window.

You can also open the Source Code Manager and Schematic Viewer from the pop-up menu to aid in diagnosis.

For more information, see Diagnosing FSM Deadlock Properties on page 244.

Diagnosing FSM Reachability and Transition Properties

To initiate diagnosis for a passing Reachability or Transition property, you can use the pop-up menu diagnosis process (see Diagnosing an FSM Property on page 258). The Diagnose feature acts in the following way:

- Executes the `DIAGNOSE` command.
- Opens the State Diagram window (see State Diagram Window on page 261).
- For FSM Reachability and Transition properties with a depth greater than 0, opens the Waveform window.

You can also access the Source Code Manager and Schematic Viewer from the pop-up menu to aid in diagnosis.

Debugging Reachability and Transition Properties

For failed FSM Reachability and Transition properties, selecting *Diagnose* on the pop-up menu opens the Source Code Manager and State Diagram window. Debugging FSM Reachability and Transition properties is similar to debugging Branch Enable properties. That is, because a fail means that something did *not* happen, the software does not display any vectors or waveforms. To debug a failed Reachability or Transition property, do the following:

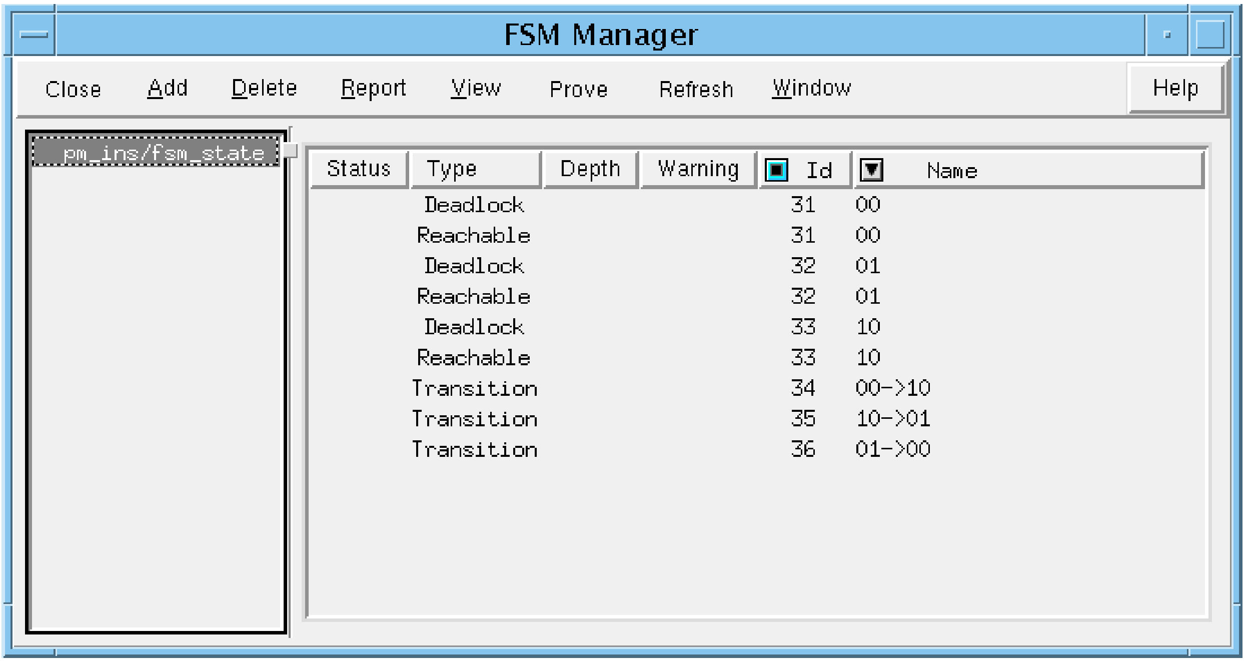
1. Select a failed property.
2. Right-click and choose *Diagnose* from the pop-up menu to open the Source Code Manager.
3. Examine the source code for the origin of FSM failures.
4. Right-click and choose *Schematics* or *Multi-Timeframe Schematics* to determine the origin of failures.

FSM Manager

The FSM Manager displays finite state machines information. To access the FSM Manager, click on *Tools* on the menu bar in the main window and choose *FSM Property*. This window is an integrated environment that allows you to:

- Select all or specified categories (Transition, Reachability, or Deadlock) to include or exclude them from the proof process
- Select specified properties to include or exclude them from the proof process
- Report FSM property data
- Invoke the Conformal Extended Checks proof engine

- Access integrated diagnosis tools (for example, the Waveform Viewer and the Schematic Viewer)



Status	Type	Depth	Warning	Id	Name
	Deadlock			31	00
	Reachable			31	00
	Deadlock			32	01
	Reachable			32	01
	Deadlock			33	10
	Reachable			33	10
	Transition			34	00->10
	Transition			35	10->01
	Transition			36	01->00

The FSM Manager includes a menu bar and two display areas. The display area on the left lists all the extracted FSMs. The display area on the right shows individual FSM properties for the highlighted FSM.

Menu Bar

The FSM Manager menu bar includes the following:

- *Close*
- Add
- Delete
- Report
- View
- Prove!
- Refresh!

- *Window*
- *Help*

While the *Close*, *Window*, and *Help* buttons are standard throughout the Conformal GUI windows, procedures related to the remaining drop-down menus are described below.

Add

In the FSM Manager use the *Add* drop-down menu for the following purposes:

- exclude properties from the prove list
- add properties to the prove list

For greater control over which FSMs and which properties are added, see Accessing the Pop-Up Menu Commands on page 254.

Note: When adding specified FSMs or FSM properties to the prove list, you cannot designate these same properties as “ignored” until you delete them from the prove list. Likewise, when you designate specified properties as ignored properties, you cannot add them to the prove list until you make them available. Use the Delete drop-down menu to delete ignored and static properties.

The correct sequence of commands is as follows:

1. Add – Ignored Property to exclude selected properties from the “Prove” list.
2. Add – Static Property to add properties to the “Prove” list.

Adding or Ignoring All Properties

Use the following procedure to add all properties to the “Prove” list or ignore them during the proof:

1. Click *Add* on the menu bar.
2. Click *Static Property – All* or *Ignored Property – All*.

If you choose *Static Property* in step 2, Conformal displays the properties with a circled red question mark to represent their not run status.

If you choose Ignored Property in step 2, Conformal displays the properties with a circled (i) to represent their ignored status.

Delete

In the FSM Manager use the *Delete* drop-down menu to reinstate all or specified ignored FSM properties, or remove all or specified FSM properties from the prove list

To exert greater control over *which* FSM properties are deleted, see Accessing the Pop-Up Menu Commands on page 254.

Note: When you add specified properties to the prove list, you cannot designate these same properties as “ignored” until you delete them from the prove list. Likewise, when you designate specified properties as ignored properties, you cannot add them to the prove list until you make them available. Use the Delete drop-down menu to delete ignored and static properties.

Deleting Properties and Removing Ignored Status

Use the following procedure to delete all properties from the “Prove” list or remove their ignored status.

1. Click *Delete* on the menu bar.
2. Click *Static Property – All* or *Ignored Property – All*.

This removes the corresponding status symbols from all FSMs and FSM properties. However, the FSM property names remain.

Deleting Specified Properties and Removing Ignored Status

Use the following procedure to delete specified properties from the “Prove” list or remove ignored status.

1. Click *Delete* on the menu bar.
2. Click *Static Property* or *Ignored Property* and one category of FSM properties.

   This removes the corresponding status symbols from the specified property category for all FSMs. However, the FSM property names remain.

Report

After adding one or more FSM properties, use the *Report* drop-down menu to configure and execute reports. Use the following procedures, which are organized according to discrete menu choices. To view reports for individual or multiple properties, refer to Pop-Up Menu Commands on page 254.

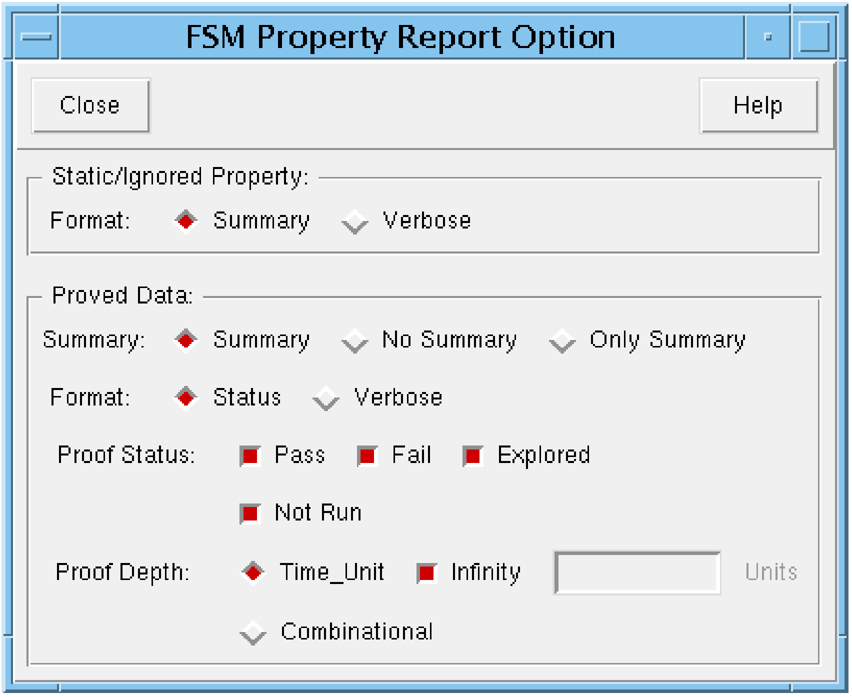
For detailed information about proof results and report formats, see Property Proofs on page 166 and “Property Reports” on page 169.

Viewing a Static Property or Ignored Property Report

After adding properties, you can specify and view a report. The Static Property report displays data on all properties currently included on the “Prove” list. The Ignored Property report displays data on all properties that have been excluded from the “Prove” list.

Note: This window includes a Static/Ignored Property section and a Proved Data section. For this process, use the Static/Ignored Property portion of the window.

1. Click the *Report* drop-down menu.
2. Click *Option* to open the FSM Property Report Option window.



Static/Ignored Property:

Format: ● Summary ○ Verbose

Proved Data:

Summary: ● Summary ○ No Summary ○ Only Summary

Format: ● Status ○ Verbose

Proof Status: ☑ Pass ☑ Fail ☑ Explored ☑ Not Run

Proof Depth: ● Time_Unit ☑ Infinity Units ○ Combinational

3. Click a *Format* check box:
4. Click *Close*.
5. From the *Report* drop-down menu, click *Static Property* or *Ignored Property*.
6. Choose *All*, *All Deadlock*, *All Reachable*, or *All Transition*.

   The specified report appears in the Transcript window of the main window.

Viewing a Proved Data Report

Note: This window includes a Static/Ignored Property section and a Proved Data section. For this process, use the Proved Data portion of the window.

After adding properties to the "Prove" list (Add – Static Property) and running the `PROVE` command (see "Prove!" on page 252), specify and view Proved Data reports with the following procedure.

1. Click the *Report* drop-down menu.
2. Click *Option*.

   The FSM Property Report Option window.

   Perform Steps 3 through 6 in the Proved Data section.
3. Click a *Summary* preference.
4. Click a *Format* type.
5. Click a *Proof Status*.

   By default, Conformal Extended Checks includes all properties in the Proved Data reports. Click a radio button to de-select it and exclude the specified status type from the report. (For example, de-select the Pass radio button if you do not want to include properties that have a Pass proof status.)
6. Click *Proof Depth* selections:
   - *For Sequential Properties:*
     - Select *Time_Unit* and *Infinity.*
     - De-select *Infinity* and enter an integer in the *Units* field.
   - *For Combinational Properties*, select *Combinational*.
7. Click *Close*.
8. Click the *Report* drop-down menu.
9. Click *Proved Data* and choose *All*, *All Deadlock*, *All Reachable*, or *All Transition*.

   The specified report appears in the Transcript window of the main window.

View

Configure the FSM Manager display to suit your needs using the *View* drop-down menu and the following procedures.

Specifying Name Format

Use the following procedure to specify name formats in the FSM Manager.

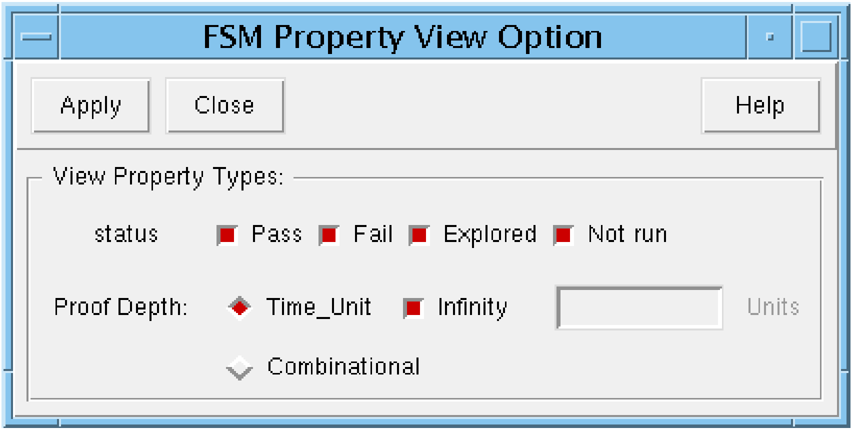
1. Click the *View* on the menu bar.
2. Click *Name Format*.
3. Click a check box to select the desired format.

   The default is Full.

Specifying View Options

The following procedure allows you to configure the FSM Manager display. This function is especially useful when you encounter a large number of FSM properties.

1. Click the *View* button on the menu bar.
2. Click *Option* to open the FSM Property View Option window.



View Property Types:

status ☒ Pass ☒ Fail ☒ Explored ☒ Not run

Proof Depth: ● Time_Unit ☒ Infinity Units

○ Combinational

The FSM Property View Option window offers configuration options. In the View Property Types section, make selections with radio buttons and check boxes as follows.

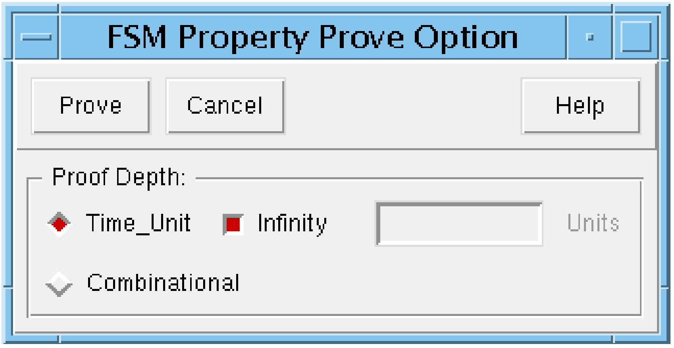
3. For *Status*, click the radio buttons to select specific status types for display.
4. For *Proof Depth*, click the depth.

- ❑ For Sequential Properties, do one of the following:
  - ❍ *Time_Unit* and *Infinity*
  - ❍ De-select *Infinity* and type an integer in the *Units* field.
- ❑ For Combinational Properties, click *Combinational*.

Prove!

After adding properties, you can initiate the proof. Click the *Prove!* button to open the FSM Property Prove Option window. Use this window to specify and execute a proof.

When Conformal completes the proof process, the status of each FSM property on the prove list is identified. See “Display Area and Pop-Up Menu” on page 253.



Proof Depth:

◆ Time_Unit ☒ Infinity Units

◇ Combinational

Proving FSM Properties Sequentially

Use the following procedure to prove FSM properties sequentially with depth infinity or a specified depth.

1. Click the *Prove!* button located on the menu bar of the FSM Manager.

   The FSM Property Prove Option window opens.

2. In the *Proof Depth* section, click *Time_Unit*.

   This check box is selected by default.

3. Do one of the following:
   - ❑ Ensure that the *Infinity* check box is selected.
   - ❑ Click Infinity to de-select it and enter the specified time units in the *Units* field.

4. Click *Prove*.

Proving FSM Properties Combinationally

1. Click the *Prove!* button located on the menu bar of the FSM Manager.

   The FSM Property Prove Option window opens.

2. In the *Proof Depth* section, click *Combinational*.
3. Click *Prove*.

Refresh!

Click *Refresh!* to refresh the contents of the FSM Manager.

Note: If you change the Pass/Fail Icon preferences, the Conformal software prompts you to refresh the window.

Display Area and Pop-Up Menu

The FSM Manager display area on the right includes the following column headings:

- Deadlock, Reachable, and Transition
  Display a status indicator
- Depth
  Displays the proof depth integer, infinity, or combinational.
- Warning
- Id
  Number assigned by the Conformal software
- Name

For additional information about property checking, refer to Chapter 8, "Predefined Checks."

Each FSM property is listed in the display area with its status. The status is one of the following:

🔴 ✗	red-filled circle (or red X)	Fail and Fail – Depth M
🟢 ✓	green-filled circle (or green check)	Pass and Pass – Depth M
?	circled red question mark (?)	Not Run

🟡	yellow-filled circled	Explored – Depth N
ⓘ	circled i	Ignored

To Sort Instances

▼■ Click the triangular sort icon in the Depth, ID, or Name column headings to sort instances numerically by ID number or alphabetically by name. The square icon indicates the column that was sorted. (The complementary columns are rearranged in the process.)

Pop-Up Menu Commands

Use the pop-up menu to perform the commands detailed below on the specified FSMs or FSM properties.

Selecting Properties

Use any of the following procedures to select properties:

- Click a property to select it.
- Click the first property in a group, depress and hold the Shift key, and click the final property in a group to select the entire group.
- Click the first property in a group and drag the cursor to the final property in a group to select the entire group.
- Depress the `Ctrl`-key and click a property to add it to the selected group.

Accessing the Pop-Up Menu Commands

To add, delete, or report properties, or to work with the integrated debugging tools relevant to specified FSMs or FSM properties, follow this general process for accessing the pop-up menu.

1. Click an FSM to view properties in the display area on the right.
2. Select an FSM property in the display area on the left or one or more FSM properties on the right.
3. Right-click to open the pop-up command menu.

Many of the following commands access the integrated diagnosis tools. To learn more about the specific tools, please refer to the relevant information in Chapter 3, "Using the Graphical User Interface."

Adding a Single FSM

When adding a single FSM to the "Prove" list (Static Property) or excluding an FSM (Ignored Property), a circled red (?) or a circled (i) appears in the appropriate column (Deadlock, Reachable, or Transition). This represents not run and ignored, respectively.

1. Click an FSM in the display area on the left to select it.
2. Right-click and choose *Add* from the pop-up menu.
3. Click *Static Property* or *Ignored Property*.

Adding a Single FSM Property

When adding a single FSM property to the "Prove" list (Static Property) or excluding a property (Ignored Property), a circled red (?) or a circled (i) appears in the appropriate column (Deadlock, Reachable, or Transition). This represents not run and ignored, respectively.

1. Click an FSM in the display area on the left.
2. Click an FSM property in the display area on the right to select it.
3. Right-click and choose *Add* from the pop-up menu.
4. Click *Static Property* or *Ignored Property*.

Adding Multiple FSM Properties

When adding multiple FSM properties to the "Prove" list (Static Property) or excluding them (Ignored Property), a circled red (?) or a circled (i) appears in the appropriate column (Deadlock, Reachable, or Transition) for each of the selected properties. This represents not run and ignored, respectively.

1. Click an FSM in the display area on the left.
2. Select multiple properties in the display area on the right with one of the methods described above. (See "Selecting Properties" on page 254).
3. Right-click and choose *Add* from the pop-up menu.
4. Click *Static Property* or *Ignored Property*.

Deleting a Single FSM

When deleting a single FSM from the “Prove” list (Static Property) or reinstating it (Ignored Property), the status symbol is removed from all the FSM properties in the display area to the right of the FSM. However, the instance identification numbers and names remain.

1. Click an FSM in the display area on the left.
2. Right-click and choose *Delete* from the pop-up menu.
3. Click the applicable option: *Static Property* or *Ignored Property*.

Deleting a Single FSM Property

When deleting a single FSM property from the “Prove” list (Static Property) or reinstating it (Ignored Property), the status symbol is removed from the specified FSM property. However, the FSM property identification number and name remain.

1. Click an FSM in the display area on the left.
2. Click an FSM property in the display area on the right.
3. Right-click and choose *Delete* from the pop-up menu.
4. Click the applicable option: *Static Property* or *Ignored Property*.

Deleting Multiple FSM Properties of a Specified FSM

When deleting multiple FSM properties from the “Prove” list (Static Property) or reinstating them (Ignored Property), the status symbol is removed from the specified FSM properties listed in the display area on the right. However, the FSM property identification numbers and names remain.

1. Click an FSM in the display area on the left.
2. Select multiple FSM properties in the display area on the right with one of the methods described above. (See “Selecting Properties” on page 254).
3. Right-click and choose *Delete* from the pop-up menu.
4. Click the applicable option: *Static Property* or *Ignored Property*.

Reporting Data on an Individual FSM

After adding an FSM to the “Prove” list (Static Property) or excluding it from the list (Ignored Property), you can display a property data report.

The report appears in the Transcript window of the main window configured according to the current Report settings. (Refer to "Report" on page 248.)

Use the following procedure:

1. Click an FSM in the display area on the left.
2. Right-click and choose *Report* from the pop-up menu.
3. Click the desired report type.

Reporting Data on an Individual FSM Property

After adding an FSM property to the "Prove" list (Static Property) or excluding it from the list (Ignored Property), you can display a property data report.

The report appears in the Transcript window of the main window configured according to the current Report settings. (Refer to "Report" on page 248.)

Use the following procedure:

1. Click an FSM in the display area on the left.
2. Click an FSM property in the display area on the right to select it.
3. Right-click and choose *Report* from the pop-up menu.
4. Click the desired report type.

Reporting Data on Multiple FSM Properties

After adding FSM properties to the "Prove" list (Static Property) or excluding them from the list (Ignored Property), you can display a property data report.

The report appears in the Transcript window of the main window configured according to the current Report settings. (Refer to "Report" on page 248.)

Use the following procedure:

1. Click an FSM in the display area on the left.
2. Select multiple FSM properties in the display area on the right with one of the methods described above. (See "Selecting Properties" on page 254).
3. Right-click and choose *Report* from the pop-up menu.
4. Click the desired report type.

Diagnosing an FSM Property

After the prove process, use this procedure to diagnose an FSM property. See the following sections for additional information:

- “Diagnosing FSM Deadlock Properties” on page 244
- “Diagnosing FSM Reachability and Transition Properties” on page 244
- “Debugging Reachability and Transition Properties” on page 244

Use the following procedure:

1. Click an FSM property in the display area on the right.
2. Right-click and choose *Diagnose* from the pop-up menu.

Generating a State Diagram of an FSM

or an orientation to the State Diagram window, refer to “State Diagram Window” on page 261.

1. Click an FSM in the display area on the left.
2. Right-click and choose *State Diagram* from the pop-up menu.

Generating a State Diagram of an FSM Property

For an orientation to the State Diagram window, refer to “State Diagram Window” on page 261.

1. Click an FSM in the display area on the left.
2. Click a property in the display area on the right to select it.
3. Right-click and choose *State Diagram* from the pop-up menu.

Displaying Related Source Code

When diagnosing properties with the integrated debugging tools, you can move from one diagnostic tool to another. The following process opens the Source Code Manager in a context-dependent mode.

1. In the FSM Manager (FSM or FSM property display), State Diagram, Flattened Schematics, Waveform Display, or Schematic window, left-click an FSM property to select it.

2. Right-click to open the pop-up command menu.
3. Choose *Source Code* to open the Source Code Manager scrolled to the relevant line of code. The appropriate location is highlighted.

Opening the Multi-Timeframe Schematics

1. Click an FSM in the display area on the left.
2. Click an FSM property in the display area on the right to select it.
3. Right-click to open the pop-up command menu.
4. Choose *Multi-Timeframe Schematics.*

   The Schematic (Multi-Timeframe) window opens showing the fan-in cone of the selected property. The valuation of the counter-example is annotated on the circuit.

Opening the Waveform Viewer

For sequential properties with a depth greater than 0, the Waveform Viewer opens automatically when you choose Diagnose from the pop-up menu.

In addition to accessing the Waveform Viewer through the Diagnose option on the pop-up menu, you can use the following procedure.

1. Click an FSM in the display area on the left.
2. Click an FSM property in the display area on the right to select it.
3. Right-click to open the pop-up command menu.
4. Choose *Debug Waveform*.

Opening the Schematic Viewer

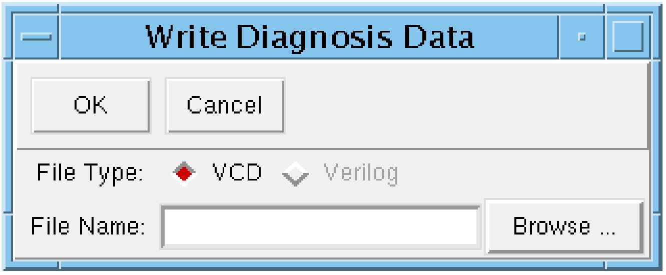
This procedure opens the viewer that displays a schematic representation of the selected FSM property.

1. Click an FSM in the display area on the left.
2. Click an FSM property in the display area on the right to select it.
3. Right-click to open the pop-up command menu.
4. Choose *Schematics* to access the viewer in the context-dependent mode.

Saving Diagnosis Data

After diagnosing a failed FSM property (see instructions above), you can save the diagnosis data to a file.

1. Click a property to select it.
2. Right-click and choose *Write Diagnosis Data* from the pop-up menu to open the Write Diagnosis Data window.



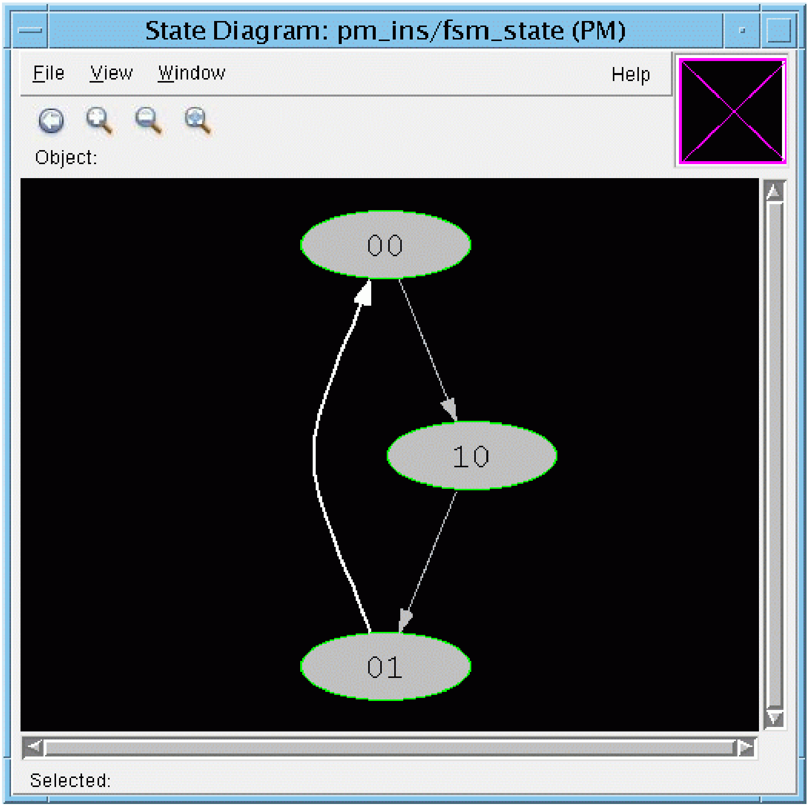
File Type: ● VCD ○ Verilog

File Name:

3. Left click a *File Type* check box.
4. Enter a name in the *File Name* field or click the *Browse* button to locate the appropriate file.
5. Click *OK*.

State Diagram Window

The State Diagram window provides an interactive diagnosis environment that links various Conformal diagnosis tools. It displays all the states and transitions in the FSM property.



View a diagram of an FSM before the proof process to explore and resolve any issues prior to executing the Prove process. After the Prove process, view a diagram as part of the diagnosis process. Error states are denoted with red ovals.

The State Diagram window includes the following sections:

- Menu Bar
- Toolbar
- Diagram Display
- Selected Object Field

Menu Bar

The menu bar includes the following drop-down menus:

- File
- View
- Window

The File and View drop-down menus are discussed in detail below.

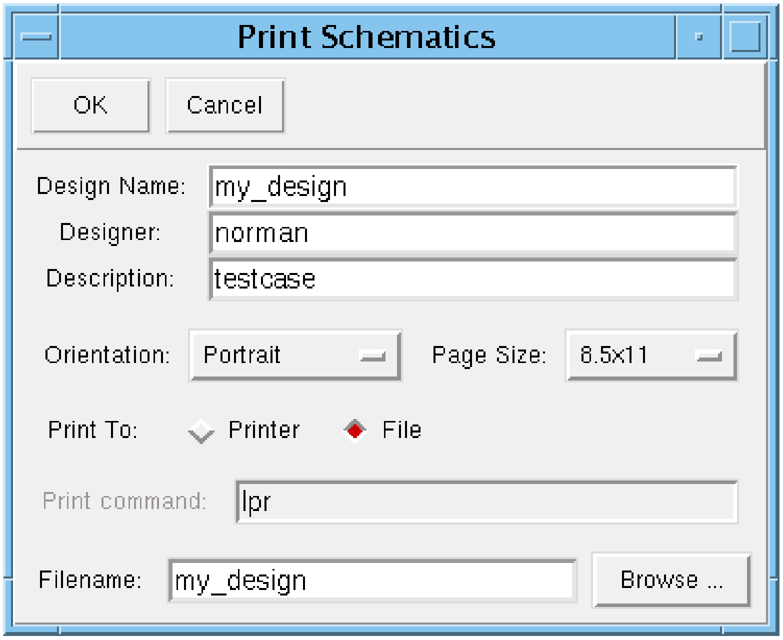
File

The File drop-down menu includes *Print*, *Report*, and *Close*.

Printing a Schematic

When printing a schematic, the information in the *Design Name*, *Designer*, and *Description* fields appears in the bottom left corner.

1. From the *File* drop-down menu, click *Print* to open the Print Schematic window.



Print Schematics

Design Name: my_design
Designer: norman
Description: testcase
Orientation: Portrait
Page Size: 8.5x11
Print To: ○ Printer ● File
Print command: lpr
Filename: my_design

2. Enter a name in the *Design Name* field.
3. Enter a name in the *Designer* field.

4. Enter a description in the *Description* field.
5. Select one of the following *Print To* destination.
   - ❑ If you select *Printer*, enter a print command in the appropriate field.
   - ❑ If you select *File*, enter a file name in the appropriate field or use the *Browse* button to locate the appropriate file through the Print File window.
6. Click *OK*.

Executing an FSM Report

Use the following procedure to execute the `REPORT FSM` command.

1. From the *File* drop-down menu, click *Report*.
2. View the report in the main window.

View

Use the View drop-down menu to specify a variety of display preferences.

Displaying Transition Names

By default, Conformal does not display Transition names in the State Diagram window. To see Transition names, choose *View – Transition Name* radio button.

Highlighting Selected Objects

By default Conformal highlights selected objects with a white outline. To disable or re-enable the highlighter: From the *View* drop-down menu, click the *Highlight Cursor Object* check box.

Execute Zoom Commands

From the *View* drop-down menu, click a zoom command.

Manipulate a Panoramic View

From the *View* drop-down menu, click *Pan* and select *Left*, *Right*, *Up*, or *Down*.

Display the Information Box

By default, when you rest the cursor over a state, Conformal displays an information box that identifies the state by name and lists the applicable FSM properties (Deadlock, Reachability, and Transition) and their statuses.

To disable or re-enable this feature, choose *View – Display Information Box*.

Toolbar

The following icons are located on the State Diagram window toolbar. With the window active, position the cursor over an icon to view its function. Some of the available icons serve as visually accessible short-cuts to *View* drop-down menu options.

Icon	Function	Definition
⬅	Last View	Return to the diagram view you previously specified.
🔍	Zoom In	Magnify a smaller area of the diagram with greater detail.
🔍	Zoom Out	Display a larger area of the diagram with less detail.
🔍	Zoom to Full	Display all the contents of the diagram.

Diagram Display

The diagram display area constitutes the major portion of this window.

Mouse Actions

With the State Diagram window active, use the mouse to perform various functions; for example, select and de-select portions of the diagram.

- Click an object:
  De-selects the previous selection and selects another object.

- Click in an area other than an object:
  De-selects all.
- Click and drag:
  Zooms in on the selected area.
- Right-click:
  Displays the pop-up menu.
- Rest the cursor over an object
  Displays an information box that identifies the state by name and lists the applicable FSM properties (Deadlock, Reachability, and Transition) and their statuses.

Keyboard Shortcuts

The State Diagram window features the following “shortcuts”. The “f” shortcut is case-sensitive.

Key	Function
+	zoom-in
-	zoom-out
f	zoom to full view
Up-arrow	move view up
Down-arrow	move view down
Page Up	move view up one page
Page Down	move view down one page
Left-arrow	move view left
Right-arrow	move view right

Pop-Up Menu Commands

With the State Diagram window active, position the cursor over an object and right click. This opens the pop-up menu, which you will use to perform the following actions. Refer to “Pop-Up Menu Commands” on page 254 for processes related to the following State Diagram window pop-up menu choices: *Diagnose*, *Source Code*, *Multi-Timeframe Schematics*, *Debug Waveform*, and *Schematics*.

Executing a Static Property Report on an FSM Property

After adding an FSM to the “Prove” list (Static Property) or excluding it from the list (Ignored Property), you can display the Static Property Report in the Transcript window of the main window.

Use the following procedure:

1. Click an FSM in the diagram area.
2. Right-click and choose *Report* from the pop-up menu.
3. Click *Static Property*.

Executing a Proved Data Report on an FSM Property

After Conformal proves an FSM, you can display a Proved Data report in the Transcript window of the main window.

Use the following procedure:

1. Click an FSM in the diagram area.
2. Right-click and choose *Report* from the pop-up menu.
3. Click *Proved Data*.

Selected Object Field

The Selected Object field is located near the bottom of the State Diagram window. When you select an object, the object type and ID number appear in this field.

11

FIFO Synchronizers

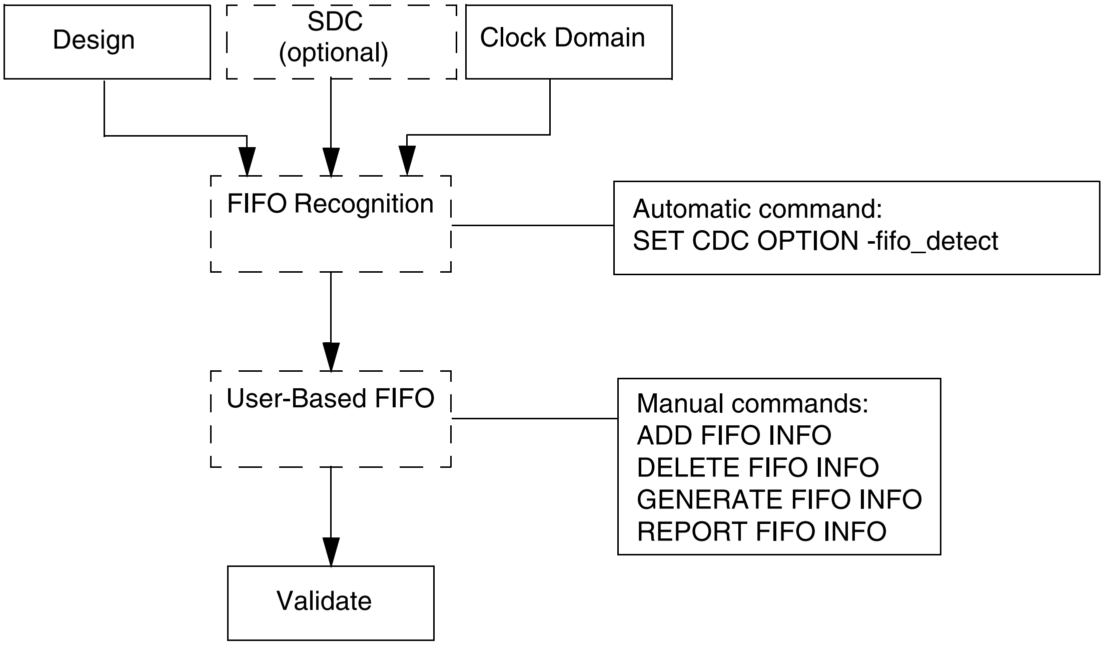
Overview

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.

The advantage of generating the FIFO information is that it will reduce the number of false errors generated by the tool for the designs containing FIFO instantiations. Tool will remove all the data crossings from memory to read registers for validation. Similarly, reconvergence checks for read and write synchronization registers will be removed. The reduced number of false errors will translate into faster debug and verification time.

FIFO Flow

The following shows the FIFO flow, where you can use automatic FIFO recognition, user-based FIFO, or a combination of both.



FIFO Recognition

You can automatically set FIFO identification with the SET CDC OPTION command's -fifo_detect option so you do not have to run the GENERATE FIFO INFO command. For example:

```
set cdc option -fifo_detect
set system mode verify
add data association read_data\[*\] -domain read_clk
add cdc check -structural -source -all -destination -all -from -all -to -all
validate
```

Debugging Non-identification of FIFOs

You can use the REPORT FIFO FAILURE command to debug the non-identification of FIFOs present in the design. In the following command example, ram_ff_reg is the name of the FIFO memory registers and rd_data_reg is the name of the FIFO read registers:

```
report fifo failure fifo5_1/ram_ff_reg* rd_data_reg*
```

If the software can detect the FIFO under the present setup, this command notifies you to run the GENERATE FIFO INFO command to identify the FIFO. For example:

```
// The software should have detected this FIFO.
// Use the "generate fifo info" command to identify the FIFO.
```

If the software cannot detect the FIFO under the present setup, the command notifies you the reason for the non-identification. For example:

```
// Warning: Two clock domains with mutual crossings do not exist.
```

FIFO Gray Code Checking for Read and Write Pointers

In addition to the automatic FIFO identification, running the GENERATE FIFO INFO command, or the ADD CDC CHECK command with automatic FIFO identification (SET CDC OPTION -fifo_detect), the software will automatically prove the correctness of gray code for read and write pointers and issue a warning if the detected read and write registers are not valid gray code generating registers. For example:

```
// 2 properties added.
// Warning: Gray code checking is not successful for index:1 FIFO
// Note: 1 FIFO present in the design
```

If the software can functionally prove the correctness of gray code registers, it will not produce any additional output.

Range Overflow Check for FIFO Memory

In addition to the automatic FIFO identification, running the `GENERATE FIFO INFO` command, or the `ADD CDC CHECK` command with automatic FIFO identification (`SET CDC OPTION -fifo_detect`), the software will automatically add the static property for range overflow for FIFO memory.

Run the `PROVE` command prior to `GENERATE FIFO INFO` to detect if the range overflow conditions exist.

In the following example, the first always block might go out of the range of the defined index for `ram` register. As a result, the code segment has a false range overflow property.

```
reg [31:0] ram [0:1];
always @( read_adr or ram[0] or ram[1])
    read_data = ram[read_adr];

always @( posedge read_clk or negedge read_reset_n ) begin
    if ( !read_reset_n )
        read_adr <= 0;
    else if ( !read_empty )
        read_adr <= (read_adr + 1’d1);
end
```

User-Based FIFO

Generating FIFO Information

Conformal Verify extract FIFOs with one or more dimensional memory component.

You can use the `GENERATE FIFO INFO` command after running the `ADD CDC CHECK` command with the `-structural` option to extract/identify FIFOs that have one-dimensional or multi-dimensional memory components.

Reporting FIFO Components

Use the `REPORT FIFO INFO` command to report the FIFOs and their components in the design.

Adding/Deleting FIFO Information

Conformal Verify requires that FIFO have all the necessary components. Use the `ADD FIFO INFO` command to add components to a FIFO.

Use the `DELETE FIFO INFO` command to delete information regarding the FIFOs in the design.

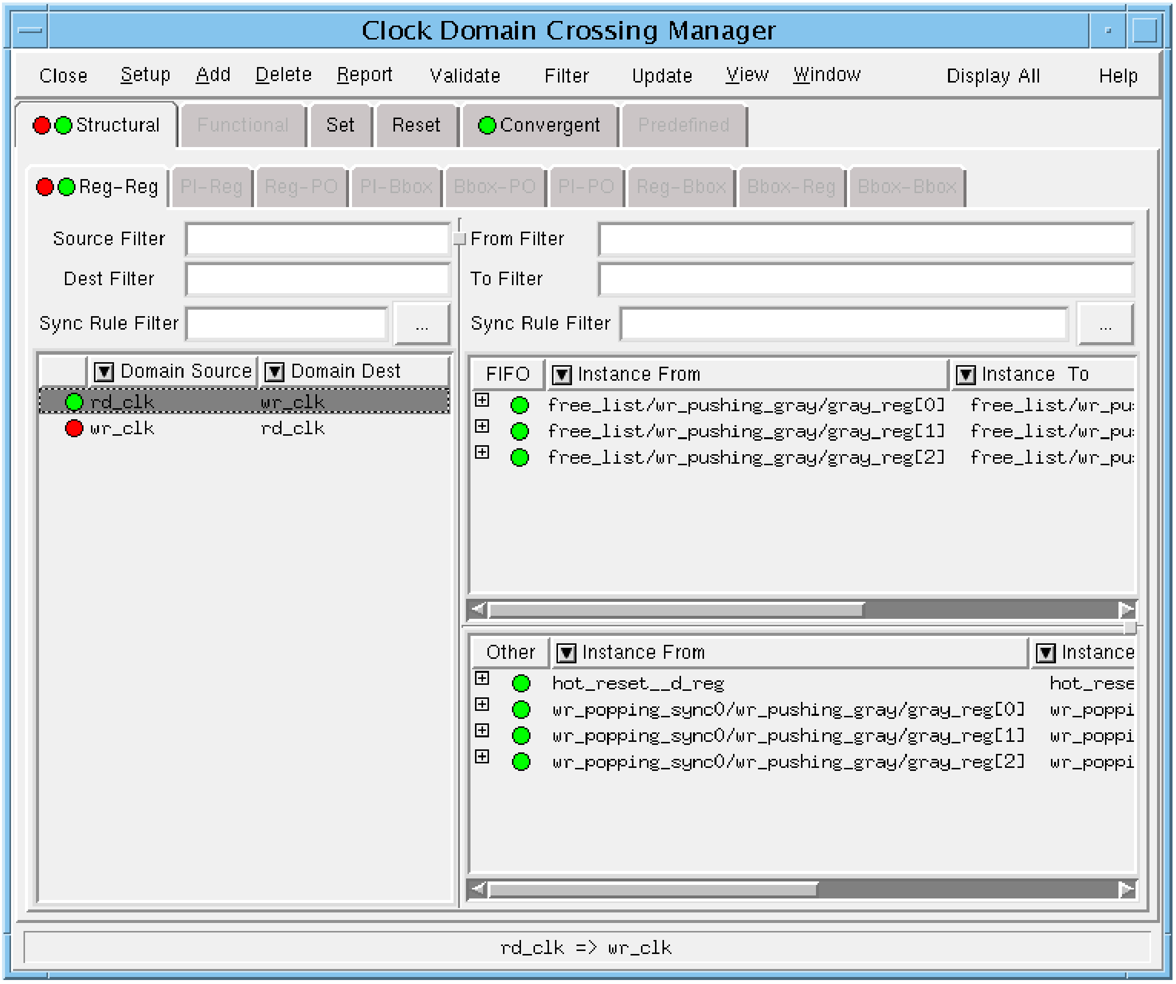
Command Example

The following commands show an example of user-based FIFO identification:

```
set system mode verify
add data association read_data\[*\] -domain read_clk
add cdc check -structural -source -all -destination -all -from -all -to -all
generate fifo info
validate
```

Using the FIFO Panel in the CDC Manager

You can use the Cross Domain Crossing Manager to categorize structural crossings.



	Domain Source	Domain Dest
●	rd_clk	wr_clk
●	wr_clk	rd_clk

FIFO		Instance From	Instance To
⊞	●	free_list/wr_pushing_gray/gray_reg[0]	free_list/wr_pu
⊞	●	free_list/wr_pushing_gray/gray_reg[1]	free_list/wr_pu
⊞	●	free_list/wr_pushing_gray/gray_reg[2]	free_list/wr_pu

Other		Instance From	Instance
⊞	●	hot_reset__d_reg	hot_rese
⊞	●	wr_popping_sync0/wr_pushing_gray/gray_reg[0]	wr_poppi
⊞	●	wr_popping_sync0/wr_pushing_gray/gray_reg[1]	wr_poppi
⊞	●	wr_popping_sync0/wr_pushing_gray/gray_reg[2]	wr_poppi

rd_clk => wr_clk

The FIFO panel contains all the crossings which appear to be valid by the software (even though they do not have synchronizers at the crossing) due to the FIFO property.

Note: These crossings are deleted when running the `VALIDATE` command and will not be displayed.

12

Example Session

Starting Conformal Extended Checks

In this example session, you will use the sample testcase in the `<install_dir>/demo/extended_checks` directory.

To start this sample session, copy the `<install_dir>/demo/extended_checks` directory to your working directory and then start Conformal Extended Checks. For example:

```
cp -r <verify_install_dir>/demo/extended_checks ~/VERIFY_DEMO
cd ~/VERIFY_DEMO
lec -verify &
```

This opens the main Conformal Extended Checks GUI window.

Reading the Design

Use the following procedures to read in the design.

1. Choose *File – Read Design* to open the Read Design form:
2. Double-click the `test.v` design file in the *Files* display to select it.
3. Click the *OK* button to approve the selection and close the window.

   Conformal reads the Verilog design.
4. Confirm that the main Conformal GUI window prints messages that indicate that the `test.v` file was successfully read.

   Note: Conformal prints messages in the transcript window while reading the design. These messages are the result of the HDL Rule Check (also referred to as the RTL Rule Check). See the *Conformal HDL Rule Check Reference*.

Defining Clock Waveforms

The demonstration design is a sequential circuit with a single clock. In this step you will add a clock waveform to the clock pin.

1. From the main Conformal Extended Checks GUI window, choose *Setup – Clock*.

   This opens the Clock Waveform window.
2. Add a default waveform to a clock signal:

a. From the Clock Waveform window, type `clk` in the *Clock Signal Name* field.

   This is the name of the clock pin for this design.

b. Leave all other default settings.

c. Click *Apply*.

   This adds the default waveform to the clock signal `clk`.

   Notice that the clock signal `clk` is listed in the display area below the *Clock Signal Name* field.

d. Click *Close* to close the Clock Waveform window.

3. Confirm that the following message displays in the main Conformal Extended Checks GUI window:

```
add clock 0 -waveform 1 1 2 clk
```

Initializing the Design

In this step, you will initialize the design with an initialization sequence file.

1. From the main GUI window, choose *Setup – Initial State.*

   This opens the Initial State form.

2. Click the *Browse* button.

   The Init State File window appears.

3. Double-click the `init.seq` file.

   This closes Init State File window.

4. From the Initial State window, confirm that `init.seq` appears in the *Filename* field.

5. Click *Apply*.

6. Click *Close*.

7. Confirm that the following message displays in the main Conformal Extended Checks GUI window:

```
read initial state my_path/demo/extended_checks/init.seq -Sequence
```

Switching to Verify Mode

Switch to the Verify mode to perform proofs.

➤ Click the *Verify* mode icon.

Tip

The percentage of completion for a process is shown in the *Progress Bar* located in the bottom right corner of the main GUI window.

During the transition from Setup mode to Verify mode, Conformal Extended Checks automatically performs Modeling Rule Checks. However, for this step, you will not address violations.

Selecting Properties for Proof

When you switch to Verify mode, Conformal Extended Checks automatically extracts all possible predefined properties. In this step you will select the properties you would like Conformal Extended Checks to prove.

1. Choose *Tools – Static Property* to open the Static Property Manager..

2. Review the extracted predefined properties by clicking the various tabs in the Static Property Manager.

   The tab headings group the predefined checks according to categories (for example, *Bus*). Click a tab to bring the desired category to the front. Grayed-out tabs indicate that Conformal Extended Checks did not extract properties in the specified category.

3. Choose *Add – Static Property – All*.

   This adds all the properties for verification.

4. Examine the tabs and the display area in the Static Property Manager to review the selected instances.

   Each of the properties and each of the tabs displays an “unprocessed” icon.

   ? E.

Note: This will change as a result of the actions you take in Step 7, which is when the Conformal Extended Checks proof takes place.

Proving Properties

In this step, Conformal Extended Checks will prove the selected properties and display the results.

1. From the Static Property Manager, click *Prove!.*

   This opens the Prove Option window.

2. Click *Prove* to accept the default options.
3. Monitor the progress of the proof with the dynamic *Progress Bar* located in the bottom right corner of the main Conformal Extended Checks GUI window.
4. Review the summary report in the Transcript window of the main Conformal Extended Checks GUI window.
5. Note any properties listed in the *Fail* column.

   In an actual session, these are the properties you would diagnose and fix.

   At this point, you may wish to review Chapter 3, "Using the Graphical User Interface" to learn more about the integrated debugging tools.

6. Exit Conformal Extended Checks as follows:
   - a. Choose *File – Exit* from the main Conformal Extended Checks window.
   - b. Click *Yes* to confirm and exit the tool.

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 285.

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 288.
Configurations:	configuration declaration	supported
	block configuration	supported
	use	supported
	attribute specifications	ignored
	component configurations	supported* See Component Configuration on page 289.
	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 292.
	variable	supported
	shared variable	supported* See Shared Variable on page 292.
	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 293.
	predefined attributes	supported* See Predefined Attributes on page 294.
	user-defined attributes	supported* See User-Defined Attributes on page 298.
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 298.
	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 299.
	assertion	ignored
	report	ignored
	guarded signal assignment	supported* See VHDL GUARDED Block Support on page 301.
	transport / after	ignored
	signal assignment	supported* See Signal Assignment on page 301.
	variable assignment	supported
	procedure call	supported* See Procedure Calls on page 303.
	if statement	supported
	case statement	supported
	for loop statement	supported* See For Loops on page 303.
	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 301.
	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 301.
	guarded concurrent signal assignment	supported* See VHDL GUARDED Block Support on page 301.
	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;
```

```
4.
5.  USE work.pack.all;
6.  ENTITY test IS
7.  … END test;
8.  ARCHITECTURE arch OF test IS
9.  …
10. glob1 <= in0 OR in1;
11. END arch;
12.
13. USE work. pack.all;
14. ENTITY test2 IS
15. …
16. END test2;
17. ARCHITECTURE arch OF test2 IS
18. …
19. glob1 <= in0 AND in1;
20. 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
- HIGH
- RANGE
- LENGTH
- LEFTOF
- PRED
- POS
- BASE
- STABLE*
- TRANSACTION*
- RIGHT
- LOW
- REVERSE_RANGE
- ASCENDING
- RIGHTOF
- SUCC
- VAL
- EVENT*
- LAST_VALUE*

*See Restriction 2 on page 295.

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
- PRED
- VAL
- RIGHTOF
- SUCC

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
```

```
3.      IF clk'EVENT AND clk = '1' THEN
4.      …
```

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 301.

■

B

VHDL 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.

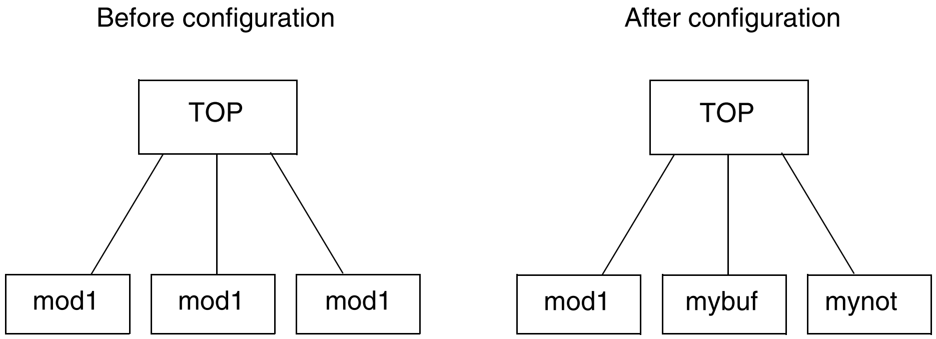
The Conformal software supports a subset of Verilog configurations, as defined in the *Verilog 2005 Language Reference Manual*. In particular, 'hierarchical use configurations,' liblist ordering, and cell configurations are not supported. The namespace mapping is supported through the `READ DESIGN` command's `-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.

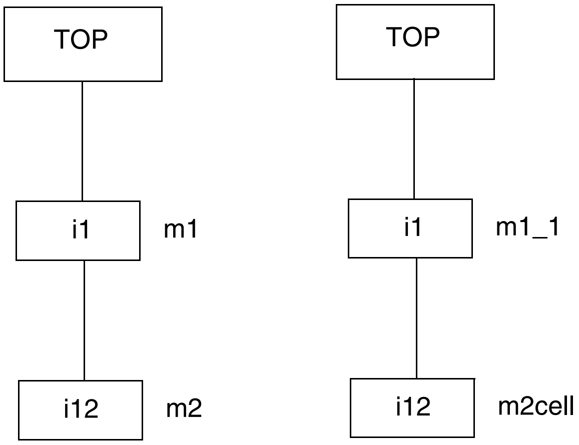
For example, if a design whose top module top has three instances `i1`, `i2` and `i3` of module `mod1`, and you read in the configuration `cfg1`, the following shows the design hierarchy before and after applying the configuration:

```
config cfg1;
    design work.top;
    instance top.i2 use mybuf;
    instance top.i3 use mynot;
endconfig
```



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

■

C

System Verilog Support

System Verilog 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 341.

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 barrel_shifter.v -root barrel_shifter -sva
read design 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 ///////////////////////
```

D

TVI Testbench and Simulation Script Files

This Appendix includes a general outline of a TVI Verilog testbench file and a description of a TVI Verilog simulation file.

The `WRITE DIAGNOSIS DATA` command with the `-verilog` option generates a Verilog simulation testbench file. The default simulation testbench filename is:

```
Vpx_<root_modulename>_<property_name>.tv
```

The following is a general outline of a Verilog testbench file generated by the Conformal Extended Checks TVI feature. This information is offered to help you customize your file.

Note: IEEE 1364-1995 standard Verilog is used in TVI-generated testbench files.

File Component	Definition
Copyright notice	This is a Verilog comment block.
Conditional macros	Conditional macros are used as customizable options. Verilog tools allow you to change command line options to conditional macros. Default macro values are listed below:
`` `define VPX_SEVERITY 2 ``	Is not currently used. It will globally set the default severity level for the Conformal Extended Checks predefined property.
`` `define VPX_RESETn VPX_ASSERT_RESET_LOW_SIGNAL ``	Specifies the internal reset low signal for the Conformal Extended Checks predefined property. This signal prevents glitches in monitored signals at simulation time zero.
`` `define VPX_FORCE force ``	Forces internal wire or reg objects to a `1‘b1` or `1‘b0` value.

‘define VPX_UNIT_TIME 10000	Sets the default time step to 10000 units. This amplifies the Conformal Extended Checks time step to allow ordering simulation events.
‘define VPX_DAT_DEL 0	Specifies that the data event has a zero delay after the Conformal Extended Checks time step.
‘define VPX_CLK_DEL 1	Specifies that the clock event has a unit delay after the Conformal Extended Checks time step.
‘define VPX_UNIT_TIMEx	(‘VPX_UNIT_TIME-’VPX_DAT_DEL) Is an internal macro used to synchronize simulation time to the VPX_UNIT_TIME boundary.
‘define ‘VPX_TIMESCALE ‘timescale lns/lns	Defines the time unit and time resolution of the TVI Verilog testbench.
‘VPX_DUMP_VCD	Has no default. If ‘VPX_DUMP_VCD is defined, a simple $dumpvars; statement is included. This enables the VCD object value dump on all Verilog simulation objects.
c. Main testbench module	The overall structure of the Verilog testbench module is outlined below to help experienced Verilog users to customize or to merge the testbench with other testbenches.

```
module TestBench_<property_name> ;
// Declare I/O actual objects
// Instantiate root module with I/O actuals
   <root_module_name> DUT_<property_name> ( I/O actuals );
// Set VCD dump and max simulation time
// Define clock signals
// Specify monitor for predefined property
   always @( {monitored signals} ) begin
     . . .
     if ( violated conditions )
     $display( "ASSERT . . . " );
     . . .
   end
// Specify input events
   // begin at time 0
     // set default 0 for all inputs
     // initialize pi/dff/dlat objects
   // advance to Time Unit 1
     // set new values for pi/dff/dlat objects
   // advance to Time Unit 2
   . . .
   // advance to LAST Time Unit
   . . .
 endmodule
```

The `WRITE DIAGNOSIS DATA` command with the `-verilog` option generates a Verilog simulation script file. The default simulation script filename is:

`Vpx_<root_modulename>_<property_name>.tv.runvlg`

The following is a description of a Verilog simulation script file generated by the Conformal Extended Checks TVI feature. This information is offered to help you customize your file.

Script	Definition
`+access+rwc`	allows Verilog simulator to have Read/Write/ Control access.
`+define+VPX_CLK_DEL=0`	specifies clock delay: zero delay.
`+define_VPX_DAT_DEL=1`	specifies data delay: unit delay.
`+define+ASSERT_ON`	turns on OVL assertion modules for simulation.

`+define+VPX_DUMP_VCD`	turns on the default VCD value dumping in the testbench.
`Vpx_<root_modulename>_<property_name>.tv`	specifies the default Verilog testbench file.
`Verilog design files`	specifies all Verilog design files used for simulation.
`-y <library directory>`	specifies the library directory used. Multiple `-y` options are permitted.
`+incdir+<include directory>`	specifies the directory that will be searched for include files. Multiple `+incdir` options are permitted.
`+libext+<extension>`	specifies the file suffix for module files in the library directory. Multiple `+libext` options are permitted.
`-v <library file>`	specifies the library file used.

E

Defining Clocks

Overview

Conformal Extended Checks Modeling Rules indicate any modeling errors encountered during the analysis and modeling of the design. These checks are performed as the system mode changes from Setup to Verify. Some violations require you to resolve ill-defined clocks with the `ADD CLOCK` command. This appendix provides the information you need to create `ADD CLOCK` commands.

This appendix consists of two sections. The first section illustrates how to use four steps to derive waveform from clock frequencies. Each step is explained in detail. The second section includes an example that implements the four steps, further illustrating how to create `ADD CLOCK` commands.

Deriving Clock Waveform

The following case demonstrates how to use the available information to create Conformal Extended Checks `ADD CLOCK` commands.

Clocks

Given N clocks, for example:

```
clk_1, clk_2, …, clk_N
```

Frequencies

With the following respective frequencies:

```
f_1, f_2, …, f_N
```

Primary Inputs

Assume that the primary inputs have the same names as the clocks:

```
clk_1, clk_2, …,clk_N
```

Step 1

In the first step, convert the frequency to period by using the following equation:

```
p_N = 1 / f_N
```

where `p_N` (the period) is expressed in terms ms, ns, or ps (as appropriate).

Now, you have the periods for all the clocks:

```
p_1, p_2, …, p_N
```

Step 2

In Step 2, determine the least common denominator for all each `p_N`.

```
p_LCD = LCD { p_1, p_2, …, p_N }
```

where `p_LCD` is the least common denominator for all the periods.

Step 3

In this step, determine the fastest clock in the design.

The fastest clock has the smallest period, thus:

```
    p_fastest <= p_j for all j = {1, …, N}
```

Step 4

In this final step, use the fastest clock as a basis to determine the `ADD CLOCK` commands for all the clocks.

Let's assume that all clocks start at 0 and have a duty cycle of 50%. If not, you can adjust the offset and width parameters of the `ADD CLOCK` command to compensate and use this scenario as an exercise.

First determine the multiplier for the fastest clock using what you already know about `p_LCD`:

```
MUL = p_LCD / p_fastest
```

Commands

Thus, the `ADD CLOCK` commands for all the clocks are:

```
add clock 0 -waveform p_i*MUL/2 p_i*MUL/2 p_i*MUL    clk_i,
```

where:

```
i = {1, …, N}
```

Creating ADD CLOCK Commands

For the exercise provided in this section you will use the four steps explained above to create an `ADD CLOCK` command.

Clocks

Assume there are three clocks for the design `TOP`:

```
clk_1, clk_2, clk_3
```

Frequencies

The following are the frequencies for the three clocks:

- `clk_1` = 100 MHz
- `clk_2` = 200 MHz
- `clk_3` = 400 MHz

Primary Inputs

The corresponding primary inputs are also called:

```
clk_1, clk_2, clk_3
```

Use the following four steps to create the `ADD CLOCK` commands for these three clocks.

Step 1

Convert the frequency to period.

```
p_1 = 1 / 100 MHz = 10 ns
p_2 = 1 / 200 MHz = 5 ns
p_3 = 1 / 400 MHz = 2.5 ns
```

Step 2

Determine the least common denominator for all `p_N`.

Since the common units is ns, use 1 time unit in Conformal Extended Checks to represent 1ns. Therefore, use 10, 5, and 2.5 in the next calculation:

```
p_LCD = LCD {10, 5, 2.5} = 10
```

Step 3

Determine the fastest clock in the design.

The fastest clock is `clk_3` with a period 2.5 units, because it is the smallest of the three periods: 10, 5, and 2.5.

Step 4

Using the fastest clock as a basis to determine the `ADD CLOCK` commands for all the clocks in Conformal Extended Checks.

```
MUL = 10 / 2.5 = 4
```

Commands

Thus, the `ADD CLOCK` commands are:

```
add clock 0 -waveform 10*4/2 10*4/2 10*4 clk_1

add clock 0 -waveform 5*4/2 5*4/2 5*4 clk_2

add clock 0 -waveform 2.5*4/2 2.5*4/2 2.5*4 clk_3
```

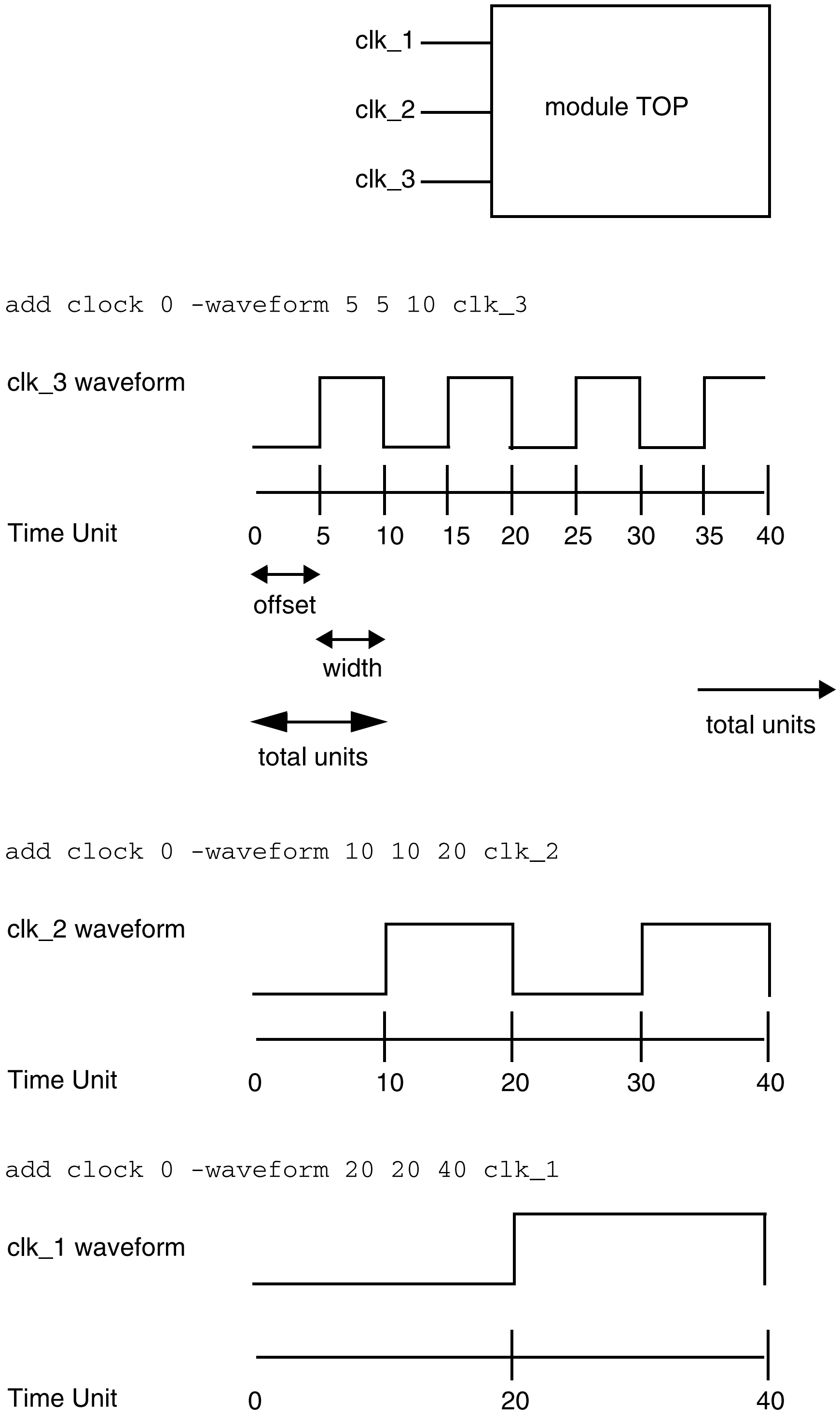
After performing the calculation, the commands become:

```
add clock 0 -waveform 20 20 40 clk_1

add clock 0 -waveform 10 10 20 clk_2

add clock 0 -waveform 5 5 10 clk_3
```

Refer to the above commands and Figure E-1 on page 358. Notice that the periods of these clocks are now 40 units, 20 units, and 10 units; representing 10 ns, 5 ns, and 2.5 ns, respectively. Thus, 4 time units equal 1 ns.

Using the whole number units (40, 20, and 10) as the periods for Conformal Extended Checks purposes is more convenient, *and* it reflects the phase relationship between the clocks.

Figure E-1 Defining Clocks



```
add clock 0 -waveform 5 5 10 clk_3
```

```
add clock 0 -waveform 10 10 20 clk_2
```

```
add clock 0 -waveform 20 20 40 clk_1
```

F

Initialization Sequence File

This appendix 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 Extended Checks time unit, which must be greater than or equal to 0.

Conformal Extended Checks 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, Conformal Extended Checks 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.
- Conformal Extended Checks 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: Conformal 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, Conformal 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 Conformal Extended Checks cannot find or does not allow the specified `cell_name`, or if the specified `cell_name` is not consistent with the `type_specifier`, Conformal Extended Checks 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, Conformal Extended Checks 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 Conformal Extended Checks is an unassignable `X`). The purpose of this assignment is to reduce the `X` occurrence in the initialization.

Note that the 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 `$`. Conformal Extended Checks supports minimal matching (for example, `$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 that `0 $random  -Z2LOGIC` only applies to `Z2LOGIC` at time `0`.

- Conformal Extended Checks 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).
- Conformal Extended Checks 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, Conformal Extended Checks issues an INIT2 error.
- Random assignment might 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]
  ```

G

Sample DoFiles

This appendix includes the following sample dofiles:

- Example One: verify_main.dof on page 370
- Example Two: verify_constraints.dof on page 371

Example One: verify_main.dof

```
//**********
//Sample Conformal Extended Checks DoFile
//verify_main.dof
//**********

//Local Conformal Extended Checks Variables
setenv SESSIONLOG logs/bt_sample.log
setenv TOPMODULE mod_top

//Log File Header
set log file $SESSIONLOG
!date
hostname
!uname -a
version

//Startup

dofile verify_setup.dof

dofile verify_constraints.dof

set system mode verify
usage

//Prove Checks
set prove effort low
add static property -dont_care
add static property -set_reset
prove
usage
report proved data -verbose -fail

//**********
//End of verify_main.dof
//**********
```

Example Two: verify_constraints.dof

```
//**********
//Sample Conformal Extended Checks dofile
//verify_constraints.dof
//**********

//Set environmental constraints
add clock 0 -waveform 0 1 4 phi1
add clock 0 -waveform 2 1 4 phi2

add pin constraint 1 log_reset_L
add pin constraint 1 sync_reset_L

//Establish legitimate starting state.
read initial state -vcd -root sys_tf/modtop mywaves.vcd

set flatten model -pio_to_bus pio

//Assertions to be specified as constraints
add assertion constraint modtop_inst/assert_one_hot_inst015
add assertion constraint modtop_inst/assert_always const002

//**********
//End of verify_constraints.dof
//**********
```