Conformal® Equivalence Checking User Guide

Conformal L, Conformal XL, and Conformal GXL

Product Version 19.2
November 2019

Contents

About This Manual

The Encounter® Conformal® logical equivalence checking tools verify RTL, gate, or transistor-level designs. As part of the functional verification platform, Conformal gives you complete equivalence checking (EC) solution available for verifying complex system-on-a-chip (SoC) designs from RTL to layout.

Conformal includes the following solutions:

- Conformal L

  Conformal L has equivalency checking capabilities with functional checks for ASIC design flows.

- Conformal XL

  Conformal XL includes Conformal L and extends equivalency checking capabilities to datapath synthesis and layout.

- Conformal GXL

  Conformal XL includes Conformal XL and extends equivalency checking capabilities to digital custom logic and custom memories.

- Conformal LowPower

  Conformal LowPower enables low power equivalence and functional checks for isolation cells, level-shifter cells, and state retention cells.

Audience

This manual is written for experienced designers of digital integrated circuits who must be familiar with RTL, synthesis, and design verification; as well as having a solid understanding of UNIX and Tcl/Tk programming.

How This Manual Is Organized

The chapters in this manual are organized to follow the flow of tasks through the design process. Because of variations in design implementations and methodologies, the order of the chapters will not correspond to any specific design flow.

Each chapter focuses on the concepts and tasks related to the particular design phase or topic being discussed.

In addition, the following sections provide prerequisite information for using the Conformal software:

- Chapter 1, "Introduction to the Conformal Equivalence Checker"

  Describes the process flow and the major components of the Conformal operation.

- Chapter 2, "Getting Started"

  Describes how to install, set up, and run the Conformal Equivalence Checker software, and use the online Help system.

Conventions

Syntax Structure

Convention	Definition
`Bold Case`	Indicates the command name.
`UPPERCASE`	Indicates the required minimum character entry.
`< >`	Indicates required arguments. Do not type the angle brackets.
`[ ]`	Indicates optional arguments. Do not type the square brackets.
|	Indicates a choice among alternatives. Do not type the vertical bar.
`\`	The backslash character (\) at the end of a line shows that the command you are typing continues on the next line.
...	Indicates multiple entries of an argument.
`*`	Indicates that Conformal lets the wildcard (*) represent any zero or more characters.

GUI Convention

Convention	Definition
Menu – Command	Indicates command sequences under a menu. For example: Choose *File – Read Design.*
Left-click	Click the left mouse button on the specified item.
Right-click	Click the right mouse button on the specified item.
Click	Click the left mouse button unless otherwise specified.
Double-Click	Click twice on the left mouse button.
Drag	Press and hold the left mouse button, and then move the pointer to the destination and release the button.

Additional Learning Resources

Cadence offers the following training courses on Conformal:

- *Custom Equivalence Checking with Conformal EC*
- *Conformal ECO*
- *Logic Equivalence Checking with Conformal EC*
- *Low-Power Verification with Conformal*

1

Introduction to the Conformal Equivalence Checker

Overview

The Conformal Equivalence Checking solutions are logical equivalence checking tools that verify RTL, gate, or transistor-level designs. As part of the Encounter® Conformal® functional verification platform, Conformal gives you the only complete equivalence checking (EC) solution available for verifying complex system-on-a-chip (SoC) designs from RTL to layout. It verifies the widest variety of circuits, including complex arithmetic logic, datapath, memories, and custom logic. Conformal has high-performance, high-capacity, and excellent debugging capabilities. These features are combined in an integrated environment.

Conformal Equivalence Checking solutions consist of three products:

- Conformal L

  EC capability with functional checks for ASIC design flow

- Conformal XL

  Extends EC capability to complex datapath synthesis and layout

- Conformal GXL

  Extends EC capability to digital custom logic and custom memories

Conformal supports standard library and design interface formats and integrates readily into existing design environments. The flexibility of these tools lets you efficiently impose constraints and apply guidance. Conformal is self-contained and is not tied to any particular synthesis environment. Thus, it gives you a higher degree of confidence than equivalence checkers integrated with a particular logic synthesis tool.

Conformal employs proprietary *key point mapping* and *formal functional comparison* algorithms that incorporate many innovative techniques for solving a wide range of problems. The comparison engine has superior performance and successfully completes verification of designs with differences.

Conformal also has excellent debugging capabilities. It automatically diagnoses design mis-matches and accurately pinpoints the source of the differences.

Conformal Features

Conformal incorporates many features that streamline and authenticate the design process, while giving you flexibility.

- Supports Full-Chip Verification

  Conformal has excellent processing speed that significantly reduces verification time for high-capacity, high-complexity, full-chip designs.

- Supports Multiple Design Formats

  Conformal supports Verilog®, VHDL, SPICE, EDIF, and NDL design formats.

- Supports Standard Library Formats
  Conformal supports Verilog simulation libraries and the Synopsys® Liberty™ Format Libraries.

- Employs Verilog/VHDL-RTL and Transistor Function Abstraction

  Conformal has a built-in Verilog/VHDL-RTL and transistor function abstraction engine that lets you verify Verilog/VHDL-RTL, gate, or transistor level designs.

- Employs Advanced, Automatic Mapping

  Conformal contains advanced and proprietary sequential element mapping algorithms that identify corresponding sequential elements automatically with minimal user resources. This feature relieves you of the tedious job of specifying corresponding flip-flops and latches.

- Employs an Efficient and Effective Comparison Engine

  Conformal has a superior formal comparison engine to ensure successful verification of non-similar designs with different hierarchical structures. Conformal contains a unique correlation learning technology that effectively explores both structural and functional relationships of the logic in two designs and dramatically reduces the verification run time. This technology does not require high memory use and is very effective for both similar and dissimilar designs.

- Includes Automatic Diagnosis

  When a logic mismatch is found, designers find that it is absolutely essential to be able to quickly locate the source of functional differences. Conformal automatically diagnoses functional differences, narrowing them to a small number of possible locations in the design. This feature helps you identify and effectively correct problems and reduce debugging time.

- Includes Integrated Debugging

  Conformal has extensive gate reporting integrated with the schematic viewer. This feature gives you flexibility and immediate feedback for debugging and diagnosis.

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`

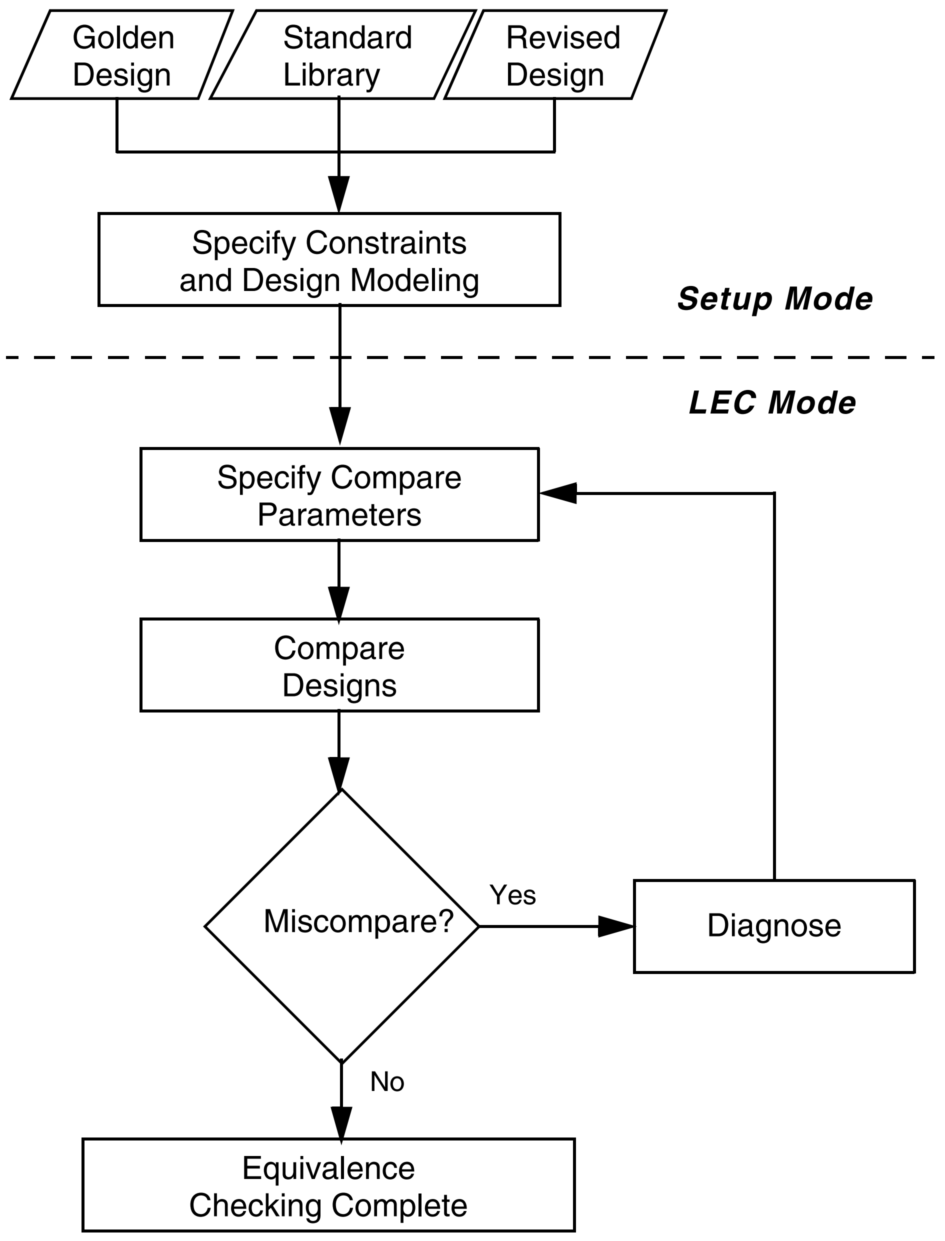
SystemVerilog	IEEE 1800-2009	`READ DESIGN -systemverilog`
		`READ LIBRARY -systemverilog`

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

CPF	1.0, 1.0e, 1.1 and 2.0	`READ POWER INTENT`

Conformal Methodology

The following flowchart illustrates the Conformal process flow.



Preparing the Designs

During the first phase of equivalence checking, Conformal reads in the Golden and Revised designs and their associated libraries. The designs can be any of the following formats:

- Verilog
- VHDL
- EDIF
- NDL
- SPICE

Conformal supports Verilog simulation libraries and Synopsys Liberty libraries.

After reading in designs and libraries, you can tailor the session to your particular needs by specifying constraints and parameters. When all of these Setup mode tasks are complete, you can change to LEC mode and the next session phase.

Mapping and Comparing

During the second and third phases of equivalence checking, Conformal automatically maps key points and compares them. When the comparison is complete, Conformal pinpoints the differences.

Diagnosing

During the final phase of the Conformal process flow, you can employ a combination of the integrated diagnosis tools to examine differences. These tools include the Schematic Viewer and Source Code Manager. After you have remedied the differences, a new verification session begins.

Conformal Operation

This section details the Conformal integrated debugging environment, describing each of the major components of the Conformal operation.

System Modes

Conformal operates in two system modes: *Setup* and *LEC*. In the Setup mode, Conformal reads two designs. You designate the design types, which are Golden and Revised. (Generally, the Revised design is a modified or post-processed design that Conformal compares to the Golden design.) Additionally, you apply constraints and design settings in the Setup mode. Finally, you can specify Conformal compare options and mapping methods. When you have set all design conditions, you can move to the LEC system mode.

Transition

The following two sections relate to events that occur as Conformal transitions from the Setup to the LEC mode.

Rule Checking

Conformal checks various rules during parsing. You can specify how Conformal will respond to rule violations before you read in the designs and libraries. Additionally, you can choose to view a report displaying all of the library and design rule violations that occurred during parsing.

For additional information about rule checking, see the *Conformal HDL Rule Check Reference*.

Key Points

In the transition from the Setup to the LEC system mode, Conformal flattens and models the Golden and Revised designs and automatically maps the key points. Key points are defined as:

- Primary Inputs
- Primary Outputs
- D Flip-Flops
- D Latches
- TIE-E Gates (error gate, created when x-assignment exists in Revised design)
- TIE-Z Gates (high impedance or floating signals)
- Blackboxes

- Cut Gates (artificial gates that break combinational loops)

Mapping

Conformal employs three name-based methods to map key points and one no-name method. Name-based mapping is useful for gate-to-gate comparisons when small changes have been made to the logic. Conversely, the no-name-mapping method is useful when Conformal must map designs with completely different names. By default, Conformal automatically maps key points with the name-first mapping method when it exits the Setup mode. Any key points that Conformal does not map are classified as unmapped points.

Unmapped Points

After Conformal maps key points, the remaining unmapped points are classified into one of three categories: extra, unreachable, or not-mapped.

- *Extra* unmapped points are key points that are present in only one of the designs, Golden or Revised.
- *Unreachable* unmapped points are key points that do not have an observable point, such as a primary output.
- *Not-mapped* unmapped key points are key points that are reachable but do not have a corresponding point in the logic fan-in cone of the corresponding design.

Comparison

After Conformal maps the key points, the next step of the verification is comparison.

Compare Points

You can designate mapped points for comparison. Comparison examines these compare points to determine if they are equivalent or non-equivalent.

Run Time and Performance

The comparison determines if the compared points are:

- Equivalent
- Non-equivalent

- Inverted-equivalent
- Aborted

In the case of aborted compare points, you can change the compare effort to a higher setting. Thus, Conformal can continue the comparison on only the aborted compare points. Conformal can also display the complete run time and total memory use for the comparison.

Reports

When Conformal completes the comparison, you can get summary reports showing which key points are equivalent and which are non-equivalent. Then, you can diagnose non-equivalent points to determine the cause of the difference.

Diagnosis

Diagnosis is the process of examining non-equivalent points and identifying the most likely error candidates. In this phase of verification, examine non-equivalent points using the integrated tools as described below.

Error Patterns and Candidates

The Conformal diagnosis feature precisely locates the cause of a non-equivalent point. During this process, Conformal displays the error patterns that caused the difference and lists the possible candidates. Included in this list is the percent of probability, which is shown in decimal form. The most likely candidate is 1.00.

Gate Reporting

With the above diagnosis information gathered, you are ready to employ gate reporting to trace the fan-in or fan-out cone of the non equivalent point. Additionally, you can view a schematic representation of the non-equivalent point.

Schematic Viewing

When you view the schematic representation of the non-equivalent point, the viewer displays the fan-in cone with the corresponding and non-corresponding supporting key points and their simulation values. The non-equivalent compared point, diagnosis input point, supporting key points, and error candidates are all color-coded for visual accessibility. The schematic viewer displays the Golden and Revised schematic representations side-by-side in separate windows, for quick comparison.

Source Code Viewing

To further diagnose non-equivalent points, you can access the Source Code Browser. This tool lets you specify a gate in the Revised or Golden design and view its relative location in the source code.

Overview of Conformal Tcl

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 Equivalence Checking 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 Equivalence Checking Reference Manual* for detailed descriptions of the native Conformal commands.

Tcl Version

To determine the current version of Tcl used by Conformal:

```
SETUP> tclmode
TCL_SETUP> puts $tcl_version
8.5
or
TCL_SETUP> info tclversion
8.5
```

To get more detailed patch level information, one can use

```
TCL_SETUP> info patchlevel
8.5.2
```

To know from where the tcl script is being used from, one can do:

```
TCL_SETUP> info library
<cadence lec installation>/share/cfm/lec/tcl8.5
```

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.

Start-Up Command Options

The following lists the licensing options for the `lec` command when starting the Encounter® Conformal® software. For example, if you have an XL or GXL license, you must specify this at startup with the `-xl` or `-gxl` option. For example, to start the Conformal software with the XL license in GUI mode, you would enter the following at the UNIX prompt:

```
UNIX% lec -xl
```

Without the `-xl` or `-gxl` option, the Conformal software starts with the default L license.

Once you start your session, you can use the `LICENSE` command to display the current license status in the transcript output.

`-L`	Launches the Encounter® Conformal® Equivalence Checker **Note:** This is the default when running the `lec` command with no options.
`-RCV`	Launches Encounter® Conformal® XL with an RTL-Compiler verification license
`-XL`	Launches Encounter® Conformal® L with Datapath and advanced equivalence checking capabilities
`-GXL`	Launches Encounter® Conformal® XL with digital custom logic and memory verification capabilities
`-ECO`	Launches Encounter® Conformal® XL with ECO
`-ECOGXL`	Launches Encounter® Conformal® XL with ECO GXL capabilities (physical design awareness)
`-LP/-LPXL`	Launches Encounter® Conformal® XL with Low Power Verification

`-CCD`	Launches Encounter® Conformal® L with a Conformal Constraint Designer L license
`-CCDXL`	Launches Encounter® Conformal® XL with a Conformal Constraint Designer XL license
`-LPGXL`	Launches Encounter® Conformal® XL with digital custom logic, memory verification, and low power verification capabilities.
`-VERIFY`	Encounter® Conformal® Extended Checks

The `lec` command has the following additional options. This list is also available using the `lec -help` command *before* you start your session.

`-Dofile <filename>`	Runs the script *`<filename>`* after starting LEC. See "Dofile Command Files" on page 43 for more information.
`-LOGfile <filename>`	Sets up a log filed called *`<filename>`*.
`-RESTART_checkpoint <filename> [-protect <password>]`	Restarts a session that was saved using the `CHECKPOINT` command.
`-INFO_CHECKpoint <filename>`	Information of the checkpoint `<filename>`
`-Gui | -NOGui`	Starts the session in GUI or non-GUI mode.
`-TclMode`	After the session starts, the tool enters Tcl mode.
`-NOColor | -Color`	Controls color-coded messaging when in non-GUI mode. By default, color-coding is off.
`-DEFault [init_filename] | -NODEFault]`	Specifies whether to process the initial command file (`init_filename` or `.conformal_lec`) by default during startup. For more information, see "Initial Command Files" on page 42.
`-Banner | -NOBanner`	Specifies whether to display the LEC banner during startup.
`-RESETrc`	Reset GUI default settings.
`-NOLIcwait`	If all licenses are checked out, exit immediately.

`-Info`	Display the product information and exit.
`-Version`	Displays the product version. Once you have started your session, you can also use the `VERSION` command to display the 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.

Initial Command Files

When you start the Conformal software, it searches for and executes initial command files (`.conformal_lec`). The software checks for the `CONFORMAL_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 `CONFORMAL_RC` variable is not set, the software continues the search as follows:

1. The installation directory
2. The home directory
3. The current working directory

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

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

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

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

Important

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

Dofile Command Files

The Conformal Equivalence Checker 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 46.
- 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 -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`:

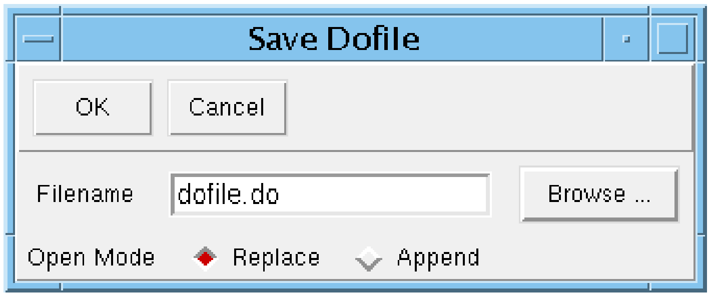
```
UNIX% lec -nogui -dofile my_dofile
```

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 Equivalence Checker session.



Filename dofile.do

Open Mode [✓ Replace] Append

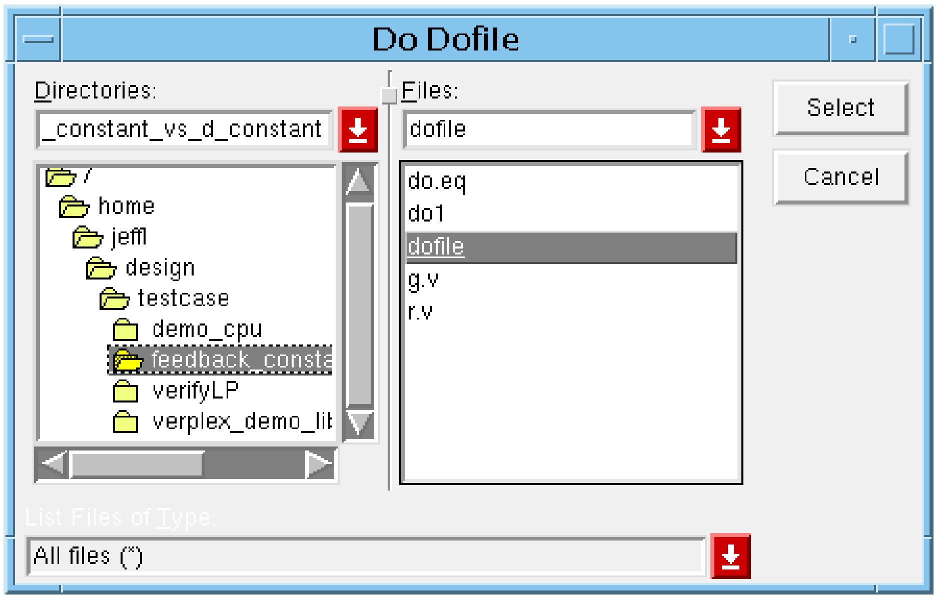
Save Dofile Fields and Options

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.



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 -both
```

   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` and `-both` options:

```
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 -both
```

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` and `-both` options:

```
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 -both
```

Saving and Restoring a Session

In the current release, there are two ways to save a session:

- Save and Restore Commands on page 47
- Checkpoint and Restart Facility on page 47

Save and Restore Commands

Use the `SAVE SESSION` command, or the Save Session form (*File – Save Session*), to save session up to a current point in binary format, which can be restored later. You can use this if priorities demand that another session preempt your session

Use the `RESTORE SESSION` command, or the Restore Session form (*File – Restore Session*), to restore a session you previously initiated and saved. Before using this procedure, Conformal must be in its initial state. Therefore, you must either reset the system to the initial state with the `RESET` command or *Reset Design* menu option (*File – Reset Design*), or exit and restart the Conformal software.

Limitation: When you use the *Restore Session* menu option, you must restore the session on the same platform and with the same Conformal version.

Checkpoint and Restart Facility

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

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

Applicable commands	`CHECKPOINT`
	`INFO CHECKPOINT`
	`<start_up_command> -restart_checkpoint <checkpoint_file_name> [-protect <password>]`

Data preserved	When you save your session as a checkpoint, the tool preserves the: ■ Hierarchical and flattened databases ■ Environment settings ■ Constraints ■ Verification results ■ User-defined variables ■ User-defined procedures
Supported Platform	Linux

Limitations:

This feature has the following limitations:

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

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

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

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

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

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

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

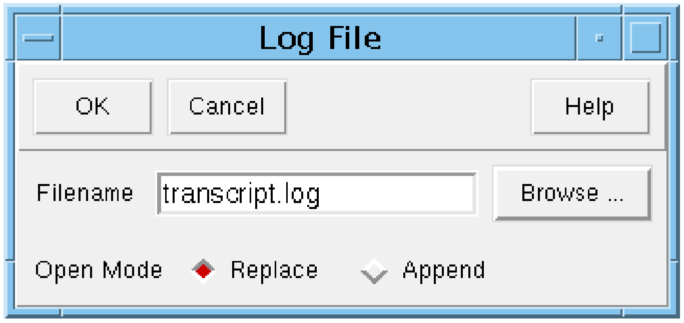
Transcript Messages

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 Equivalence Checker retains the transcript in a file. To save the transcript output in a log file, see Recording Transcript Log Files on page 50.

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

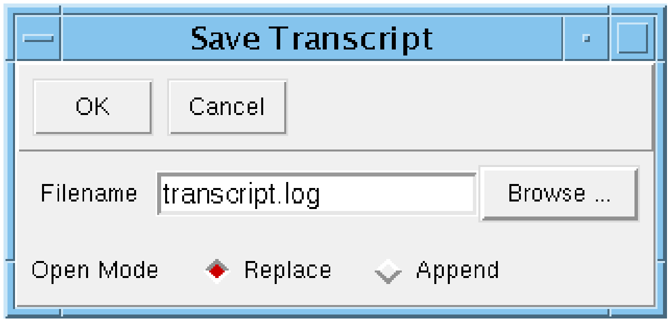
Recording in this file begins after you click *OK*; therefore, you might want to create a log file at the beginning of your session. However, if you begin a session and decide to save the transcript at a later point, see Saving a Transcript File on page 50 to capture a transcript of the beginning of 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:

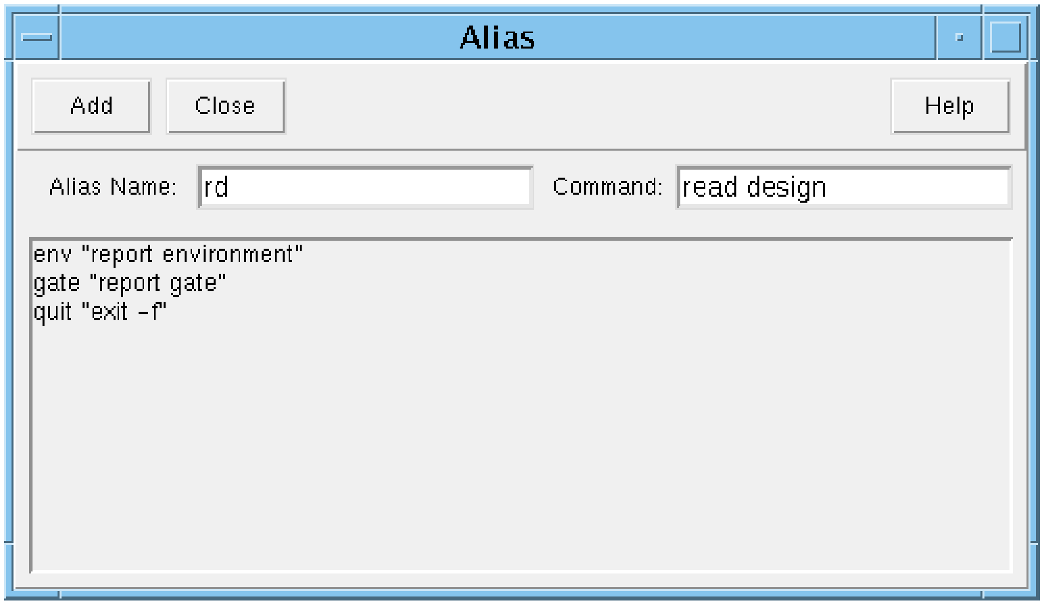
Example command:

```
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*.

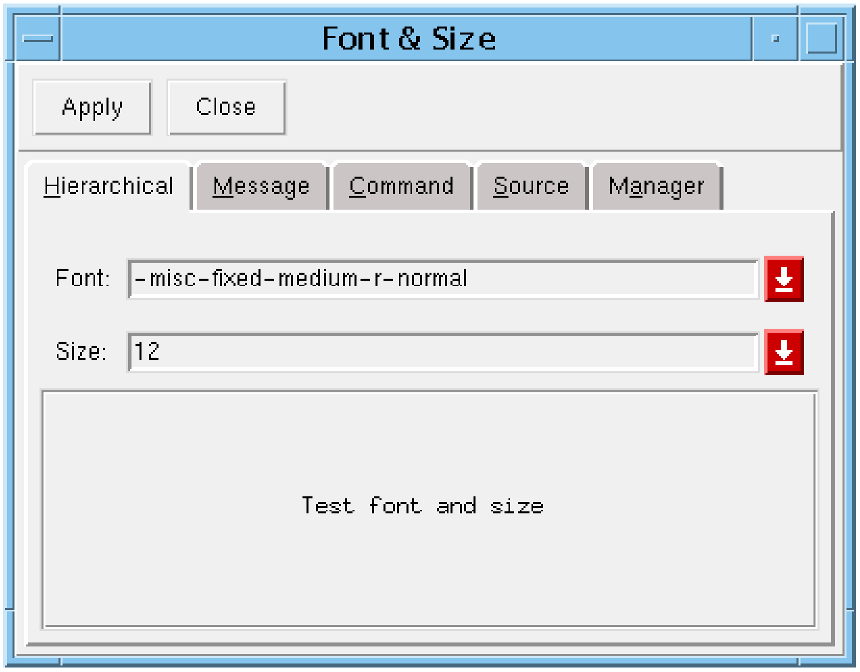
Setting Preferences

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*.

Hierarchical Browser On

Displays or hides the Module Browser in the main window.

Hierarchical Browser Sync. up

Synchronizes the hierarchical schematic between the Golden and Revised designs. Paired HRC sync-up schematic will sync-up the following events between paired schematics:

- Selected object, which includes instance, net, pin)
- Load/Drive
- Coloring object which include instance, net and pin.
- Move schematic hierarchy up/down.

The Conformal software will open a paired HRC sync-up schematic when selecting Schematics from the the following areas:

- Selecting Schematics from the Hierarchical Browser popup menu.
- Selecting Show Hierarchical View from the Flattened Schematic window.
- Selecting Show Hierarchical HLite View from the Flattened Schematic window.

Icon Bar

Displays or hides the Icon Bar in the main window.

Show Static Infobox

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

Simplified Schematic Viewing

With simplified schematic viewing, you can control how many netlists are displayed in the Schematics Viewer.

Without simplified viewing, the tool determines the number of netlists to display. Very large schematic displays can take a long time to load, and it can be difficult to pinpoint/trace logic.

With simplified viewing, you can drag/drop any module/instance you want to view and then trace corresponding loads/drivers.

Simplified schematic viewing is enabled by default. To disable simplified schematics, use the Preferences - *Simplified Schematics Viewing Options* menu item from the main Conformal window.

Accessing Online Help and Documentation

Launching Cadence Help

The online documentation system is called Cadence Help.

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

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

Getting Help for Cadence Help

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

Getting Help on Commands to Run Tools

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

```
% tool_name -help
```

Example:

```
% ccd -help
```

Getting Help on Commands and Messages

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

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

  ```
  man read design
  ```

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

  ```
  man read design -verbose
  ```

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

  ```
  man f10
  ```

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

  ```
  man -message
  ```

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

```
%man man
```

Searching the Help Database for Specified Strings

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

Using the Help Menu

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

Accessing Help from Command Windows

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

Accessing User Documentation

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

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

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

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

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

Accessing Product Information

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

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

Accessing License Information

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

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

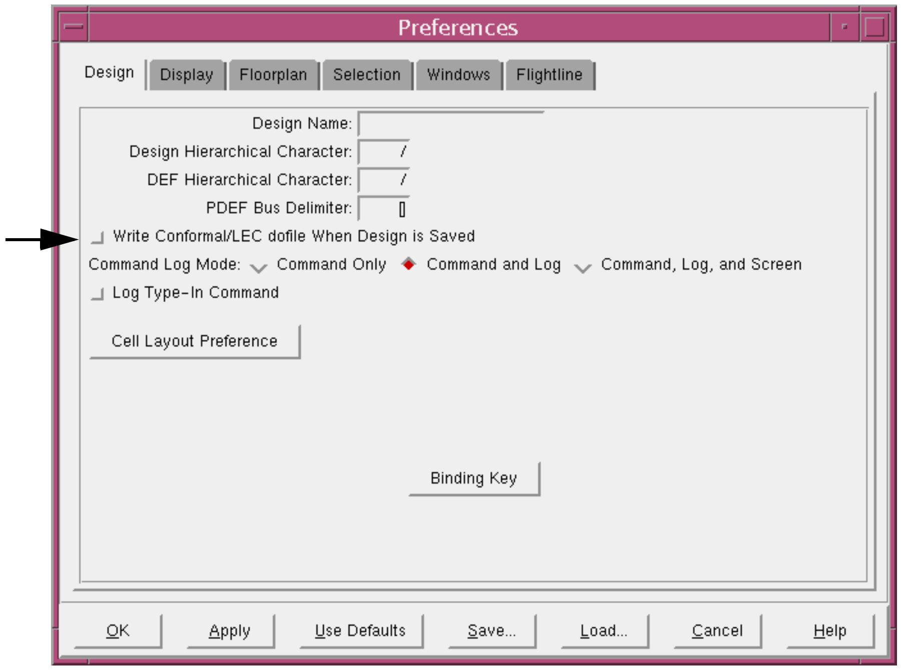
Platform Integration

Dofile Generation from First Encounter

You can use the First Encounter graphical user-interface (forms) to generate dofiles for Conformal equivalence checking verification. You can create and save dofiles using two Encounter forms: Preferences and Conformal Equivalence Checker.

Preferences

In the Encounter main window, select *Design – Preferences* to open the Preferences form, and click on the *Design* tab.



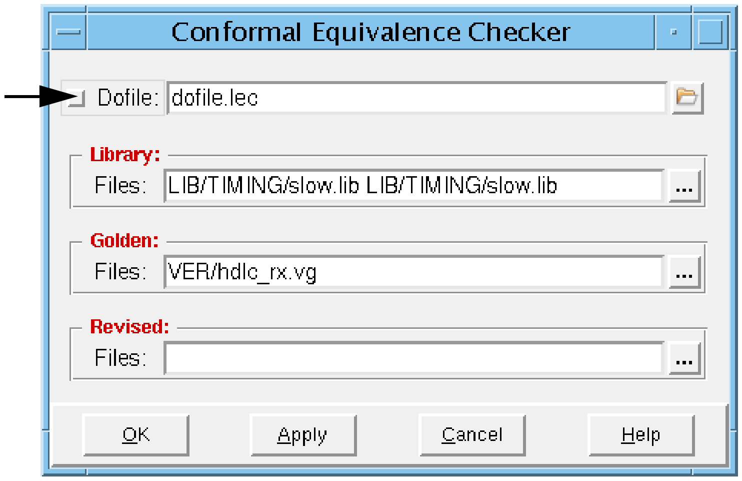
Preferences

Design Name:
Design Hierarchical Character: /
DEF Hierarchical Character: /
PDEF Bus Delimiter: []
Write Conformal/LEC dofile When Design is Saved
Command Log Mode: Command Only [✓ Command and Log] Command, Log, and Screen
Log Type-In Command

Click *Write Conformal/LEC dofile When Design is Saved* to specify that when saving a netlist, this automatically saves a dofile for the Conformal Equivalence Checking capabilities.

Conformal Equivalence Checker

In the Encounter main window, select *Tools – Conformal – Verify Design Equivalence* to open the Conformal Equivalence Checker form.



Dofile: dofile.lec

Library: Files: LIB/TIMING/slow.lib LIB/TIMING/slow.lib

Golden: Files: VER/hdlc_rx.vg

Revised: Files:

Click *Dofile* to specify the dofile creation. You can enter the name of the dofile, use the `dofile.lec` default, or click *Browser* icon select a dofile from the Open Dofile browser window.

Dofile Generation from RC Synthesis Tools

This section describes how to translate RTL Compiler settings to Conformal LEC settings in order to compare an RTL design against a RTL Compiler synthesized netlist. This translation is automatically done by the `write_do_lec` command.

Design Hierarchy in a Conformal LEC Comparison

There are two ways Conformal LEC performs the comparison between the Golden design and the revised design: a hierarchical comparison and a flattened comparison. A flattened comparison is sufficient when comparing two gate-level designs.

However, when at least one of the two designs is a complex RTL design, a hierarchical Conformal LEC comparison might be necessary. By default `write_do_lec` always prescribes a hierarchical comparison if the Golden design is at the RTL level.

`write_do_lec` assumes:

- The design loaded into RTL Compiler is an RTL design
- The design specified with the `-Golden_design` option is a gate-level netlist. A flat compare will be performed with this option.
- The design specified by the `-revised_design` option is a gate-level netlist

If the `-Golden_design` switch is not specified, and a single-shot do file was created, the RTL design loaded with the `read_hdl` command will be considered the Golden design and a hierarchical comparison will ensue. If the `-Golden_design` switch is not specified and an incremental comparison was performed, then the Golden netlist is described in the last checkpoint file, and a flat comparison will ensue.

Combining Liberty Files

RTL Compiler and Conformal LEC handle the combination of Liberty files in the same way. Both can combine multiple Liberty files that are syntactically incomplete into a single Liberty file. For example, in the following RTL Compiler example, `file1.lib` and `file3.lib` Liberty files are complete, and `file2.lib` and `file4.lib` Liberty files are missing the `library` keyword and the corresponding opening and closing braces ("{}"):

```
set_attribute library { { file1.lib file2.lib } { file3.lib file4.lib } }
```

The `file1.lib` and `file2.lib` libraries will nonetheless be combined and treated as a single Liberty file. The `file3.lib` and `file4.lib` files will also be combined and treated as a single Liberty file.

If all the Liberty files were syntactically correct, in this case with their `library` keywords and braces, then the above example will look like this in Conformal LEC:

```
SETUP> read library -liberty file1.lib file2.lib file3.lib file4.lib
```

- If an incomplete Liberty file only contains elements that Conformal LEC does not need, like wire-load models, simply remove them from the dofile.
- If an incomplete Liberty file contains elements that Conformal LEC needs, like extra cells, you must manually combine the main library and the partial library and then load the new single library in Conformal LEC. To do this:

  a. Take the main library and delete the final "`}`" (brace) character

  b. Concatenate all partial libraries into the main library

  c. Add the previously deleted "`}`" character at the end of the main library.

Liberty and Simulation Libraries

All library files are loaded into Conformal LEC through a single `read library` command that is specified in the generated dofile. The command will always be specified with its `-both` option. The following example loads all the Liberty libraries:

```
rc:/> read library -both -liberty $lib_files
```

An alternative to using the Liberty libraries is to use simulation libraries with the `write_do_lec` command.

Currently, the simulation libraries are only supported in Verilog-1995 format. Multiple simulation libraries can be specified as a Tcl list.

State Table in the Libraries

Every `read library` Conformal LEC command in the RTL Compiler generated dofiles is always accompanied by the `-statetable` option.

RTL Code

- Each `read_hdl -verilog` RTL Compiler command translates to a `read design -verilog` Conformal LEC command.
- All the `read_hdl -vhdl` commands (if there are more than one) are combined into one `read design -vhdl` command.
- The `read design -vhdl` command will be placed after the last `read design -verilog` commands.

Each parameter that was explicitly specified with the `elaborate -parameters` RTL Compiler command will become an argument to the `-parameter` option of the `read design` command in the generated dofile. For example:

```
rc:/> elaborate -parameters {{p1 16} {p2 10}}
```

will translate to:

```
read design -parameter p1 16 -parameter p2 10
```

Blackboxes

The `write_do_lec` command will always insert the following Conformal LEC command in the dofiles:

```
set undefined cell -noascend black_box -both
```

Although `write_do_lec` will always use this command, do not use `set undefined cell black_box` when you are creating dofiles manually: it can mask a user-error in which not all the HDL files are loaded into Conformal LEC but Conformal LEC declares an EQ after matching the black boxes.

Undriven Signals

RTL Compiler has three undriven-related attributes:

```
set_attribute hdl_undriven_signal_value        <0|1|X|none> /
set_attribute hdl_undriven_output_port_value   <0|1|X|none> /
set_attribute hdl_unconnected_input_port_value <0|1|X|none> /
```

Conformal LEC has one undriven-related command:

```
set undriven signal <Z|0|1|X> [-both | -Golden | -revised]
```

In Conformal LEC, the default undriven setting is `Z`. In RTL Compiler, the default undriven setting is `none` for all three scenarios.

- The `0` setting in RTL Compiler is the same as the `0` setting in Conformal LEC.
- The `1`setting in RTL Compiler is the same as the `1` setting in Conformal LEC
- The `X` setting in RTL Compiler is the same as the `X` setting in Conformal LEC.
- The `none` setting in RTL Compiler is the same as the `Z` setting in Conformal LEC.

Conformal LEC uses one undriven setting to control all three undriven scenarios. RTL Compiler has one undriven setting for each scenario. To translate from RTL Compiler undriven settings to Conformal LEC undriven setting, the following two RTL Compiler attributes are ignored:

```
hdl_undriven_output_port_value
hdl_unconnected_input_port_value
```

Translating the undriven setting is only needed for RTL designs. Therefore, translation of the undriven attributes is done only for the Golden design when it is the RTL loaded into RTL Compiler. In this case, the following actions are taken:

- If `get_attribute hdl_undriven_signal_value /` returns `0`, then the dofile has
  ```
  set undriven signal 0 -Golden
  ```
- If `get_attribute hdl_undriven_signal_value /` returns `1`, then the dofile has
  ```
  set undriven signal 1 -Golden
  ```
- If `get_attribute hdl_undriven_signal_value /` returns `X`, then the dofile has
  ```
  set undriven signal x -Golden
  ```

- If `get_attribute hdl_undriven_signal_value /` returns `none`, then the dofile has

  ```
  set undriven signal z -Golden
  ```

Only RTL designs require the translation of undriven settings. Therefore, undriven attributes only need to be translated when the Golden design is the RTL that is loaded into RTL Compiler.

Unreachable Key Points

If the `hdl_preserve_unused_registers` attribute is `true` on RTL designs, every Conformal LEC `read design` command will be specified with its `-keep_unreach` switch.

If both the `delete_unloaded_seqs` and `delete_unloaded_insts` attributes are `false` on RTL designs, the following command will be written to the dofiles:

```
set mapping method -unreach
```

Tip

Adding an instance constraint on a DFF removes it from the compare list. As a result, it becomes unreachable because it is no longer needed. However, if you still want to compare it, use the `SET MAPPING METHOD` -unreach command.

Constant Flop and Latch Optimizations

If you specify any of the following attributes (regardless of their values):

- `optimize_constant_0_flops`
- `optimize_constant_1_flops`
- `optimize_constant_latches`

The following Conformal LEC command will be written to the dofiles:

```
set flatten model -seq_constant -seq_constant_x_to 0
```

Latches and Muxes with Feedback

If `hdl_latch_keep_feedback` attribute is `false`, then the following Conformal LEC command will be written into the dofiles:

```
set flatten model -loop_as_dlat
```

Clock Gating Features

Among the Low Power transformations and optimizations performed by RTL Compiler, `write_do_lec` supports MSV and clock-gating insertion.

It ignores the Conformal LEC impact (if any) of the SRPG and operand isolation Low Power features

`write_do_lec` supports the ff and none styles of clock-gating insertion, and ignores the Conformal LEC impact (if any) of the latch style.

DFT Scan Flops

Conformal LEC performs functional verification of the core logic of a design and not the scan-insertion logic. Therefore, the RTL Compiler generated dofile may carry the following types of design constraints:

- For every scan-data-out port added by RTL Compiler, there is a Conformal LEC `add ignored outputs` command to exclude it from the Conformal LEC comparison.
- For every test signal that is declared by the RTL Compiler `define_dft` command, there is a Conformal LEC `add pin constraints` command to tie it to the inactive state.
- For every primary input port serving as a test control signal of an IEEE 1500 core wrapper, there is a Conformal LEC `add pin constraints` command to tie it to the inactive Conformal state.

For LSSD designs, provide Conformal LEC with a simulation model for every LSSD cell used in the design.

The `write_do_lec` command ignores the `unmap_scan_flops` and `dft_mapped` attributes in its translation process.

Instantiated DesignWare or ChipWare Models

If the RTL code instantiates a ChipWare or DesignWare model, the generated dofile loads its simulation model using the `read design` command. However, currently this process will not work for the pipelined models (like `CW_mult_2_stage`), or non-bit-accurate models (like `CW_multp`).

Exiting Conformal LEC

By default, the last line in a generated dofile is an exit command, telling Conformal LEC to quit its session after finishing the dofile.

You can suppress the `exit` command in the generated dofile by specifying the `-no_exit` switch of the `write_do_lec` command. Doing so will not terminate the Conformal LEC session after the last command is executed.

Using Conformal With Virtuoso

For information on how to use the Virtuoso to Conformal interface, refer to the README file and demo located at:

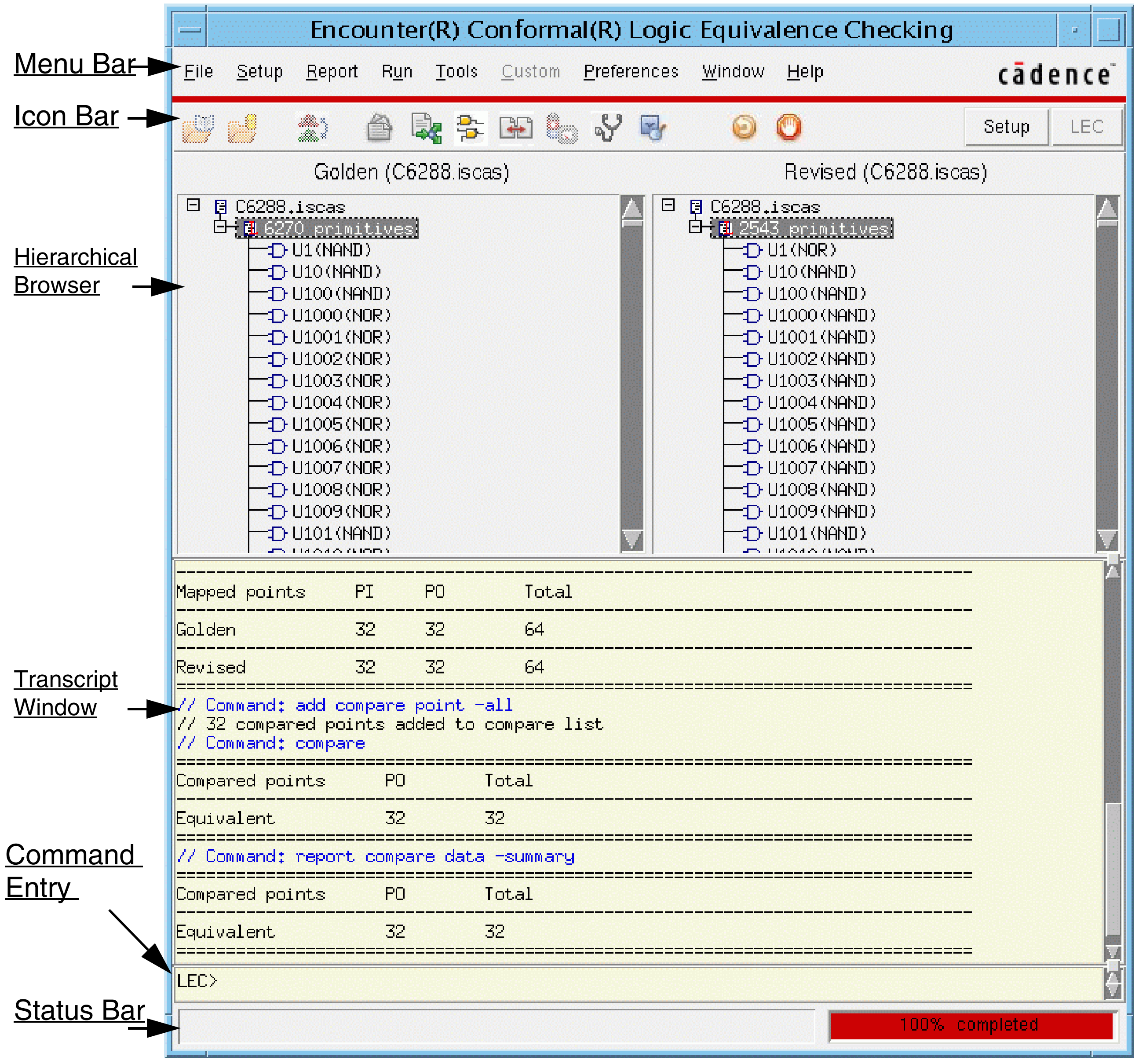
```
<install_dir>/share/cfm/lec/demo/virtuoso
```

3

Using the Graphical User Interface

Main Window

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



```
C6288.iscas
  6270 primitives
    U1(NAND)
    U10(NAND)
    U100(NAND)
    U1000(NOR)
    U1001(NOR)
    U1002(NOR)
    U1003(NOR)
    U1004(NOR)
    U1005(NOR)
    U1006(NOR)
    U1007(NOR)
    U1008(NOR)
    U1009(NOR)
    U101(NAND)
```

```
C6288.iscas
  2543 primitives
    U1(NOR)
    U10(NAND)
    U100(NAND)
    U1000(NAND)
    U1001(NAND)
    U1002(NAND)
    U1003(NAND)
    U1004(NAND)
    U1005(NAND)
    U1006(NAND)
    U1007(NAND)
    U1008(NAND)
    U1009(NAND)
    U101(NAND)
```

```
--------------------------------------------------------------------------------
Mapped points      PI      PO          Total
--------------------------------------------------------------------------------
Golden             32      32          64
--------------------------------------------------------------------------------
Revised            32      32          64
================================================================================
// Command: add compare point -all
// 32 compared points added to compare list
// Command: compare
================================================================================
Compared points         PO          Total
--------------------------------------------------------------------------------
Equivalent              32          32
================================================================================
// Command: report compare data -summary
================================================================================
Compared points         PO          Total
--------------------------------------------------------------------------------
Equivalent              32          32
================================================================================
LEC>
```

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.

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 LEC). 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 Icon Bar, which is located in the main window, contains icon buttons that run specific commands. Click an icon to run the related command or access the related tool. If an icon is not highlighted, it is not available in your current system mode. For example, when the system is in Setup mode, the *Mapping Manager* and *Diagnosis Manager* icons are not highlighted, since they are available in the LEC operating mode only. The following tables defines the icons.

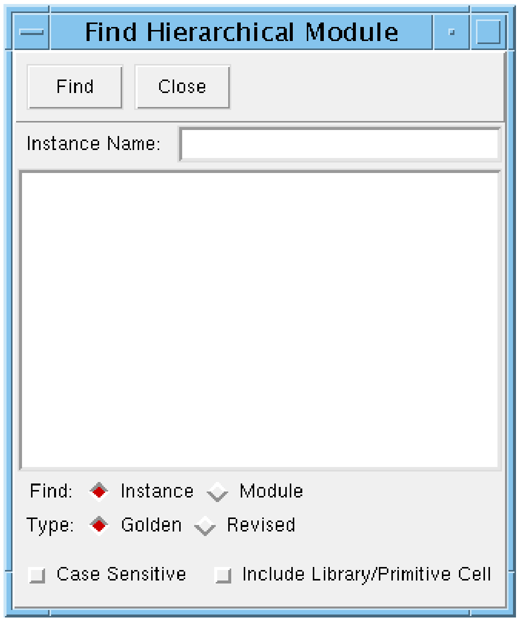
To display or hide the Icon Bar in the main window, choose the *Preferences – Icon Bar* check box.

Icon	Icon Name	Description
	Read Library	Specifies library filenames.
	Read Design	Specifies design filenames.
	Hierarchical Compare	Compares two hierarchical designs. See Hierarchical Module Comparison Window on page 242.
	Source Code Manager	Displays the design’s source code.
	HDL Rule Manager	Displays library and design rule checks.
	Gate Manager	Helps to diagnose and debug designs.
	Mapping Manager	Helps to manage unmapped, mapped, and compared points.
	Diagnosis Manager	Displays the error patterns and error candidates for non-equivalent points.
	Find	Opens the Find Hierarchical Module form to locate an instance in the Hierarchical Browser.

	Refresh	Refreshes the main window display and compresses all modules.
	Stop	Interrupts mapping and comparison.
Setup	Setup Mode button	Changes the system mode to Setup.
Verify	Verify Mode button	Changes the system mode to Verify.
cādence™	*About*	Opens the Company and 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 Hierarchical Module

Find Close

Instance Name:

Find: [✓ Instance] Module
Type: [✓ Golden] Revised

☐ 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.
Type	Specifies either a *Golden* or *Revised* object type 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 in the Golden or Revised design 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 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 in the Golden or Revised design to select it.
2. Right-click and drag the cursor to choose a command from the pop-up menu.

The following tables list the executable commands.

Running Commands on the Root Module

In the Setup system mode, you can run certain commands from within the Hierarchical Browser window. The following commands relate to the root module.

Pop-Up Menu Command	**Description**
Pin Constraints	Opens the Pin Constraints form. See Pin Constraints on page 119.
Pin Equivalences	Opens the Pin Equivalences form. See Pin Equivalences on page 122.
Add Primary Input	Opens the Add Primary Inputs form. See Adding Primary Inputs on page 125.
Add Primary Output	Opens the Add Primary Outputs form. See Adding Primary Outputs on page 127.
Add Cut Point	Opens the Add Cut Point form. See Adding Cut Points on page 134.
Tied Signals	Opens the Tied Signals form. See Tied Signals on page 130.
Source Code Manager	Opens the Source Code viewer for the module.
Schematics	Opens the schematic of the root module.

Running Commands on a Module or Instance Other Than Root

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

Pop-Up Menu Command	**Description**
Root Module	Opens the Root Module form. See Changing the Root Module on page 96.

Pop-Up Menu Command	Description
Add Blackbox	*Instance* defines the selected instance as a blackbox. *Module* defines the selected module as a blackbox. Conformal inserts a blackbox symbol next to the module or instance name. See Transcript Window on page 72.
Add Primary Input	Opens the Add Primary Inputs form. See Adding Primary Inputs on page 125.
Add Primary Output	Opens the Add Primary Outputs form. See Adding Primary Outputs on page 127.
Add Cut Point	Opens the Add Cut Point form. See Adding Cut Points on page 134.
Tied Signals	Opens the Tied Signals window. See Tied Signals on page 130.
Report Gate	Opens the Gate Manager. See Gate Manager on page 176.
Source Code	*Instance* opens the Source Code viewer and highlights the selected instance. *Module* opens the Source Code viewer and highlights the selected module.
Schematics	Opens the schematic of the selected module.
Flat Schematics	Opens the schematic for the highlighted instance.
Parent Module	Highlights the parent module of the selected instance.

Transcript Window

The Transcript window is located in the main Conformal Equivalence Checker 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
- Mapping Manager sorting option
- Mapping Manager display object class selection (for example, show only non-equivalent points)
- Mapping Manager region display option

Using Default Settings in Future Sessions

If you exit from the Exit window and save preferences, you can later reverse this choice and use default settings in future sessions. Remove the `.conformal_gui.rc` file from your home directory or reset the default settings with the `-resetrc` options when you begin a session:

```
rm ~/.conformal_gui.rc
conformal -resetrc
```

Exit Status Codes

If you have done some debugging, the Conformal software returns a status code. The exit status code consists of flags that represent different conditions.

For more information, including a table that lists of flags, their descriptions, and examples, see <u>Exit Status Codes</u> on page 218.

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 105.

- *Read Design*—Specify the design filenames the Conformal software reads in as the Golden and Revised designs.

  See Read Design Form on page 109.

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

  See Saving a Dofile on page 43.

- *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 44.

- *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 Saving a Transcript File on page 50.

- *Reset Design*—Reset the system to its initial state. This deletes all existing designs and libraries and cancels all previously issued commands.

- *Save Session*—Saves your session up to a current point and outputs the session file in a binary format. You can then restore the session later using the *Restore Session* command. You can use this command if priorities demand that another session preempt your session.

- *Restore Session*—Restores a session you previously initiated and saved using the *Save Session* command. When invoked, Conformal will go back to its initial state then restores the state from the saved session.

  See Saving and Restoring a Session on page 47.

- *Exit GUI*—Switch Conformal from the GUI mode to the non-GUI command line mode.

- *Exit*—Exit the Conformal software completely.

  See Exiting the GUI and Software on page 73.

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 49.
- *Alias*—Add, delete, or view alias names in a session.

  See Alias Form on page 51.
- *Search Path*—Create, modify, or delete directory search paths.

  See Adding Search Paths on page 97.
- *Environment*—Set global options for the Golden and Revised designs and for mapping and comparison operations.

  See Setting Global Options For Mapping and Comparison on page 100.
- *Pin Constraints*—Add and delete pin constraints to primary input pins.

  See Pin Constraints on page 119.
- *Pin Equivalences*—Add and delete pin equivalences.

  See Pin Equivalences on page 122.
- *Primary Input*—Add and delete primary inputs.

  See Primary Inputs on page 124.
- *Primary Output*—Add and delete primary outputs.

  See Primary Outputs on page 127.
- *Cut Point*—Add and delete cut points.

  See Cut Points on page 134.
- *Tied Signals*—Add and delete tied signals to floating nets and pins.

  See Tied Signals on page 130.
- *Root Module*—Change the Conformal automatic root module assignment and to specify the name of the new root module in the Golden and Revised designs.

  See Changing the Root Module on page 96.

- *Renaming Rule*—Add or delete renaming rules to guide mapping, help map modules for hierarchical comparisons, or rename pin names of blackboxes.

  See Renaming Rules on page 147.

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

  See Adding Notranslate Modules on page 98.

- *Flatten Model*—Set global options for flattening and modeling, which occur when exiting Setup system mode.

  See Flatten Model Form on page 144.

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 49.

Reporting contains the following categories. You can either select these from the *Report* menu, or from the form's *Report Type* pull-down menu.

- *Black Box*—Displays black boxes from the Golden and Revised designs.
- *Cut Points*—Displays cut points from the Golden and Revised designs.
- *Design Data*—Displays the number of design modules, library cells, inputs, outputs, primitives, and one-to-one mapped state points on the Golden and Revised designs.
- *Environment*—Displays global settings for the Golden and Revised designs and system settings.
- *Floating Signals*—Displays all floating signals in the Golden and Revised designs or in specified modules of a design.
- *Instance Constraints*—Displays the constraints placed on instances in the Golden and Revised designs.
- *Instance Equivalences*—Displays the equivalences placed on instances in the Golden and Revised designs.
- *Messages*—Displays either a summary or complete list of the warning messages that come from the modeling, mapping, or comparison process.
- *Modules*—Displays module information for the Golden and Revised designs.

- *Notranslate Modules*—Displays all of the library and design modules that Conformal will not compile when reading in libraries and designs.
- *Pin Constraints*—Displays the constraints placed on primary input pins in the Golden and Revised designs.
- *Pin Equivalences*—Displays a list of added pin equivalences and inverted pin equivalences.
- *Primary Inputs*—Displays primary input pins from the Golden and Revised designs.
- *Primary Outputs*—Displays primary output pins from the Golden and Revised designs.
- *Renaming Rule*—Displays all of the library and design modules that Conformal will not compile when reading in libraries and designs.
- *Search Path*—Displays the paths Conformal searches to locate filenames included in the Conformal software.
- *Tied Signals*—Displays the list of renaming rules for mapping, module, and pin renaming.

The following options are for the Mapping Manager window. For more information about these features and related functionality, see Mapping Manager on page 188.

- *Mapped Points*—Opens the Mapping Manager and displays the mapped points that were automatically identified or added with the Conformal software. Each mapped point from the Golden and Revised design is displayed along with a summary of all Golden and Revised mapped points.
- *Unmapped Points*—Opens the Mapping Manager and displays a list of unmapped points, along with a summary of all of the unmapped points in the Golden and Revised designs.
- *Compared Points*—Opens the Mapping Manager and displays the compared points that were added with the Conformal software.
- *Compare Data*—Opens the Mapping Manager and displays a list of all or specified compared points.
- *Statistics*—Displays the mapping and comparison statistics for the Golden and Revised designs.

Run Menu - Compare

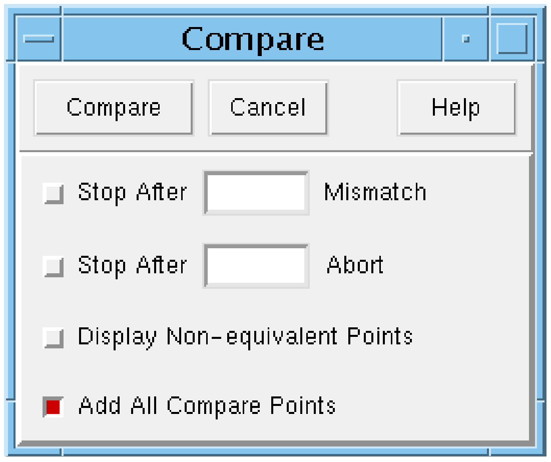
Use the Compare form to add all of the compare points and run the equivalency checking comparison between the Golden and Revised designs. You can also stop the comparison after Conformal encounters a specified number of abort points or mismatches.

When Conformal completes the comparison, it displays a summary table of the number of equivalent and non-equivalent compared points in the Transcript window.

Tip

You can also run the `COMPARE` command in the Mapping Manager to compare specified points or all points (see Comparing Key Points on page 197).

➤ Choose *Run – Compare*.



☐ Stop After [ ] Mismatch

☐ Stop After [ ] Abort

☐ Display Non-equivalent Points

☑ Add All Compare Points

Compare Form Fields and Options

Stop After # Mismatch	Specifies the number of non-equivalent points where the comparison stops.
Stop After # Abort	Specifies the number of abort points where the comparison stops.
Display Non-equivalent Points	Displays the non-equivalent points as they are found during the comparison.
Add All Compare Points	Automatically adds all compare points (the default).

Tools Menu

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

- *HDL Rule Manager*—Display all of the library and design rule checks the Conformal software runs during parsing.

  See HDL Rule Manager on page 86.

  For more information on the HDL Rule Checks that Conformal performs, see the *Conformal HDL Rule Check Reference*.

- *Gate Manager*—Helps diagnose and debug your designs.

  See Gate Manager on page 176

- *Mapping Manager*—Serves as a gateway to the integrated debugging environment.

  See Mapping Manager on page 188

- *Diagnosis Manager*—Display the error patterns and error candidates for non-equivalent points.

  See Diagnosis Manager on page 205

- *Hierarchical Compare*—Run a module-by-module, bottom-up, hierarchical comparison on two hierarchical designs.

  See Hierarchical Module Comparison Window on page 242

- *LowPower Manager*—Display unmapped and checked points, check key points, and report the status of each checked point.

  See Low Power Manager in the *Conformal Low Power User Guide* for more information.

4
Command Line Features

- Command Line Editing on page 81
- Command Line Completion on page 83

Command Line Editing

The non-GUI terminal of any Conformal tool supports the following editing functions:

In the following table, `^F` indicates pressing the `Ctrl` key and the `F` key simultaneously. Function keys have their names enclosed in angle brackets, for example, `<ESC>` is the Escape key.

The key sequences for basic editing functions are summarized in the following table.

Figure 4-1 Basic Editing Functions

Keys	Function
`^F` or `<right-arrow>`	Move the cursor one character to the right
`^B` or `<left-arrow>`	Move the cursor one character to the left
`^A`	Move the cursor to the beginning of the line
`^E`	Move the cursor to the end of the line
`^U`	Delete the entire line
`^K`	Delete all characters from the cursor position to the end of the line
`<ESC>f`	Move the cursor forward by one word
`<ESC>b`	Move the cursor backward by one word
`^D`	Delete the character under the cursor

`^H` or `<Backspace>` or `^?`	Delete the character to the left of the cursor
`^R`	Redisplay the current line
`^L`	Clear the screen and show the current line at the top of the screen
`^I` or `<TAB>`	Complete word (See section below)

Every command that is successfully entered is saved in a history list. You can recall commands in the history list to avoid repeated typing. The history list has a size limit of 256k bytes and the oldest commands in the list will be discarded when this limit is exceeded.

Figure 4-2 Command Line History

Key	Function
`^P` or `<up-arrow>`	Recall the previous history line
`^N` or `<down-arrow>`	Recall the next history line
`<ESC>p`	History search backward
`<ESC>n`	history search forward
`^Xh`	List the history
`<ESC><`	Recall the first history line
`<ESC>>`	Recall the last history line
`^D`	List all possible completions when cursor is at end of line

The history search capability looks into the history list for one that matches the beginning of the current line. If the search string contains wildcards (`*`, `?`), then the entire pattern is matched. This is useful for searching patterns in the middle of a line.

For example:

```
SETUP> usage
SETUP> echo hello
SETUP> echo bye
SETUP> us<ESC>p
SETUP> usage
SETUP> *hello*<ESC>p
SETUP> echo hello
```

The command line history can also be saved into a file using the command `SAVE DOFILE`.

Command Line Completion

Command line completion (or tab completion) is when the tool automatically fills in partially typed commands. The tool supports command line completion in VPX and TCL mode (using the appropriate commands for each mode).

Using Command Line Completion

Completion mode is activated by the `<TAB>` key. For example, if you press the `<TAB>` key after typing "`re`", you will get the following possible command completions:

```
SETUP> re<TAB>
read... remodel* report... reset... restore... reduce... remove* reset*
```

Partial command completions are listed with the postfix "`...`"; complete command completions are listed with the postfix "`*`". In the example above, the `remodel` command is complete, commands like "`read`" are not. To narrow the choices, type more characters. For example, press `<TAB>` after typing "`read`" will show the commands that begin with "`read`":

```
SETUP> read<TAB>
read cpf* read lef... read memory... read rule... read design* read library* read
pattern* read testcase* read fsm... read mapped... read rom...
```

Valid abbreviated commands are understood during command completions as illustrated below:

```
SETUP> ana m<TAB>
ana module* ana multiplier*
```

When there is only one choice, the command is completed automatically. For example, pressing `<TAB>` after typing "`read li`" will complete the command "`read library`". Typing `^D` (when the cursor is at the end of a line) lists the possible completions without making any completions. *Beware that using ^D on an empty line will terminate the tool.*

Command completion understands the `VPX` and `MAN` commands, and will complete the commands that come after. For example,

```
SETUP> man write l<TAB> SETUP> man write library
```

After the command is completed, pressing `<TAB>` will activate filename completion.

Repeating Actions

The effect of pressing a key can be repeated automatically by giving it a repeat count using the key sequence `<ESC>`*`numberX`* where *`number`* is the count in one or more digits, and `X` is the key to be repeated. For example, the following example repeats the single character deletion using the `<Backspace>` key 20 times.

```
SETUP> abcdef01234567890123456789<ESC>20<Backspace>
SETUP> abcdef
```

Notes

- Command completion completes one word at a time. For example, the partial input `"write ru"` needs two completions, one for `"rule"`, and one for `"check"` to result in the completed `"write rule check"` command. However, since there are no other commands that begin with `"write ru"`, only one completion should be necessary.
- Options are not completed.

5

Managing Rule Checks

HDL Rule Manager

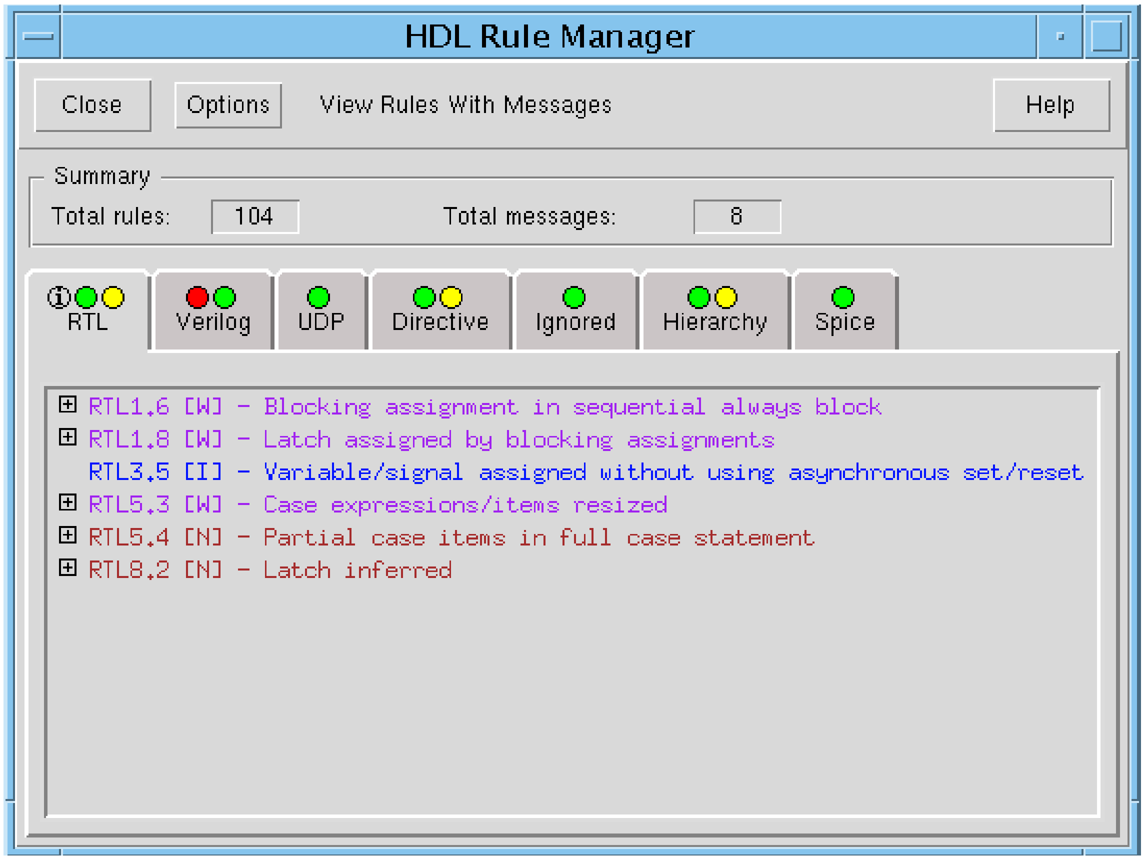
HDL rules consist of a group of desirable rules that should be observed during design analysis, elaboration, and RTL construction. For example, the checker notifies you of the presence of UDPs, directives, and hierarchical coding; and alerts you to code that might lead to RTL and gate-level simulation 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.



```
⊞ RTL1.6 [W] - Blocking assignment in sequential always block
⊞ RTL1.8 [W] - Latch assigned by blocking assignments
  RTL3.5 [I] - Variable/signal assigned without using asynchronous set/reset
⊞ RTL5.3 [W] - Case expressions/items resized
⊞ RTL5.4 [N] - Partial case items in full case statement
⊞ RTL8.2 [N] - Latch inferred
```

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

- Changing the Severity of Rule Checks on page 89
- Enabling and Disabling Rule Checks on page 89
- Running Incremental Rule Checks on page 89
- Reporting Messages for Rule Checks on page 90
- Viewing a Specific Message on page 90
- Viewing Source Code for an Occurrence on page 90

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), or *All* to display a complete list of rules and the messages (violations) for each page. 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.
Summary	For each page, this displays the total number of rules for the specified category, and the total number of rule violation occurrences (messages).

Severity Levels

The severity levels are listed below from the most serious to the least serious:

- Error—The Conformal software might not allow you to begin verification until you resolve the error.
- Warning—The Conformal software allows you to begin verification; however, it warns you of potential errors in the design.
- Note—The Conformal software allows you to begin verification; however, it flags potential errors in the design.
- Ignore—The Conformal software does not report this severity by default.

You can change the level of severity for rule violations with the `SET RULE HANDLING` command. You must use this command during Setup *before* reading in library or design files. Alternatively, you can use the HDL Rule Manager to change the severity (see Changing the Severity of Rule Checks on page 89.)

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: Golden        Severity: Warning
    Type: Revised       Severity: Warning
    Type: Golden library       Severity: Warning
    Type: Revised 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: Golden        Severity: Error
    Type: Revised       Severity: Error
    Type: Golden library       Severity: Error
    Type: Revised 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 the Severity of Rule Checks

To change the severity of the rule handling in the HDL Rule Manager, use the following procedure in Setup mode and *before* you read in the library, designs, and SDC files, do the following:

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

   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 `SET RULE FILTER` command to filter out rules that occur in modules outside the root hierarchy.

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> -Golden` (or `-revised`) 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 HDL 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.

Viewing a Specific Message

Use the following procedure in the HDL Rule Manager to view the verbose listing of a specific rule messages.

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.

Viewing Source Code for an Occurrence

To investigate HDL source code violations from the HDL Rule Manager, do the following:

1. Click the *+* symbol preceding a highlighted rule to fully expand the entry.
2. Click an occurrence to select it.

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

   The Source Code Manager opens and Conformal highlights the relevant line of code.

Modeling Messages

Modeling messages indicate any modeling warnings encountered during the analysis and modeling of the design. Each modeling message is prefixed by `F*` because they require a flattened design.

You can view a summary or expanded report of all rule violations with one of the following commands:

- `report messages -modeling -verbose`
- `report messages -modeling -summary`

To view information for a specific message, use the `HELP` command followed by the message name.

```
HELp [message_name]
```

For example:

```
help F1
```

To view all rule check messages, use the `HELP` command with the `-message` option:

```
help -message
```

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

6

Using the Setup Mode

Overview

The Conformal software has two operating modes, Setup and LEC. After startup, the Conformal software begins operation in the Setup mode, as indicated by the `SETUP>` prompt in the command entry window. In the Setup mode, you can read in the library and design, apply constraints, and set up options for verification.

Setting Options

The following sections describe the settings you can apply *before* reading in the library and design files.

- Change the Severity of Rule Checks

  See Chapter 5, "Managing Rule Checks" for more information.

- Specify Case Sensitivity

  Use the `SET CASE SENSITIVITY` command to specify whether any names you use 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 all or specified vendor 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 Equivalence Checking Reference Manual*, or type `help set directive -verbose` in the command line.

- Specify Text Handling Rules

  Use the `SET NAMING RULE` command to specify 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 Verilog and VHDL design files.

- Set Undefined Cells

  All referenced modules must be either defined or blackboxed. When Conformal finds an undefined cell, it reports an error. To prevent an error, choose to blackbox undefined cells using the `SET UNDEFINED CELL` command.

  Note: For information on replacing blackboxed modules with synthesized modules, see the `WRITE BLACKBOX WRAPPER` or `SUBSTITUTE BLACKBOX WRAPPER` commands.

■ Set Undefined Ports

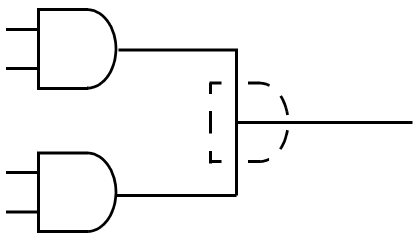
Undefined ports in the design or library cause an error message unless you choose to ignore them using the `SET UNDEFINED PORT` command.

■ Specify Undriven Signals

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

■ Set Wire Resolution

To specify the output behavior of multi-driven nets as either an AND or OR gate, use the `SET WIRE RESOLUTION` command. The following illustrates Wired-AND behavior.



■ Set a Design Root Level

Specify a different root module so that when reading in the Golden and Revised designs, the top module of each is *not* treated as the root module by default.

For more information, see Changing the Root Module on page 96.

■ Add Search Paths

Specify the location of HDL files or libraries that must be included in the session, but are not in the current directory.

For more information, see Adding Search Paths on page 97.

■ Add Notranslate Modules

Choose not to compile specific library or design modules so that the specified modules (for example, non-synthesizable and memory modules) automatically become blackboxes.

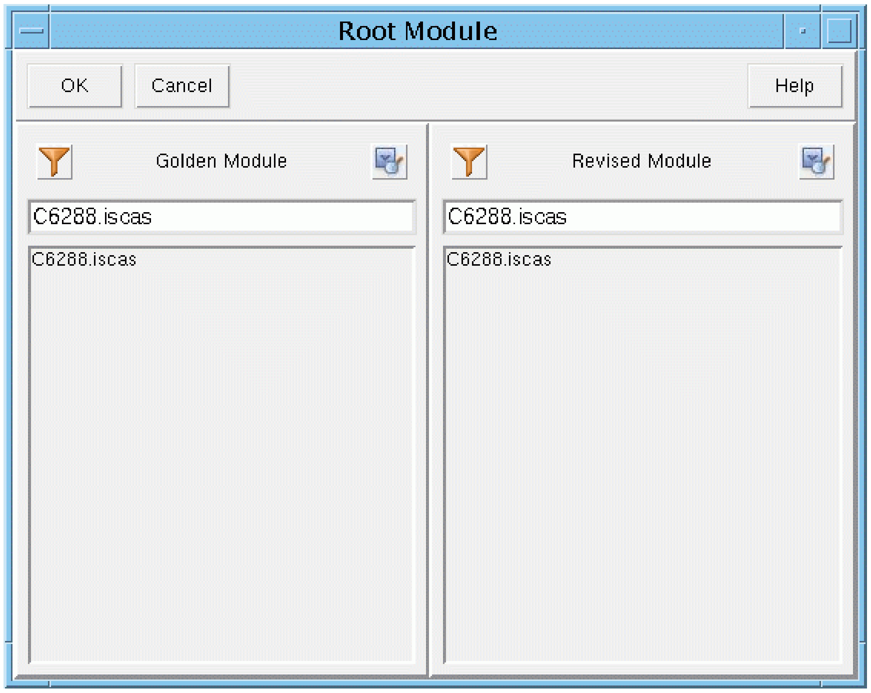
For more information, see Adding Notranslate Modules on page 98.

Changing the Root Module

You can change the Conformal automatic root module assignment and to specify the name of the new root module in the Golden and Revised designs. Use the SET ROOT MODULE command or in the Hierarchical Browser, do the following to open the Root Module form:

1. Click a module name in the *Golden* or *Revised* column to select the name.
2. Right-click to open the pop-up menu and choose *Root Module*.

Alternatively, you can open this form from the main window (*Setup – Root Module*).



Golden Module: C6288.iscas
C6288.iscas

Revised Module: C6288.iscas
C6288.iscas

The current root module is displayed in the *Golden Module* and *Revised Module* fields.

Specifying a New Root Module

To manually specify a new root module, double-click a *module name* in the *Golden Module* or *Revised Module* list to add the root module in the field above the list and click *OK*.

Sorting Module Lists

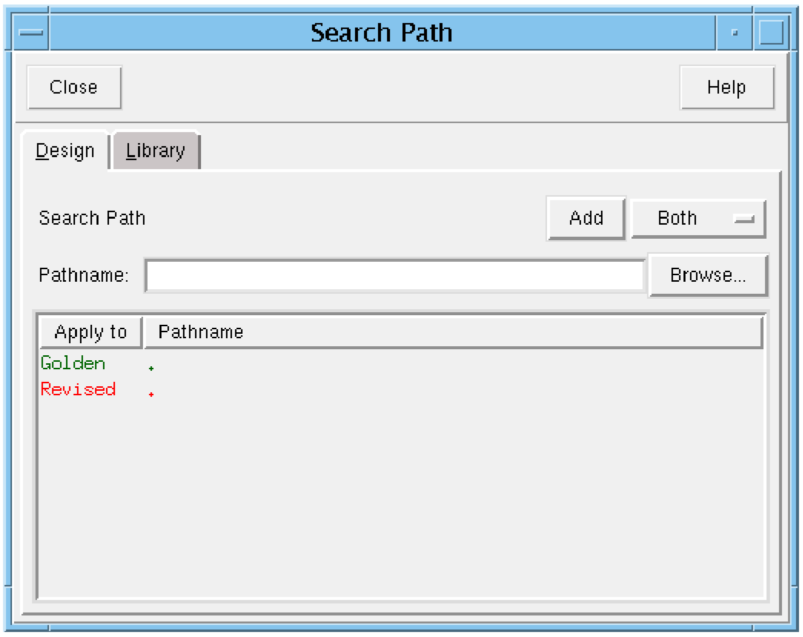
To alphabetically sort the *Golden Module* and *Revised Module* list boxes, right-click in the appropriate column and choose *Sort* from the pop-up menu.

Adding Search Paths

Use the `ADD SEARCH PATH` command, or the Search Path form (*Setup – Search Path*), to 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. The Conformal software searches for library and design files in the order of the paths listed from left to right in the command string.

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.



Apply to	Pathname
Golden	.
Revised	.

Search Path Form Fields and Options

Pathname	Specifies the search path. You can type the directory path or click *Browse* to locate the path.
Add	Adds the directory path to the list in the *Pathname* list box. Use the pull-down window to select *Both*, *Golden*, or *Revised* design type.

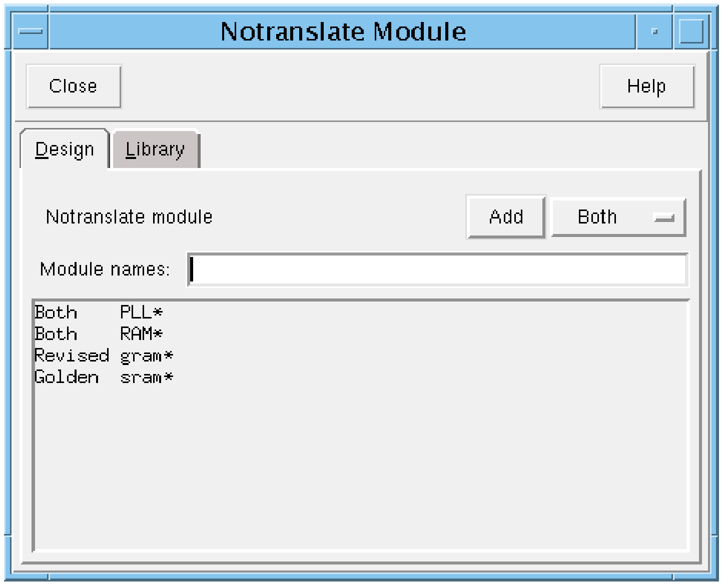
Pathname list box	Lists the directory search paths. To delete directory search paths, right-click on a path to bring up the pull-down menu and select either a *Delete Search Path* or *Delete All Search Paths*.

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.



```
Both    PLL*
Both    RAM*
Revised gram*
Golden  sram*
```

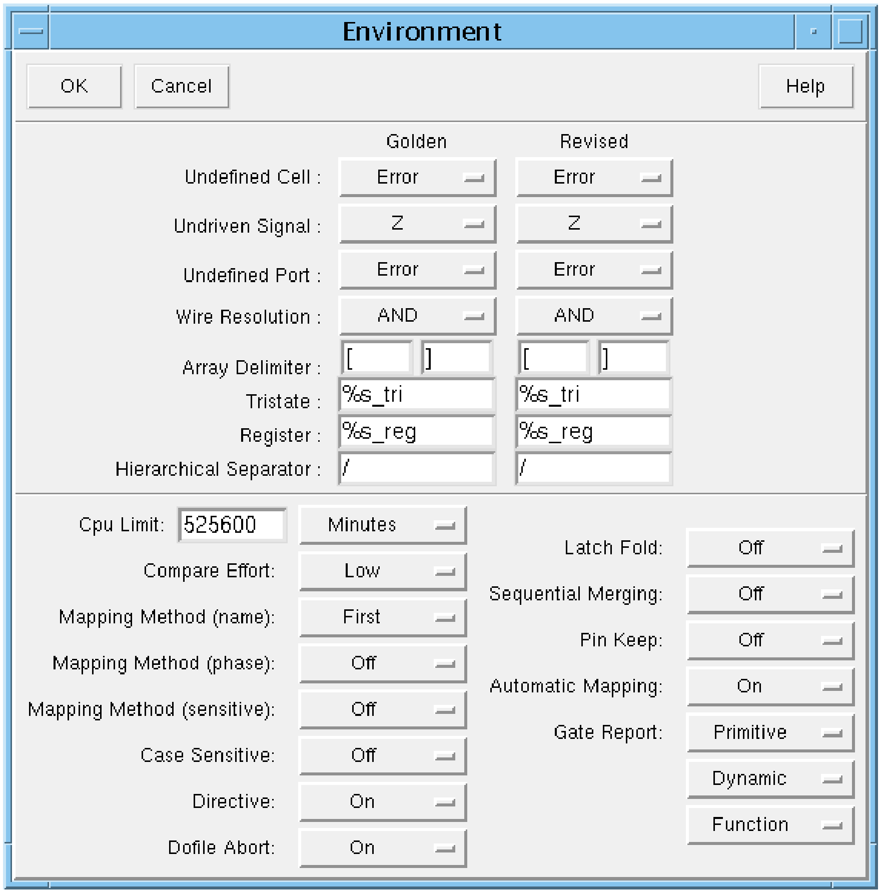
To delete one or all notranslate modules from designs and libraries, click a module name in the list box in the *Design* or *Library* page, and right click to open the pop-up menu and choose *Delete Notranslate Module* to delete a single notranslate module, and *Delete All Notranslate Module* to delete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

Notranslate Module Form Fields and Options

Add	Add the notranslate modules, and adds the notranslate module names to the list box.
Both	Applies the notranslate modules to both the Golden and Revised designs (the default). You can use this pull-down menu to select *Golden* to apply them to Golden designs, or *Revised* to apply them to Revised designs
Module name list box	To delete notranslate modules, right-click on a path to bring up the pull-down menu and select *Delete Notranslate Module* to delete a single notranslate module, or *Delete All Notranslate Modules.*

Setting Global Options For Mapping and Comparison

Use the Environment form (*Setup – Environment*) to set global options for the Golden and Revised designs and for mapping and comparison operations.



Environment

OK | Cancel | Help

	Golden	Revised
Undefined Cell :	Error	Error
Undriven Signal :	Z	Z
Undefined Port :	Error	Error
Wire Resolution :	AND	AND
Array Delimiter :	[ ]	[ ]
Tristate :	%s_tri	%s_tri
Register :	%s_reg	%s_reg
Hierarchical Separator :	/	/

Cpu Limit:	525600 Minutes	Latch Fold:	Off
Compare Effort:	Low	Sequential Merging:	Off
Mapping Method (name):	First	Pin Keep:	Off
Mapping Method (phase):	Off	Automatic Mapping:	On
Mapping Method (sensitive):	Off	Gate Report:	Primitive
Case Sensitive:	Off		Dynamic
Directive:	On		Function
Dofile Abort:	On		

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*.
Wire Resolution	Specifies how the Conformal software treats the output behavior of multi-driven nets. Click the pull-down menu to choose either *And* or *Or*.
Array Delimiter	Specifies the array delimiter rule for reading in a Verilog RTL or hierarchical design.
Tristate	Specifies the tristate rule for reading in a Verilog RTL or hierarchical design.
Register	Specifies the register rule for reading in a Verilog RTL or hierarchical design.
Hierarchical Separator	Specifies the hierarchical separator rule for reading in a Verilog RTL or hierarchical design.
Cpu Limit	Specifies CPU time limit for the Conformal equivalence checking compare effort. Type a positive integer to change the CPU time, and click on the pull-down menu to choose *Minutes*, *Hours*, or *Days*.
Compare Effort	Specifies the amount of effort equivalency checking applies for a particular gate. Click the pull-down menu to choose *Low*, *Medium*, or *High*.
Mapping Method (name)	Specifies the mapping method for names. Choose one of the following (to a certain degree, the mapping method operates under the following modes.) ■ *Name First*—Conformal first maps the key points with the paths of the gates, then uses the mapping algorithm to map the rest of the key points. ■ *Name Only*—only maps the key points based on the paths of the gates. ■ *Name Guide*—Conformal first maps key points with a mapping algorithm, then maps the rest of the key points based on paths of the gates. ■ *Name None*— will not map key points based on the paths of the gates. If the mapping algorithm cannot map a key point, it remains unmapped.

Mapping Method (phase)	In the *On* position, Conformal maps key points with an inverted phase. Inverted-phase compared points can either be Inverted-equivalent or Non-equivalent.
Mapping Method (sensitive)	In the *On* position, Conformal maps key points with case-sensitive key point names.
Case Sensitive	*On* specifies that names you use are case-sensitive.
Directive	*On* enables the effects of 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.
Latch Folding	In the *On* position, Conformal converts two latches in a simple LSSD format into a single DFF gate.
Sequential Merging	In the *On* position, Conformal merges common groups of sequential elements into one sequential element in the clock cone of a DFF or D-latch.
Pin Keep	In the *On* position, Conformal retains all of the gate pin information for gate reporting. Use this option when reporting gate information at the design level. It increases the memory use.
Automatic Mapping	In the *On* position, Conformal automatically maps key points when you change the system mode from Setup to LEC.

Gate Report	Specifies the level of detail in the Conformal gate information display. This applies to the three pull-down menus that precede it. ■ *Primitive*—displays the gate report information at the primitive level. ■ *Design*—displays the gate report information at the design level. ■ *Dynamic*—displays the dynamic constraints in the gate report information. ■ *No Dynamic*—does not display the dynamic constraints in the gate report information. ■ Click the bar on the fan-in cones cyclic field and choose one of the following to specify the display: ■ *Function*—does not display the fan-in cone of the zero/one gates in the gate report information. ■ *Structure*—displays the fan-in cone of the zero/one gates in the gate report information.

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 read in the library *before* reading in the design. You can use the `READ LIBRARY` command, or the Read Library form in GUI mode (*File – Read Library*). See Read Library Form on page 105 for a description of the fields and options.

If there are duplicate modules, Conformal uses the first module found and ignores all others. However, you can use `READ LIBRARY -lastmod` to specify that Conformal use the last module and ignore the earlier ones. The library can also be in the Synopsys Liberty format.

Note: For RTL to gate formal equivalence checking, use simulation libraries instead of synthesis libraries because design verification sign-off happens for simulation libraries—not for synthesis libraries.

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 of the library files after the `READ LIBRARY` command explicitly or using 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 library file1.v file2.v file3.v... \
-verilog -Golden
```

Or

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

Method 2:

1. Create a file containing all of 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. Append the name of this newly created file to the `READ LIBRARY` command:

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

Method 3:

Read multiple library files of different languages:

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

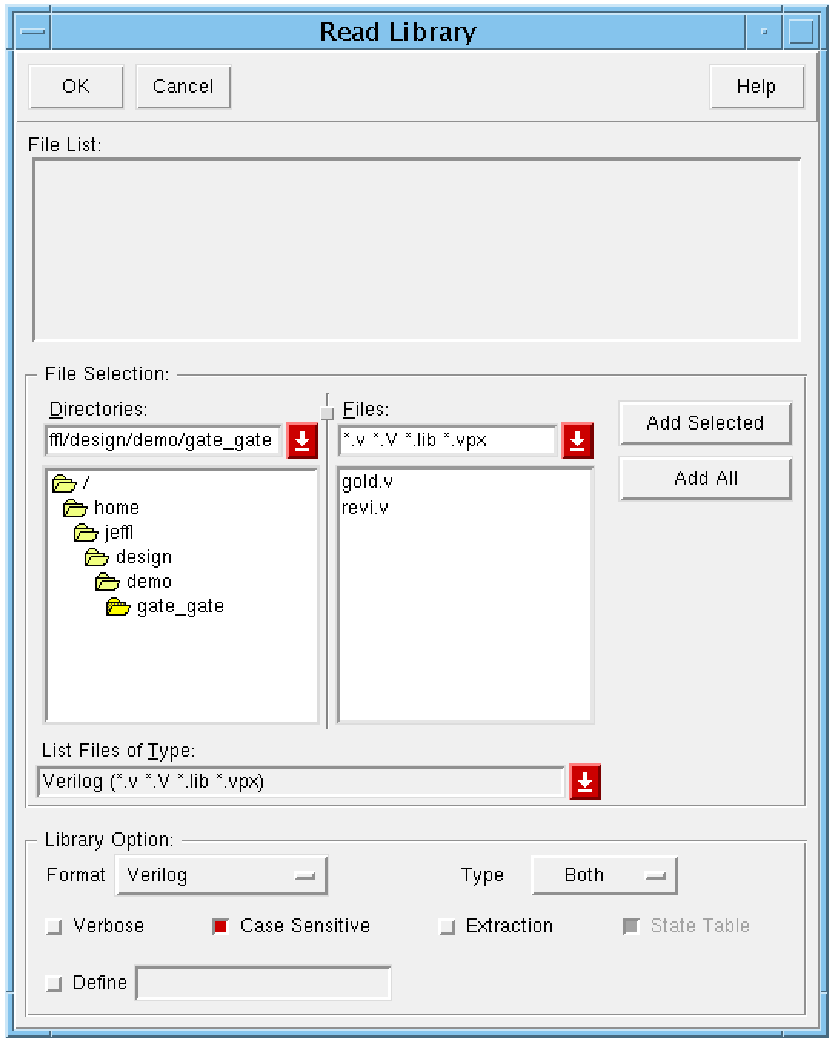
Writing Libraries

After you read in a library, Conformal can write it out in Verilog format. This command is useful for learning 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 to specify library filenames you will include with a design.

➤ Choose *File – Read Library*.



Directories: ffl/design/demo/gate_gate

/
home
jeffl
design
demo
gate_gate

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

gold.v
revi.v

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

Format: Verilog

Type: Both

Verbose

[✓ Case Sensitive]

Extraction

State Table

Define

Read Library Fields and Options

File List	Lists the library files that the Conformal Equivalence Checker reads in for this session. As you build the list of files, the Conformal Equivalence Checker 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.
Type	Specifies the library type. You can choose *Golden*, *Revised*, or *Both*.
Verbose	Displays the verbose messages for parsing and translating each library module.
Case Sensitive	Specifies that the Conformal Equivalence Checker should handle the library as case sensitive. **Note:** This option is not available for VHDL.
Extraction	Specifies that the Conformal Equivalence Checker is to abstract transistor models into gate models.
State Table	Specifies that the library contains Synopsys Liberty state tables. Conformal can handle state tables that have single asynchronous inputs and no overlapping rule outputs. This option is only available when selecting *Liberty* from the *Format* pull-down menu.
Define	Specifies the text macro name you want to define. For Verilog formats, enter your Verilog `` `ifdef `` macro in this field.

Reading in Design Files

To read in the Golden and Revised design files, you can use the `READ DESIGN` command, or the Read Design form in GUI mode (*File – Read Design*). See Read Design Form on page 109 for a description of the fields and options.

The supported design formats are Verilog, Verilog2K, SystemVerilog, VHDL, SPICE, EDIF and NDL (LSI Logic's netlist format). When you must replace a design, use `READ DESIGN -replace`. If Conformal finds multiple modules with the same name, it uses the first module and ignores later modules with that name. You can use `READ DESIGN -lastmod` to specify that Conformal use the last module and ignore the earlier ones.

Reading in Mixed-Language Design Files

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

```
read design sub1.vhdl -vhdl -mapfile lib1 lib/pkg1.vhd -Golden -noelaborate
read design sub2.vhdl -vhdl -mapfile lib2 lib/pkg2.vhd -Golden -noelaborate
read design top.v -verilog -Golden
```

Please refer to "VHDL Support" on page 379 and proper library mapping setup for the `READ DESIGN` command. Library in this context refers to the technology library, such as ASIC cell and memory definitions. See `READ DESIGN` for information on reading VHDL libraries and packages

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 -Golden
```

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 `Golden.v` might contain the following:

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

2. Append the name of this newly created file to the `READ DESIGN` command as shown below.

```
read design Golden.v -verilog
```

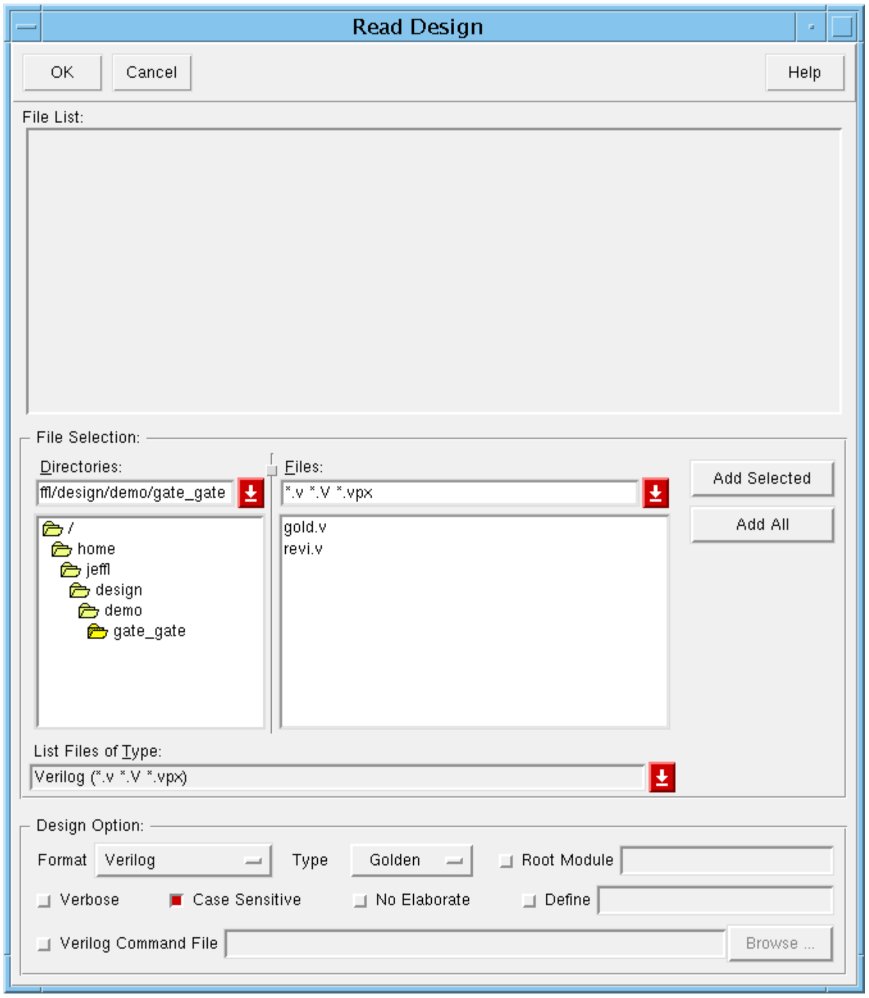
Writing Designs

Use the `WRITE DESIGN` command after you read in a design to write it out in Verilog format. This feature is useful for learning how Conformal parses RTL or transistor-based designs.

Read Design Form

Use the Read Design form to specify the design filenames the Conformal software reads in as the Golden and Revised designs.

➤ Choose *File – Read Design*.



Read Design

File List:

File Selection:

Directories: fl/design/demo/gate_gate

/
home
jeffl
design
demo
gate_gate

Files: *.v *.V *.vpx

gold.v
revi.v

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

Design Option:

Format: Verilog

Type: Golden

Root Module: ☐

Verbose: ☐

Case Sensitive: ☑

No Elaborate: ☐

Define: ☐

Verilog Command File: ☐

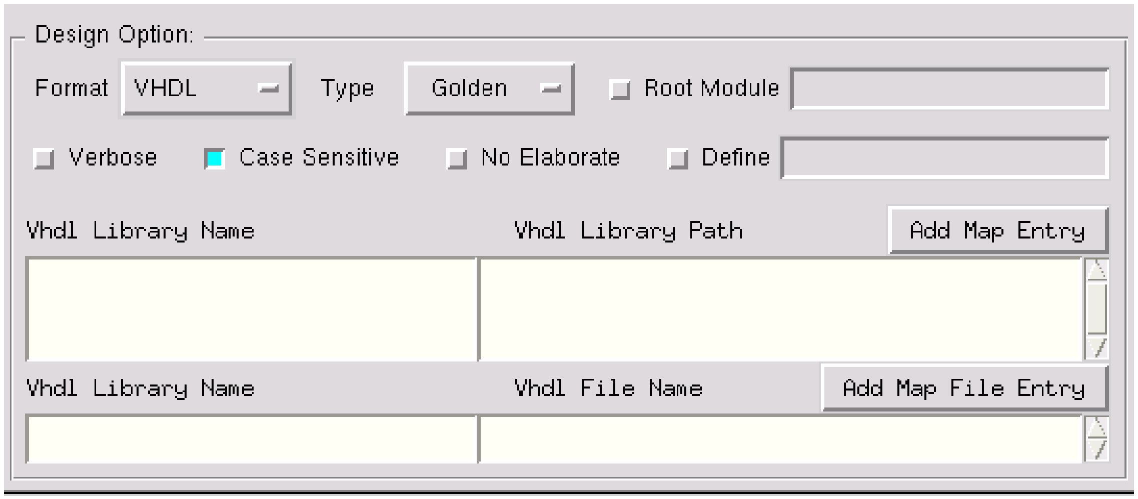
Read Library Fields and Options

File List	Lists the design files that the Conformal Equivalence Checker reads in for this session. As you build the list of files, the Conformal Equivalence Checker 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 111 for more information. When selecting *EDIF*, the bottom portion of the form expands. See Specifying Design Options for EDIF Designs on page 112 for more information.
Type	Specifies the design type. You can choose *Golden*, *Revised*, or *Both*.
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 Equivalence Checker 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.

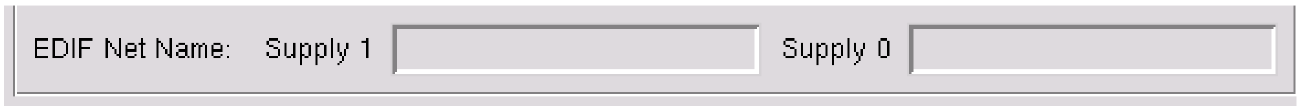

Design Option:
Format VHDL Type Golden Root Module
Verbose [✓ Case Sensitive] No Elaborate Define

Add Map Entry	Opens the Add Vhdl Library Mapping window where you can select the library name and path of the specific VHDL libraries.
Add Map File Entry	Opens the Add Vhdl Library Mapfile window where you can specify exactly which files belong to a given library.

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

Specifying Design Options for EDIF Designs

f the design format is EDIF, the bottom portion of the form expands.


EDIF Net Name: Supply 1 Supply 0

EDIF Net Name	Type EDIF Net Names in the *Supply 1* and *Supply 0* fields to assign a value of 1 or 0 to specified variables.

Using Verilog Command Filelists

The following describes a time-saving method that lets you 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, a directory named `/user/library/verilog` contains the following library files:

```
and.v or.v
dff.v lat.v
```

And in the current working directory, there is a Golden and Revised directory with a gate netlist in each:

```
$CWD/Golden/Golden.v
$CWD/revised/revised.v
```

1. Create a Verilog filelist with the above contents.

   In this example, the names of the Verilog command filelist are `Golden.vc` and `revised.vc`:

   ```
   Golden.vc:
   -y /user/library/verilog
   Golden/Golden.v

   revised.vc:
   -y /user/library/verilog
   revised/revised.v
   ```

2. Run the `READ DESIGN` command with the `-file` option to read in the designs and libraries without using the `READ LIBRARY` command. (See the following examples.)

   ```
   read design -file Golden.vc -verilog -Golden
   read design -file revised.vc -verilog -revised
   ```

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

Comparing Design Hierarchies

The `WRITE HIER_COMPARE DOFILE` command lets you write a dofile script that Conformal can use to compare two hierarchical designs. The `dofile` script verifies the two hierarchical designs starting at the lowest-level modules and progressing to the root module. At the end of the dofile, Conformal displays the total number of equivalent and non-equivalent modules.

The sample dofile below does the following:

- Reads in the two hierarchical designs
- Writes out the hierarchical dofile script
- Compares the design hierarchy

  ```
  read library Golden.lib -verilog -Golden
  ```

```
read design Golden.v -verilog -Golden
read library revised.lib -verilog -revised
read design revised.v -verilog -revised
write hier_compare dofile hier.do -replace
set log file hier.log -replace
dofile hier.do
exit -force
```

See Chapter 10, “Hierarchical Module Comparison Window” for additional information.

Comparing Libraries

In addition to comparing design hierarchies, the `WRITE HIER_COMPARE DOFILE` command compares two libraries, such as Liberty and Verilog libraries. The sample dofile below does the following:

- Reads in a synthesis library and simulation library
- Writes out all of the library models
- Compares the libraries

```
read design syn.lib -liberty -Golden
read design simulation.v -verilog -revised

write hier_compare dofile lib_ver.do -all -replace
set log file lib_ver.log -replace
dofile lib_ver.do
exit -force
```

Design Constraints

After Conformal successfully reads the designs and libraries, you can place constraints on the designs to exclude sections of a design from verification, specify behavior and relationships, and constrain internal nets, instances, and feedback.

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

Blackboxes

When running the Conformal software, there are several ways to blackbox a module in Conformal:

Note: Although the internal logic within a blackbox is not compared, the connections feeding in and out of the blackbox is still verified.

- Use the `ADD NOTRANSLATE MODULE` command

  This command is run before running the `READ DESIGN` and `READ LIBRARY` commands. It instructs Conformal to blackbox the specified module(s).

  With this command, the actual code of the module is not parsed; only the directions for the input and output ports are parsed and used for the blackbox. As a result, it requires less memory in Conformal to process blackboxes. Typically, RAMs and ROMs are blackboxed this way because it would require too much memory in Conformal to process them for comparison. Analog blocks are also blackboxed this way because the code is not synthesizable.

  For more information on this method of adding blackboxes, see Adding Notranslate Modules on page 98.

- Use the `ADD BLACK BOX` command

  This command is used to blackbox a module after the module has already been read in, and is often used in hierarchical comparison.

  For more information on adding blackboxes with this command, see Adding a Blackbox Instance or Module on page 117.

- Use the `SET UNDEFINED CELL -black_box` command

  With both `ADD NOTRANSLATE MODULE` and `ADD BLACK BOX`, the code for the modules, or a dummy module with port declarations, are required for the commands to work. If a module does not exist, you can use `SET UNDEFINED CELL -black_box` to instruct Conformal to treat any missing references as blackboxes. Like `ADD NOTRANSLATE MODULE`, this command must be run before running the `READ DESIGN` and `READ`

`LIBRARY` commands. Conformal, when constructing the blackboxes, will attempt to automatically determine the port directions of the blackboxes. After completion, you can use the `WRITE DESIGN` command to write out the design to check if Conformal correctly identify all the port directions of the blackboxes. This is important for comparison because blackbox input pins are considered as compare points while blackbox output pins are considered fanins to the next logic cones.

- Automatic Blackboxing

  Conformal can also blackbox a module during `READ DESIGN` or `READ LIBRARY` if it encounters non-synthesizable code or if it encounters synthesis directives, such as `translate_off` or `synthesis_off`.

Regardless of how blackboxes are created, the number of the blackboxes must match between the Golden and Revised designs for Conformal to perform the comparison correctly. Running the `REPORT BLACK BOX` command shows whether the blackboxes are paired up properly. The following shows an example of a set of balanced blackboxes:

```
SETUP> report black box
SYSTEM: (G R) ram8x1024
SYSTEM: (G R) rom8x256
```

If the blackboxes are not balanced, you might see the following:

```
SETUP> report black box
SYSTEM: (G R) ram8x1024
SYSTEM: (G)  rom_wrapper
SYSTEM: (R)  rom8x256
```

In this case, you can check to see how the blackboxes are constructed by running the `REPORT BLACK BOX` command with the `-detail` option.

in addition to the blackbox modules matching up, the blackbox input and output pins must also match up for the comparison to work properly. You can run the `REPORT MAPPED POINTS` command to check if the blackbox pin names got changed. For example:

```
report mapped points bbox_u1 -input -output
```

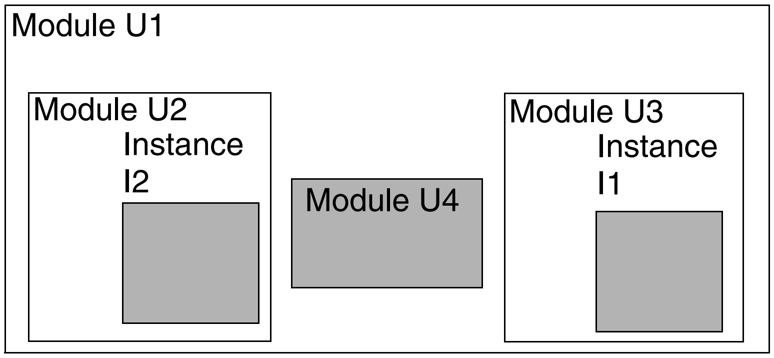
By default, blackboxes are mapped by their module names. To map by instance names instead, use the `SET MAPPING METHOD -nobbox_name_match` command.

If blackbox pins are not paired up correctly because pin names got changed, you can use the `ADD RENAMING RULE` command to rename those pins. For example:

```
add renaming rule ... -pin -bbox ...
```

Adding a Blackbox Instance or Module

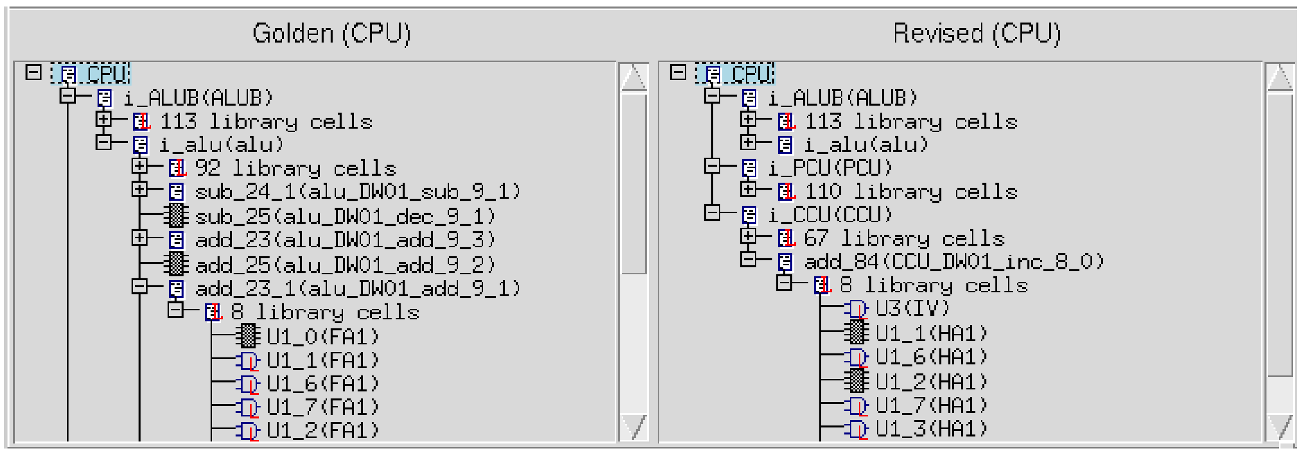
To treat any module or instance as a blackbox, use the ADD BLACK BOX command.



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

Alternatively, you can use the Hierarchical Browser in the main window.

The *Black Box* menu option adds or deletes instances or modules as blackboxes in the Hierarchical Browser window display. The *Black Box* icon appears or disappears, accordingly.



Golden (CPU)

```
CPU
  i_ALUB(ALUB)
    113 library cells
    i_alu(alu)
      92 library cells
      sub_24_1(alu_DW01_sub_9_1)
      sub_25(alu_DW01_dec_9_1)
      add_23(alu_DW01_add_9_3)
      add_25(alu_DW01_add_9_2)
      add_23_1(alu_DW01_add_9_1)
        8 library cells
          U1_0(FA1)
          U1_1(FA1)
          U1_6(FA1)
          U1_7(FA1)
          U1_2(FA1)
```

Revised (CPU)

```
CPU
  i_ALUB(ALUB)
    113 library cells
    i_alu(alu)
  i_PCU(PCU)
    110 library cells
  i_CCU(CCU)
    67 library cells
    add_84(CCU_DW01_inc_8_0)
      8 library cells
        U3(IV)
        U1_1(HA1)
        U1_6(HA1)
        U1_2(HA1)
        U1_7(HA1)
        U1_3(HA1)
```

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

   A blackbox symbol appears next to the instance or module name.

Deleting a Blackbox Instance or Module

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

1. Click a blackbox instance or module to select it.
2. Right-click and choose *Delete Black Box* and *Instance* or *Module* from the pop-up menu.

   The blackbox icon will disappear.

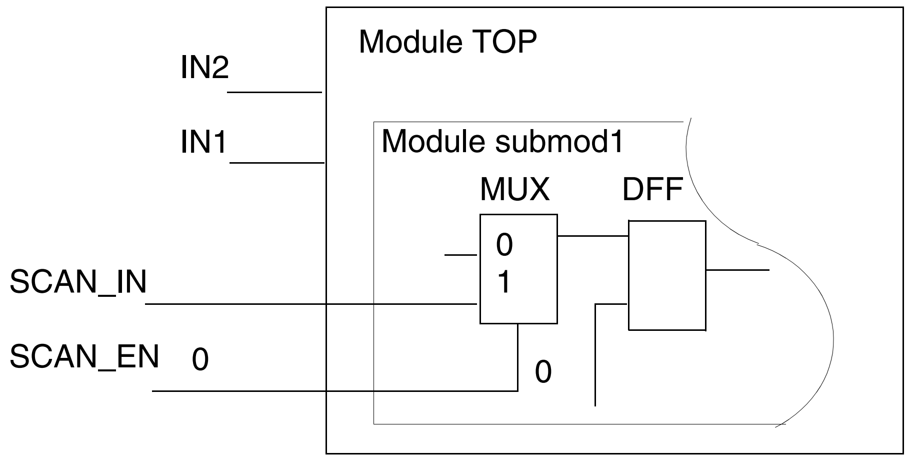
Net Constraints

To add one-hot or one-cold constraints to specified net paths, use the `ADD NET CONSTRAINTS` command. You can delete either the Golden or Revised design net constraints that are added with this command using the `DELETE NET CONSTRAINTS` command.

Use the `REPORT NET CONSTRAINTS` command to display a list of all added net constraints

Pin Constraints

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



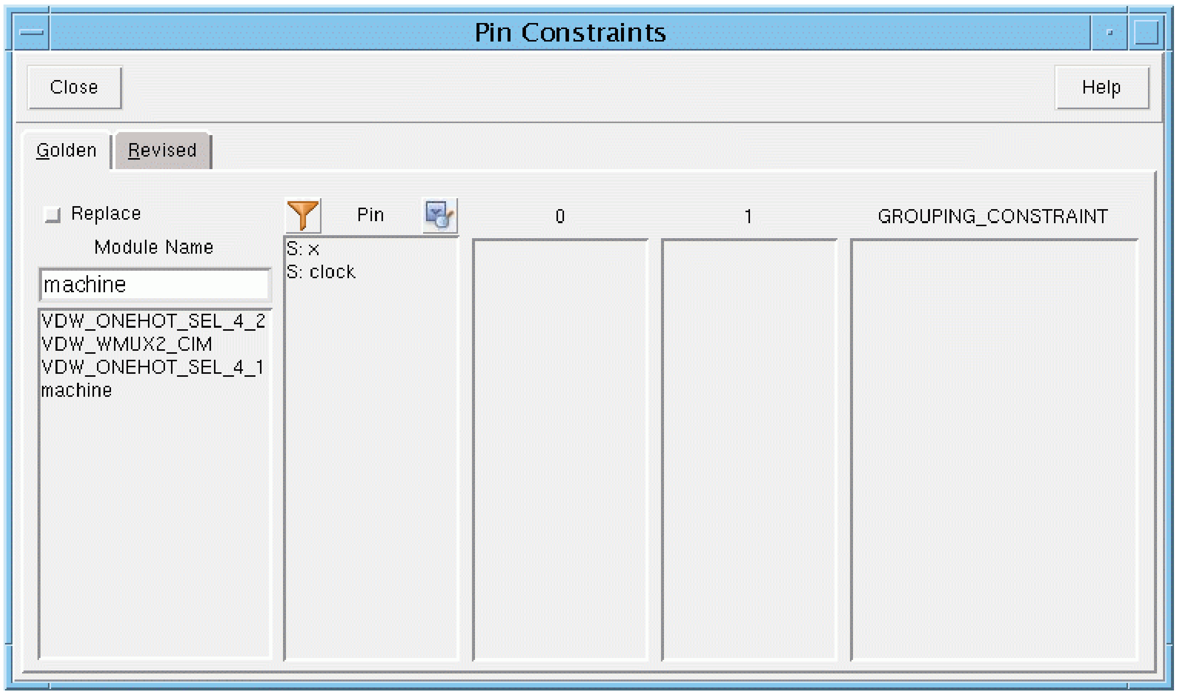
```
SETUP> add pin constraints 0 SCAN_EN -revised
```

To delete pin constraints to primary input pins, use the `DELETE PIN CONSTRAINTS` `C`ommand.

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

➤ Choose *Setup – Pin Constraints*.

The Pin Constraints window includes two tabs: Golden and Revised. Click on the Golden or Revised tab to switch between the two lists.



machine

VDW_ONEHOT_SEL_4_2
VDW_WMUX2_CIM
VDW_ONEHOT_SEL_4_1
machine

S: x
S: clock

For each list there are four columns with the headings: *Pin*, *0*, *1*, and *GROUPING_CONSTRAINT*. The primary input list is shown in the Pin column. Each primary input is either a system class primary input (*S:* `name`) or a user-defined class primary input (*U:* `name`).

Selecting Primary Inputs

In the following procedures, you are asked to select primary inputs. Use any of the following procedures to select primary inputs:

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

The following procedures explain how to add and delete pin constraints.

Adding a Pin Constraint to a Primary Input

Use the following procedure to add a constraint to a single primary input using the Pin Constraints form:

1. In the *Pin* column, click a primary input to select it.
2. Right-click and choose the *Constraint 0* or *Constraint 1* constraint from the pop-up menu.

   The selected primary input appears in the appropriate column.

Adding a Constraint to a Group of Primary Inputs

Use the following procedure to add a constraint to a group of primary inputs using the Pin Constraints form:

1. In the *Pin* column, select multiple pins with one of the methods described in "Selecting Primary Inputs" on page 120.
2. Right-click to open the pop-up menu and choose a constraint.

Deleting Pin Constraints

Use the following procedure to delete one or all constraints using the Pin Constraints form:

1. Click a primary input in the *0*, *1*, or *GROUPING_CONSTRAINT* column to select it.
2. Right-click and choose one of the following from the pop-up menu:

   To delete a constraint from the selected pin, choose *Delete Pin Constraint*. Conformal removes the pin constraint. And in the case of *GROUPING_CONSTRAINT*, Conformal deletes the entire group.

   To delete all constraints, choose *Delete All Pin Constraints*. Conformal deletes all constraints from all columns.

Sorting Pin Lists

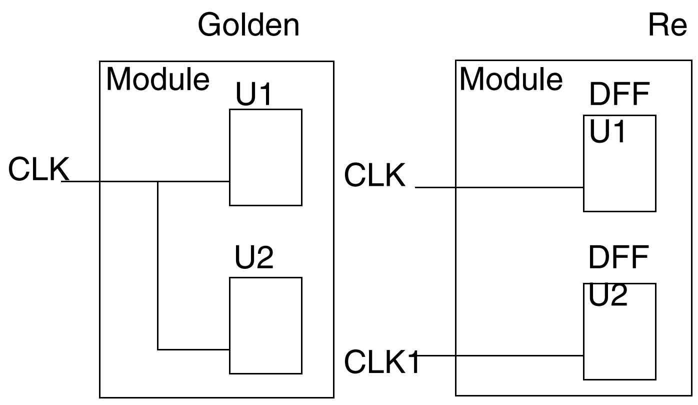
You can alphabetically sort the primary input lists.

1. Right-click in the *Pin*, *0*, or *1* column.
2. Choose *Sort* from the pop-up menu.

Pin Equivalences

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

Note: If you use the `-both` option, every primary input pin you list must exist in *both* designs (Golden and Revised). If they do not, Conformal returns an error message.

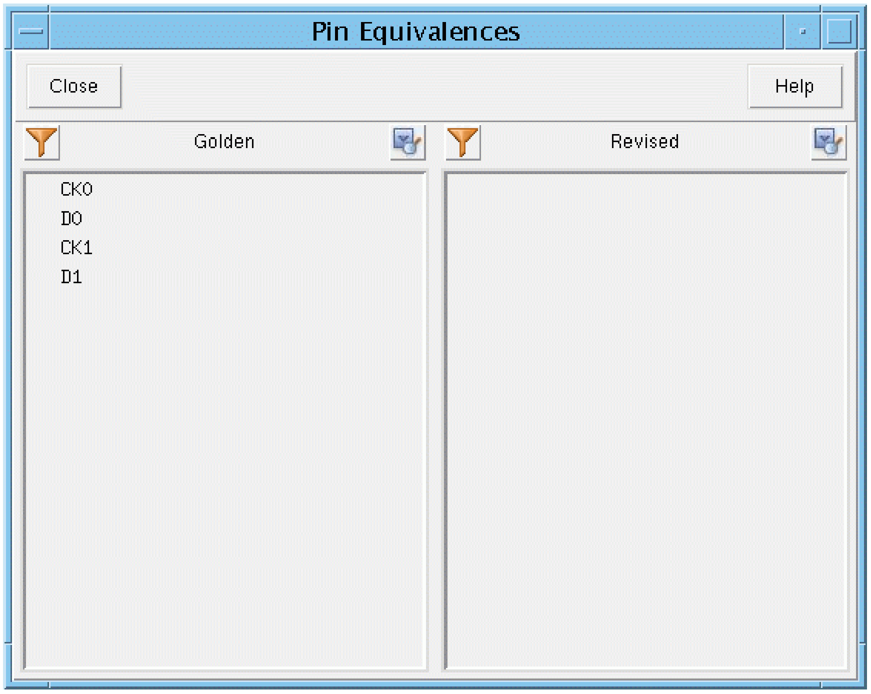


```
SETUP> add pin equivalences CLK CLK1 -revised
```

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.

➤ Choose *Setup – Pin Equivalences*.



The primary input lists for each of the Golden and the Revised designs are displayed in their respective columns. Conformal displays added pin equivalences below the target primary input with a connecting line. Inverted pin equivalences are denoted with (-) following the primary input name.

Adding a Pin Equivalence

1. Click a primary input in either in one of the columns 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 to show that it is the target primary input.
3. Click the second primary input (in the same column) that must be equivalent to the target primary input.
4. Right-click and choose *Add Pin Equivalence* or *Add Invert Pin Equivalence* from the pop-up menu.

Deleting Pin Equivalences

Use the following procedure to delete one or all pin equivalences.

1. Choose *Setup – Pin Equivalences*.

   The Pin Equivalences window appears.

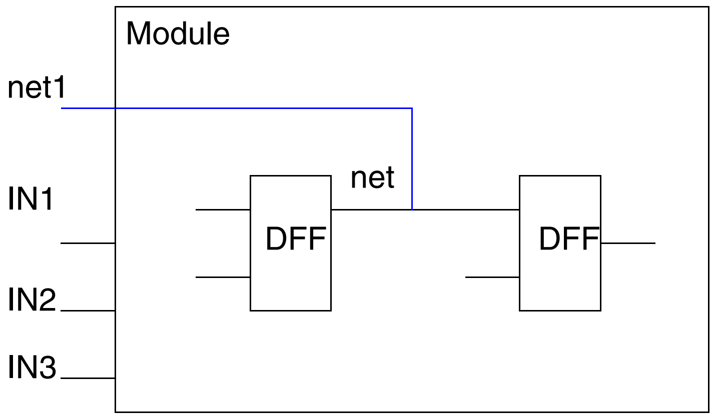
2. Click an equivalent primary input under one of the columns to select it.
3. Right-click and choose *Delete Pin Equivalence* or *Delete All Pin Equivalences* from the pop-up menu:

Sorting the Primary Input Lists

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

Primary Inputs

To add additional primary inputs to corresponding nets, use the ADD PRIMARY INPUT command.



```
SETUP> add primary input net1 -net -revised
```

To delete specified primary inputs that were originally added, use the DELETE PRIMARY INPUTS command. After you delete the primary input pins from either the Golden or Revised design, the associated nets become floating nets, unless there are other net drivers.

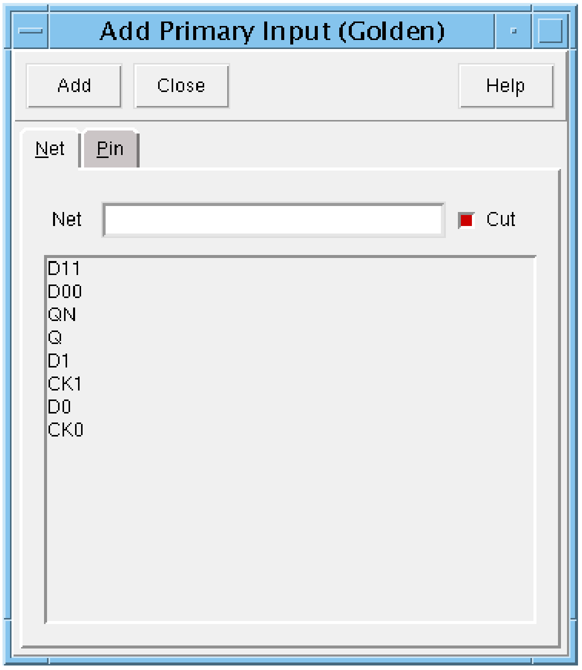
Alternatively, you can use the *Primary Input* menu from the main window to add and delete primary inputs.

Adding Primary Inputs

Choosing *Primary Input – Add* opens a dialog box instructing you to use the Hierarchical Browser window to add primary inputs. You must use the Hierarchical Browser window because primary inputs are added to hierarchical net names and, for a hierarchical design, it is not possible to list all hierarchical net names from the root module.

To open the Add Primary Input form to add primary inputs to net names from the Hierarchical Browser window, do the following:

1. Click an object to select it.
2. Right-click to open the pop-up menu and choose *Add Primary Input*.



D11
D00
QN
Q
D1
CK1
D0
CK0

3. Click on the *Net* or *Pin* tab to view a list of all of the net or pin names in the design.
4. Double-click a net (or pin) name to show the name in the *Net* (or *Pin*) field.
5. Click the *Cut* check box to specify if the other drivers of the net are to be disconnected so that the new added primary input is the only driver of the net.

   The default is Cut.
6. Click *Add*.

   Conformal adds the primary input to the net and appends `(added cut)` to the name in the Add Primary Input window. This notation means a primary input that is the only driver to the net. Otherwise, Conformal appends `(added nocut)` to the name to mean a primary input has been added.

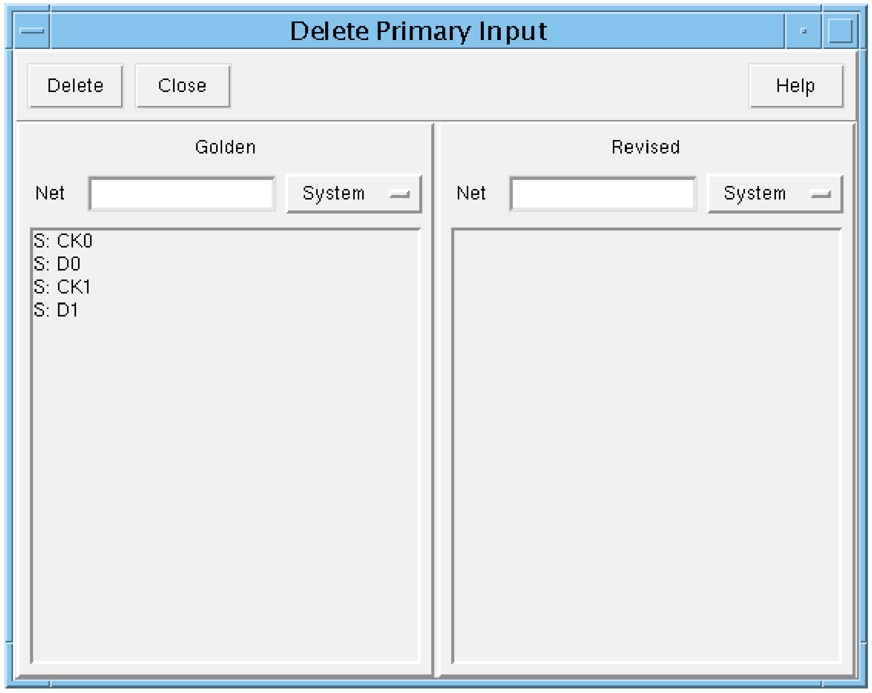
Sorting a Net or Pin List in the Add Primary Input Window

To alphabetically sort a *Net* or *Pin* list, right-click in the list display area to open the pop-up menu and choose *Sort*.

Deleting Primary Inputs

Use the Delete Primary Input form to delete user-defined primary inputs.

➤ Choose *Setup – Primary Input – Delete*.



S: CK0
S: D0
S: CK1
S: D1

Conformal displays the primary input list for each of the Golden and Revised designs in the left and right columns. Each primary input is either a system class primary input (S: name) or a user-defined class primary input (U: name).

Important

The class cyclic field adjacent to the *Net* field automatically displays whether the selected primary input is a system or user-defined primary input. You can only delete user-defined primary inputs

Deleting a Primary Input

To delete a single user-defined primary input, double-click a user-defined (U) primary input and click *Delete*.

Deleting All Primary Inputs

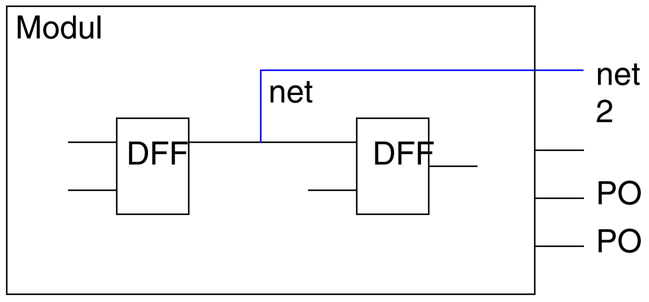
To delete all user-defined primary inputs, right-click in one of the columns to open the pop-up menu and choose *Delete All – User*.

Sorting Primary Input Lists

To alphabetically sort the primary input lists, right-click in either column and choose *Sort* from the pop-up menu.

Primary Outputs

To add additional primary outputs to corresponding nets, use the `ADD PRIMARY OUTPUT` command.



```
SETUP> add primary output net2 -revised
```

To delete specified primary outputs that were originally added, use the `DELETE PRIMARY OUTPUTS` command. When you delete the primary output pins from the Golden or Revised design, the nets become floating nets, unless there are other net drivers.

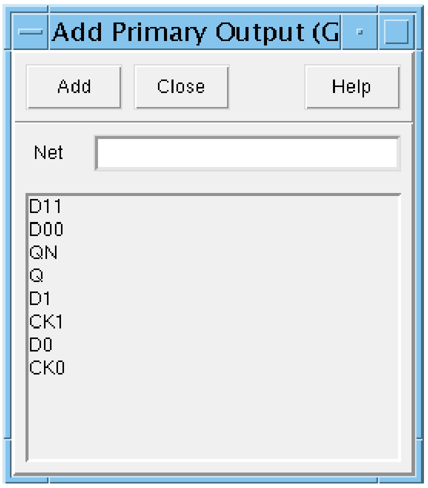
Alternatively, you can use the *Primary Output* menu from the main window to add and delete primary outputs.

Adding Primary Outputs

Choosing *Primary Output – Add* opens a dialog box instructing you to use the Hierarchical Browser window to add primary outputs. You must use the Hierarchical Browser window because outputs are added to hierarchical net names and, for a hierarchical design, it is not possible to list all hierarchical net names from the root module.

To open the Add Primary Output form to add primary inputs to net names from the Hierarchical Browser window, do the following:

1. Click an object to select it.
2. Right-click to open the pop-up menu and choose *Add Primary Output*.



D11
D00
QN
Q
D1
CK1
D0
CK0

3. Click on the *Net* or *Pin* tab to view a list of all of the net or pin names in the design.
4. Double-click a net (or pin) *name* to show the name in the *Net* (or *Pin*) field.
5. Click the *Add* button.

   `(added)` appears following the name.

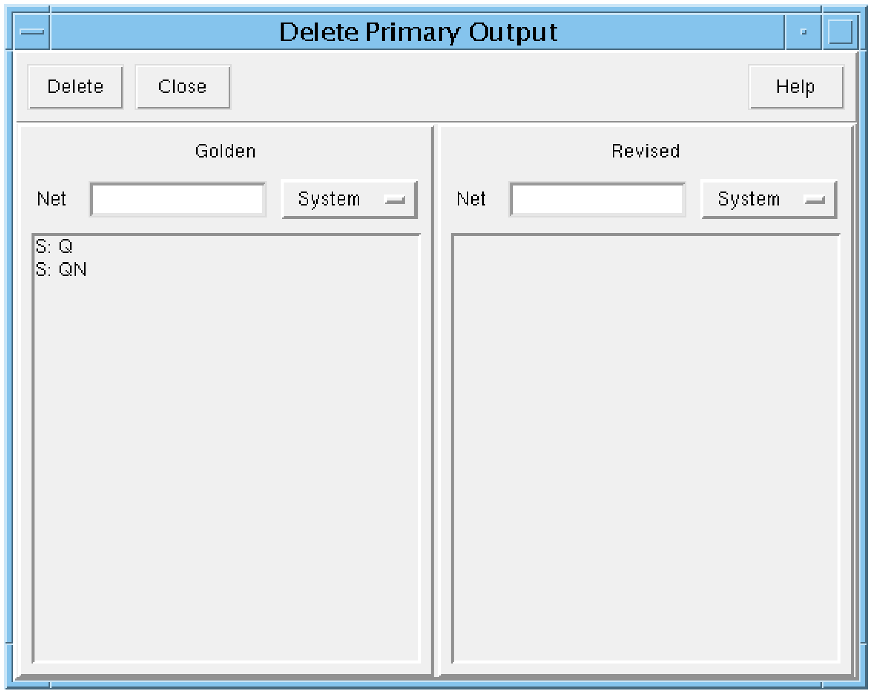
Sorting a Net or Pin List in the Add Primary Output Window

To alphabetically sort a *Net* or *Pin* list, right-click in the list display area to open the pop-up menu and choose *Sort*.

Deleting Primary Outputs

Use the Delete Primary Output form to delete user-defined (U) primary outputs.

➤ Choose *Setup – Primary Output – Delete*.



S: Q
S: QN

Conformal displays the primary output list for the Golden and Revised designs in the respective columns. Each primary output is either a system class primary output (S: name) or a user-defined class primary output (U: name).

Important

The class button adjacent to the *Net* field automatically displays whether the primary output is a system or user-defined primary output. You can only delete a user-defined primary output.

Deleting a Primary Output

To delete a single user-defined primary output, double-click a user-defined (U) primary input and click *Delete*.

Deleting All Primary Outputs

To delete all user-defined primary outputs, right-click in one of the columns to open the pop-up menu and choose *Delete All – User*.

Sorting Primary Output Lists

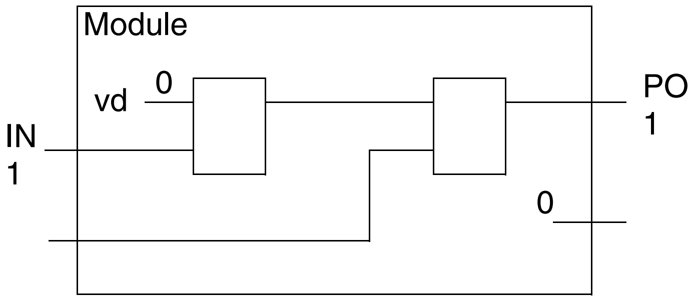
To alphabetically sort the primary output lists, right-click in either column and choose *Sort* from the pop-up menu.

Reporting Primary Outputs

Use the REPORT PRIMARY OUTPUTS command, or use the Primary Outputs Report form (*Report – Primary Outputs*) to display primary output pins from the Golden and Revised designs. You can open this form in two ways from the main window.

Tied Signals

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

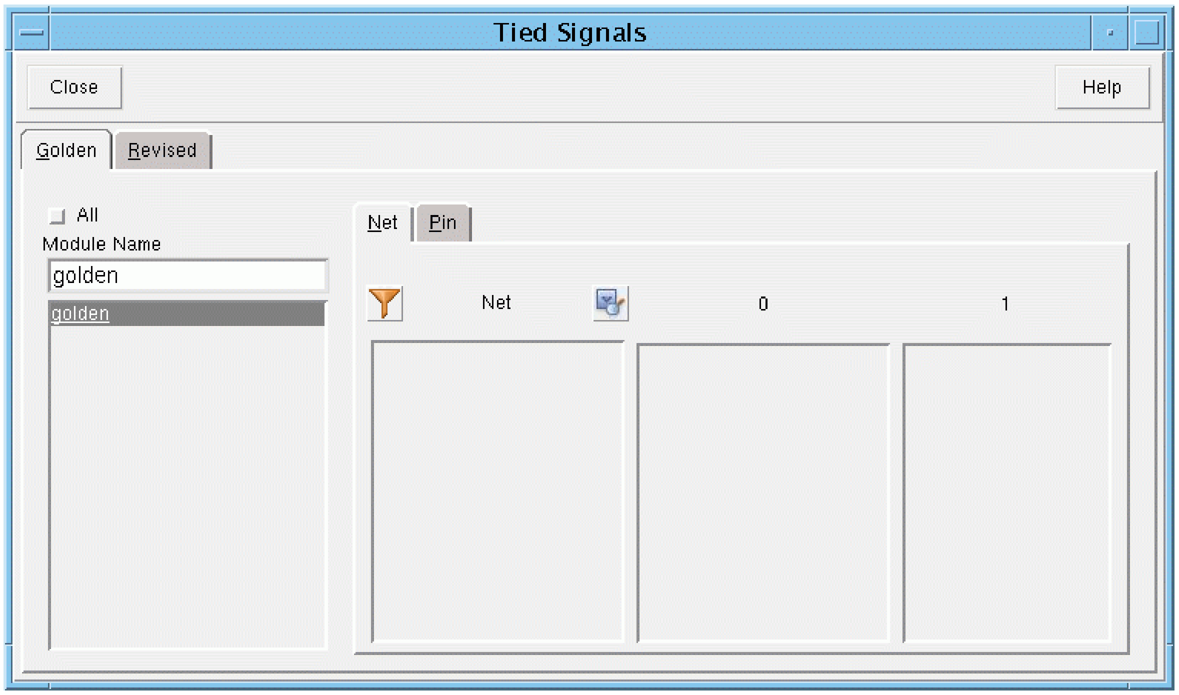


```
SETUP> add tied signals 0 vdd -pin -module U1 -revised
SETUP> add tied signals 0 SO -net -module U1 -revised
```

To delete specified tied signals from the Golden or Revised design, use the DELETE TIED SIGNALS command.

Alternatively, you can use the Tied Signals form from the main window to add and delete tied signals to floating nets and pins.

➤ Choose *Setup – Tied Signals*.



Click the *Golden* or *Revised* tab to switch between the two lists. For each list there is a *Module Name*, *Net* or *Pin*, *0*, and *1* column.

The names of all of the design's modules are displayed in the *Module Name* list box. Click the *Net* or *Pin* tab to display all of the selected module's floating nets or pins.

Adding a Tied Signal

Use the following procedure to add a tied signal to a net or pin.

1. Double-click a module name.
2. Click on the *Net* or *Pin* tab.
3. In the *Net* or *Pin* column, click a floating net or pin to select it.
4. (Optional) If the floating net or pin must be tied in all of the modules, click *All*.
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. Each tied signal belongs to one of two classes: system class tied signal (S: name) or user-defined class tied signal (U: name).

Deleting a Tied Signal

Use the following procedure to delete a tied signal from a net or pin.

1. Double-click a module name.
2. Click on the *Net* or *Pin* tab.
3. Click the net or pin name in the *0* or *1* column.
4. Right-click to open the pop-up menu and choose *Delete Tied Signal*.

Deleting All Tied Signals

Use the following procedure to delete all System or User tied signals from nets or pins.

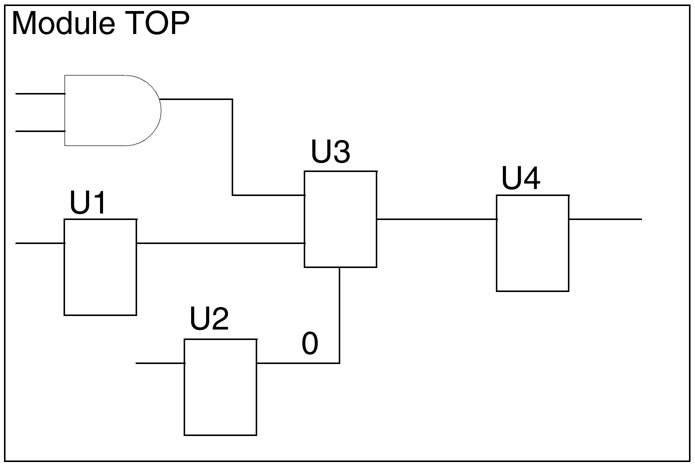
1. Click on the *Golden* or *Revised* tab.
2. Right-click in the *0* or *1* column to open the pop-up menu and choose *Delete All – User* or *Delete All – System*.

Sorting the Module, Net, and Pin Lists

To alphabetically sort the lists in all columns of this window, right-click in any column and choose *Sort* from the pop-up menu.

Instance Constraints

To constrain any internal DFF or DLAT output to Logic-0 or Logic-1, use the `ADD INSTANCE CONSTRAINTS` command.



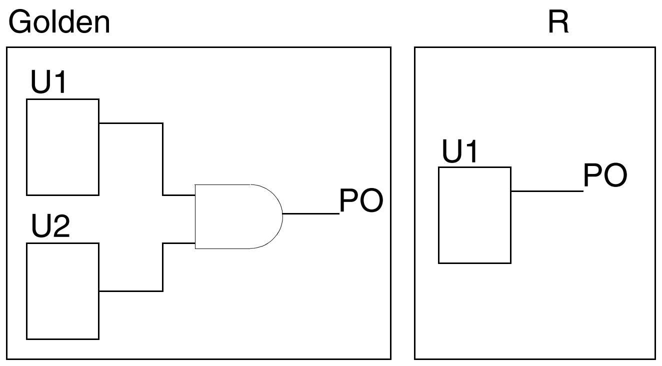
```
SETUP> add instance constraints 0 /TOP/U2
```

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.

Instance Equivalences

The `ADD INSTANCE EQUIVALENCES` command to specify internal equivalence or inverted equivalence between DFFs or D-Latches.

Note: This command affects comparisons when you use `add compared points -all`. In that situation, Conformal merges the instances specified with the `ADD INSTANCE EQUIVALENCES` command, and then it verifies them at the end of the comparison.

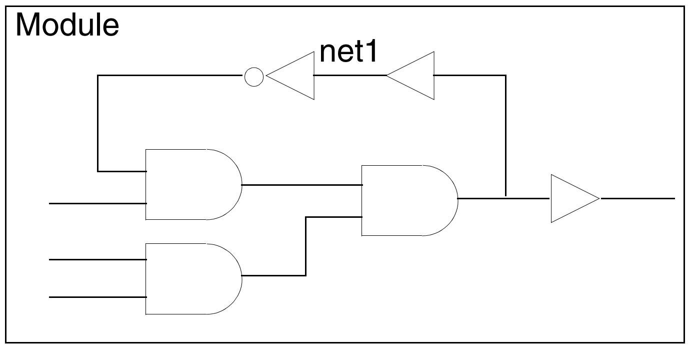


```
SETUP> add instance equivalence U1 U2 -Golden
```

Use the `DELETE INSTANCE EQUIVALENCES` command to delete instance equivalences that were added. Use the `REPORT INSTANCE EQUIVALENCES` command to display a list of all added instance equivalences.

Cut Points

To specify the cut points for breaking combinational feedback loops, use the `ADD CUT POINT` command. If you do not use this command and combinational feedback loops exist, Conformal automatically cuts the loops when you exit the Setup mode.



```
SETUP> add cut point /U1/net1 -revised
```

Use the `DELETE CUT POINT` command to delete cut points that were added.

Alternatively, you can use the *Cut Point* menu from the main window to add and delete cut points. *Cut Point* has the following submenus:

- Adding Cut Points on page 134
- Deleting Cut Points on page 136

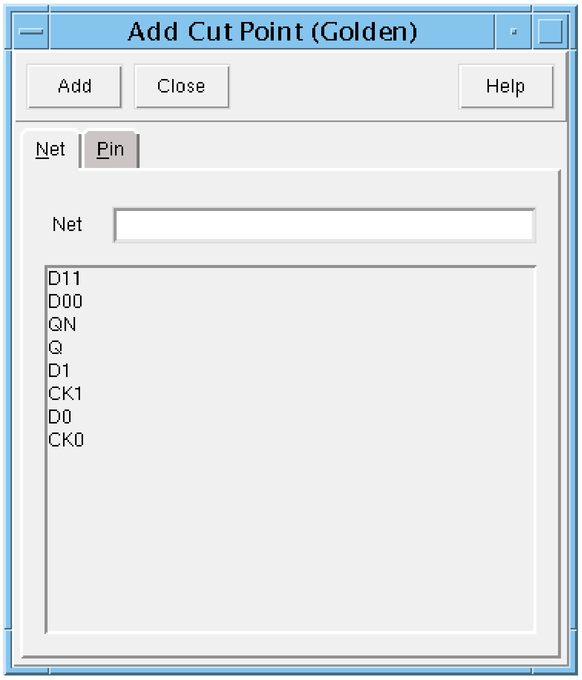
Primary Output is a menu item on the *Setup* drop-down men. This section explains the submenu choices for *Primary Output*.

Adding Cut Points

Choosing *Cut Point – Add* opens a dialog box instructing you to use the Hierarchical Browser window to add cut points. You must use the Hierarchical Browser window because cut points are added to hierarchical net names and, for a hierarchical design, it is not possible to list all hierarchical net names from the root module.

To open the Add Cut Point form to add user-defined cut points to net and pin names to break combinational feedback paths, do the following:

1. Click an object to select it.
2. Right-click to open the pop-up menu and choose *Add Cut Point*.



Net

D11
D00
QN
Q
D1
CK1
D0
CK0

3. Click on the *Net* or *Pin* tab to view a list of all of the net or pin names in the modules you selected from the Hierarchical Browser.
4. Double-click a net or pin name.

   The name appears in the *Net* or *Pin* field.
5. Click *Add*.

   Conformal affixes `added` to the net or pin name to show that a cut gate will be added to break the combinational feedback path.

Sorting a Net or Pin List in the Add Cut Point Window

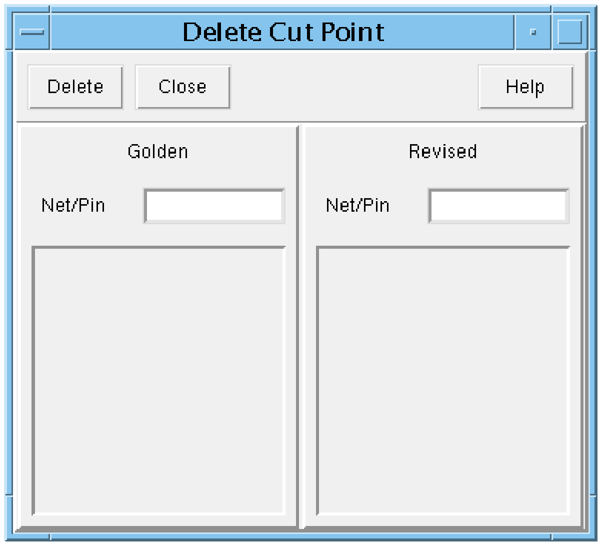
You can alphabetically sort a net or pin list by doing the following:

1. Click the *Net* or *Pin* tab.
2. Right-click in the list display area and choose *Sort* from the pop-up menu.

Deleting Cut Points

Use the Delete Cut Point form to delete user-defined cut points that were established to cut combinational feedback paths.

➤ Choose *Setup – Cut Point – Delete*.



Deleting a Cut Point

To delete a cut point, double-click a user-defined (U) primary input and click *Delete*.

Deleting All Cut Points

To delete all cut points, right-click in one of the columns to open the pop-up menu and choose *Delete All*.

Sorting Cut Point Lists

To alphabetically sort net names, right-click in either column and choose *Sort* from the pop-up menu.

Flattening Options

The SET FLATTEN MODEL command allows you to specify certain conditions for flattening the circuit. This section includes flattening information to help you tailor the command for your designs.

Alternatively, you can use the Flatten Model form to set some of the options described here. For a description of the form, see Flatten Model Form on page 144.

Specifying Key Point Mapping Options

To specify whether Conformal automatically maps key points when it exits the Setup system mode, use the `SET FLATTEN MODEL` command with the `-map` option.

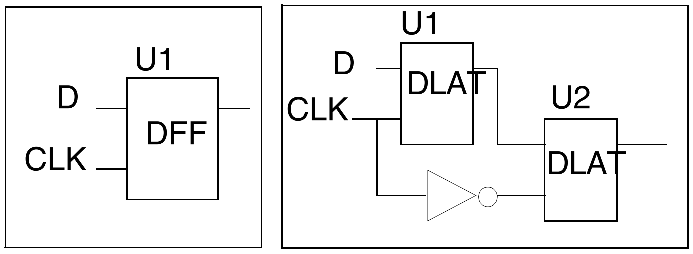
Note: You can also specify whether Conformal automatically maps key points during flattening with the `SET SYSTEM MODE` command, using the `-map` or `-nomap` option.

Retaining Gate Pin Information

To retain the gate pin information to report gate information at the design level (`REPORT GATE` command), use the `SET FLATTEN MODEL` command with the `-pin_keep` option. This option increases memory use.

Converting DLATs to DFFs

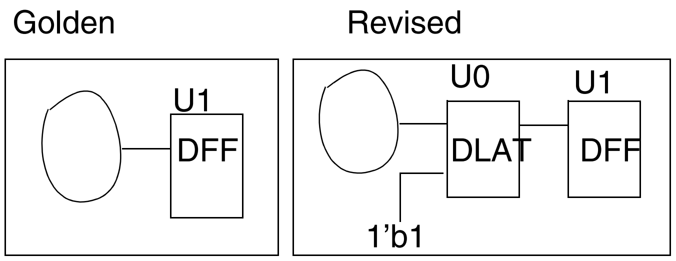
To convert two master/slave D-latches (DLATs) into a single D flip-flop (DFF) gate, use the `SET FLATTEN MODEL` command with the `-latch_fold` option.



```
SETUP> set flatten model -latch_fold
```

Converting DLATs to Buffers

To remodel DLATs (whose clock ports are always enabled) into buffers (transparent latches), use the `SET FLATTEN MODEL` command with the `-latch_transparent` option.

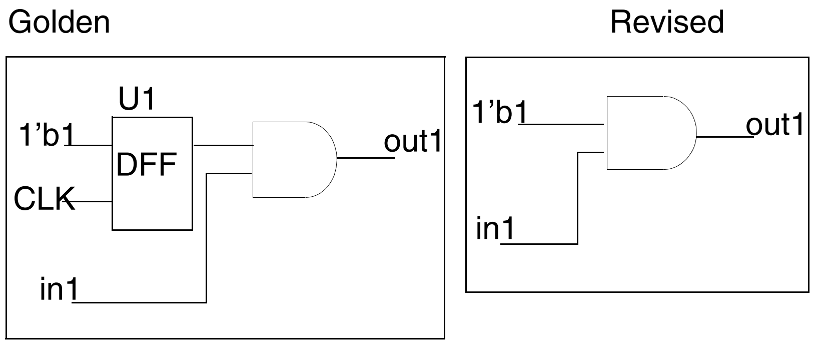


```
SETUP> set flatten model -latch_transparent
```

Converting DFF or DLAT to Zero or One Gate

Optimizing a Constant Flop to a Constant Value

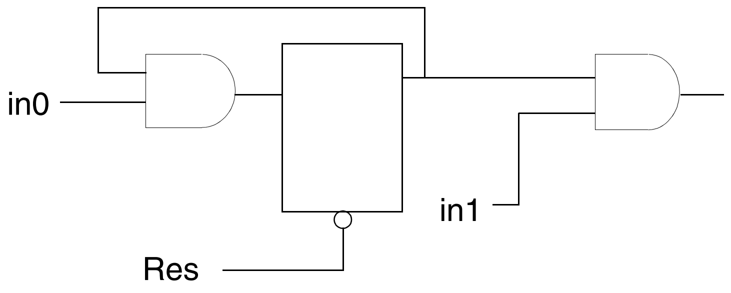
To convert a DFF or DLAT to a ZERO/ONE gate if the data port is set to 0/1, use the `SET FLATTEN MODEL` command with the `-seq_constant` option.



```
SETUP> set flatten model -seq_constant
```

Optimizing a Constant Feedback Flop to a Constant Value

To remodels registers that also have feedback to constants, use the SET FLATTEN MODEL command with the `-seq_constant_feedback` option. This is also enabled by default when you select the `-seq_constant` option. Once the flop is set at ZERO it will remain ZERO.



```
SETUP> set flatten model -seq_constant_feedback
```

Optimizing an Uninitialized Flop to a Constant Value

To optimize a flop to a constant value (either zero or one) when the flop is always in a *don't care* (`X`) state, use the SET FLATTEN MODEL command with the `-seq_constant_x_to` option.

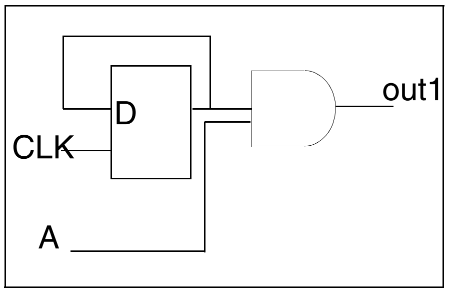
Note: Use this in conjunction with the `-seq_constant` option.

Tip

The `-seq_constant_x_to` switch can trigger the following modeling messages:

```
F18: Converted DFF/DLAT(s) to ZERO/ONE
F34: Convert X assignment(s) as don't care(s)
```

For example, the following figure illustrates a Golden and Revised version of a circuit when using this switch. The Golden circuit has a flop where its output `Q` is feedback to its input `D` and its asynchronous set and reset are disabled.



```
SETUP> set flatten model -seq_constant_x_to 0 -seq_constant
```

Gated-Clock Learning

When clock gating styles are causing problems during a comparison, you can use gated-clock learning to remodel the gated-clock logic during flattening.

There are two types of clock gating styles:

- Latch-based clock gating—*Latch-based gated clock modeling* uses a latch to hold the enable signal during the active phase of a clock.
- Latch-free clock gating—This type of modeling is used when enable signals are known to be stable during the active phase of a clock.

You can use the `-gated_clock` and `-gated_clock_latch_free` options of the `SET FLATTEN MODEL` to remodel these types of gating during flattening.

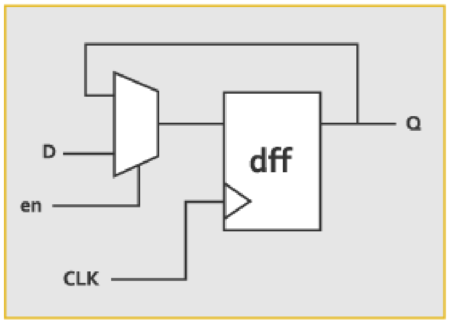
Remodeling Latch-Based Clock Gating

To remodel latch-based gated clocks, use:

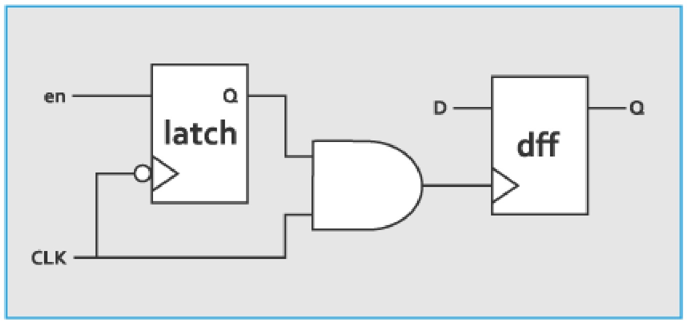
```
SETUP> set flatten model -gated_clock
```

During flattening, Conformal will model the latch-based clock gating structure into a MUX-DFF feedback type circuit.

For example:



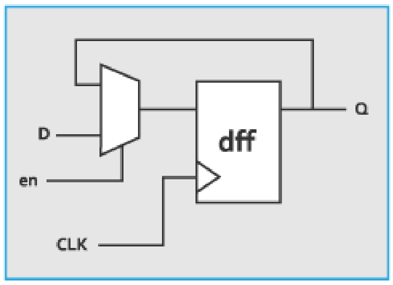
Golden



Revised

```
set flatten model -gated_clock
```

After enabling gated-clock learning, the latch-based gated clock is remodeled as follows:



Revised

Latch-Free Clock Gating

To remodel latch-free clock gating, you must specify that the enable signals for the clock gating circuits are stable with respect to the clocks. This can be done automatically using the `-gated_clock_latch_free` option of the `SET FLATTEN MODEL` command, or manually using the `ADD CLOCK` command.

Note: For the latch-free clock gating modeling to be valid, the enable signals must be stable. You can verify this using external circuits that are outside of the design, but that are also subject to LEC verification.

Automatically Remodeling Latch-Free Clock Gating

To transform latch-free clock gating into a MUX-DFF-feedback type circuit, use the `SET FLATTEN MODEL`'s `-gated_clock_latch_free` and `-gated_clock options` (they must be used together).

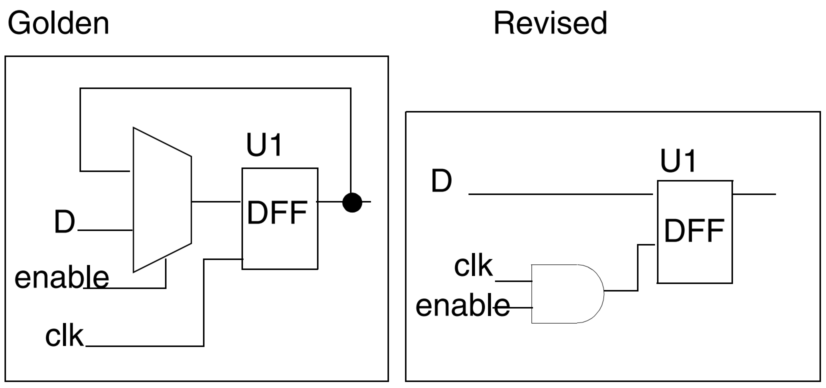
Manually Remodeling Latch-Free Clock Gating

You can also transform latch-free clock gating into a MUX-DFF-feedback type circuit by explicitly specifying the clocks. For example:

```
SETUP> add clock 0 clk -revised
SETUP> set flatten model -gated_clock
```

By doing this, the tool assumes that the enable signal is stable with respect to the clock specified in the `ADD CLOCK` command.

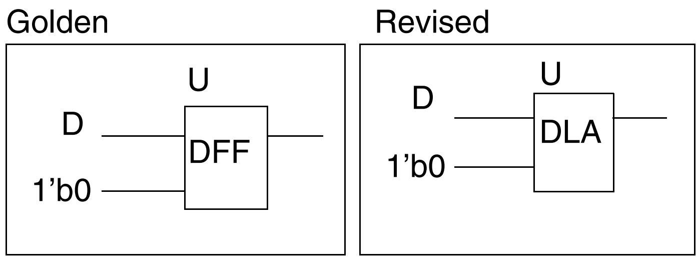
In this example, the `clk` value holds the clock signal (low/high) when inactive and allows for a gated clock transformation into a MUX-feedback type circuit.



```
SETUP> add clock 0 clk -golden
SETUP> add clock 0 clk -revised
```

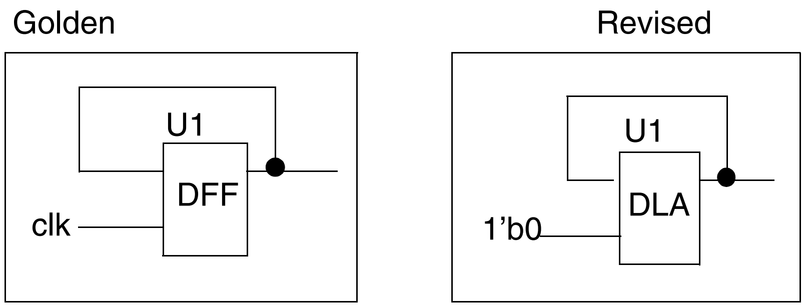
Converting DFFs to DLATs

To convert a DFF to a DLAT if the clock signal is zero, use the SET FLATTEN MODEL command with the `-DFF_TO_DLAT_ZERO` option.



```
SETUP> set flatten model -dff_to_dlat_zero
```

To convert a DFF to a DLAT when there is a direct feedback loop from the Q port of the DFF to its D port, use the SET_FLATTEN_MODEL command with the -DFF_TO_DLAT_FEEDBACK option.



```
SETUP> set flatten model -dff_to_dlat_feedback
```

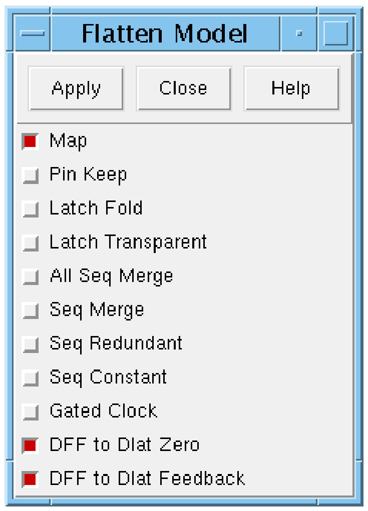
Using DLATs to Model Combinational Loops

To use a DLAT to model a combinational loop, you can use the SET_FLATTEN_MODEL command with the -loop_as_dlat option.

Flatten Model Form

Use the Flatten Model form from the main window to set global options for flattening and modeling, which occur when exiting Setup system mode.

➤ Choose *Setup – Flatten Model.*



- [✓] Map
- [ ] Pin Keep
- [ ] Latch Fold
- [ ] Latch Transparent
- [ ] All Seq Merge
- [ ] Seq Merge
- [ ] Seq Redundant
- [ ] Seq Constant
- [ ] Gated Clock
- [✓] DFF to Dlat Zero
- [✓] DFF to Dlat Feedback

Flatten Model Form Fields and Options

Map	Does automatic key point mapping. When this option is disabled, this will skip the automatic key point mapping when the system mode is changed from Setup to LEC.
Pin Keep	Keeps all gate pin information for gate reporting. Use this option when reporting gate information at the design level.
Latch Fold	Folds a master-slave latch into a D flip-flop.
Latch Transparent	Converts D-latches (DLATs) into buffers if the clock ports of the DLATs are always enabled.
All Seq Merge	Merges state elements that are functionally equivalent.
Seq Merge	Merges common groups of sequential elements into one sequential element in the clock cone of a DFF or DLATs.
Seq Redundant	Removes redundant fan-out gates from DFFs and DLATs.
Seq Constant	Propagates constant data through latches and registers.
Gated Clock	Remodels the gated-clock logic of the clock port of a DFF. If the clock pin cannot be automatically determined, use the `ADD CLOCK` command to define the clock pin.
DFF to Dlat Zero	Converts a DFF to a DLAT if the clock port is zero.
DFF to Dlat Feedback	Converts a DFF to a DLAT if the Q output feeds back to the D input.

Mapping Settings

When you move from the *Setup* system mode to the *LEC* system mode, Conformal automatically maps the key points with the name-first default mapping method. Before you exit the *Setup* mode, you can use the `SET MAPPING METHOD` command to change the default settings and specify the following:

- Mapping Method—If the Golden and Revised designs have the same names, change the mapping method to name-only. This option makes mapping more efficient and less time consuming.
- Mapping Phase—You can specify inverted phase mapping.

Note: When using the `SET MAPPING METHOD -phase` command, the Conformal software compares the set logic of the Golden design to the reset logic of the Revised design (and vice-versa) for inverted-equivalence.

- Case Sensitivity—The default is no case sensitivity. Conformal considers key point names case-sensitive, if you prefer.
- Unreachable Points—If you want Conformal to map unreachable points, use the `-unreach` option. If you do not want to report unreachable points, use the `-noreport_unreach` option.
- Name Effort—You can specify the amount of effort you want Conformal to apply to key point mapping. The system default level is `hi`, which eliminates the need for simple renaming rules.

  Note: This option applies to DFFs and DLATs.
- Blackboxes—You can specify mapping for blackboxes based on instance name matches (by default, Conformal maps blackboxes if both the module names *and* instance names match).

Mapping Methods

Conformal employs three name-based methods to map key points and one no-name method. If most of the key point names in the Golden and Revised designs are the same, choose a name-based mapping method. This method is useful for gate-to-gate comparisons when small changes have been made to the logic. Conversely, the no-name-mapping method is useful when Conformal must map designs with completely different names. By default, Conformal automatically maps key points with the name-first mapping method when it exits the Setup mode.

In addition to the name-first method, Conformal includes two other name-based methods: name-guide and name-only. Use the name-only and name-first methods when the Golden and Revised designs have the same names. (This approach speeds up mapping.) And create naming rules to help Conformal during mapping when corresponding key points have different names. (See “Renaming Rules” on page 147.) Any key points that Conformal does not map are classified as unmapped points.

Name-First Mapping

For the name-first mapping method, which is the default, the Conformal software maps key points in two steps.

1. When the names are the same in the Golden and Revised designs.

2. Attempts to map remaining key points with a mapping algorithm.

Any key points that remain unmapped after the second step are identified as unmapped points.

Name-Guide Mapping

For the name-guide mapping method, the Conformal software also maps key points in two steps.

1. Maps key points with a mapping algorithm.
2. Attempts to map remaining key points by matching names in the Golden and Revised designs.

Any key points that remain unmapped after the second step are identified as unmapped points.

Name-Only Mapping

When using the name-only mapping method, Conformal maps key points only if the names are the same in the Golden and Revised designs. Any key points that do not have the same name are identified as unmapped points.

No-Name Mapping

When using the no-name-mapping method, Conformal relies solely on the mapping algorithm to map key points. Any remaining key points are identified as unmapped points.

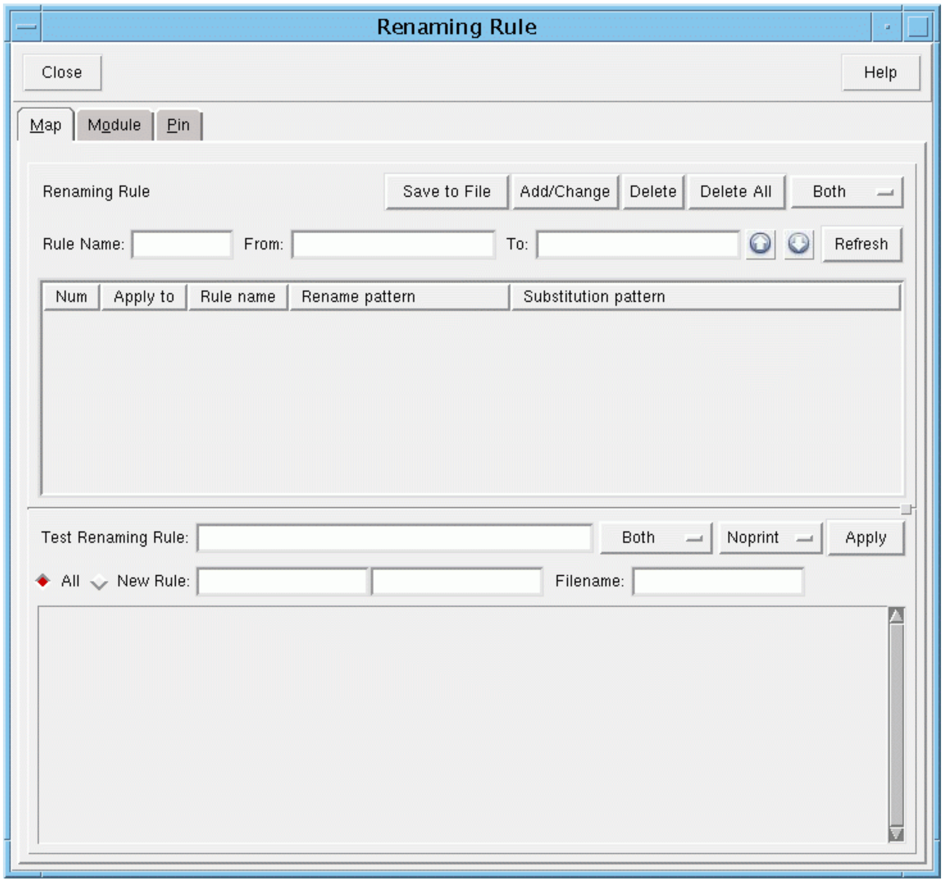
Renaming Rules

When the naming conventions in the Golden and Revised designs are not the same, augment the name-based mapping methods described above by introducing naming rules. These rules are a way to translate key point names. When you use the `ADD RENAMING RULE` command, you identify string patterns and define temporary substitute string patterns, thus enabling Conformal to automatically map additional key points when names are not the same.

The `ADD RENAMING RULE` command lets you include or exclude particular types of key points for renaming. 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 to guide mapping, help map modules for hierarchical comparisons, or rename pin names of blackboxes.

You can also use the `TEST RENAMING RULE` command (see Testing Renaming Rules on page 151) or the Renaming Rule form to test renaming rules for mapping performance based on name mapping.



The Renaming Rule window includes a *Renaming Rule* and *Test Renaming Rule* sections and three pages:

- *Map* for renaming rules that apply to key point mapping.
- *Module* for renaming rules that apply to module renaming when the library and design are read in
- *Pin* for renaming rules that apply to pin names of blackboxes.

Renaming rules are color-coded according to the specified design:

- Both: blue
- Golden: green
- Revised: orange

Renaming Rule Form Fields and Options

Most of the fields and options are the same for the *Map*, *Module*, and *Pin* pages, except where noted in the following descriptions:

Renaming Rule	The options on this row are as follows: ■ *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. ■ *Delete All* - Deletes all renaming rules. *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. ■ *Both* (the default) - Applies the renaming rule to both the Golden and Revised designs. You can use this pull-down menu to select *Golden* to apply the renaming rule to the Golden designs, or *Revised* to apply the renaming rule to the Revised designs.
Black Box (for the *Pin* page)	Applies a renaming rule to a specified blackbox module.
Rule Name	Specifies a unique rule name.
From	Specifies the renaming pattern.
To	Specifies the substitution pattern.

Up Arrow/Down Arrow	Changes the sequence of the renaming rules in the display. Click a rule in the rule list, and click the up- or down-arrow.
Refresh	Refreshes the rule list.

After adding renaming rules, you can use bottom part of the form to check their effectiveness.

Test Renaming Rule	Specifies the renaming rule to be checked. ■ *Both* (the default) - Applies the renaming rule test to both the Golden and Revised designs. You can use this pull-down menu to select *Golden* to apply the test to the Golden designs, or *Revised* to apply the test to the Revised designs. ■ *Noprint* (the default) - Does not display mapping pairs, groups, or single key points. You can use this pull-up menu to select *Single*, *Pair*, or *Group*. ■ Apply - Updates the new or changed renaming rule test entries.
All	Specifies all renaming rules will be tested.
New Rule	Specifies a specific string. Type the first pattern the Conformal software will search for, and the second pattern it will substitute.
Filename	Specifies the name of the file where the Conformal software will write the results.

See the `ADD RENAMING RULE` command for the following information:

- The structure of renaming rules
- Keywords that require escape characters
- Sample expression pattern matching and substitution strings

Specifying Sets of Naming Rules

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

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

Then ran the following command for Verilog as rule 2:

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

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

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 for mapping, module, and pin renaming. The list displays a rule number along with a renaming rule. If you do not enter options, the Conformal software displays all renaming rules.

Testing Renaming Rules

Use the `TEST RENAMING RULE` command to test specific or all renaming rules before you add them. Other uses for this command follow:

- Use this command to apply a new or existing rule without committing to it.
- Apply a renaming rule to a sample or the entire design to see how Conformal matches the key points.
- Get a summary or complete listing of the design's key point pairs, groups, and singles.

Analyze Renaming Rules

To enable renaming rule analysis, use

```
set analyze option -analyze_renaming_rule
```

This feature is effective only when balanced modeling is enabled, for example, with the setting `SET FLATTEN MODEL -SEQ_CONST`.

During the flattening process, LEC reports the renaming rule analysis. For example:

```
// Command: set system mode lec
// Renaming Rule Analysis (DFF/DLATs)
```

```
============================================================
Rule       Matches(G) Matches(R)     Mapped    %
------------------------------------------------------------
                                          0    0
r1              20         0             20   80
============================================================
// Balanced modeling (auto) mapped 21 out of 25 DFF/DLATs
```

Understanding this example:

- The analysis is currently applied to only DFFs and DLATs.
- The first line shows the number of mapped DFF/DLATs and the percentage that do not have renaming rules.
- After that, each renaming rule has a line showing the number of matches in Golden and Revised, and the number of mapped DFF/DLATs and the percentage after this renaming rule is applied.
- The effectiveness of each renaming rule is shown by the number of matches, and the name mapping results.
    - An unexpected low number of matches may indicate a bad matching expression
    - A lack of improvement in the mapping count may indicate a bad substitution expression

Creating Renaming Rules to Map Array Pins

Use the `ADD RENAMING RULE` command's `-PIN_MULTIDIM_TO_1DIM` option to automatically create renaming rules to map multi-dimensional array pins to one-dimensional array pins, where the rules are determined by examining all Golden module ports independently from the Revised design.

For example, in the following command, `y2[1:0][2:0]` in module `test2` is renamed `y2[5:0]`:

```
add renaming rule -pin_multidim_to_1dim
// Rule created for (test2) y2[1:0][2:0]
// Rule created for (test1) y1[1:0][1:0]
// Rule created for (top) y2[1:0][2:0]
// Rule created for (top) y1[1:0][1:0]
// Rule created for (top) ym[2:3][2:0][1:0]
// 5 rules created. Rules for top module must be manually validated.
```

You can use the `REPORT RENAMING RULE` command to view the added rules.

7

Using the LEC Mode

Moving to LEC Mode

After setup is complete, move to the LEC mode. When you move from the Setup system mode to the LEC system mode, Conformal automatically maps the key points with the name-first default mapping method. Review the sections related to “Mapping Settings” on page 145 and “Renaming Rules” on page 147 before leaving the Setup mode.

To switch system modes, use the `SET SYSTEM MODE` command. When you exit the *Setup* system mode, Conformal automatically flattens the designs and maps key points between the Golden and Revised designs. However, when you use the `-nomap` option, you prevent Conformal from automatically mapping key points on entering the *LEC* mode.

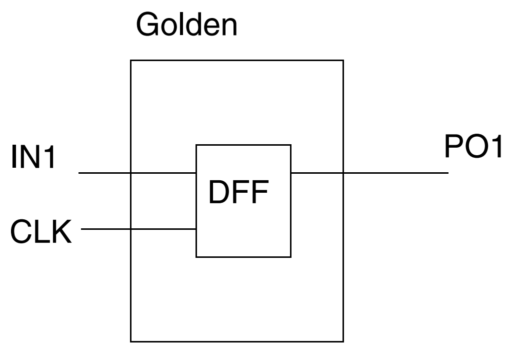
In the LEC mode, you can view warning messages and compare and debug a design. This chapter describes the commands that allow you to choose verification options and run the verification.

Mapping Modifications

On entering the LEC system mode, you might find you need to make modifications to improve mapping results. If the results are not satisfactory (that is, Conformal did not map a high percentage of key points) you can change the mapping method (see Mapping Settings on page 145) and introduce renaming rules (see Renaming Rules on page 147). The following information guides you as you remap key points, manually add mapped points, and save mapping results for future sessions.

Altering Key Point Mapping

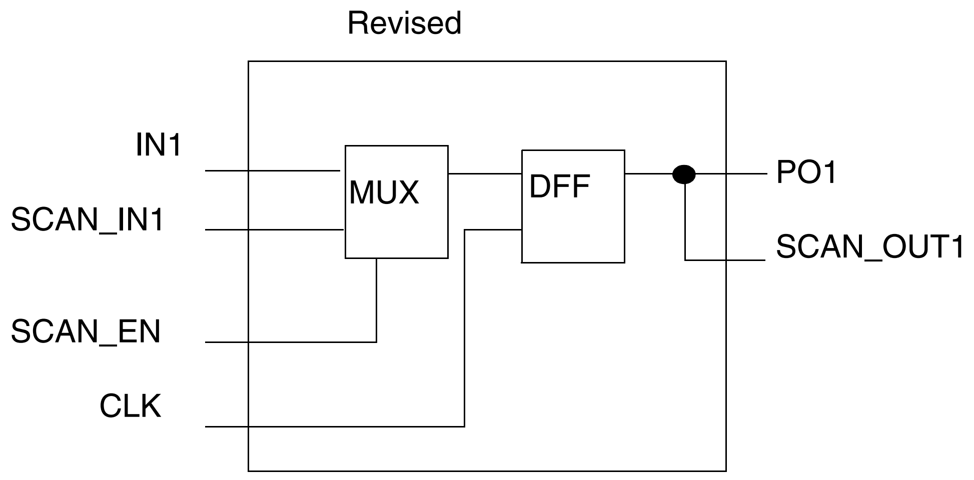
After altering mapping methods and renaming rules, use the MAP KEY POINTS command to re-map the key points with your modifications.



Mapped Points

Golden	Revised
IN1	IN1
CLK	CLK
PO1	PO1

Unmapped Points
Revised
SCAN_IN1 Extra
SCAN_EN Extra



```
LEC> map key points
```

Adding Mapped Points

When Conformal completes automatic mapping, you can add mapped points that were not automatically identified with the ADD MAPPED POINTS 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.

Inverting Mapping Phase

Use the `INVERT MAPPED POINTS` command to apply inverted mapping phase for specified points.

Saving Mapping Results

You can write the mapping results to a file and read them back in a future run to speed up mapping, using the `WRITE MAPPED POINTS` and `READ MAPPED POINTS` commands.

Note: By default, Conformal automatically maps key points during the transition from *Setup* to *LEC* mode. And if the key points are already mapped, Conformal ignores any mapped point information in the file. Thus, to prevent Conformal from automatically mapping key points during the transition from *Setup* to *LEC* mode and enable Conformal to read in mapped point information completely from the file, do one of the following:

```
...
set system mode -nomap
read map point <map_file>
...
```

or

```
...
set flatten model -nomap
set system mode lec
read map point <map_file>
...
```

Compare Options

Conformal can compare all mapped points or a sub-set of mapped points. The comparison tells whether key points are equivalent or non-equivalent. Compared points are:

- Primary Outputs
- D Flip-Flops (DFFs)
- D-Latches (DLATs)
- RAMs
- Blackboxes
- Cut Gates that were identified as mapped points

Adding Compared Points

To specify which mapped points Conformal compares, use the `ADD COMPARED POINTS` command.

Setting the Compare Effort

To set the compare effort (the amount of effort the algorithm expends to make a comparison for a compared point) use the `SET COMPARE EFFORT` command.

Setting a CPU Limit

To limit the amount of time Conformal spends comparing key points, use the `SET CPU LIMIT` command. Conformal stops the comparison and exits the session when it reaches the specified limit.

Reporting Compare Time

To report the CPU time consumed during a comparison, use the `REPORT COMPARE TIME` command.

You must enable this feature before starting a comparison; otherwise, Conformal does not record any information. For example:

```
command> compare
command> report compare time -enable
command> compare
command> report compare time
```

In this example, Conformal records the CPU time for the second comparison only.

Note: Conformal does not record compare time for trivial cones.

Comparison

To start the comparison, use the `COMPARE` command.

Reporting Compare Data

After Conformal completes the comparison, use the `REPORT COMPARE DATA` command to view a list of all compared points and their status (equivalent or non-equivalent).

Reporting Statistics

The `REPORT STATISTICS` command summarizes the mapping and compare statistics.

Reporting CPU Use

The `USAGE` command displays the total CPU time and the total memory used during the current Conformal session.

Report Verification

Conformal can report a table of all violated checklist items for the following categories:

1. Non-standard modeling options used
2. Incomplete verification
3. Design modifications
4. Conformal recommended extended checks
5. Design ambiguity

Use the `REPORT VERIFICATION` command to run the report. You can prints out each category and the count of violations, or print out all items for each category where the violated items are marked with an asterisk (`*`).

Running Additional 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 LEC mode.

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

- Black Boxes Report on page 160
- Cut Points Report on page 160
- Design Data Report on page 161
- Environment Report on page 161
- Floating Signals Report on page 161
- Instance Constraints Report on page 162
- Instance Equivalences Report on page 162
- Messages Report on page 162
- Modules Report on page 163
- Notranslate Modules Report on page 164
- Pin Constraints Report on page 164
- Pin Equivalences Report on page 165
- Primary Inputs Report on page 165
- Primary Outputs Report on page 165
- Renaming Rules Report on page 166
- Search Paths Report on page 166
- Tied Signals Report on page 166
- Mapped Points Report on page 167
- Unmapped Points Report on page 167
- Compared Points Report on page 167
- Compare Data Report on page 167

- Statistics Report on page 167

Black Boxes Report

Use the `REPORT BLACK BOX` command or the Black Box Report form (*Report – Black Box*) to display black boxes from the Golden and Revised designs.

Black Box Report Form Fields and Options

Class	Displays the specified class of blackboxes. Use the pull-down menu to select *Full* for blackboxes from both the User and System classes (the default), *User* for blackboxes previously added with the Conformal software, or *System* for blackboxes included in the original design.
Type	Select *Module* to report only the blackbox modules (the default), or *Instance* to report only the blackbox instances.
Hier	Displays hierarchical compare blackboxes
Hidden	Displays blackboxes within other blackboxes

Cut Points Report

Use the `REPORT CUT POINTS` command, or use the Cut Point Report form (*Report – Cut Point*) to display cut points from the Golden and Revised designs.

Cut Point Report Form Fields and Options

Design	Specifies the design to display the cut points. Choose *Revised*, *Golden*, or *Both* (the default).

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

Golden Module Name	Specifies the module name for the Golden design.
Revised Module Name	Specifies the module name for the Revised design.
Extra	Reports the extra input, output, or I/O pins for pair-able modules between the Golden and Revised designs. Choose *Input* for input pins, *Output* for output pins, or *Inout* for inout pins.
Key Point	Reports the total one-to-one mapped state points. **Note:** If you use this with *Verbose*, Conformal reports all one-to-one mapped state points.
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 designs 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 Golden and Revised designs 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.
Design	Specifies the design to display the floating nets. Choose *Revised*, *Golden*, or *Both* (the default).
Signal	*Net* displays only floating nets, *Pin* displays only floating pins, and *Full* displays both floating nets and floating pins.
All	Display all floating signals in all modules

Instance Constraints Report

Use the `REPORT INSTANCE CONSTRAINTS` command or the Instance Constraints Report form (*Report – Instance Constraints*) to display constraints placed on instances in the Golden and Revised designs.

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

Instance Equivalences Report

Use the `REPORT INSTANCE EQUIVALENCES` command or the Instance Equivalences Report form (*Report – Instance Equivalences*) to display the equivalences placed on instances in the Golden and Revised designs.

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

Messages Report

When you exit Setup system mode, there can be summary warning messages related to modeling the Golden or Revised designs, mapping key points, or comparing. Use the `REPORT MESSAGES` command or the Messages Report form (*Report – Messages*) to report a detailed, or verbose, listing of the warning messages.

Messages Report Form Fields and Options

Type	*Modeling* (the default) displays only warning messages from the processing and modeling of the Golden and Revised designs. *Mapping* displays warning messages only from the automatic key point mapping process. *Compare* displays warning messages only from the comparison process.
Design	Specifies the design to display the messages. Choose *Revised*, *Golden*, or *Both* (the default).
Summary	Displays only a summary message for common warning messages. By default, the Conformal software displays all warning messages.
Rule Name	Specifies the named rule that should only be displayed.

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

Golden Module Name	Specifies the module name for the Golden design.
Revised Module Name	Specifies the module name for the Revised 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.

Design	Specifies the design to display the modules. Choose *Revised*, *Golden*, or *Both* (the default).
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 Golden and Revised designs.

Pin Constraints Report Form Fields and Options

Golden Module Name	Specifies the module name for the Golden design.
Revised Module Name	Specifies the module name for the Revised design.
Design	Specifies the design to display the pin constraints. Choose *Revised*, *Golden*, or *Both* (the default).
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

Golden Module Name	Specifies the module name for the Golden design.
Revised Module Name	Specifies the module name for the Revised design.
Design	Specifies the design to display the pin equivalences. Choose *Revised*, *Golden*, or *Both* (the default).
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.

Renaming Rules Report

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

Renaming Rule Report Form Fields and Options

Map	*Map* (the default) displays only mapping renaming rules. *Module* displays only module renaming rules. *Pin* displays only pin renaming rules. *Full* displays all renaming rules.
Design	Specifies the design to display the renaming rules. Choose *Revised*, *Golden*, or *Both* (the default).

Search Paths Report

Use the `REPORT SEARCH PATH` command or the Search Path Report form (*Report – Search Path*) to display all paths used to search for library and design files.

There are no customized options for the Search Path Report. Click *Apply* to view the report in the Transcript window.

Tied Signals Report

Use the `REPORT TIED SIGNALS` command or the Tied Signals Report form (*Report – Tied Signals*) to display tied signals from the Golden and Revised designs.

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.
Design	Specifies the design to display the tied signals. Choose *Revised*, *Golden*, or *Both* (the default).

Mapped Points Report

Use the `REPORT MAPPED POINTS` command or the *Mapped Points* menu command (*Report – Mapped Points*) to display the mapped points that were automatically identified or added with the Conformal software. Each mapped point from the Golden and Revised design is displayed along with a summary of all Golden and Revised mapped points.

Unmapped Points Report

Use the `REPORT UNMAPPED POINTS` command or the *Unmapped Points* menu command (*Report – Unmapped Points*) to display a list of unmapped points, along with a summary of all of the unmapped points in the Golden and Revised designs.

Compared Points Report

Use the `REPORT COMPARED POINTS` command or the *Compared Points* menu command (*Report – Compared Points*) to display the compared points that were added with the Conformal software.

Compare Data Report

Use the `REPORT COMPARE DATA` command or the *Compare Data* menu command (*Report – Compare Data*) to display a list of all or specified compared points.

Statistics Report

Use the `REPORT STATISTICS` command or the Statistics Report form (*Report – Statistics*) to display the mapping and comparison statistics for the Golden and Revised designs.

There are no customized options for the Statistics Report. Click *Apply* to display the Statistics Report in the Transcript window.

8

Debugging

Diagnosing Non-Equivalent Points

Use the `DIAGNOSE` command to diagnose a non-equivalent compared point (to diagnose groups of points, see "Diagnosing Multiple Points" on page 170). The `DIAGNOSE` command pinpoints areas of difference so you can identify the probable causes of non-equivalent points. Using this command in conjunction with the schematic viewer can also help you locate mismatches. The diagnosis displays all of the non-corresponding support key points with a list of all likely error candidates from the Revised design.

Use the command with the `-summary` option to display a diagnosis summary table listing all of the non-equivalent points. The summary table helps you determine which non-equivalent point has the smallest cone size. You can then use this information to diagnose the non-equivalent point that has the smallest cone size first and work through non-equivalent points by degrees of complexity.

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.

Diagnosing Multiple Points

In many cases, multiple non-equivalences are due to the same error. For example, an error in a clock gating circuit can cause all DFFs driven by the same clock gating circuit to be non-equivalent. As another example, a wrong keypoint mapping can cause multiple frontier keypoints of the same wrong mapping to be non-equivalent. In these cases, diagnosing the entire group of non-equivalences simultaneously can better reveal the single root cause of the non-equivalences.

Use the `-group` option of the `DIAGNOSE` command to group non-equivalences that are likely caused by the same error. For example:

```
diagnose -noneq -group
```

analyzes all the non-equivalences and reports groups with more than one non-equivalence. Since the non-equivalences in the same group are caused by the same error, you can focus on the one with the smallest logic cone to diagnose. The following sample report groups the key points by non-equivalences, common supports, and common test vectors:

```
Group 0 contains 8 Non-equivalent key points:
 0.Non-equivalent key points:
  (G) + 16  PO   /OUT[0]
  (R) + 16  PO   /OUT[0]
 1.Non-equivalent key points:
  (G) + 15  PO   /OUT[1]
  (R) + 15  PO   /OUT[1]
 2.Non-equivalent key points:
  (G) + 14  PO   /OUT[2]
  (R) + 14  PO   /OUT[2]
 ......
 There are 2 common supports:
 0.
  + 4   PI  /A[0]                  + 4   PI  /A[0]
 1.
  + 8   PI  /B[0]                  + 8   PI  /B[0]

 Supports for this group:
  Corresponding supports:
  + 1   PI  /A[3]                  + 1   PI  /A[3]
  + 2   PI  /A[2]                  + 2   PI  /A[2]
  + 3   PI  /A[1]                  + 3   PI  /A[1]
  + 4   PI  /A[0]                  + 4   PI  /A[0]
  + 5   PI  /B[3]                  + 5   PI  /B[3]
  + 6   PI  /B[2]                  + 6   PI  /B[2]
  + 7   PI  /B[1]                  + 7   PI  /B[1]
  + 8   PI  /B[0]                  + 8   PI  /B[0]

  0: 10001111 DDDddddD
  1: 10011111 dddDDDDd
  2: 10101100 00DddddD
  3: 10111100 00dDDDDd
  4: 11001010 0D0ddddD
  5: 11011010 0d0DDDDd
  6: 10100110 0DdDddd0
  7: 10110110 0dDdDDD0
  8: 01001110 0DDDddd0
 ......
```

The non-equivalences are sorted by support size, in increasing order. By default, non-equivalences are grouped together if they share a common support or test vector. You can specify more stricter conditions for grouping. For example, you can require that the non-equivalences in one group have at least 5 common supports:

```
diagnose -noneq -group -common_support 5
```

The common test vectors are ordered according to the number of non-equivalences they cause, where the vector that causes the most non-equivalences is shown first.

You can also use `-group` option on specific keypoint. For example, the following reports the group that contains the compare point 14:

```
diagnose 14 -group
```

Diagnosing Using Test Vector Patterns

In multipoint diagnosis, test vectors are ordered so that you can identify patterns that can help diagnose the causing issue. The following are examples of the test vector patterns for non-equivalences.

d: Golden value is 0 and Revised value is 1
D: Golden value is 1 and Revised value is 0

- Phase mapping

  The following illustrates the test vector ordering for a non-equivalence caused by a phase mapping issue. Specifically, the support `a` in Golden is correctly mapped to the support `a'` in Revised, but the phase should be inverted.

  ```
  0: 10001111 DDDddddD
  1: 10011111 dddDDDDd
  2: 10101100 00DddddD
  3: 10111100 00dDDDDd
  .....
        ^ (a is the 4th column)
  ```

  If a vector `vec1` is a counter example, inverting the value of `a` in it is also a counter example. Note the output values are inverted too.

- Mapping

  The following illustrates the test vector ordering for a non-equivalence caused by a mapping issue. Specifically, the supports `a, b` in Golden are incorrectly mapped to the supports `b', a'`.

  ```
  0: 1001100001 dddDd
  1: 0101100001 DDDdD
  2: 1001010001 ddddD
  3: 0101010001 DDDDd
  ....
     ^^ (a is the first column, and b is the second column)
  ```

  Test vectors contain only patterns of `a, b` non- equivalence. The output values are also inverted when a, b change from 01 to 10.

- Sequential constant

The following illustrates the test vector ordering for a non-equivalence caused by a sequential constant issue. Specifically, the Golden key points have non-corresponding DFF supports, and some pattern of these non-corresponding supports is missing in all test vectors.

```
0: 11101110 01 Dddd0
1: 11011110 01 Ddd0d
2: 11101110 11 Dddd0
3: 11011110 11 Ddd0d
4: 11101110 10 0ddd0
5: 11011110 10 0dd0d
6: 11001110 01 Ddd00
7: 11001110 11 Ddd00
8: 11001110 10 0dd00
            ^^ (non corresponding supports that can be sequential
                constant 00)
```

For the above example, pattern 00 is missing, which is the value that these DFFs become after sequential constant propagation.

- Sequential merge

  The following illustrates the test vector ordering for a non-equivalence caused by a sequential merge issue. Specifically, both Golden and Revised key points have non-corresponding DFF supports. The all 0 or all 1 patterns of the non-corresponding DFF supports are missing from the test vectors.

```
0: 10000000000000000000000000000000 1 dddddddddddd1ddddddddddddddddddd
1: 01000000000000000000000000000000 1 dddddddddddddddddddddddddddd1ddd
2: 11000000000000000000000000000000 1 dddddddddddd1ddddddddddddddd1ddd
3: 11100000000000000000000000000000 0 0000000000D0D000000000000000D000
4: 11010000000000000000000000000000 0 000000000000D0000000D0000000D000
5: 11001000000000000000000000000000 0 000000000000D000000000000000D0D0
......
```

  In the above example, Golden has 32 DFFs that should be merged, which are shown as the first 32 supports. These DFFs directly feed to POs, which are proven non-equivalent. The last support shown is the Revised non-corresponding support. The error happens when Golden DFFs are taken at different values from the Revised support.

Proving Equivalence

Another command that is useful for debugging is the PROVE command. This command checks for equivalence and shows whether the specified gates from the Golden or Revised designs are equivalent or non-equivalent. Use the ADD DYNAMIC CONSTRAINTS command to specify constraints you want to use during the proof.

Use the following command to check equivalency for one of the following pairs:

- One gate in each of the Golden and Revised designs

- Two gates in the Golden design
- Two gates in the Revised design

Adding Dynamic Constraints

Use the `ADD DYNAMIC CONSTRAINTS` command, in conjunction with the `PROVE` and `DIAGNOSE` commands, to diagnose a non-equivalent point. This command gives you a time-efficient method of debugging that does not require you to exit the LEC system mode to add pin constraints, and then rerun the comparison to see how the constraints affect the comparison results. However, if you do exit the LEC system mode and rerun comparison, dynamic constraints no longer have any effect.

Displaying Error Patterns

Use the `REPORT TEST VECTOR` command to display the Revised design error patterns that caused non-equivalence at the diagnosis point.

Tip

If you intend to use the schematic viewer to aid diagnosis, you must start it while Conformal is in GUI mode. If the current session is in non-GUI mode, use the `SET GUI` command.

Reporting Design Similarities

The degree of structural similarity between two designs reflects the complexity of comparison. It usually is easier to compare two designs which have more similar structures. The dissimilar regions of the designs are possible root causes of aborts when comparing two designs. The design similarity report can help you understand these aborts and help to guide the adoption of methodologies to increase the design similarity and resolve aborts.

The dissimilar structures of two designs can be due to the following:

- usage of different datapath implementations
- resource sharing
- synthesis with don't cares
- low power synthesis

The Conformal software can help increase design similarity with features such as datapath analysis and functional partitioning. Designers can also increase design similarity with RTL recoding techniques, such as parenthesizing adder trees or avoiding coding with don't cares. You can also increase design similarity by re-synthesizing the netlist.

Use the `REPORT DESIGN SIMILARITY` command to report the degree of similarity between two designs. Design similarity reflects the complexity of comparison and it can be used to measure the effectiveness of methods used to increase design similarity. The value of similarity ranges from 0% to 100%, and is obtained by measuring the number of corresponding points in the two designs.

For example, if aborts occur when comparing two designs, `RTL1` and `GATE1`, you can run the `REPORT DESIGN SIMILARITY` command to get a report that shows that the `GATE1` netlist has a structure that is possibly very different from `RTL1`, which could possibly cause aborts.

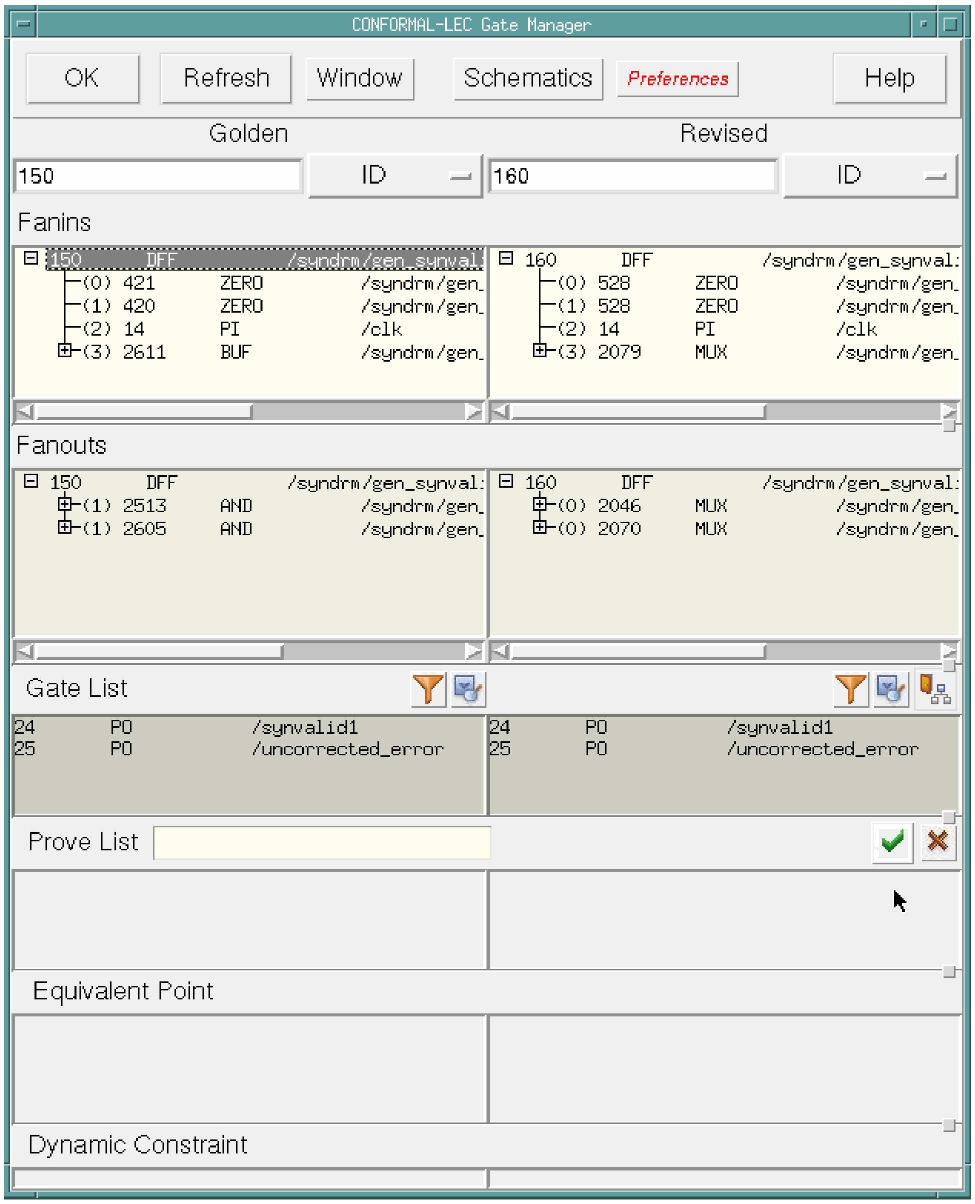
```
============================================================
Similarity          Region (Golden)
    30%              (root module)
============================================================
```

Then you can run several commands, such as `ANALYZE DATAPATH` or the automatic abort resolution `ANALYZE ABORT` command.

Run the `REPORT DESIGN SIMILARITY` command again after these commands to see if the design similarity has been increased. If the aborts remain, you could consider recoding the RTL or resynthesize the netlist to increase the similarity between the designs. The design similarity can be reported any time to track the progress.

Gate Manager

Use the Gate Manager (*Tools – Gate Manager*) to help you diagnose and debug your designs. You can also access this form from the Mapping Manager, Diagnosis Manager, and Schematic Viewer. See the sections on the related integrated debugging tools for more information.



Golden: 150 (ID) — Revised: 160 (ID)

Fanins:
```
Golden                                          Revised
⊟ 150        DFF       /syndrm/gen_synval:      ⊟ 160        DFF       /syndrm/gen_synval:
   ├(0) 421     ZERO      /syndrm/gen_             ├(0) 528     ZERO      /syndrm/gen_
   ├(1) 420     ZERO      /syndrm/gen_             ├(1) 528     ZERO      /syndrm/gen_
   ├(2) 14      PI        /clk                     ├(2) 14      PI        /clk
  ⊞(3) 2611    BUF       /syndrm/gen_            ⊞(3) 2079    MUX       /syndrm/gen_
```

Fanouts:
```
Golden                                          Revised
⊟ 150        DFF       /syndrm/gen_synval:      ⊟ 160        DFF       /syndrm/gen_synval:
  ⊞(1) 2513    AND       /syndrm/gen_            ⊞(0) 2046    MUX       /syndrm/gen_
  ⊞(1) 2605    AND       /syndrm/gen_            ⊞(0) 2070    MUX       /syndrm/gen_
```

Gate List:
```
24     PO     /synvalid1                  24     PO     /synvalid1
25     PO     /uncorrected_error          25     PO     /uncorrected_error
```

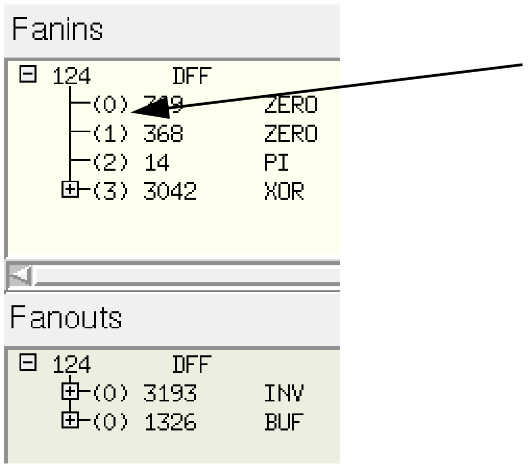
The Gate Manager includes two columns in each of the major sections. The left column contains Golden design information and the right column contains Revised design information.

For the Gate Manager, see the following for more information:

- Gate Manager Fields and Options on page 178
- Refreshing the Window on page 179
- Opening Schematics from the Gate Manager on page 179
- Using the Preferences Drop-Down Menu on page 179
- Filtering the Gate List on page 180
- Finding Gates on page 181
- Reporting Gate Information on page 182
- Customizing the Gate List Section with Specified Gates on page 182
- Proving Equivalency for Two Specified Gates on page 182
- Removing Gates from the Prove List on page 183
- Locating an Equivalent Gate on page 183
- Adding and Deleting Dynamic Constraints on page 183
- Locating a Gate in the Design Hierarchy on page 184
- Highlighting a Point in the Hierarchical Browser on page 184
- Viewing a Gate’s Location in the Source Code on page 185
- Highlighting a Point in the Source Code Manager on page 185
- Viewing a Schematic Representation of One Gate on page 185
- Gate Reporting on page 186

Gate Manager Fields and Options

Fanins	The *Fanins* section is a hierarchical gate browser for the fan-ins of specified gates. Click the (+) or (-) symbols to expand or compress the hierarchical gate display for a logic cone. The number within the parentheses indicates the pin location:



```
Fanins
⊟ 124       DFF
   ├(0) 719      ZERO
   ├(1) 368      ZERO
   ├(2) 14       PI
   ⊞(3) 3042     XOR

Fanouts
⊟ 124       DFF
   ⊞(0) 3193     INV
   ⊞(0) 1326     BUF
```

Fanouts	The *Fanouts* section is a hierarchical gate browser for the fan-outs of specified gates. Click the (+) or (-) symbols to expand or compress the hierarchical gate display for a logic cone.
Gate List	Select a gate listed in the Golden or Revised column of this section to: ■ Display its fan-ins and fan-outs ■ Add it to the Prove List section ■ Show its equivalent gate Open the Hierarchical Browser, Schematic Viewer, and Source Code Manager in the context-dependent mode
Prove List	When you add any two gates shown in the *Fanins*, *Fanouts*, or *Gate List* sections to the Prove List, Conformal displays them in this section. In this section, you can run the `PROVE` command for specified gates.

Equivalent Point	After you have run the `COMPARE` command, Conformal can display the corresponding equivalent gate of a specified gate in this section.
Dynamic Constraint	When you add a dynamic constraint to any gate in the *Fanins* or *Fanouts* sections of the Gate Manager, Conformal lists the gates in this section.

Refreshing the Window

Click the *Refresh* button to collapse the fan-ins and fan-outs to either the default level, which is 1, or a level you specified through the *Preferences* drop-down menu. (Refer to "Using the Preferences Drop-Down Menu" on page 179.)

Opening Schematics from the Gate Manager

When you use the procedure described below, two separate schematic windows open to display the Golden and Revised gates that are specified at the top of the Gate Manager. (You will have side-by-side viewing.)

1. Double-click a gate listed in the *Golden* column of the *Gate List* section of the Gate Manager.

   Conformal displays this gate in the *Golden* field at the top of the window.

2. Double-click a gate listed in the *Revised* column of the *Gate List* section of the Gate Manager.

   Conformal displays this gate in the *Revised* field at the top of the window.

3. Click on the *Schematic* icon located on the menu bar and click the *Show Inserted Buffer Gate* check box to display all of the buffer gates.

   The default is to collapse all buffers and not display them.

4. Click the *Schematic* icon and choose *Open*.

   Two schematic windows open. They display the fan-in cone of the selected gate and highlight the gate (unless it is a PI or PO).

Using the Preferences Drop-Down Menu

Use the following procedures to specify viewing preferences.

Customizing the Display

This procedure lets you display specified sections of the Gate Manager.

1. Click the *Preferences* button located on the menu bar.
2. Click the check boxes to display only the sections you specify.

Specifying Gate Display Options

By default, Conformal displays all inverters and buffers in fan-in and fan-out cones at the primitive level. Use the following steps to change the preferences to exclude inverters and buffers from the display and change the number of displayed levels in fan-in and fan-out cones.

1. Click the *Preferences* button located on the menu bar.
2. Click *Options*.

   The Gate Manager Display Option window appears.
3. Click the *Collapse Internal Buffers* check box according to your preferences.
4. Click the *Collapse Internal Inverters* check box according to your preferences.
5. Click in the *Fanin/Fanout Expand Level* field and type a number, or click the adjacent up- or down-arrow to specify the number of levels Conformal automatically displays in the fan-in and fan-out cones.

   You can also manually expand and collapse the levels by clicking on the + and - markers.
6. Click *Apply*.
7. Click *Close*.

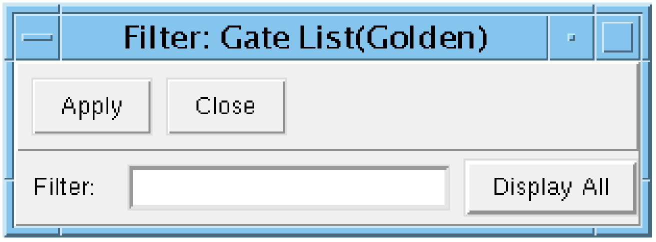
Filtering the Gate List

Use the following procedure to filter the display of gate IDs that match a specified string.

Note: Conformal supports wildcards. For example, type `*35*` to display all points that include `35` in their name or gate ID.

1. Click the *Filter* icon.

The section-specific filter window opens, for example, Filter: Gate List (Golden).



2. Type a string in the *Filter* field.
3. Click *Apply*.
4. To return to the original display, click *Display All*.
5. Click *Close*.

Note: If you click the *Refresh* button on the menu bar of the Gate Manager, the original unfiltered display returns.

Finding Gates

Use the following procedure to locate gate IDs based on a search string.

1. Click the *Find* icon button located in the upper right corner of the appropriate section, or press `Ctrl-f`. The section-specific Find window opens, for example, Find: Gate List (Golden).
2. Type any string or partial string of a key point name in the *Find* field.
3. Click the *Find Forward* or *Find Backward* check box to specify the direction of the search.
4. Click the *Case Sensitive* check box, if applicable.
5. Click the *Find* button to search for the name.
6. Repeat step 5 to find the next point that fulfills the search criteria.

Reporting Gate Information

The following procedure reports the corresponding gate information in the *Fanins* and *Fanouts* sections of the Gate Manager and in the Transcript window of the main window.

1. Click a gate in *Fanins*, *Fanouts*, or *Gate List* to select it:
2. Do one of the following:
   - ❑ Right-click and choose *Show Fanin/Fanout* from the pop-up menu.
   - ❑ Double-click on the gate.

Customizing the Gate List Section with Specified Gates

Use the following procedure to filter out categories of gates from the *Gate List* section of the Gate Manager, and then report on gates from the shortened list.

1. Click the *Class* icon located at the top right corner of the *Gate List* section of the window.
2. From the drop-down menu, choose the types of gates you want to view.

   For example, *All*, *PI*, *0*, *1*, *X*, *Z*, and so forth.

Proving Equivalency for Two Specified Gates

Use the following procedure to prove equivalence for two gates:

1. Click a gate in the *Golden* column in *Fanins*, *Fanouts*, or *Gate List* to select it:
2. Right-click and choose *Add Prove List* from the pop-up menu.

   Conformal adds the gate to the *Prove List* section in the appropriate column.
3. Repeat steps 1 and 2 for a second gate in the *Revised* column.
4. Click the *Prove* button located in the upper right corner of the *Prove List* section.

Conformal runs the `PROVE` command for the specified gates and prints the proof result in the status field at the top of the *Prove List* section and in the Transcript window of the main window.

Removing Gates from the Prove List

To remove a gate from the Prove list, click a gate in the *Prove List* section to select it, then right-click to open the pop-up menu and choose *Delete Prove List*.

Deleting All Gates from the Prove List

Do the following to remove all gates from the *Prove List* section.

➤ ✕ Click the *X* icon located in the upper right corner of the *Prove List* section.

Locating an Equivalent Gate

Use the following procedure to show the equivalent of a gate in the *Fanins*, *Fanouts*, or *Gate List* section in the *Golden* or *Revised* columns of the Gate Manager.

1. Click a gate in the *Gate List* section to select it.
2. Right-click and choose *Show Equivalent* from the pop-up menu.

   Conformal displays the corresponding equivalent gate, if any, in the *Equivalent Point* section. For example: select a gate from the *Golden* column and Conformal displays its equivalent in the *Revised* column of the *Equivalent Point* section.

Adding and Deleting Dynamic Constraints

Adding a Dynamic Constraint

Use the following procedure to add dynamic constraints.

1. Click a gate in *Fanins* or *Fanouts* to select it:
2. Right-click to open the pop-up menu and select *Add Dynamic Constraints 0* or *Add Dynamic Constraints 1*.

Deleting a Dynamic Constraint

Use the following procedure to delete dynamic constraints.

1. Click a gate in the *Dynamic Constraint* section to select it.
2. Right-click and choose *Delete Dynamic Constraints* from the pop-up menu.

   This choice removes the gate from the Dynamic Constraints section.

Locating a Gate in the Design Hierarchy

Use the following procedure to open the Hierarchical Browser window of the main window, scrolled to the appropriate location in the design hierarchy. Conformal applies an aqua highlight to the specified gate.

Note: This procedure does not apply to primary inputs (PI) or primary outputs (PO).

1. Click a gate in *Fanins*, *Fanouts*, or *Gate List* to select it:
2. Right-click and choose *Hierarchical Browser* from the pop-up menu.

Highlighting a Point in the Hierarchical Browser

Use the following procedure to select a point in the Gate Manager and find its location in the Hierarchical Browser.

Tip

Begin this procedure with the main window open on the desktop and the Gate Manager active.

1. Click to select a point in *Gate List*, *Fanins*, or *Fanouts*:
2. Using the middle mouse button, click and drag the selected point from the Gate Manager to the main window.

   When you click with the middle button, the Gate Manager displays the name of the point you clicked in an ivory text box. As you move the box to the new window, the background of the text box changes if the object is in a window where you can drop items.
3. Release the middle button, and the Hierarchical Browser window scrolls to and highlights the applicable line of code.

Viewing a Gate’s Location in the Source Code

Use the following procedure to open the Source Code Manager scrolled to the appropriate line of code. (An aqua square marks the first character of the line in question.) From the Source Code Manager, you can open an editor.

1. Click a gate in the *Fanins*, *Fanouts*, or *Gate List* section of the Gate Manager.
2. Right-click and choose *Source Code* from the pop-up menu.

Highlighting a Point in the Source Code Manager

Use the following procedure to select a point and find its location in the source code.

Tip

Begin this procedure with the Source Code Manager open and the Gate Manager active.

1. Click a point in the *Gate List*, *Fanins*, or *Fanouts* section.
2. Using the middle mouse button, click and drag the selected point from the Gate Manager to the Source Code Manager.

   When you click with the middle button, the Gate Manager displays the name of the point you clicked in an ivory text box. As you move the box to the new window, the background of the text box changes if the object is in a window where you can drop items.
3. Release the middle button, and the Source Code Manager scrolls to and highlights the applicable line of code.

Viewing a Schematic Representation of One Gate

When you use the following procedure, the schematic viewer displays the fan-in cone of the selected gate and highlights the gate (unless it is a PI or PO).

1. Click a gate in either the *Golden* or *Revised* column of the *Gate List* section.
2. Right-click and choose *Schematics*.

Gate Reporting

Use the REPORT GATE command report the gate ID, type, name, and its fanins and fan-outs at the primitive level. For information the structure of the gate report information at the primitive level, see Gate Report Structure on page 186.

Setting the Gate Report

Use the SET GATE REPORT command prior to running REPORT GATE to specify the detail level of gate reports.

- To report gate information at the design level, run the SET GATE REPORT command with the -design option.
- To exclude dynamic constraints in the gate report, run the SET GATE REPORT command with the -nodynamic option.
- To include fan-in cone of the zero/one gates in the gate report, run the SET GATE REPORT command with the -structure option.

Use the REPORT ENVIRONMENT command to display the gate report level settings.

Gate Tracing

The BACKWARD and FORWARD commands allow you to back trace and forward trace a gate. These commands show the fan-in and fan-out cones of a gate's inputs and outputs. The integer further specifies the trace; for example, backward 1 denotes the first fan-in.

Gate Report Structure

The structure of the gate report information at the primitive level is as follows:

Pin-name	ID (nnn)	Type	Tie	Gate-name (Library: mmm)
	gate_number	gate_type	(ttt)	uuu
---------- Fan-ins ---				
###: ppp	gate_number	gate_type	(ttt)	uuu
###: ppp	gate_number	gate_type	(ttt)	uuu

Pin-name	ID (nnn)	Type	Tie	Gate-name (Library: mmm)
###: ppp	gate_number	gate_type	(ttt)	uuu
###: ppp	gate_number	gate_type	(ttt)	uuu
---------- Fan-outs --				
###:	gate_number	gate_type		uuu

Where the following assignments are defined as:

nnn	Reported gate is from the Golden or Revised design
mmm	Reported gate has a corresponding library model name
###:	Fan-in or fan-out index integers
	(This integer is used in conjunction with the `BACKWARD` and `FORWARD` commands.)
ppp	Reported gate has corresponding pin names
gate_number	Identification number assigned to the gate
gate_type	Gate type of the reported gate
ttt	Reported gate or fanin pin can be one of the following:
	D0 or D1: added dynamic constraint of 1
	L0 or L1: learned internally to be 0 or 1
	C0 or C1: added pin constraint of 0 or 1
	R0 or R1: redundant learned internally to be 0 or 1
	F - Z gate created from a floating net or pin
	B - Z gate created from a tristate buffer or I/O pin modeling
uuu	The fanin or fan-out instance name connected to or from reported gate

The structure of the gate report information at the design level is as follows:

Pin-name	ID (nnn)	Cell-type	Pin-name	Name
	gate_number	cell_type	ttt	

Pin-name	**ID (nnn)**	**Cell-type**	**Pin-name**	**Name**
---------- Inputs ---				
###: ppp	gate_number	cell_type	ttt	uuu
###: ppp	gate_number	cell_type	ttt	uuu
###: ppp	gate_number	cell_type	ttt	uuu
###: ppp	gate_number	cell_type	ttt	uuu
---------- Outputs ---				
###: ppp	gate_number	cell_type	ttt	uuu
###: ppp	gate_number	cell_type	ttt	uuu

Where the following assignments are defined as:

`nnn`	Reported gate is from the Golden or Revised design
`###:`	Input or output index integers (This integer cannot be used in conjunction with the `BACKWARD` and `FORWARD` commands.)
`ppp`	Reported gate has corresponding pin names
`gate_number`	Identification number assigned to the gate
`cell_type`	Library cell name of the reported gate
`ttt`	Reported input or output pin connected to or from the reported gate
`uuu`	Input or output instance name connected to or from the reported gate

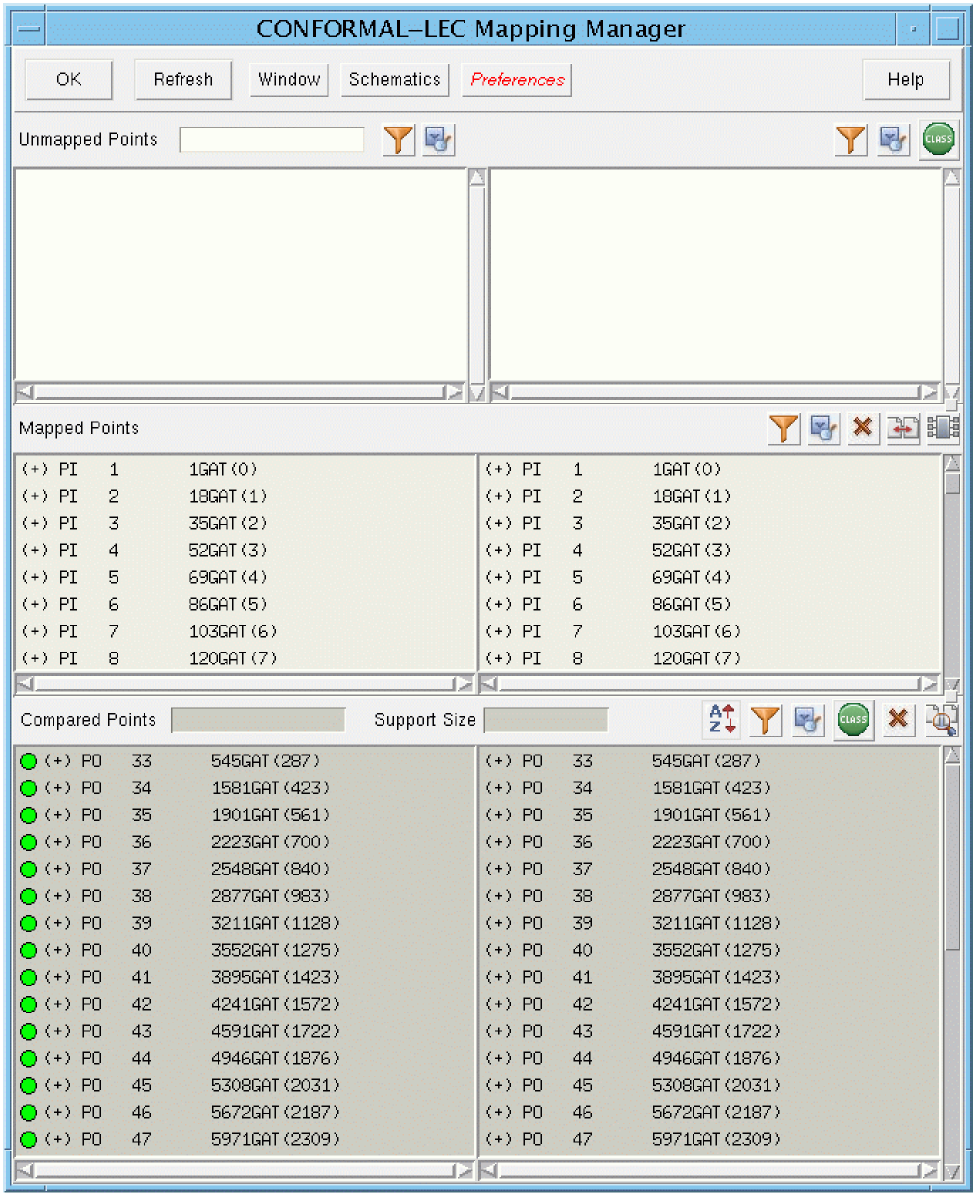
Mapping Manager

Use the Mapping Manager as a gateway to the integrated debugging environment. The Mapping Manager serves several functions:

- Displays the unmapped, mapped, sequential merge, and compared points
- Adds and deletes mapped and compared points accordingly
- Compares key points

- Gives the status of each compared point (equivalent, non-equivalent, inverted-equivalent, abort, or not-compared)

➤ Choose *Tools – Mapping Manager*.



```
(+) PI   1        1GAT(0)          (+) PI   1        1GAT(0)
(+) PI   2        18GAT(1)         (+) PI   2        18GAT(1)
(+) PI   3        35GAT(2)         (+) PI   3        35GAT(2)
(+) PI   4        52GAT(3)         (+) PI   4        52GAT(3)
(+) PI   5        69GAT(4)         (+) PI   5        69GAT(4)
(+) PI   6        86GAT(5)         (+) PI   6        86GAT(5)
(+) PI   7        103GAT(6)        (+) PI   7        103GAT(6)
(+) PI   8        120GAT(7)        (+) PI   8        120GAT(7)
```

```
● (+) PO   33       545GAT(287)      (+) PO   33       545GAT(287)
● (+) PO   34       1581GAT(423)     (+) PO   34       1581GAT(423)
● (+) PO   35       1901GAT(561)     (+) PO   35       1901GAT(561)
● (+) PO   36       2223GAT(700)     (+) PO   36       2223GAT(700)
● (+) PO   37       2548GAT(840)     (+) PO   37       2548GAT(840)
● (+) PO   38       2877GAT(983)     (+) PO   38       2877GAT(983)
● (+) PO   39       3211GAT(1128)    (+) PO   39       3211GAT(1128)
● (+) PO   40       3552GAT(1275)    (+) PO   40       3552GAT(1275)
● (+) PO   41       3895GAT(1423)    (+) PO   41       3895GAT(1423)
● (+) PO   42       4241GAT(1572)    (+) PO   42       4241GAT(1572)
● (+) PO   43       4591GAT(1722)    (+) PO   43       4591GAT(1722)
● (+) PO   44       4946GAT(1876)    (+) PO   44       4946GAT(1876)
● (+) PO   45       5308GAT(2031)    (+) PO   45       5308GAT(2031)
● (+) PO   46       5672GAT(2187)    (+) PO   46       5672GAT(2187)
● (+) PO   47       5971GAT(2309)    (+) PO   47       5971GAT(2309)
```

The Mapping Manager includes two columns in each of the major sections. The left column contains Golden design information and the right column contains Revised design information.

For the Mapping Manager, see the following for more information:

- Reporting Gate Information on page 201
- Displaying Fan-in and Fan-Out Information on page 201
- Locating a Point in the Source Code on page 202
- Locating a Point in the Design Hierarchy on page 203
- Viewing a Schematic of a Point on page 203

Mapping Manager Fields and Options

Note: You can also hover over a point to view details about it, or click on a point and its classification will display in the Unmapped Points text field.

- *Unmapped Points*— Lists all of the unmapped points in the Golden and Revised designs. Each unmapped point is classified as one of the following:

🟥	Extra
🟥	Unreachable
🔴	Not-mapped

- *Mapped Points*—Lists all of the mapped points in the Golden and Revised designs. When you select a mapped point in one of the columns, Conformal highlights its corresponding mapped point in the adjacent column.
- *Compared Points*—Lists all of the compared points from the Golden and Revised designs. Additionally, when you click a point, Conformal displays the status and support size in the text fields at the top of the Compared Points section. The status of the compared points is one of the following:

🟢✓	Equivalent
⊖	Inverted-Equivalent
🔴✗	Different
🟡	Abort

?	Not-Compared
+/- S	Sequential Merge (+/- indicate mapping phase)

Split status indicators are for top-level sequential merge instances. The left side shows the design-level compare result. The right side shows any sequential merge results:

●	Design level: Equivalent Sequential merge: Abort
●	Design level: Equivalent Sequential merge: Equivalent
●	Design level: Equivalent Sequential merge: Not compared
●	Design level: Equivalent Sequential merge: Not equivalent
●	Design level: Not equivalent Sequential merge: Abort
●	Design level: Not equivalent Sequential merge: Not compared
●	Design level: Not equivalent Sequential merge: Not equivalent
●	Design level: Not equivalent Sequential merge: Equivalent
●	Design level: Not compared Sequential merge: Abort
●	Design level: Not compared Sequential merge: Equivalent
●	Design level: Not compared Sequential merge: Not compared
●	Design level: Not compared Sequential merge: Not equivalent

Note: A compared point can have an S to indicate that it is a sequential merge—in that registers merge to it.

Tip

Conformal displays a red circle or a green circle to show whether points are equivalent or different.

Setting Preferences

Click the *Preferences* pull-down menu on the menu bar to specify the following viewing preferences:

Unmapped Points On	Customizes the display to show unmapped points.
Checked Points On	Customizes the display to show checked points.
Sort by Name	Sorts the displayed gates by name.
Sort by ID	Sorts the displayed gates by ID.
Library Name On	Display the suffixes when viewing mapped points.
Renaming Rule On	Displays renamed rule names. This is especially useful for determining how renaming rules affect a group of keypoints.

Copying Information from the Mapping Manager

You can copy information from the Mapping Manager 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.

Selecting Points

In the following text, when you are directed to select one or more points, you can do any of the following:

- Click a point to select it.
- Click the first point in a group, depress and hold the Shift key, and click the final point in a group to select the entire group.

- Depress the `Ctrl`-key and click a point to add it to the selected group.

Adding Unmapped Points as Mapped Points

Use the following procedure in the *Unmapped Points* section to manually map points.

1. Click an unmapped point in either the *Golden* or *Revised* column of the *Unmapped Points* section to select it.
2. Right-click and choose *Set Target Mapping Point* from the pop-up menu.

   The target mapping point is displayed in the color red.
3. Click the corresponding mapped point in the adjacent column of the *Unmapped Points* section.
4. Right-click on *Add Mapping Point* to open the pop-up menu and choose *Non-invert* or *Invert*.

   The two points appear in the *Mapped Points* section.

Adding Unmapped Points as Mapped Points (Keyboard Shortcut)

To manually map points in the *Unmapped Points* section, click a point to select it and press one of the following keys:

Key	**Function**
t	Sets the selected Golden or Revised point as the target mapping point.
n	Makes a corresponding mapped point in the adjacent column a non-inverted mapping point.
i	Makes a corresponding mapped point in the adjacent column an inverted mapping point.

Reporting Information on an Unreachable Gate

Use the following procedure to display an unreachable gate's information in the Transcript window.

1. Click an unreachable gate in either the *Golden* or *Revised* column of the *Unmapped Points* section to select it.

2. Right-click and choose *Report Unreachable Info* from the pop-up menu.

Reporting Renaming Rules

Use the following procedure to view renaming rules that apply to a specified point. The report shows the original and renamed paths in the Transcript window of the main window.

1. Click a point in either the *Golden* or *Revised* column of the *Unmapped Points* section.
2. Right-click and choose *Show Renamed Rule* from the pop-up menu.
3. View the results in the Transcript window of the main window.

Re-Mapping Key Points

➤ Click the *Re-map* icon located in the upper right corner of the *Mapped Points* section to run the `MAP KEY POINTS` command.

Adding All Compared Points

When you add compared points, they appear in the *Compared Points* section. A question mark (?) next to these points means that Conformal has not compared them.

➤ Click the *Add* icon located in the upper right corner of the *Mapped Points* section.

Deleting One or More Mapped Points

Use the following procedure to remove selected mapped points from the *Mapped Points* section and list them in the *Unmapped Points* section. See Deleting Mapped or Compared Points on page 201 to delete *all* mapped points.

1. Select one or more mapped points.

   For more information, see Selecting Points on page 193.)
2. Right-click and choose *Delete Mapping Point* from the pop-up menu.

Adding One or More Compared Points

Use the following procedure to add selected mapped points to the *Compared Points* section. A question mark (?) appears next to these compared points to show that Conformal did not compare them.

1. Select one or more mapped points.

   For more information, see Selecting Points on page 193.)

2. Right-click and choose *Add Compared Point* from the pop-up menu.

   Conformal adds the individual mapped point as a compared point.

Changing the Mapping Phase of a Mapped Point

Use the following procedure to invert the mapping phase of specified points.

1. Select one or more mapped points.

   For more information, see Selecting Points on page 193.)

2. Right-click and choose *Change Mapping Phase* from the pop-up menu.

   The mapped point becomes inverted-mapped. The display changes the + or - in the *Revised* column.

Changing Mapping Phase (Keyboard Shortcut)

Use the following shortcut procedure to invert the mapping phase of a specified point.

1. Click a point to select it.
2. Press `c` on the keyboard.

Highlighting a Mapped Point in the Compared Points Section

Use the following procedure to find a mapped point in the *Compared Points* section.

Note: This procedure does not apply to primary inputs (PI).

1. Click a point in the *Mapped Points* section.
2. Using the middle mouse button, click and drag the selected point from the *Mapped Points* section to the *Compared Points* section.

When you click with the middle button, the selected point is displayed in an ivory text box. As you move the point to the *Compared Points* section, the background of the text box changes if the object is in a window where you can drop items.

3. Release the middle button.

   Conformal scrolls the *Compared Points* section to and highlights the selected point.

Comparing Key Points

➤ Click the *Compare* icon located in the upper right corner of the *Compared Points* section.

When the comparison is complete, the equivalent compared points are marked with a green circle or a green check, according to your specifications. (See "Pass/Fail Icon Style" on page 239.) The non-equivalent compared points are marked with a red circle or red X.

Deleting One or More Compared Points

Do the following to remove one or more compared points from the *Compared Points* section. See Deleting Mapped or Compared Points on page 201 to delete *all* compared points.

1. Select one or more compared points.

   For more information, see Selecting Points on page 193.)

2. Right-click and choose *Delete Compared Point* from the pop-up menu.

Diagnosing a Non-Equivalent Point in the Compared Points Section

To open the Diagnosis Manager and display the diagnosis information for a specified non-equivalent compared point, click a non-equivalent compared point, and right-click to open the pop-up menu and choose *Diagnose*.

Tip

See *Diagnosis Manager* on page 205 for additional information about using this integrated tool.

Sorting Compared Points by Support Size

To sort compared points according to their support size, click a compared point, and right-click to open the pop-up menu and choose *Sort by Support Size*.

Sorting Compared Points by Non-Corresponding Support Cones

For debugging, you can sort points in the *Compared Points* section according to non-corresponding support. With this sort capability, the Conformal software shows all of the points with non-corresponding support cones first (from smallest to largest) followed by points with corresponding support cones (smallest to largest).

1. Position the cursor over the *Compared Points* section.
2. Right-click and choose *Sort by Non-corresponding* from the pop-up menu.

Changing the Mapping Phase of a Compared Point

Use the following procedure to change the mapping phase of a pair of points in the *Compared Points* section and rerun the comparison.

1. Click a compared point.

   The Conformal software highlights the Golden-Revised pair.
2. Right-click and choose *Change Mapping Phase* from the pop-up menu.

   The mapped point becomes inverted-mapped. The display changes the + or - in the *Revised* column. And the status indicator changes to not-compared.
3. Click the *Compare* icon.

Highlighting a Compared Point in the Mapped Points Section

Use the following procedure to find a compared point in the *Mapped Points* section.

1. Click a point in the *Compared Points* section.
2. Using the middle mouse button, click and drag the selected point from the *Compared Points* section to the *Mapped Points* section.

   When you click with the middle button, the selected point is displayed in an ivory text box. As you move the point to the *Mapped Points* section, the background of the text box changes if the object is in a window where you can drop items.

3. Release the middle button.

   Conformal scrolls the *Mapped Points* section to and highlights the specified point.

Displaying the Information Box

By default, when you rest the cursor over a point, Conformal displays an information box that identifies the point by name and lists pertinent details about it.

Filtering the Display

Use the following procedure in the *Unmapped Points*, *Mapped Points*, and *Compared Points* sections to display points that match a specified string. Conformal bases the filter on instance names, gate type, and gate IDs.

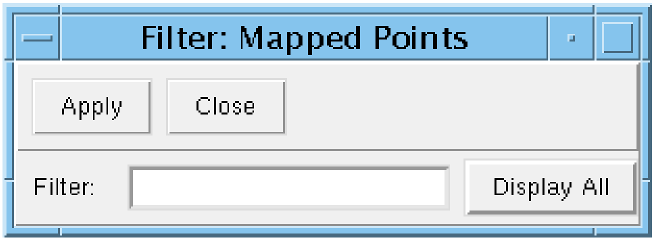
Note: Conformal supports wildcards. For example, type `*17*` to display all points that include `17` in either their name or gate ID.

Tip

To find a single point, refer to Finding Key Points on page 200“.

1. Click the *Filter* icon.

   The section-specific filter window opens (for example, Filter: Mapped Points).



2. Type a string in the *Filter* field.
3. Click *Apply*.
4. To return to the original display, click *Display All*.
5. Click *Close*.

Note: If you click the *Refresh* button on the menu bar of the Mapping Manager, the original unfiltered display returns.

Finding Key Points

Do the following in the *Unmapped Points*, *Mapped Points*, and *Compared Points* sections to locate points that contain the specified search string.

1. Click the *Find* icon button located in the upper right corner of the appropriate section, or press `Ctrl-f`. The section-specific Find window opens (for example, Find: mapped points).
2. Type any string or partial string of a key point name in the *Find* field.
3. Click the *Find Forward* or *Find Backward* check box to specify the direction of the search.
4. Click the *Case Sensitive* check box, if applicable.
5. Click the *Find* button to search for the name.
6. Repeat step 5 to find the next point that fulfills the search criteria.

Displaying Specified Classes of Points

In the *Unmapped Points* and *Compared Points* sections, use the following procedure to display specified classes of points.

1. Click the *Class* icon located in the upper right corner of the *Unmapped Points* or *Compared Points* section.
2. Choose one or more classes for the *Unmapped Points* or *Compared Points* section.

Deleting Mapped or Compared Points

Do the following to clear the *Mapped Points* or *Compared Points* display. When you clear the *Compared Points* section, Conformal removes the points. However, when you clear the *Mapped Points* section, Conformal moves points to the *Unmapped Points* section.

➤ Click the *X* icon located in the upper right corner of the *Mapped Points* section or the *Compared Points* section.

Displaying Diagnosis Data

After comparison, begin this procedure with the Diagnosis Manager open and the Mapping Manager active.

1. Click a point in the *Mapped Points* or *Compared Points* section.
2. Using the middle mouse button, click and drag the point from the Mapping Manager to the Diagnosis Manager (*Compared Point* field).

   When you click with the middle button, the name of the point you clicked is displayed in an ivory text box. As you move the point to the new window, the background of the text box changes if the object is in a window where you can drop items.
3. Release the middle button, and the Diagnosis Manager refreshes to show the applicable diagnosis data.

Reporting Gate Information

Use the following procedure in any of the three Mapping Manager sections to open the Gate Manager displaying information about the selected gate.

1. Click an unmapped, mapped, or compared point in either the *Golden* or *Revised* column of the *Unmapped Points*, *Mapped Points*, or *Compared Points* section.
2. Right-click and choose *Report Gate* from the pop-up menu.

   The Gate Manager opens. For more information, see Gate Manager on page 176.

Displaying Fan-in and Fan-Out Information

Use the following procedure to see the fan-in and fan-out information of a specified point.

Tip

Begin the following procedure with the Gate Manager open and the Mapping Manager active.

1. Click a point in the *Unmapped Points*, *Mapped Points*, or *Compared Points* section of the Mapping Manager:
2. Using the middle mouse button, click and drag the point from the Mapping Manager to the Gate Manager (*Fanins* or *Fanouts* section).

   When you click with the middle button, the name of the point you clicked is displayed in an ivory text box. As you move the point to the new window, the background of the text box changes if the object is in a window where you can drop items.
3. Release the middle button, and the Gate Manager refreshes the *Fanins* and *Fanouts* sections accordingly.

Locating a Point in the Source Code

Use the following procedure in any of the three Mapping Manager sections to open the Source Code Manager in the context-dependent mode.

1. Click a point in either the *Golden* or *Revised* column of the *Unmapped Points*, *Mapped Points*, or *Compared Points* section.
2. Right-click and choose *Source Code* from the pop-up menu.

   The Source Code Manager opens and scrolls to the selected point, which is highlighted aqua.

Tip

From the Source Code Manager, you can open an editor.

Locating a Point in the Source Code Manager (Drag and Drop)

Use the following procedure to find a mapped or compared point in the Source Code Manager.

Tip

Begin this procedure with the Source Code Manager open and the Mapping Manager active.

1. Click a point in the *Mapped Points* or *Compared Points* section.
2. Using the middle mouse button, click and drag the point from the Mapping Manager to the Source Code Manager.

   When you click with the middle button, the name of the point you clicked is displayed in an ivory text box. As you move the point to the new window, the background of the text box changes if the object is in a window where you can drop items.
3. Release the middle button, and the Source Code Manager scrolls to and highlights the applicable line of code.

Locating a Point in the Design Hierarchy

Use the following procedure to select a point in the *Unmapped Points*, *Mapped Points*, or *Compared Points* section of the Mapping Manager and find it in the Hierarchical Browser window of the main window.

Note: This procedure does not apply to primary inputs (PI) or primary outputs (PO).

1. In the Mapping Manager, click a point in either the *Golden* or *Revised* column of the *Unmapped Points*, *Mapped Points*, or *Compared Points* section.
2. Right-click and choose *Hierarchical Browser* from the pop-up menu.

   Conformal applies an aqua highlight to the specified gate in the Hierarchical Browser window.

Viewing a Schematic of a Point

Use the following procedure to select a point in the *Unmapped Points*, *Mapped Points*, or *Compared Points* section of the Mapping Manager and view a schematic of the selected point.

Note: When you select equivalent compared points, the schematics do not show simulation values.

1. In the Mapping Manager, click a point in either the *Golden* or *Revised* column of the *Unmapped Points*, *Mapped Points*, or *Compared Points* section.
2. Right-click and choose *Schematics* from the pop-up menu.

   The schematic viewer opens and highlights the selected point.

Locating a Point in the Schematic (Drag-and-Drop)

Use the following procedure in the *Unmapped Points*, *Mapped Points*, or *Compared Points* section of the Mapping Manager to locate and highlight a specified point in the schematic. This feature also works in the reverse drag-and-drop order (drag from the Flattened Schematics window to a section in the Mapping Manager).

Tip

Begin this procedure with the schematic viewer open and the Mapping Manager active.

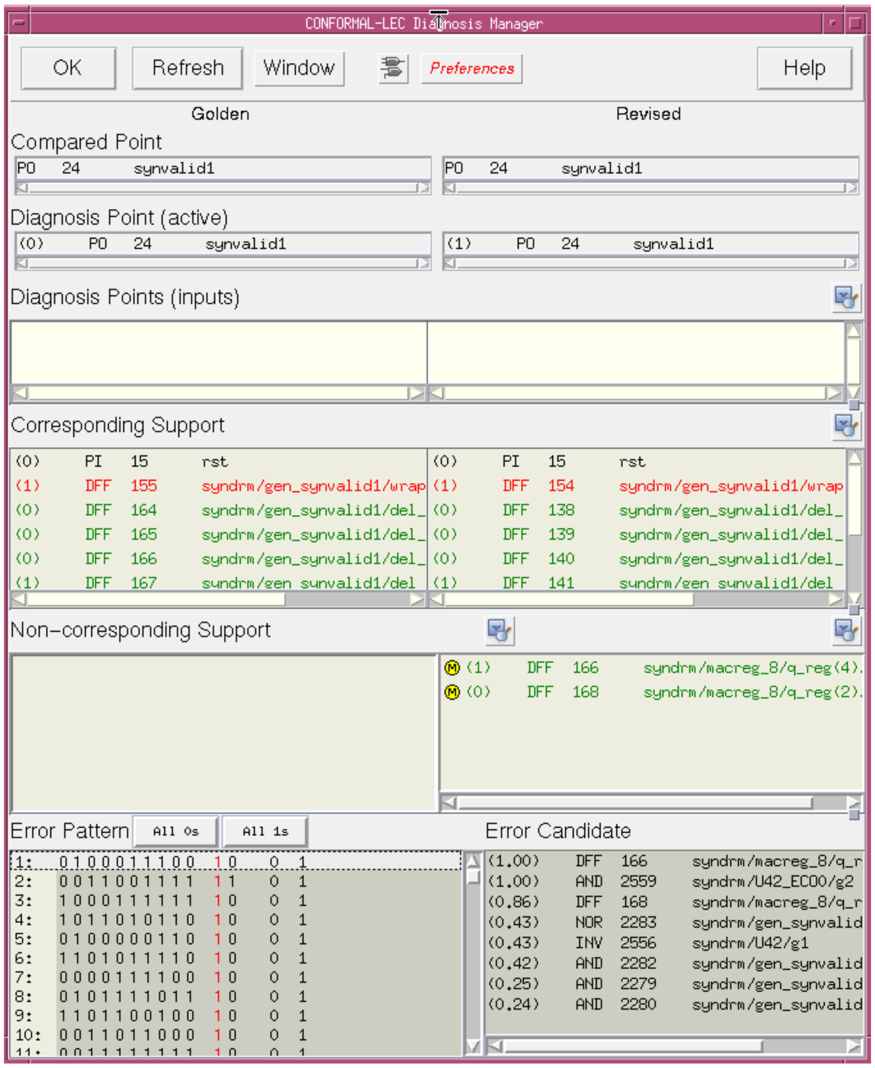
1. In the Mapping Manager, click and hold the middle mouse button over a point in the *Unmapped Points*, *Mapped Points*, or *Compared Points* section.
2. Drag the point from the Mapping Manager to the Flattened Schematic window.

   When you click with the middle button, the Mapping Manager displays the name of the point you clicked in an ivory text box. As you move the point to the new window, the background of the text box changes to black if the object is in a window where you can drop items.
3. Release the middle mouse button, and the point is highlighted in the Flattened Schematic window.

Diagnosis Manager

Use the Diagnosis Manager to display the error patterns and error candidates for non-equivalent points. Additionally, it lists the corresponding and non-corresponding support points in the logic fan-in cone for both the Golden and Revised designs. Error patterns can be written as a testbench for simulation on another tester.

To access the Diagnosis Manager for a non-equivalent point, right click on the non-equivalent point from the *Compared Points* section of the Mapping Manager and click on Diagnose. Or, Choose *Tools – Diagnosis Manager*. See Diagnosing a Non-Equivalent Point in the Compared Points Section on page 197 for additional details about how to get to the Diagnosis Manager.



Compared Point

```
Golden:  PO   24      synvalid1
Revised: PO   24      synvalid1
```

Diagnosis Point (active)

```
Golden:  (0)    PO   24      synvalid1
Revised: (1)    PO   24      synvalid1
```

Corresponding Support

```
(0)   PI    15    rst                            (0)   PI    15    rst
(1)   DFF   155   syndrm/gen_synvalid1/wrap      (1)   DFF   154   syndrm/gen_synvalid1/wrap
(0)   DFF   164   syndrm/gen_synvalid1/del_      (0)   DFF   138   syndrm/gen_synvalid1/del_
(0)   DFF   165   syndrm/gen_synvalid1/del_      (0)   DFF   139   syndrm/gen_synvalid1/del_
(0)   DFF   166   syndrm/gen_synvalid1/del_      (0)   DFF   140   syndrm/gen_synvalid1/del_
(1)   DFF   167   syndrm/gen_synvalid1/del       (1)   DFF   141   syndrm/gen_synvalid1/del
```

Non-corresponding Support

```
(1)   DFF   166   syndrm/macreg_8/q_reg(4).
(0)   DFF   168   syndrm/macreg_8/q_reg(2).
```

Error Pattern

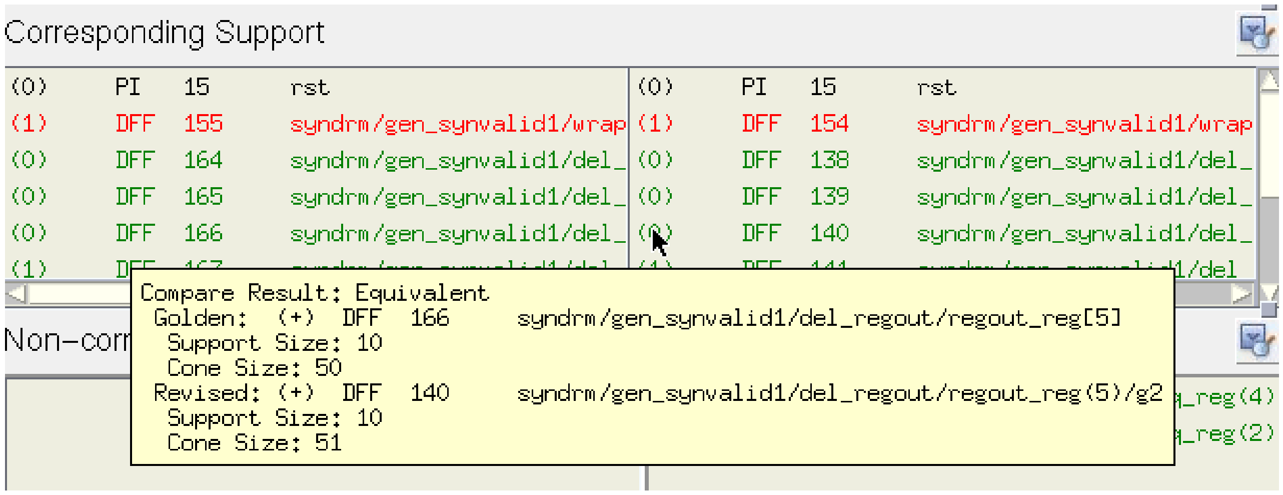
```
1:   0 1 0 0 0 1 1 1 0 0   1 0   0  1
2:   0 0 1 1 0 0 1 1 1 1   1 1   0  1
3:   1 0 0 0 1 1 1 1 1 1   1 0   0  1
4:   1 0 1 1 0 1 0 1 1 0   1 0   0  1
5:   0 1 0 0 0 0 0 1 1 0   1 0   0  1
6:   1 1 0 1 0 1 1 1 1 0   1 0   0  1
7:   0 0 0 0 1 1 1 1 0 0   1 0   0  1
8:   0 1 0 1 1 1 1 0 1 1   1 0   0  1
9:   1 1 0 1 1 0 0 1 0 0   1 0   0  1
10:  0 0 1 1 0 1 1 0 0 0   1 0   0  1
11:  0 0 1 1 1 1 1 1 1 1   1 0   0  1
```

Error Candidate

```
(1.00)   DFF   166    syndrm/macreg_8/q_r
(1.00)   AND   2559   syndrm/U42_ECO0/g2
(0.86)   DFF   168    syndrm/macreg_8/q_r
(0.43)   NOR   2283   syndrm/gen_synvalid
(0.43)   INV   2556   syndrm/U42/g1
(0.42)   AND   2282   syndrm/gen_synvalid
(0.25)   AND   2279   syndrm/gen_synvalid
(0.24)   AND   2280   syndrm/gen_synvalid
```

The Diagnosis Manager includes two columns in each section. The left column contains Golden design information and the right column contains Revised design information.

You can obtain a quick summary about a compared point by hovering over it.



Corresponding Support

```
(0)  PI   15   rst                          (0)  PI   15   rst
(1)  DFF  155  syndrm/gen_synvalid1/wrap    (1)  DFF  154  syndrm/gen_synvalid1/wrap
(0)  DFF  164  syndrm/gen_synvalid1/del_    (0)  DFF  138  syndrm/gen_synvalid1/del_
(0)  DFF  165  syndrm/gen_synvalid1/del_    (0)  DFF  139  syndrm/gen_synvalid1/del_
(0)  DFF  166  syndrm/gen_synvalid1/del_    (0)  DFF  140  syndrm/gen_synvalid1/del_
```

```
Compare Result: Equivalent
 Golden:  (+)  DFF  166     syndrm/gen_synvalid1/del_regout/regout_reg[5]
  Support Size: 10
  Cone Size: 50
 Revised: (+)  DFF  140     syndrm/gen_synvalid1/del_regout/regout_reg(5)/g2
  Support Size: 10
  Cone Size: 51
```

For the Diagnosis Manager, see the following for more information:

- Diagnosis Manager Fields and Options on page 207
- Setting Preferences on page 210
- Copying Information from the Diagnosis Manager on page 212
- Refreshing the Window on page 212
- Displaying the Information Box on page 212
- Selecting a New Active Diagnosis Point on page 212
- Changing the Simulation Value on page 212
- Saving Modified Values as an Error Pattern on page 213
- Viewing a Schematic on page 213
- Changing the Mapping Phase of a Mapped Point on page 214
- Deleting Mapped Points on page 214
- Reporting Renaming Rules on page 215
- Adding Unmapped Points as Mapped Points on page 215
- Viewing a Schematic Representation of Diagnosis Points on page 215

Diagnosis Manager Fields and Options

Compare Point	Displays the non-equivalent point that was selected for diagnosis for both the Golden and Revised designs. You will have selected this point from the *Compared Points* section of the Mapping Manager.
Diagnosis Point (active)	Displays the point at which the equivalency check failed. It displays the compare point for both the Golden and Revised designs. A simulation value is shown in parentheses ( ). If there is no simulation value, Conformal displays a (-). For inverted-mapped points, (+/-) indicates the mapping phase. For DLATs where there is feedback interaction between the Q outputs and input cone, (X/Y) indicates the current state and the next state. The first value (X) represents the current state of the DLAT, the second value (Y) represents the next state of the DLAT. For example, (0/1) indicates that the DLAT currently has a value of 0 at its Q output. After feeding in all of the test vectors from its support points, the DLAT will have a value of 1 at its Q output for the next state.
Diagnosis Points (inputs)	Lists all of the fan-in diagnosis points for the compare point. For inverted-mapped points, (+/-) indicates the mapping phase. In cases where there is more than one diagnosis point, double click on a point to make it the active diagnosis point.

Corresponding Support *Non-corresponding Support*	Displays the simulation values with all of the mapped points that are in the fan-in cone of the diagnosis point. Both the Golden and Revised designs are represented. To change the simulation value, right-click on the mapped point and use the pop-up menu. By default, support points are color coded as. See Support Points on page 209. For inverted-mapped points, (+/-) indicates the mapping phase For more information on corresponding and non-corresponding support points, see Figure 8-2 on page 210.
Error Pattern	Displays all of the test vectors that prove the diagnosis point to be non-equivalent. If the bit is 0 in all error patterns, the point is highlighted in green. If the bit is 1 in all error patterns, it is highlighted in red. When you select a support point, Conformal applies a pink highlight to the associated column in the test vector set. You can also select the following options to filter the column display. ■ *All 0s* – displays the bit(s) that are 0 in all error patterns. Use the *p* keyboard stroke to move the highlight to the next column of 0 in all error patterns; or the *n* key to move to the previous column of all 0. ■ *All 1s* – displays the bit(s) that are 1 in all error patterns. Use the *p* keyboard stroke to move the highlight to the next column of 1 in all error patterns; or the *n* key to move to the previous column of all 1.

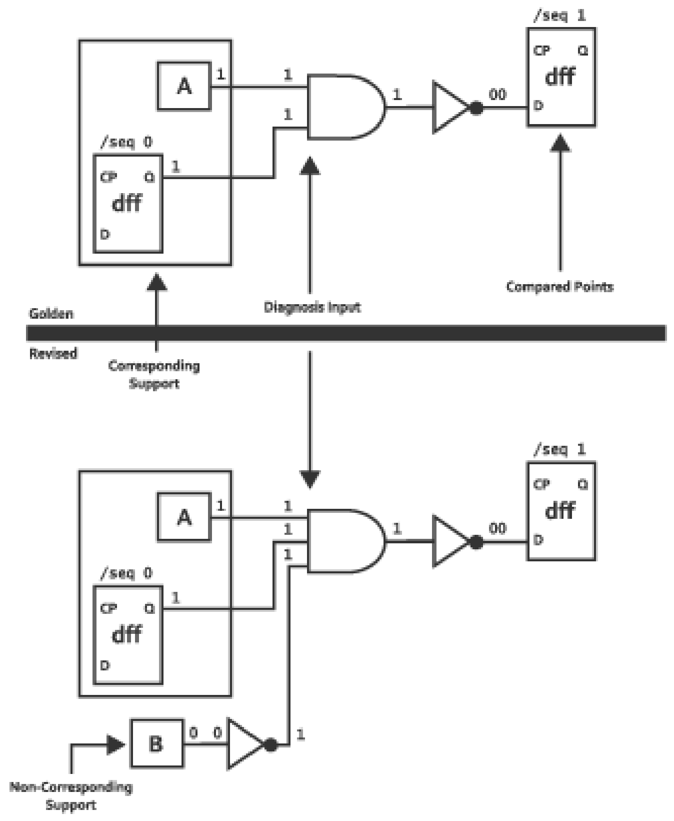
Error Candidate	Lists the gates in the Revised design with the highest probability of causing non-equivalence. The list is ordered from greatest to least probability, and displays a weighted percentage number displayed in decimal form. Thus, (1.00) signifies that the gate in the Revised design has the highest probability of causing non-equivalence. For a graphic description of corresponding and non-corresponding support, see Figure 8-2 on page 210.

Figure 8-1 Support Points

Support points are color coded as follows:

🟢	(Green) The support point is an equivalent compare point.
🔴	(Red) The support point is a non-equivalent compare point.
🟡	(Brown) Abort
	(Black) The support point either has not been compared yet or cannot be compared (for example, PI).
	(Yellow with an R) The support point is redundant logic associated with a don't care gate.
Ⓜ	(Yellow with an M) The support point is a mapped point. The point exists for this logic cone but not its corresponding mapped point.
	(Red circle) Support point is an unmapped point

Figure 8-2 Corresponding and Non-Corresponding Support Points



Setting Preferences

Use the following procedures to specify viewing preferences.

Showing Specified Sections of the Diagnosis Manager

To turn on and off sections of the Diagnosis Manage, click the *Preferences* button on the menu bar and click the check boxes to select specific sections for the display.

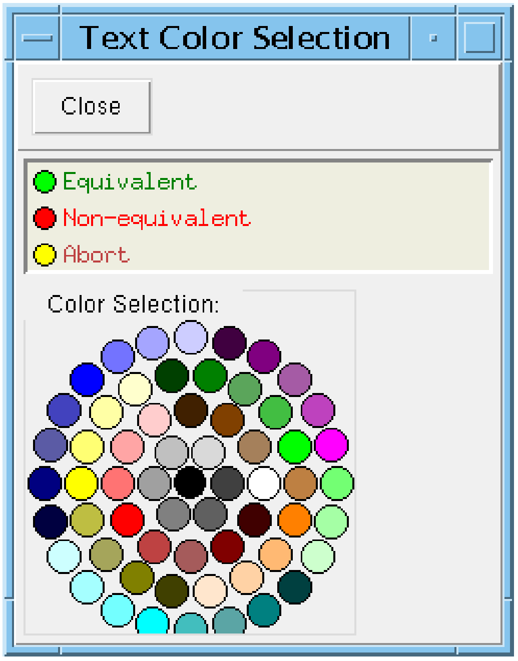
Sorting Gates in the Diagnosis Manager

To sort gates alphabetically by name or numerically by ID, click the *Preferences* button on the menu bar and choose *Sort by Name* or *Sort by ID*.

Setting the Text Color for Points in the Diagnosis Manager

Use the following procedure to set the text color for equivalent, non-equivalent, and abort points.

1. Click the *Preferences* button located on the menu bar.
2. Click *Set EQ/NEQ/Abort* text color to open the Color Selection window.



3. Click to select an *Equivalent*, *Non-equivalent*, or *Abort* point type in the list box.
4. Click a color on the *Color Selection* wheel.
5. Repeat steps 3 and 4 if necessary.
6. Click *Close*.

Copying Information from the Diagnosis Manager

You can copy information from the Diagnosis Manager 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.

Refreshing the Window

Click *Refresh* to return the displayed gates to their original numerical order.

Displaying the Information Box

By default, when you rest the cursor over a point in the *Corresponding Support* or *Non-Corresponding Support* section in the Diagnosis Manager, Conformal displays an information box that identifies the point by name and lists pertinent details about it.

Selecting a New Active Diagnosis Point

Use the following procedure to select a new diagnosis point for the Diagnosis Manager. Conformal updates the Diagnosis Manager display with the applicable points in the *Corresponding Support*, *Non-corresponding Support*, *Error Pattern*, and *Error Candidate* sections according to the selected diagnosis point.

1. Click one of the points in the *Diagnosis Points (inputs)* or *Corresponding Support* section.
2. Right-click and choose *Diagnose* from the pop-up menu.

Changing the Simulation Value

In the Diagnosis Manager, you can change the simulation value for each pair of mapped support points that appears in the *Golden* and *Revised* columns. Refer to Figure 8-2 on page 210 for a graphic description of corresponding and non-corresponding support points.

1. Click a point in one of the *Corresponding Support* or *Non-corresponding Support* section.
2. Right-click to view the pop-up menu and choose one of the following:

- ❑ *Set Value 0* to change the simulation value to 0.
- ❑ *Set Value 1* to change the simulation value to 1.

Note: The simulation value of the *Diagnosis point (active)* can change when you change the simulation value of a support point.

Saving Modified Values as an Error Pattern

You can save the modified simulation values in the *Corresponding Support* or *Non-corresponding Support* section of the Diagnosis Manager as an error pattern using the following procedure.

1. Click a point in the *Corresponding Support* or *Non-corresponding Support* section.
2. Right-click and choose *Save Pattern* from the pop-up menu.

   Conformal writes the current test vector and appends it to the list of error patterns in the *Error Pattern* section.

Viewing a Schematic

In the Diagnosis Manager, use the following procedure to open schematics displaying the specified points.

1. Click a point in the *Diagnosis Points (inputs)*, *Corresponding Support*, *Non-corresponding Support*, or *Error Candidate* section.
2. Right-click and choose *Schematics* from the pop-up menu.

   Conformal opens the schematic viewer showing the selected point. In the case of pairs of points, (Golden and Revised), two schematic viewer windows open for side-by-side viewing.

   You can hover over an object to view more information on it. The information box will also display information on corresponding points, if applicable.

Error pattern values are not displayed when opening the schematics from these sections. To view error pattern values, click on the Schematic icon located on the menu bar.

Flattened Schematics Predefined Color Scheme

The following table shows the color defaults for the flattened schematic viewer when opening from the Diagnosis Manager:

Color	**Description**
Purple	Gates proven to be equivalent between the Golden or Revised designs.
Red	Gate or net that is a potential candidate for an error path, or a trace load object.
Blue	Key point (DFF or DLAT) that needs to be compared. This is typically a flip-flop or latch.
Yellow	Non-corresponding support key point (DFF, DLAT, PI, BBOX or cut gate), or a trace driver object.
Green	Key point (DFF or DLAT) that proves to be a constant zero.
Pink	Key point (DFF or DLAT) that proves to be a constant one.
Cyan	Gate that drives the input of the compare key point (DFF or DLAT)

Changing the Mapping Phase of a Mapped Point

To change the mapping phase of a pair of points in the *Corresponding Support* section of the Diagnosis Manager, click a pair of corresponding points, and right-click to open the pop-up menu and choose *Change Mapping Phase*.

The mapped point becomes inverted-mapped. Conformal changes the `1` or `0` in the *Revised* column.

Deleting Mapped Points

To remove specified mapped points from the *Corresponding Support* section of the Diagnosis Manager and list them in the *Non-Corresponding Support* section, click a pair of corresponding points, and right-click to open the pop-up menu and choose *Delete Mapping Point*.

Reporting Renaming Rules

Use the following procedure to view renaming rules that apply to a non-corresponding support point. The report shows the original and renamed paths in the Transcript window of the main window.

1. Click a point in either the *Golden* or *Revised* column of the *Non-Corresponding* section to select it.
2. Right-click and choose *Show Renamed Rule* from the pop-up menu.
3. View the results in the Transcript window of the main window.

Adding Unmapped Points as Mapped Points

Use the following procedure to manually map points in the *Non-Corresponding* section of the Diagnosis Manager and move them to the *Corresponding Support* section.

1. Click a point in either the *Golden* or *Revised* column of the *Non-Corresponding* section to select it.
2. Right-click and choose *Set Target Mapping Point* from the pop-up menu.

   The target mapping point text color changes to red.
3. Click the corresponding mapped point in the adjacent column of the *Non-Corresponding* section.
4. Right-click and choose *Add Mapping Point* and *Non-invert* or *Invert* from the pop-up menu.

Viewing a Schematic Representation of Diagnosis Points

You can view the schematic representation of the diagnosis point for both the Golden and Revised designs. When you access the schematic viewer using the procedure that follows, two separate schematic windows open to give you side-by-side viewing for the Golden and Revised designs. The schematic viewer displays the fan-in cone of the diagnosis point.

Note: Multiple diagnosis pairs cannot be viewed simultaneously.

1. Click the *Schematic* icon located on the menu bar.
2. Click the *Show Inserted Buffer Gate* check box to display all of the buffer gates. (By default, the schematic viewer collapses all buffers and does not display them.)
3. Click the *Schematic* icon and choose *Open*.

Conformal opens a pair of schematic windows showing the Golden and Revised diagnosis points.

Locating a Point in the Schematic (Drag-and-Drop Feature)

Use the following procedure in the *Diagnosis Points (inputs)*, *Corresponding Support*, *Non-corresponding Support*, or *Error Candidate* section as a convenient way to locate and highlight a specified point in the schematic. This feature also works in the reverse drag-and-drop order (drag from the Flattened Schematics window to the Diagnosis Manager).

1. With the Flattened Schematics window open, click and hold the middle mouse button over a point in the Diagnosis Manager.
2. Drag the point from the Diagnosis Manager to the Flattened Schematic window.

   When you click with the middle button, the Diagnosis Manager displays the name of the point you clicked in an ivory text box. As you move the point to the new window, the background of the text box changes to black if the object is in a window where you can drop items.
3. Release the middle mouse button, and the object is highlighted in the Flattened Schematic window.

Displaying the Fill Fanin Cone

To display the initial (flattened) schematics with full fan-in logic cone, as opposed to pruned cone, click the *Show Schematic Full Fanin Cone* check box from the *Preference* pull-down menu.

Displaying Gate Information

To open the Gate Manager with the selected gate information displayed, click a gate to select it, and right click to open the pop-up menu and choose *Report Gate*.

For more information, see Gate Manager on page 176.

Displaying Gate Information (Drag-and-Drop Feature)

With the Gate Manager open and the Diagnosis Manager active, use this drag-and-drop procedure to display selected gate information.

1. In the Diagnosis Manager, click a gate to select it.

2. Click and hold the middle mouse button over the selected point.
3. Drag the point from the Diagnosis Manager to the Gate Manager (*Fanins* or *Fanouts* section).

   When you click with the middle button, the Diagnosis Manager displays the name of the selected point in an ivory text box. As you move the point to the new window, the background of the text box changes to black if the point is in a window where you can drop items.
4. Release the middle mouse button, and the Gate Manager refreshes the *Fanins* and *Fanouts* sections accordingly.

Showing a Gate's Location in the Source Code

Use the following procedure to open the Source Code Manager with the selected gate highlighted. From the Source Code Manager, you can open an editor.

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

Showing a Gate's Location in the Source Code (Drag-and-Drop)

With the Source Code Manager open and the Diagnosis Manager active, use this drag-and-drop procedure to locate a key point or error pattern in the Source Code Manager.

1. In the Diagnosis Manager, click a key point or error candidate.
2. Click and hold the middle mouse button over the selected item in the Diagnosis Manager.
3. Drag the item from the Diagnosis Manager to the Source Code Manager.

   When you click with the middle button, the Diagnosis Manager displays the name of the item you selected in an ivory text box. As you move the item to the new window, the background of the text box changes to black if it is in a window where you can drop items.
4. Release the middle mouse button, and the Source Code Manager scrolls to and highlights the applicable location in the code.

Showing Where a Gate is Located in the Design Hierarchy

Do the following to open the main window with a selected gate highlighted in the Hierarchical Browser window section.

1. Click a gate to select it.
2. Right-click and choose *Hierarchical Browser* from the pop-up menu.

Exit Status Codes

Upon exiting (using the `EXIT` text command or *File - Exit* menu command), the Conformal software returns a status code. A nonzero status code shows a potential error; that is, either no comparison was done or unmapped, abort, or non-equivalent points exist.

The exit status code consists of flags that represent different conditions. Bit 0 is the least significant bit. Refer to the table below for a list of flags. This table is followed by three case examples of nonzero exit status codes.

Tip

To view the status codes without exiting Conformal, use the `SET EXIT CODE` command or the Tcl `get_exit_code` command. (See the Tcl Command Entry Mode Support chapter of the *Conformal Equivalence Checking Reference Manual* for additional information about Conformal Tcl commands.

Bit	**Condition**
0	Internal Error
1	Exit status before comparison
2	Command error
3	Unmapped points or extra POs
4	Non-equivalent points during comparison
5	Abort or uncompared points exist during any comparisons.
6	Abort or uncompared points exist during the last comparison or hierarchical comparison.

Note: For bits 0, and 2 through 5, once they are set to 1, they will remain at 1.

Note: For bit 1, once it is set to 0, it will remain at 0.

- Case 1:

  Start Conformal and then exit immediately
  Status = 2 (00010 in binary). There are no equivalent points since there was no comparison. Thus, bit 1 is set.

- Case 2:

  Comparison produced a non-equivalent point, an abort point, and an equivalent point.
  Status = 48 (110000 in binary). Bits 4 and 5 are set to flag the abort and non-equivalent points.

- Case 3:

  Comparison produced all non-equivalent points.
  Status = 18 (010010 in binary). Bits 1 and 4 are set to show two conditions: During this session, Conformal found no equivalent points *and* the comparison produced non-equivalent points.

9

Resolving Aborts

Overview

Aborts are compare points that have not been conclusively compared. By the time aborts are reported, the Conformal software has applied multiple algorithms without a deterministic result. Aborts can have several causes including don’t cares, large cones, and large numbers of inputs. Cones with these attributes will result in increased runtime. To deal with extremely long runtimes, developers have limited the amount of time which the tool can use on a specific compare point. When the Conformal software exceeds this limit, the compare point will be reported as an abort. By setting a time limit for each compare point, the Conformal software avoids the appearance that it is locked up when it is still processing the compare point.

Avoiding Aborts

The best method for handling aborts is to avoid them in the first place. This implies the use of equivalency checker friendly coding practices and modularization of large cones of logic. This section offers recommendations on how to avoid aborts.

RTL Guidelines

Coding can have a huge impact on the outcome of a comparison. Don’t cares can add extra inputs to the cone of logic, doubling the number of vectors for verification with every ’don’t care’. Resource sharing can merge multiple operators and require the comparison of a huge cone of logic. Using ungroup in synthesis can create larger blocks to work on. The following sections identify issues in RTL and synthesis that can cause aborts, with recommendations on how to handle them.

Hierarchy

Hierarchy creates boundaries for the Conformal software to work with. Creating boundaries around datapath operators can help simplify the verification task. Placing difficult arithmetic operators in modules isolates difficult comparisons. By simplifying the verification task, comparisons will take less time and aborts will be reduced. Do not write too large a Verilog module or VHDL entity. Attempt to separate the structural code from the behavioral.

Synthesis

Synthesis tools can do special optimizations to achieve timing and area goals. Some of these optimizations cause problems for equivalency checking. These include ungrouping, inversion pushing, resource sharing, name changes, and other boundary optimizations.

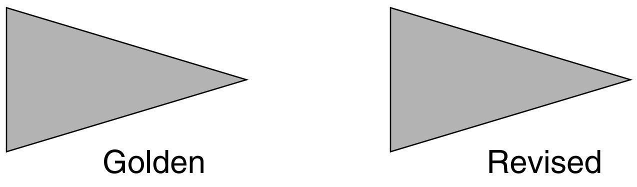
If possible, avoid ungrouping and inversion pushing. These methods complicate the verification task and most often do not provide much improvement in quality of results (QoR). If you plan to use advanced optimization methods, set up your flow to utilize the Module-Based Datapath (MDP) Flow (see MDP Flow on page 225).

Operator Optimization

Designers often cluster groups of mathematical operators into a single line. This is a fast and easy coding style.; however, this allows the synthesis engine to merge and combine operators with no distinguishing factors. This results in huge cones of logic and aborts.

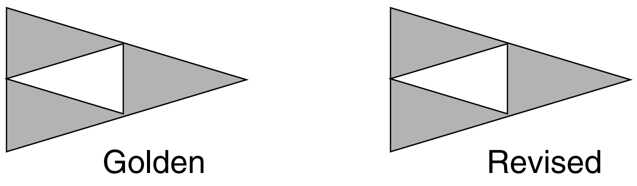
Adding parentheses guides the synthesis tool to group the operators so they can be distinguished. Assign statements also increase the likelihood of equivalency checking success. For example:

```
assign pmo = (((x * y) & mask) + offset);
```



By breaking the original lengthy assign statement into multiple statements, the logic cones are shorter and operator grouping is more easily identifiable. For example:

```
assign p = (x * y);
assign pm = (p & mask);
assign pmo = (pm + offset);
```



X-Sources and Don’t Cares

Avoid using ’X’ or Don’t care in your RTL code. Setting default values to X creates ’X’ sources and decrease the chance of success. You will need to find the ’X’ sources in the RTL and modify the code to eliminate them.

Before writing the RTL code, review rule checks reported by the Conformal software that indicates potential ‘X’ sources in the code. The following shows examples of how these rule check messages are reported:

- `x_assignment`

  In the following example, the design assigns a `1'bx` value to output `o`. See line 6 (in bold).

  ```
  module test(o,i);
  output o;
  reg o;
  input i;
  always
  o = 1'bx;
  endmodule
  // Warning: (RTL6.1) X created due to the assignment of value X (occurrence:1)
  directory: x_assignment/
  ```

- `range_overflow`

  The following example uses an index where its right hand side might be out of range. See line 10 (in bold):

  ```
  module test (clk, idx, in0, out0);
  input clk;
  input [3:0] idx;
  input [3:0] in0;
  output out0;
  reg out0;

    always @ ( posedge clk)
      begin
        out0 <= in0[idx];
      end
    endmodule
  // Warning: (RTL7.3) Array index in RHS might be out of range (occurrence:2)
  ```

  The following example uses an index where its left hand side might be out of range. See line 10 (in bold):

  ```
  module test (clk, idx, in0, out0);
  input clk;
  input [3:0] idx;
  input in0;
  output [3:0] out0;
  reg [3:0] out0;

    always @ ( posedge clk)
      begin
        out0[idx] <= in0;
      end
    endmodule
  // Warning: (RTL7.4) Array index in LHS might be out of range (occurrence:2)
  directory: range_overflow/
  ```

MDP Flow

Module-Based Datapath (MDP) analysis performs datapath analysis at a module level and can be used to resolve aborts. MDP analysis is performed in addition to and prior to the regular operator level analysis. The goal of this analysis is to improve the quality of the operator-level analysis. For MDP analysis to be successful, the synthetic datapath module must be preserved in the netlist. Therefore, ungrouping and boundary optimization must be disabled during synthesis.

MDP analysis generates an intermediate netlist in DC using the `mdp.tcl` script, included in the Conformal software. Providing an intermediate netlist reduces the amount of difference between the compared designs, thus simplifying the effort level for Conformal and handles most aborts. This automatically creates an intermediate netlist and a resource file for advanced datapath analysis.

For more information, see Module-Based Datapath Analysis on page 255.

RTL Compiler Flow

The RTL Compiler recommended synthesis flow is detailed in the “Interfacing with Conformal Logical Equivalence Checker” chapter of the *Interfacing between RTL Compiler and Conformal User Guide* (this document is available within the RTL Compiler document set).

However, the recommended flow in this chapter does not caution against setting the synthesis effort level to high when running synthesis. In avoiding aborts, Cadence recommends keeping the effort levels at their default levels. Using high synthesis efforts uses more aggressive optimizations and architecture selections that might not be supported by Conformal’s datapath analysis.

The following is the recommended flow modified with this effort level caution:

```
read_hdl ...
elaborate ...
read_sdc ...
synthesize -to_generic
//Do not set -effort high. Use the default effort level
synthesize  -to_mapped
//Do not set -effort high. Use the default effort level
write_hdl -lec > first_mapped.v
write_do_lec -revised first_mapped.v
//ungroup in any way)
//no more datapath architecture change)
synthesize -incremental (as many times as you wish)
```

```
write_hdl > final.v
write_do_lec -revised first_mapped.v -logfile rtl2firstmap.log > rtl2firstmap.do
write_do_lec -Golden first_mapped.v -revised final.v \
             -logfile first2finalmap.log > first2finalmap.do
exit
```

Resolving Aborts

Assuming all the precautions were taken as detailed in Avoiding Aborts on page 222, if aborts are reported, you should resolve them quickly. The steps for resolution depend on the cause of the abort and other factors with the design. This section describes some advanced techniques on how to resolve aborts.

Hierarchical Comparison

Running hierarchical comparison provides a convenient method for handling aborts. To run hierarchical comparison, run the `WRITE HIER_COMPARE DOFILE` command to write out a dofile that compares the design module by module. This dofile compares modules separately or in groups, depending on tool settings. After each module or module group comparison, results are recorded, the module is blackboxed, and the next module is processed. This process continues until all modules are processed.

For more information, see Chapter 10, "Running Hierarchical Comparison."

Analyzing Abort Points

After comparison, run the `ANALYZE ABORT` command to automatically recommend steps to resolve the abort points in your design. This command can also perform the recommendations and recompare the design, which might automatically solve the aborts without any further input.

Automatic Comparison Example

```
LEC> compare
================================================================================
Compared points      PO     DFF       Total
--------------------------------------------------------------------------------
Equivalent           196    577       773
--------------------------------------------------------------------------------
Abort                0      23        23
```

```
================================================================================

LEC> analyze abort -compare
...
================================================================================
Compared points       PO      DFF        Total
--------------------------------------------------------------------------------
Equivalent            196     600        796
================================================================================
```

Multithreading

Parallel comparison is best suited for large gate-to-gate comparisons, where the comparison can be distributed to multiple comparison threads. To possibly resolve more abort points and reduce the time spent on RTL-to-gate comparisons, the parallel analyze abort feature might be more effective (see Analyzing Abort Points on page 226).

You can only run multithreaded comparison with the `COMPARE` command's `-abort_stop` option to stop the comparison after finding the specified number of abort points.

For more information on multithreading, see Multithreading Process on page 277.

Partitioning

Partitioning can be used to help break down large cones of logic that can result in aborts. You can do this with the `ADD PARTITION POINTS` command (see Adding Partition Points on page 284 for more information)

If manually adding partition points is too difficult, the Conformal software can add functional partition points on the abort points based on the number of key points for a partition. In the following example, the software selects four common key points from the abort logic cones as a partition, then performs 16 comparisons for each bit combination (2n4=16):

```
...
add compare point -all
compare
usage
run partition_compare -number 4 -verbose
usage
```

Isolating Abort Modules

Comparison may report aborts that may be due to low quality MDP analysis. However, from the comparison report, it is not easily known if the aborts are coming from a particular DP_OP module that had low quality analysis results. By using '`-isolate_abort_module`' option with the '`analyze datapath -module -resourcefile <file>`' command, during MDP analysis, any low quality DP_OP module analysis that aborts will be isolated. This means that the DP_OP module is abstracted into the RTL and the synthesis netlist has been simplified to allow LEC to compare the RTL with the simplified synthesis netlist. If LEC comparison results are now EQ, then LEC will report the isolated DP_OP module. This represents that all the initial aborts occurred within this DP_OP module. By isolating this DP_OP module, LEC will able to compare successfully the rest of the logic.

Further investigation to resolve the isolated DP_OP module is necessary, but at least the root cause of the initial aborts are known. If there are additional abort points reported along with the aborted DP_OP module, this means that those aborts occur outside the DP_OP module boundary. Usually, continuing with multithreaded abort analysis will resolve these remaining aborts.

When using this option:

- LEC will automatically identify and isolate the aborted DP_OP modules. It will report the abort DP_OP module beneath the comparison results.
- Any isolated aborted DP_OP modules that were reported during a hierarchical module comparison will all be reported again at the end of the hierarchical comparison, when the final results are reported.
- This option is only used during MDP analysis with a resource file.
- Any reported isolated DP_OP modules must still be investigated. It does not mean the LEC run is completely EQ.

Note: Alternatively, you can specify the module to be isolated with the `SET DATAPATH OPTION` command's `-isolate_abort_module` option.

The following shows the results of running `ANALYZE DATAPATH` without isolating the abort modules.

```
// Command: analyze datapath -module -verbose -resourcefile resourcefile.rpt
// Note: add_5822_DP_OP_308_4437_42 : quality evaluated 100% success
// Note: add_5821_DP_OP_306_4437_41 : quality evaluated 100% success
// Note: add_1055_159_DP_OP_311_2879_8 : quality evaluated 99% success
// Note: add_5823_DP_OP_310_4437_43 : quality evaluated 100% success
// Note: add_5820_DP_OP_304_4437_40 : quality evaluated 100% success
// Note: add_9399_S2_DP_OP_314_2331_11 : quality evaluated 38% success
```

```
...
// Command: compare
======================================================================
Compared points      PO         Total
----------------------------------------------------------------
Equivalent           64         64
----------------------------------------------------------------
Abort                3          3
=======================================================================
```

The following shows the results of running `ANALYZE DATAPATH` with isolating the abort modules.

```
// Command: analyze datapath -module -verbose -resourcefile resourcefile.rpt \
                              -isolate_abort_module
// Note: add_5822_DP_OP_308_4437_42 : quality evaluated 100% success
// Note: add_5821_DP_OP_306_4437_41 : quality evaluated 100% success
// Note: add_1055_159_DP_OP_311_2879_8 : quality evaluated 99% success
// Note: add_5823_DP_OP_310_4437_43 : quality evaluated 100% success
// Note: add_5820_DP_OP_304_4437_40 : quality evaluated 100% success
// Note: add_9399_S2_DP_OP_314_2331_11 : quality evaluated 38% success
// Warning: add_9399_S2_DP_OP_314_2331_11 is isolated as an aborted instance.
...
// Command: compare
======================================================================
Compared points      PO         Total
----------------------------------------------------------------
Equivalent           67         67
======================================================================
Compared results of isolated instances in Revised design (top)
======================================================================
Status          Instance (Module)
----------------------------------------------------------------
Abort           i5/add_9399_S2_DP_OP_314_2331_11
                (NV_GR_PE_STRI_core_add_9399_S2_DP_OP_314_2331_0)
======================================================================
```

Dofile Template Scripts

Hierarchical Compare with MDP Flow

```
/// hier compare
read design -Golden ...
read design -revised ...
usage
report design data
report black box
uniquify -all -nolibrary
<constraints>
write hier dofile <hier.do> -replace -usage -constraint -noexact -run_hier \
  -prepend_string "report design data; analyze setup -verbose; usage; \
     analyze datapath -module -resourcefile <file> -verbose; usage; \
     analyze datapath -verbose; usage"
usage
run hier <hier.do> -analyze_abort
usage
```

Hierarchical Compare with MDP and Multithreading

```
/// hier compare
read design -Golden ...
read design -revised ...
usage
report design data
report black box
uniquify -all -nolibrary
<constraints>
write hier dofile <hier.do> -replace -usage -constraint -noexact -run_hier \
  -prepend_string "report design data; analyze setup -verbose; usage; \
     analyze datapath -module -resourcefile <file> -verbose; usage; \
     analyze datapath -verbose; usage"
usage
set parallel option -threads 4
run hier <hier.do> -analyze_abort
usage
```

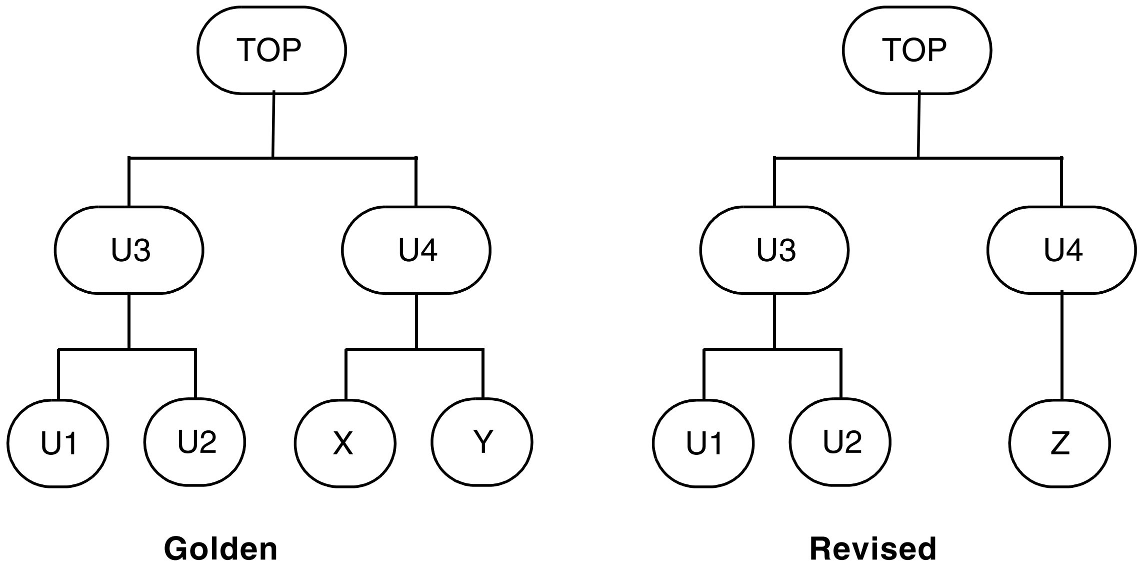
10

Running Hierarchical Comparison

Comparing Designs at the Module Level

Hierarchical comparison is a method of comparing designs at the module level in a bottom-up fashion. Comparison begins with the lowest-level modules, proceeds through higher-level modules, and culminates when the Conformal software reaches the top-level module. The efficiency of your comparison depends on the hierarchical similarity in the designs that the the Conformal software is comparing; that is, the more similar the hierarchical structure of the two designs, the faster the comparison.

The following figure and example demonstrate how a hierarchical comparison proceeds from the lower-level modules to the top root module. The following figure illustrates the hierarchical structures of a Golden and Revised design.



The following example lists the events that occur when the Conformal software compares the two designs. Refer to the figure above as you review this example.

1. Compares U1 Module:
    a. Sets root module to U1 (Both designs)
    b. Compares U1
    c. Blackboxes module U1 (Both designs)
2. Compares U2 Module:

   a. Sets root module to U2 (Both designs)
   b. Compares U2
   c. Blackboxes module U2 (Both designs)

3. Compares U3 Module:
   a. Sets root module to U3 (Both designs)
   b. Compares U3
   c. Blackboxes module U3 (Both designs)

Modules X and Y in the Golden design do not have corresponding modules in the Revised design. Similarly, module Z in the Revised design does not have a corresponding module in the Golden design.

In this case, the Conformal software finds correspondence one level higher, at module U4. The submodules of U4 in both the Golden and Revised designs are flattened during the comparison at the U4 module level.

1. Compares U4 Module:
   a. Sets root module to U4 (Both designs)
   b. Compares U4
   c. Blackboxes module U4 (Both designs)
2. Compares at TOP Module:
   a. Sets root module to TOP (Both designs)
   b. Compares TOP

Running Dynamic Hierarchical Comparison

Note: This feature requires XL license.

After generating a dofile, you can dynamically run hierarchical comparison. The following are the major features of dynamic hierarchical comparison:

- Dynamic Module Comparison

  Automatically flattens the selective modules to propagate the design error (if any) to the top module. The flattened modules are merged to the next level in the hierarchy and automatically compared at that level. This feature dynamically determines the modules that must not be blackboxed for successful hierarchical comparisons.

- Dynamic Module Selection

  Gives you the ability to execute multiple hierarchical runs without regenerating the dofile. You can generate the dofile only once for the top-level module, and the subsequent hierarchical comparisons for any specific submodules can be carried out using different flow-control options. In addition, you can specifically target the aborted or retimed modules without having to modify the hierarchical compare script or dofile.

- Demand Driven Module Comparison

  Gives you the ability to perform demand driven module comparison, such that only the un-compared modules will be compared in successive hierarchical runs. In addition, you can interrupt and continue hierarchical comparison any time.

To run dynamic hierarchical comparison, use the `RUN HIER_COMPARE <dofile>` command, where `<dofile>` is the dofile generated with the `WRITE HIER_COMPARE DOFILE` command.

Interrupting a Hierarchical Comparison.

To interrupt a hierarchical comparison, type `Ctrl-c`. To continue the hierarchical comparison, run the `RUN HIER_COMPARE` command.

Hierarchical Comparison Command Flow

This command flow applies to both non-graphical and graphical modes. See Hierarchical Module Comparison Window on page 242 for more information on how to run a hierarchical comparison using the various command menus and windows.

Read the Libraries and Designs

As with any other comparison, you must first specify the designs and the corresponding libraries. You will read in designs and libraries in the Setup system mode.

Generate a Hierarchical Compare Dofile

After successfully reading in the libraries and designs, run the `WRITE HIER_COMPARE DOFILE` command. This command generates a dofile script that compares two hierarchical designs.

For the purposes of a hierarchical comparison, you can use a dofile to read the library and designs, and write the hierarchical compare dofile script to a file.

The following sample dofile writes the hierarchical compare dofile script into a file named `hier_compare.do`:

```
read library lib.v -both
read design Golden.v -verilog -Golden
read design revised.v -verilog -revised
write hier_compare dofile hier_compare.do
```

Tip

You can also use the Hierarchical Module Comparison window. See Writing the Hierarchical Compare Dofile to a File on page 244.

Example hier_compare.do Dofile

Using the procedure outlined above, the Conformal software generates a hierarchical dofile script named `hier_compare.do`.

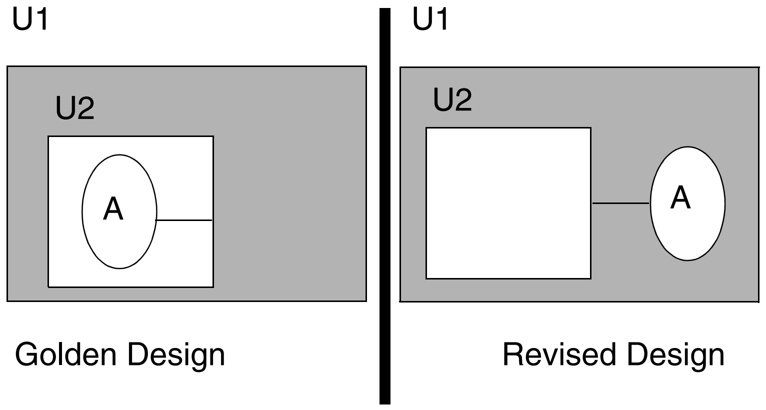
Note: In the following script, the hierarchical comparison first pairs corresponding modules from the two designs, then compares each of these pairs one at a time in a bottom-up fashion.

```
set system mode setup
//
// Comparing module ‘U1’
//
set root module U1 -Golden
set root module U1 -revised
report black box -NOHidden
set system mode lec
add compare point -all
compare
save hier_compare result
set system mode setup
add black box U1 -module -Golden
add black box U1 -module -revised
//
// Comparing module ‘U2’
//
set root module U2 -Golden
set root module U2 -revised
report black box -NOHidden
set system mode lec
```

```
add compare point -all
compare
save hier_compare result
set system mode setup
add black box U2 -module -Golden
add black box U2 -module -revised
//
// Comparing module 'U3'
//
set root module U3 -Golden
set root module U3 -revised
report black box -NOHidden
set system mode lec
add compare point -all
compare
save hier_compare result
set system mode setup
//
// Comparing module 'U4'
//
set root module U4 -Golden
set root module U4 -revised
report black box -NOHidden
set system mode lec
add compare point -all
compare
save hier_compare result
set system mode setup
add black box U4 -module -Golden
add black box U4 -module -revised
//
// Comparing module 'TOP'
//
set root module TOP -Golden
set root module TOP -revised
report black box -NOHidden
set system mode lec
add compare point -all
compare
save hier_compare result
set system mode setup
add black box TOP -module -Golden
add black box TOP -module -revised
report hier_compare result
report hier_compare result -nonequivalent
report hier_compare result -Abort
report hier_compare result -Uncompared
```

No Blackboxing

For some designs, you might want to skip comparison at a certain module hierarchy level, even though the Conformal software can successfully pair the corresponding modules. The following diagram illustrates one such situation. An explanation follows.



In this example, a portion of the logic (labeled “A”) of module U2 in the Golden design (shown on the left) was optimized to produce the Revised design. This optimization occurred across hierarchical boundaries. As a result, logic block “A” is a part of module U1 in the Revised design.

If the Conformal software compares at the U2 level, it detects differences. Similarly, if U2 is blackboxed and the Conformal software compares at the U1 level, it detects differences. However, if every element under the U1 modules is flattened before comparison, the Conformal software reports that the U1 modules are equivalent.

In conditions such as the one described above, use the `ADD NOBLACK BOX` command.

Note: You must use the `ADD NOBLACK BOX` command before the `WRITE HIER_COMPARE DOFILE` command, but after the Conformal software reads the designs.

This section illustrates a hierarchical comparison on a design that requires the `ADD NOBLACK BOX` command. Using the information above, your dofile should appear as shown in the following example. Note the proper placement of the `ADD NOBLACK BOX` command. Additionally, with the `-replace` option, the Conformal software replaces the original `hier_compare.do` file.

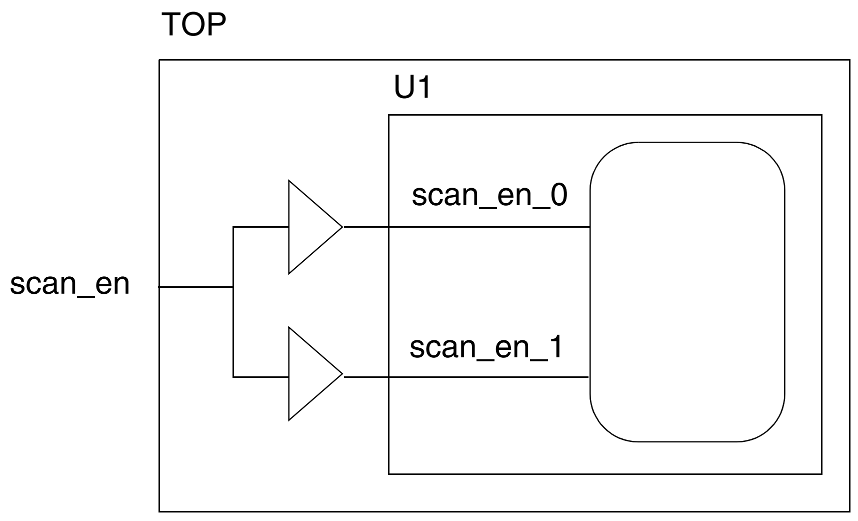
```
read library lib.v -both
read design Golden.v -verilog -Golden
read design revised.v -verilog -revised
add noblack box U2 -both
write hier_compare dofile hier_compare.do -replace
```

Constraint Propagation

In a *flat* comparison, the constraint value on scan_en is automatically propagated throughout the design. As a result, scan circuitry is correctly isolated from comparison regardless of its placement in the hierarchy.

Unlike a flat comparison, a hierarchical comparison occurs at the submodule level. Therefore, you must include all information at the module level.

In the following example, the Conformal software does a hierarchical comparison on two designs, one without scan and one with scan inserted. The following figure shows a design with scan inserted:



Because the scan circuitry only exists in the Revised design, the comparison is only relevant for the functional portion of the designs. To place the scan design (Revised) in functional mode, you must constrain the `scan_en` pin to logic-0.

In a *hierarchical* comparison you must use the `ADD PIN CONSTRAINT` command to propagate this constraint. Thus, the constraint is available at lower module levels. The information needed in this case is:

```
scan_en_0 = scan_en_1 = 0;
```

Adding a Pin Constraint

Use the following command:

```
add pin constraints 0 scan_en -revised
```

Storing Constraint Information

The Conformal software stores constraint information in the hierarchical compare dofile. Generate the dofile by including the `-constraint` option in the `WRITE HIER_COMPARE DOFILE` command as follows:

```
write hier_compare dofile hier_compare.do-constraint -replace
```

The resulting hierarchical compare dofile is similar to the one that was shown in Example hier_compare.do Dofile on page 235. However, constraint information has been added:

```
…
//
// Comparing module ‘U1’
//
set root module U1 -Golden

set root module U1 -revised
add pin constraint 0 scan_en_0 -revised
add pin equivalences scan_en_0 scan_en_1 -revised
report black box -NOHidden
set system mode lec
add compare point -all
compare
save hier_compare result
set system mode setup
add black box U1 -module -both
…
```

In this section you found that the hierarchical comparison required a pin constraint. Using the information above, your modified dofile should appear as follows:

```
read library lib.v -both
read design Golden.v -verilog -Golden
read design revised.v -verilog -revised
add pin constraint 0 scan_en -revised
write hier_compare dofile hier_compare.do -constraint -replace
```

Renaming Rules

When running the `WRITE HIER_COMPARE DOFILE` command, you might encounter warning messages related to name mismatches between the Golden and Revised designs. When you encounter this type of warning message, apply module renaming rules to resolve the mismatches between the Golden and Revised designs.

In the course of generating a hierarchical compare dofile, the Conformal software attempts to pair modules from the two designs based on the exact spelling of module names. However, when synthesis and the backend flow alter module names in the Revised design, the

Conformal software is unable to match some module pairs. For example, a module named `cpu` in the Golden design is renamed to `cpu_0` and `cpu_1` in the Revised design (often, this renaming is a result of the `UNIQUIFY` command in synthesis).

To map these modules, define a renaming rule as follows:

```
add renaming rule hier_rule1 "%s_%d$" "@1" -module -revised
```

Hierarchical Compare Dofile Execution

This section describes how to perform dynamic and static hierarchical comparison.

Dynamic Comparison

When a hierarchical compare dofile is successfully generated, you can perform dynamic comparison as follows:

```
run hier_compare hier_compare.do
```

If all of the situations presented in this chapter are applied, the final version of the dofile is:

```
read library lib.v -both
read design Golden.v -verilog -Golden
read design revised.v -verilog -revised
add pin constraint 0 scan_en -revised
add renaming rule hier_rule1 "%s_%d$" "@1" -module -revised
write hier_compare dofile hier_compare.do -constraint -replace
run hier_compare hier_compare.do
```

Static Comparison

When a hierarchical compare dofile is successfully generated, you can perform static comparison as follows:

```
dofile hier_compare.do
```

If all of the situations presented in this chapter are applied, the final version of the dofile is:

```
read library lib.v -both
read design Golden.v -verilog -Golden
read design revised.v -verilog -revised
add pin constraint 0 scan_en -revised
add renaming rule hier_rule1 "%s_%d$" "@1" -module -revised
write hier_compare dofile hier_compare.do -constraint -replace
dofile hier_compare.do
```

Hierarchical Comparison for Abort Resolution

Hierarchical comparison compares submodules in a bottom-up fashion. Each submodule comparison has lower complexity with respect to the complexity when comparing the entire design. Thus, this hierarchical comparison methodology is very helpful in resolving aborts. To minimize the complexity of each submodule comparison, you can maximize the number of submodules written out in a hierarchical dofile. Using the `UNIQUIFY` command before performing hierarchical comparison, you can remedy the incompatible instantiation warnings during hierarchical script generation and therefore maximize the number of modules included in the hierarchical dofile.

The following example shows how the `UNIQUIFY` command works with hierarchical comparison:

```
...
uniquify -all
write hier_compare dofile hier.do
run hier_compare hier.do
```

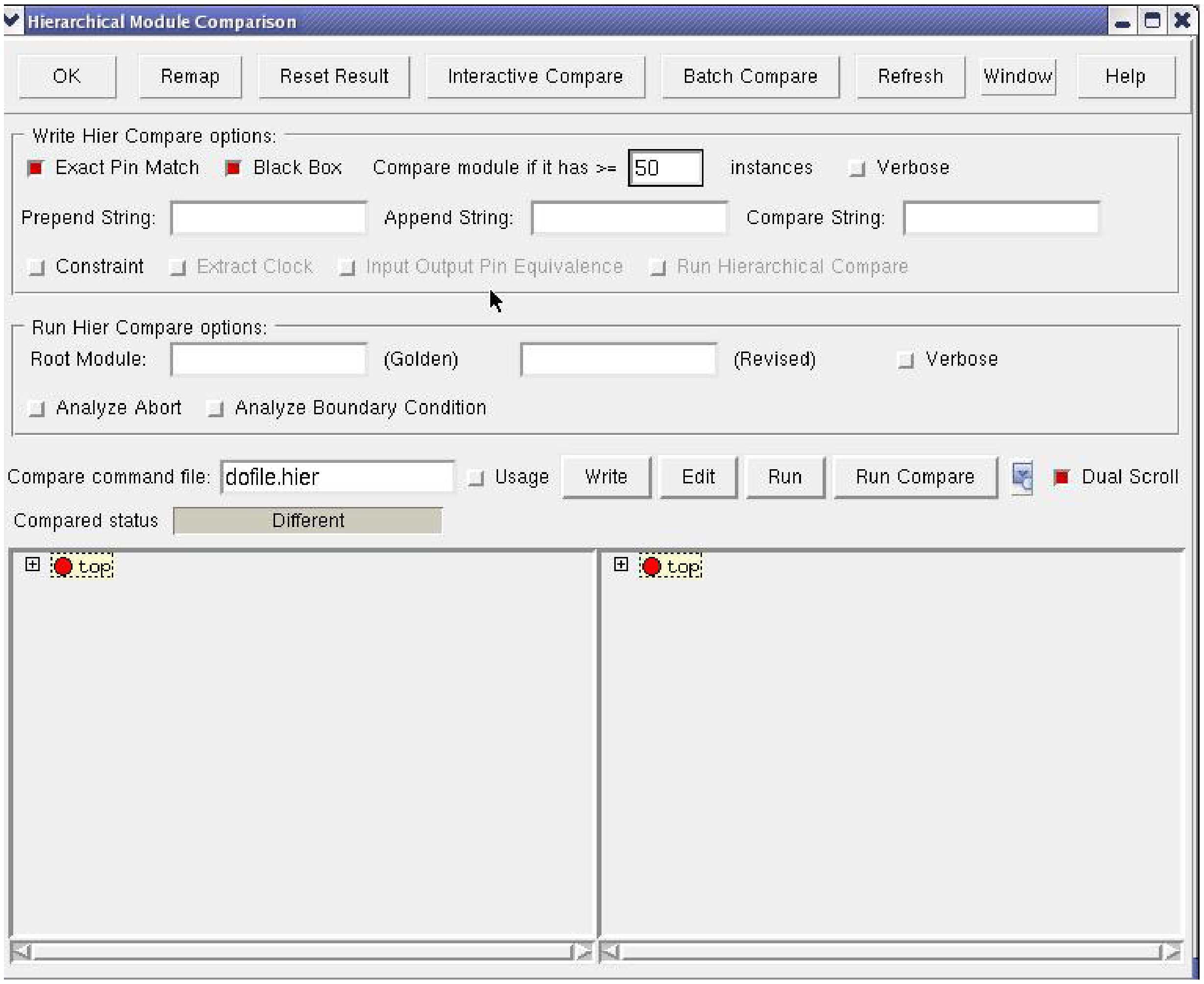
In the above command example, running the `UNIQUIFY -all` command makes all the hierarchical modules unique in the design. Then running the `WRITE HIER_COMPARE DOFILE` command creates a hierarchical dofile script named `hier.do` containing the compare script for the submodules and the root module. And finally, hierarchical comparison is executed using the `RUN HIER_COMPARE` command. Using `UNIQUIFY -all` allows you to maximize the number of submodules written out in the hierarchical dofile `hier.do`.

Hierarchical Module Comparison Window

Use the Hierarchical Module Comparison window to run a module-by-module, bottom-up, hierarchical comparison on two hierarchical designs.

Before using the Hierarchical Module Comparison window, modules must be paired through mapping. If modules are not paired, use the `ADD RENAMING RULE` command with the `-module` switch to remedy this situation. See Renaming Rules on page 239 for more information about renaming. After you have paired all modules, click the *Remap* button located on the menu bar in the main window.

- Choose *Tools – Hierarchical Compare*.



Write Hier Compare options:
Exact Pin Match [✓] Black Box [✓] Compare module if it has >= 50 instances Verbose
Prepend String: Append String: Compare String:
Constraint Extract Clock Input Output Pin Equivalence Run Hierarchical Compare

Run Hier Compare options:
Root Module: (Golden) (Revised) Verbose
Analyze Abort Analyze Boundary Condition

Compare command file: dofile.hier Usage Write Edit Run Run Compare Dual Scroll [✓]
Compared status: Different

The Hierarchical Module Comparison window includes two columns in the *Compare Status Display*. The left column is for the Golden design, and the right column is for the Revised design.

For the Hierarchical Module Comparison window, see the following for more information:

Hierarchical Module Comparison Fields and Options

Compare Options	Specifies your general hierarchical module comparison options. For more information, see Setting General Options on page 244.
Compare Command File	Area where you can write the hierarchical compare command file (dofile) and edit and execute it from this section. Also, search the *Compare Status* section for module names using the *Find* icon or press `Ctrl-f`.
Compare Status (Display)	This display area shows the module pairings. Scroll up or down to see all of the modules. Use the + and - icons to expand and compress the hierarchical display.

Setting General Options

Before Conformal can begin the hierarchical comparison, you must take several steps to ensure proper comparison. Follow the procedures for setting general hierarchical compare options.

1. Click *Exact Pin Match* to specify whether Conformal must compare modules that have the same number of pins. The default is to compare only modules that have the same number of pins.
2. Click *Constraint* to specify whether Conformal must propagate constraints and equivalences to lower-level modules for comparison. By default, Conformal does not propagate the constraints and equivalences.
3. Click *Black Box* to specify whether Conformal must blackbox the module after comparison. By default, Conformal blackboxes each module after comparison.
4. Change the number of instances in the *Compare module if it has >=* field to specify the minimum number of instances a module must have in order for Conformal to compare it (threshold). The default is 50 instances in a module.

   If a module name in the module display has (Small module) noted next to it, Conformal does not compare it hierarchically. To compare these small modules, change the threshold number to a smaller number and left-click *Remap* located on the toolbar at the top of the window.

Reporting CPU Use

To display the CPU time and memory use, click *Usage*. By default, *Usage* is off.

Working with Hierarchical Compare Dofiles

Writing the Hierarchical Compare Dofile to a File

After you have set the general options, fine-tune your comparison options by typing a dofile name in the *Compare command file* field and click the *Write* button.

Editing the Hierarchical Compare Dofile

To open the LEC File Editing window to edit the Hierarchical Compare Dofile, click *Edit* in the *Compare command file* section.

Running the Hierarchical Compare Dofile

To run a Hierarchical Compare Dofile, click *Run* in the *Compare command file* section.

Finding Module Names

1. Click the *Find* icon located on the right side of the *Compare Command File* section, or press `Ctrl-f` to open the Find: Hierarchical Module window.
2. Type any string or partial string of a module name in the *Find* field.
3. Click the *Find Forward* or *Find Backward* check box to specify the direction you want to proceed in the list.
4. Click the *Case Sensitive* check box, if applicable.
5. Click the *Find* button or press `Ctrl-f` to search for the name.

Deselecting the Dual Scroll Option

Use the *Dual Scroll* check box located in the *Compare command file* section to choose whether to scroll the windows individually or in tandem (dual). By default, the *Dual Scroll* button is selected.

Viewing a Module’s Compare Status

To view a module’s compare status, click a module to display the compare status in the *Compared status* field directly above the module lists.

Specifying Blackbox Modules

To specify modules that must be defined as blackboxes, click a module in the *Compare Status Display* to select it, then right-click to open the pop-up menu and choose *Add Black Box*. A blackbox symbol appears next to the module name.

Deleting Previously Added Blackbox Modules

To delete specified blackboxes from the design, click the module in the *Compare Status Display* to select it, then right-click to open the pop-up menu and choose *Delete Black Box* from the pop-up menu. Conformal removes the blackbox symbol that was next to the module name.

Ignoring Modules during Comparison

You can specify modules that will be ignored for hierarchical comparison because of cross-hierarchy optimization, click the module in the *Compare Status Display* to select it., then right-click to open the pop-up menu and choose *Add No-Black Box* from the pop-up menu. A no-blackbox symbol appears next to the module name.

Deleting No-Blackbox Status

To delete the no-blackbox symbols you added in the previous procedure, click the module in the *Compare Status Display* to select it, then right-click to open the pop-up menu and choose *Delete No-Black Box*. Conformal removes the no-blackbox symbol next to the module name.

Running a Hierarchical Comparison

After you complete the hierarchical comparison setup, run a hierarchical comparison on the entire hierarchy with one of the following methods:.

- Interactive Comparison

  When you click *Interactive Compare*, Conformal incorporates the compare options and blackbox and no-blackbox settings you specified with the previous procedures.

- Batch Comparison

  When you click *Batch Compare*, Conformal uses the hierarchical dofile that you can also write to a file and edit. Click *Batch Compare* (located on the menu bar at the top of the screen), or click *Run* in the *Compare command file* section.

Comparing Lower-Level Modules

1. Click the module in the module display list to select it.
2. Right-click and choose *Compare Sub-hierarchy* from the pop-up menu.

   After Conformal completes the hierarchical comparison, the module list displays a green circle preceding each “equivalent” module pair and a red circle preceding each “different” module pair.

Highlighting Non-Equivalent Modules

To highlight each non-equivalent module in the module list, click any module in the module list display to select it, then right-click to open the pop-up menu and choose *Next Module with Error*. The first non-equivalent module in the list is highlighted.

Deleting and Resetting Hierarchical Compare Results

To delete the hierarchical compare results and restart at any time, click *Reset Result* (located on the menu bar near the top of the window).

Specifying the Root Module

To set any lower-level module as the root module for batch comparison, click the module in the display list to select it, then right-click to open the pop-up menu and choose *Set Both Modules as Root*. The specified pair of modules become the root modules for the Golden and Revised designs.

11

Advanced Capabilities

Overview

Many of today's designs for video, graphics, and DSP require complex datapath structures with high performance. To satisfy those needs, major synthesis companies have developed tools that aim directly for datapath modules. Using a traditional equivalence checking method to verify such designs can often lead to abort points and excessive run times.

Conformal XL has features that can help address datapath-oriented designs.

Supported Datapath Structures and Optimizations

This section covers the Conformal XL-supported structures and optimizations.

Multipliers

Both Conformal L and Conformal XL support multipliers with standard architectures. However, Conformal XL also supports multipliers implemented with dynamic structures.

Standard Architectures

Conformal supports multipliers implemented with known, standard architectures such as CSA, RCA, NBW, WALL, and BKA.

Requirements

The multiplier module boundary must be kept and the product size must equal the combined size of the inputs, that is:

```
Product [N+M-1:0] = In1[N-1:0] * In2[M-1:0]
```

Procedure

While in LEC mode, analyze the multiplier using the `ANALYZE MULTIPLIER` command. This command requires the standard Conformal L licenses. Refer to the *Conformal Equivalence Checking Reference Manual* for detailed descriptions of the `ANALYZE MULTIPLIER` command and other related commands.

Multipliers with Dynamic Structures

Conformal XL supports multipliers that have been implemented with dynamic structures by datapath synthesis tools from Synopsys Module Compiler and Cadence RTL Compiler.

Requirements

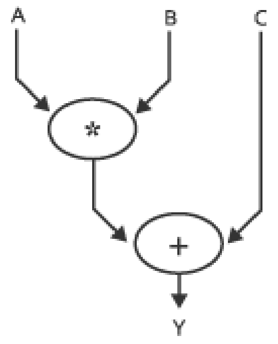
None

Procedure

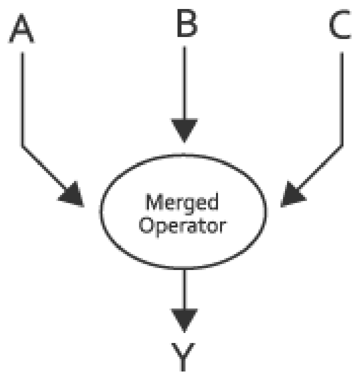
While in LEC mode, analyze the datapath using the `ANALYZE DATAPATH` command. You must have a Conformal XL license to use this command. Refer to the *Conformal Equivalence Checking Reference Manual* for detailed descriptions of the `ANALYZE DATAPATH` command and other related commands.

Operator Merging

Conformal XL supports datapath structures with operator merging, which is a method to implement a combination of two or more arithmetic operators. In the following figure, for example, an arithmetic expression `Y = A * B + C` is implemented with a multiplier and an adder using a standard synthesis tool.



However, when the intermediate `A*B` result is not required, the datapath synthesis tool (such as the `partition_dp` or `transform_csa` command from Design Compiler XL) can implement one merged operator for the entire arithmetic expression, as show the following figure:



Procedure

While in LEC mode, use the `ANALYZE DATAPATH` option. You must have a Conformal XL license to use this command. Refer to the *Conformal Equivalence Checking Reference Manual* for detailed descriptions of the `ANALYZE DATAPATH` command and other related commands.

Resource Sharing

The Conformal software can automatically solve long compare times or aborts caused by resource sharing with the `ANALYZE DATAPATH` and `SET DATAPATH OPTION` commands' `-SHARE` option, which applies the resource sharing technique on all multipliers with low datapath analysis quality. Datapath analysis is performed on new resources created after sharing.

Sequential Merge Optimization

Sequential merge handling typically reduces the number of sequential elements in the golden design such that it matches the number of sequential elements in the revised netlist, which is synthesized with sequential merge optimizations. With the `Conformal XL SET ANALYZE OPTION -AUTO` command, Conformal can automatically perform this analysis.

See "Sequential Merge Analysis" on page 270 for more information.

Module-Based Datapath Analysis

Module-Based Datapath (MDP) Analysis runs datapath analysis at a module level to help improve the quality of the operator-level analysis in an effort to reduce the number of potential abort points. This analysis is run in addition to and prior to the regular operator level analysis, and only on synthetic modules containing datapath operators in the Revised design netlist.

Note: For MDP analysis to be successful, the synthetic datapath module must be preserved in the netlist. Therefore, ungrouping and boundary optimization must be disabled during synthesis.

Datapath Module Abstraction

Datapath analysis has two parts:

1. Datapath Module Abstraction on the revised netlist
2. Datapath Operator Learning on the Golden netlist

They are applied by using the following commands in the dofile:

```
ANALYZE DATAPATH -MODULE [-RESOURCEFILE <file>] -verbose
ANALYZE DATAPATH [-WORDLEVEL] -verbose
```

The first command (datapath abstraction on the revised netlist) can help to improve the quality of second command (Datapath Operator Learning on the Golden netlist).

The second command (without `-MODULE` option) performs Datapath Operator Learning on the Golden RTL design to make it structurally similar to the revised netlist. See "Datapath Operator Learning" on page 258.

Datapath module abstraction on the revised netlist is applied by using the following command the dofile:

```
ANALYZE DATAPATH -MODULE [-RESOURCEFILE <file>] -verbose
```

The first command (with `-MODULE` option) performs datapath abstraction on the revised synthesis netlist. During synthesis, synthesis tools usually group several datapath operators into a module. The first command replaces the module's gate-level netlist with the corresponding RTL. As a result, the synthesis netlist is abstracted as RTL and it makes it easier to perform datapath learning on the Golden netlist.

In order for LEC to apply datapath abstraction on the revised netlist, it needs to follow the recommendations for the synthesis script. The generated netlist keeps the datapath module boundary for datapath abstraction.

```
// Synthesis script for RC:
read_hdl
elaborate
synthesize -to_mapped
write_hdl -lec > first_mapped.v
write_do_lec -revised first_mapped.v
synthesize -incremental
write_hdl > final.v
write_do_lec -Golden first_mapped.v -revised final.v
```

```
//Synthesis script for DC:
read_hdl
set compile_report_dp true
set compile_ultra_ungroup_dw false
compile_ultra -no_autoungroup -no_boundary_optimization \
      -no_seq_output_inversion
report_resource -hierarchy > resource.rpt
write -format verilog -hierarchy -output first_mapped.v
compile -incremental -map_effort high
write -format verilog -hierarchy -output final.v
```

Datapath Module Abstraction Reporting and Diagnosis

The following is an example of a datapath abstraction report generated by the `ANALYZE DATAPATH -MODULE` command. The synthesis netlist is generated by RC and the instance in the revised netlist has been analyzed in several iterations shown as pass 1 to 3. If the instance has the high quality evaluated, it will make the datapath learning on the Golden design easier.

```
// Command:
analyze datapath -module -verbose
// Analyzing modules in 'filename.v'
// Note: mod_csa_tree_231 : quality evaluated 100% success
// Note: mod_final_adder_234: quality evaluated 100% success
// Analyzing module instances (pass 1)
// Note: csa_tree_231 : quality evaluated 100% success
// Note: final_adder_234 : quality evaluated 100% success
// Analyzing module instances (pass 2)
// Note: final_adder_234 : quality evaluated 100% success
// Analyzing module instances (pass 3)
// Note: csa_tree_231 : quality evaluated 100% success
```

The following table outlines recommendations for resolving issues during the synthesis and verification process to ensure datapath abstraction can be applied successfully.

Table 11-1 Resolving Datapath Abstraction Issues

Scenario	Recommendation
Abstraction is not used prior to learning	Check dofile
Lack of resource file	Check synthesis script
Module cannot be found in the netlist	Check synthesis script
Module has no RTL definition	Check synthesis script
Module cannot be abstracted due to aborts	Isolate abort module

Handling Aborts in Datapath Module Abstraction

If the analysis quality evaluated is between 30% to 39% in the command `ANALYZE DATAPATH -MODULE`, it means that there are aborts while datapath abstraction is verifying the module's gate-level netlist with the RTL modeling. LEC can isolate these aborted datapath modules in the revised netlist by using the following command option.

```
ANALYZE DATAPATH -MODULE [-RESOURCEFILE <file>] -ISOLATE_ABORT_MODULE
```

This command abstracts the datapath modules from gate-level into RTL by assuming that these two models are functionally equivalent. This feature has several advantages:

- Accurately identifies the verification bottlenecks
- Prevents module abort effects from propagating to the top-level modules. As a result, you can verify remaining logics without being stuck at the aborted module.
- Once the remaining logic is verified, you can focus on the isolated abort module at any time by using more computing resources and algorithms with higher effort.

The following is an example of a report for the datapath abstraction with an abort module isolated. One datapath module has been isolated during datapath abstraction, and the comparison reports that the key points of the design are equivalent with the condition that one module in the revised netlist has been aborted and isolated.

```
// Command: analyze datapath -module -isolate_abort_module
// Note: i5/add_3200_DP_OP : quality evaluated 38% success
// Warning: i5/add_3200_DP_OP is isolated as an aborted instance.
// Command: compare
====================================================================
Compared points PO Total
--------------------------------------------------------------------
Equivalent 67 67
====================================================================
Compared results of isolated instances in Revised design (top)
====================================================================
Status Instance (Module)
--------------------------------------------------------------------
Abort i5/add_3200_DP_OP
      (mod_add_3200_DP_OP_0)
====================================================================
```

When there are aborts in datapath abstraction, you can also use the following command to generate the test case and report to LEC supports.

```
REPORT TESTCASE -DATAPATH_MODULE -QUALITY 70 -file <filename>
```

Datapath Operator Learning

Datapath operator learning on the revised netlist is applied by using the following command the dofile:

```
ANALYZE DATAPATH [-WORDLEVEL] -verbose
```

Datapath analysis in LEC has been enhanced for advanced adder tree clusters, complex multiplier architectures (such as product-of-sum multipliers), and XOR trees.

Datapath learning, through the `ANALYZE DATAPATH` command, makes the Golden datapath operators structurally similar to the revised netlist so that comparing the two designs is easier.

The `-WORDLEVEL` option applies additional analysis that handles advanced datapath optimizations in the following areas:

- Complex adder tree clustering: Supports clustering multiple adder trees that share common sub-expressions, and whose input operand has least-significant-bit (LSB) truncation.
- Product-of-sum optimizations: Supports optimization of merging the cluster adder with the multiplier.
- Associative Law on multipliers: Supports reordering of the cascaded multipliers based on associative law.
- XOR tree optimizations: Supports optimization of reordering XOR tree.

- Constant multipliers and adder trees optimizations: Supports adder tree optimizations with input constraints.

You can use the `-WORDLEVEL` option with the `SET DATAPATH OPTION` and `ANALYZE DATAPATH` commands.

The following is an example of the datapath learning report generated by the `ANALYZE DATAPATH -WORDLEVEL` command:

```
// Command: analyze datapath -wordlevel -verbose
// Note: mult_12: quality evaluated 85% success
// Note: mult_18(pos): quality evaluated 90% success
// Note: add_602_2(clustered): quality evaluated 85% success
```

The command analyzes the Golden design's datapath operators and merges the cluster adders with the product-of-sum operators.

When the quality of datapath learning is low, you can improve the quality of datapath learning by using

- Datapath abstraction on the revised netlist. See "Datapath Module Abstraction" on page 255.
- Increased datapath learning effort
- The `-WORDLEVEL` option

  Note: The datapath operators can only be learned once with the `-WORDLEVEL` option. Thus, when you switch to this option, you have to relearn the datapath by changing the dofile and running it from the beginning.

The following table outlines some recommended methods for resolving datapath aborts, depending on the given scenario.

Table 11-2 Resolving Datapath Aborts

Scenario	Recommendation
Datapath abstraction is not applied in dofile	`analyze datapath -module`
Resource sharing is applied in synthesis netlist	Functional partition
Don't care space exists	Recode RTL to add CUT points
XOR tree is in RTL	`analyze datapath -wordlevel`

Product-of-sum is in RTL	`analyze datapath -wordlevel`
Complex clustering adders are in RTL	`analyze datapath -wordlevel`

DC Synthesis Flow

In MDP Analysis, there is one intermediate netlist between the RTL code and the final netlist that divide the flow into two comparisons:

1. RTL to intermediate.

   See Dofile Example for Intermediate Netlists on page 263.

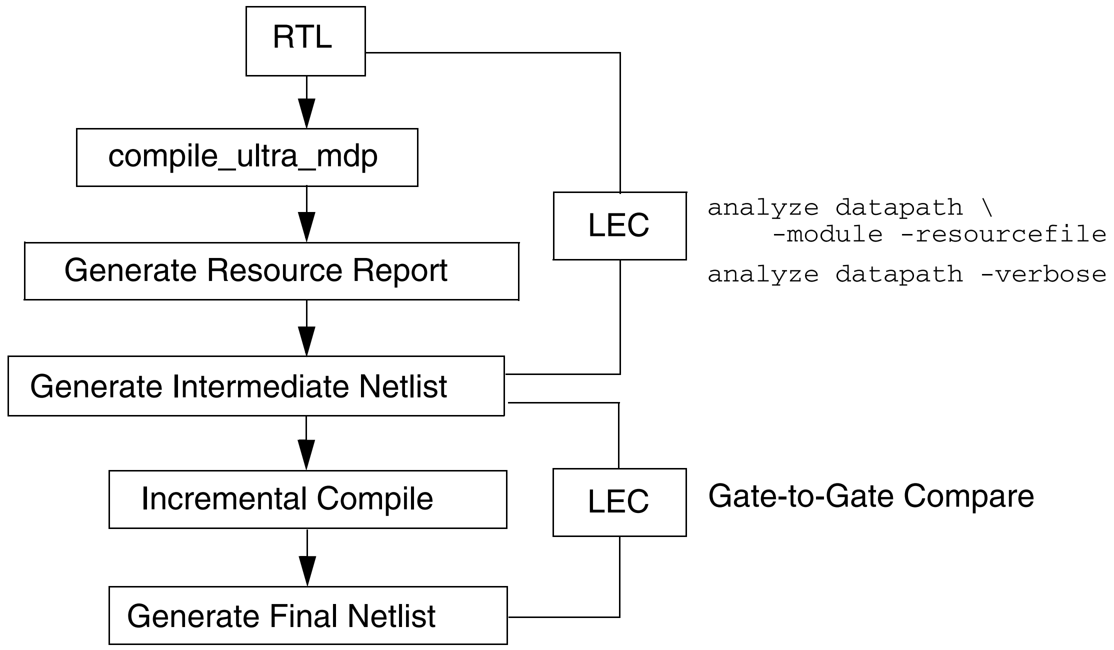
2. Intermediate to final

   See Dofile Example for Intermediate to Final Netlist on page 264.

This flow is applicable to both Design Compiler (DC) synthesis and RTL Compiler (RC) synthesis; however, this section is dedicated to the DC synthesis flow.

The RC synthesis flow is more automated than the DC synthesis flow, where the intermediate netlist and Conformal dofile are generated automatically with the `write_hdl -lec` and `write_do_lec` RC commands. For more information, see the “Interfacing with Conformal Equivalence Checker” chapter of the *Interfacing Between RTL Compiler and Conformal* guide (this document is available within the RTL Compiler document set).

The following shows the DC synthesis flow:



```
analyze datapath \
    -module -resourcefile
analyze datapath -verbose
```

The following shows the steps with examples:

1. Source the `mdp.tcl` script.

   ```
   source mdp.tcl
   ```

   where `mdp.tcl` is located at:

   ```
   <release_path>/share/cfm/lec/scripts/mdp.tcl
   ```

   For more information on the DC commands that are included in the `mdp.tcl` script, see DC Commands on page 262.

2. Run the `compile_ultra_mdp` command with the effort level (0, 1, 2, 3, or 4) and the design module, which is the name of the top module that is synthesized.

   ```
   compile_ultra_mdp 4 top1
   ```

   Note: Effort level 0 is a pass-through effort where there are no option changes to the synthesis script.

   For more information on the effort levels, see MDP Effort Levels on page 263.

3. Run the `compile_ultra` command with the `-incremental` option to allow incremental compilation.

   ```
   compile_ultra -incremental
   ```

   Note: If this option is not used after `compile_ultra_mdp`, new mapping or implementation selection can be done to change original results

4. Continue running original DC synthesis script commands.
5. Run `report_qor` to get final QoR results.

   ```
   report_qor
   ```

6. Exit the software to get final results.

   ```
   quit
   ```

Sample DC Script

```
read_vhdl {../src/address.vhd ../src/cfft1024X12.vhd ../src/cfft.vhd \
  ../src/mulfactor.vhd ../src/p2r_cordic.vh d ../src/sc_corproc.vhd \
  ../src/blockdram.vhd ../src/cfft4.vhd ../src/div4limit.vhd \
  ../src/p2r_CordicPipe.vhd . \
  ./src/rofactor.vhd}
source ../../script/mdp.tcl
compile_ultra_mdp 4 revised1
compile -incremental -map_effort high
write -format verilog -hierarchy -output revised4.2.v
report_qor
quit
```

DC Commands

This section describes the DC commands that are included in the `mdp.tcl` script.

- `set compile_ultra_ungroup_dw false`

  By default, all DesignWare hierarchies are unconditionally ungrouped in the second pass of the compile. The default is `true`.

- `compile_ultra`
  - `-no_autoungroup`

    Turns off automatic ungrouping. By default, the `compile_ultra` command performs delay-based auto-ungrouping. It ungroups hierarchies along the critical path and is used essentially for timing optimization.

  - `-no_boundary_optimization`

    Turns off boundary optimization. By default, the `compile_ultra` command optimizes across hierarchical boundaries. Boundary optimization is a strategy that can improve a hierarchical design by allowing the compile process to modify the port interface of lower-level designs.

  - `-no_seq_output_inversion`

    Does not allow sequential elements to have their output phase inverted.

- `-report_resource hierarchy`

  Displays information about the resource implementation

MDP Effort Levels

The following describes the options that are included in effort levels 1 through 4.

Options	**1**	**2**	**3**	**4***
Preserves DesignWare Hierarchy	YES	YES	YES	YES
Boundary Optimization	YES	NO	NO	NO
Sequential Output Inversion	YES	YES	NO	NO
Preserves Design Module Hierarchy	NO	NO	NO	YES

*Cadence recommends using an effort level of 4 when possible.

Dofile Example for Intermediate Netlists

The following Conformal dofile hierarchically compares the RTL and the intermediate netlist using the resource report file for MDP analysis.

The `SET ANALYZE OPTION -auto` command invokes both `ANALYZE SETUP` and `ANALYZE ABORT -compare` commands. The first `ANALYZE DATAPATH -module` command starts MDP analysis. The second `ANALYZE DATAPATH` is required to complete the datapath analysis.

```
read design <rtl_files> -Golden
read design intermediate_gate.v -revised
report design data
report black box
..
set analyze option -auto
write hier_compare dofile hier.do -replace -usage -constraint -noexact \
    -prepend_string "report design data; analyze datapath -module -resourcefile \
    resource.rpt -verbose; analyze datapath -verbose"
run hier_compare hier.do
usage
```

The following shows the output, where the synthetic modules `add_*` have been analyzed:

```
LEC> analyze datapath -resourcefile resource.rpt -module -verbose
```

```
// Note: add_16_DP_OP_6: quality evaluated 100% success
// Note: add_14_DP_OP_7: quality evaluated 100% success
// Note: add_19_DP_OP_8: quality evaluated 100% success
```

In this output, each synthetic module contains several individual datapath operators. If this had high evaluation quality, the operator-level datapath analysis would have higher quality results, while low evaluation quality has no impact on the operator-level datapath analysis.

Dofile Example for Intermediate to Final Netlist

The following Conformal dofile compares the intermediate netlist and the final netlist:

```
read design intermediate_gate.v -Golden
read design find_gate.v gate.v -revised
...
set system mode lec
add compare point -all
compare
...
```

Extracting Testcases for Datapath Modules

Note: This feature is only applicable to Design Compiler (DC) synthesis.

MDP analysis results sometimes have low evaluation quality when the datapath module is too complex. You can automatically extract a testcase with only the information of the failed modules into a single compact testcase file. The testcase file, in XML format, contains a netlist of the datapath module, a netlist of the gate-level implementation (from the Revised design’s netlist), and other attributes associated with the datapath module, such as resource information. The extracted netlists consist of primitive gates and do not contain direct information on the library.

There are two ways to extract the testcase:

1. Specify the instance name of the datapath modules.

   For example, the following will report testcases on datapath modules whose instance name starts with “`add`“ into the file `add.xml` under the directory `LEC_testcase`:

   ```
   report testcase -datapath_module -inst_name add* -file add.xml
     -dir_name LEC_testcase -replace
   // Note: datapath module "add_16_DP_OP_6" reported.
   // Note: datapath module "add_14_DP_OP_7" reported.
   // Note: datapath module "add_19_DP_OP_8" reported.
   // Note: add.xml generated into directory LEC_testcase.
   // Note: Testcase is extracted into directory LEC_testcase.
   ```

2. Specify a number that those datapath modules whose evaluated quality less than or equal to the specified number will be reported.

   For example, the following will report testcase on datapath modules whose evaluated quality are less than or equal to 38% into the file `low_quality.xml` under the directory `LEC_testcase`.

   ```
   LEC > report testcase -datapath_module -quality 38 -file low_quality.xml
     -dir_name LEC_testcase -replace
   // Note: datapath module "add_9399_S2_DP_OP_314_2331_11" reported.
   // Note: low_quality.xml generated into directory LEC_testcase.
   // Note: Testcase is extracted into directory LEC_testcase.
   ```

   To enable testcase extraction of datapath modules in the hierarchical flow, you can use the following command:

   ```
   write hier_compare dofile -prepend_string

   report testcase -datapath_module -quality 38 -file low_quality.xml -dir LEC_testcase -append
   ```

   Note: Use the `REPORT TESTCASE` command's `-append` option instead of `-replace` to avoid overwriting the same file. The `-append` option will prepend the current root module name to the `low_quality.xml` file

Recreating Testcases for Datapath Modules

Note: This feature is only applicable to Design Compiler (DC) synthesis.

The extracted testcase file contains the information of reported datapath modules. You can reproduce the problem of failed modules in original design. The embedded information in the testcase file is separated into files under the specified directory, along with one generated dofile. Running the generated dofile can reproduce the problem of failed modules in original design.

The following example reproduces the problem with the files:

- `testcase.rpt`: resource information of the failed modules.
- `testcase.v`: netlist of the failed modules. Note that the constraints associated with the datapath modules will be annotated into the netlist.
- `testcase.do`: the generated dofile. You can run this dofile to reproduce the problem.

Isolating Aborted Datapath Modules

Note: This feature is applicable to both Design Compiler (DC) or RTL Compiler (RC) synthesis and requires a Conformal XL license.

The Conformal software sometimes has abort points when comparing designs with complicated datapaths. In such circumstances, you might choose to accept those abort points as the final result. While you can accept a certain level of risks associated with datapaths not being formally compared, you should make sure the rest of the designs are fully compared.

You can isolate the aborted datapath modules and allow the comparison to proceed without the aborted datapath modules. The aborted datapath module will be reported along with the final comparison results.

For example, without isolating aborted datapath modules, the report might look like the following:

```
// Command: analyze datapath -module -verbose -resourcefile resourcefile.rpt
// Note: add_5822_DP_OP_308_4437_42 : quality evaluated 100% success
// Note: add_5821_DP_OP_306_4437_41 : quality evaluated 100% success
// Note: add_1055_159_DP_OP_311_2879_8 : quality evaluated 99% success
// Note: add_5823_DP_OP_310_4437_43 : quality evaluated 100% success
// Note: add_5820_DP_OP_304_4437_40 : quality evaluated 100% success
// Note: add_9399_S2_DP_OP_314_2331_11 : quality evaluated 38% success
...
==============================================================================
Compared points      PO        Total
------------------------------------------------------------------------------
Equivalent           64        64
------------------------------------------------------------------------------
Abort                3         3
==============================================================================
```

When isolating the aborted datapath modules, the report looks like the following:

```
// Command: analyze datapath -module -verbose -resourcefile resourcefile.rpt \
  -isolate_abort_module
// Note: add_5822_DP_OP_308_4437_42 : quality evaluated 100% success
// Note: add_5821_DP_OP_306_4437_41 : quality evaluated 100% success
// Note: add_1055_159_DP_OP_311_2879_8 : quality evaluated 99% success
// Note: add_5823_DP_OP_310_4437_43 : quality evaluated 100% success
// Note: add_5820_DP_OP_304_4437_40 : quality evaluated 100% success
// Note: add_9399_S2_DP_OP_314_2331_11 : quality evaluated 38% success
// Warning: add_9399_S2_DP_OP_314_2331_11 is isolated as an aborted instance.
...
==============================================================================
Compared points      PO        Total
------------------------------------------------------------------------------
Equivalent           67        67
==============================================================================
Compared results of isolated instances in Revised design (top)
==============================================================================
Status          Instance (Module)
--------------------------------------------------------------------------------
Abort           i5/add_9399_S2_DP_OP_314_2331_11
                (NV_GR_PE_STRI_core_add_9399_S2_DP_OP_314_2331_0)
================================================================================
```

Note: This feature does not help resolve aborts. It reduces the report to make it easier to show where the abort points lie. Instead of showing several abort key points, it shows all EQ key points and shows only the abort `DP_OP` module. This type of reporting allows you to make a more informed decision about the risks associated with reported abort modules

You can specify the module to be isolated using either of the following commands:

```
analyze datapath -module -isolate_abort_module
set datapath option -module -isolate_abort_module
```

During MDP analysis, the aborted modules are extracted into RTL. The software reports all isolated modules as part of the final comparison results.

To enable isolation of aborted modules in the hierarchical flow, you can use the following commands:

```
write hier_compare dofile -prepend_string \
  "analyze datapath -module -isolate_abort_module"
analyze datapath -module -isolate_abort_module
```

Or set the following option before running the `RUN HIER_COMPARE` command:

```
set datapath option -module -isolate_abort_module
```

During hierarchical comparison, the software automatically isolates the aborted datapath module for each root in the synthesis netlist. Run the `REPORT HIER_COMPARE RESULTS` command to report the isolated module.

Word-Level Datapath Analysis

Datapath analysis in LEC has been enhanced for advanced adder tree clusters, complex multiplier architectures (such as product-of-sum multipliers), and XOR trees.

Datapath learning, through the `ANALYZE DATAPATH` command, makes the Golden datapath operators structurally similar to the revised netlist so that comparing the two designs is easier.

Datapath Learning

The `-WORDLEVEL` option applies additional analysis that handles advanced datapath optimizations in the following areas:

- Complex adder tree clustering: Supports clustering multiple adder trees that share common sub-expressions, and whose input operand has least-significant-bit (LSB) truncation.
- Product-of-sum optimizations: Supports optimization of merging the cluster adder with the multiplier.
- Associative Law on multipliers: Supports reordering of the cascaded multipliers based on associative law.
- XOR tree optimizations: Supports optimization of reordering XOR tree.
- Constant multipliers and adder trees optimizations: Supports adder tree optimizations with input constraints.

You can use the `-WORDLEVEL` option with the `SET DATAPATH OPTION` or `ANALYZE DATAPATH` commands.

Reporting and Diagnosis of Datapath Analysis

The following is an example of the datapath learning report generated by the `ANALYZE DATAPATH -WORDLEVEL` command:

```
// Command: analyze datapath -wordlevel -verbose
// Note: mult_12: quality evaluated 85% success
// Note: mult_18(pos): quality evaluated 90% success
// Note: add_602_2(clustered): quality evaluated 85% success
```

The command analyzes the Golden design's datapath operators and merges the cluster adders with the product-of-sum operators.

When the quality of datapath learning is low, you can improve the quality of datapath learning by using

- Datapath abstraction on the revised netlist.

  ```
  analyze datapath -module -reseourcefile <filename>
  ```

- Increased datapath learning effort

  ```
  analyze datapath -effort high
  ```

- The `-WORDLEVEL` option

  ```
  analyze datapath -wordlevel
  ```

  Note: The datapath operators can only be learned once with the `-WORDLEVEL` option. Thus, when you switch to this option, you have to relearn the datapath by changing the dofile and running it from the beginning.

The following table outlines some recommended methods for resolving datapath aborts, depending on the given scenario.

Table 11-3 Resolving Datapath Aborts

Scenario	Recommendation
Datapath abstraction is not applied in dofile	For a DC netlist: `analyze datapath -module -resourcefile \ resource.f` For an RC netlist: `analyze datapath -module`
Resource sharing is applied in synthesis netlist	Functional partition: `add partition points` (For more information and the complete syntax, use the `MAN ADD PARTITION POINTS` mmand)
Don't care space exists	Recode RTL to add CUT points
XOR tree is in RTL	`analyze datapath -wordlevel`
Product-of-sum is in RTL	`analyze datapath -wordlevel`
Complex clustering adders are in RTL	`analyze datapath -wordlevel`

Sequential Merge Analysis

Sequential merge handling typically reduces the number of sequential elements in the golden design such that it matches the number of sequential elements in the revised netlist, which is synthesized with sequential merge optimizations.

Conformal includes automatic sequential merge analysis and diagnostic capabilities for sequential merge.

Sequential Merge Flow

Use the following commands to enable sequential merge analysis:

```
SET ANALYZE OPTION -AUTO
ANALYZE SETUP
```

For more information on these commands, refer to the Conformal Equivalence Checking Reference or use the `MAN` command.

Synthesis Requirements

DC Synthesis Flow

In the DC synthesis flow, you must turn off inverted sequential merging so that Conformal can correctly verify the design. Or, you must specify the phase information using the `ADD INSTANCE EQUIVALENCE -INVERT` command.

RC Synthesis Flow

In RC synthesis flow, you must use the `-lec` option when writing out the netlist:

```
write_hdl -lec
```

This enables sequential merge annotation, where merges are specified as comments in the netlist. For example:

```
// synthesis_merge
// merged rep i1/q1_reg
//    merge + i1/q2_reg
// merged rep i2/q1_reg
//    merge + i2/q2_reg
```

Conformal uses this information during sequential merge analysis. For example, when a netlist with such annotations is set as the revised design, LEC automatically reads in the list of sequential elements, applies the appropriate sequential merges, and displays the following modeling message:

```
// (F21) Merged 123 DFF/DLAT(s) due to added instance equivalences
```

Sequential Merge Verification

To verify the correctness of sequential merges, use the `ADD COMPARED POINTS -ALL` command. The following is an example of the proof results displayed during the compare process.

For example, when a group of four DFFs is merged into a single DFF, this is called a *representative* or a merge group, LEC will first perform a single design mergeability proof and validate whether all four DFFs can be merged together without causing conflict.

When the functions of merged registers contain don't-care (DC) conditions, the mergeability relationship is no longer transitive. When this happens, LEC will perform a sufficient number of proofs for the merge group. As a result, the number of reported "Compared points" may exceed the actual number of merged DFFs. The proof result is reported as follows:

```
Compare results of instance/output/pin equivalences and/or seq_merge
============================================================
Compared points      DFF       Total
------------------------------------------------------------
Equivalent           4         4
============================================================
```

When the merge representative contains a don't-care condition, the sequential merge could enlarge the permissible functional space, thereby masking any synthesis errors. In this case, LEC creates additional compare points, one for each merged DFF. These special compare points, called *merged compare points*, are compared against the representative's mapped point in the revised design.

A new comparison result section called "Compare results of merged compare points" for these merged compare points is reported after a comparison, as follows:

```
Compare results of merged compare points
====================================================================
Compared points      DFF       Total
--------------------------------------------------------------------
Equivalent           4         4
====================================================================
```

Notice that the comparison result of the representative will be reported in the regular comparison data.

Setting the Effort Level

With automatic sequential analysis, registers must be proven equivalent before they can be merged. During the modeling process, the proof effort is limited and can overlook some sequential merges, causing mismatches. If the `COMPARE` command reports mismatches, try `ANALYZE SETUP` with a higher effort (like high or ultra) after the `COMPARE` command:

```
ANALYZE SETUP -EFFORT HIGH
ANALYZE SETUP -EFFORT ULTRA
```

Diagnosing Instance/Sequential Merge Nonequivalence

Use the `DIAGNOSE -MERge` command to debug instance/sequentially merged compare points.

Note: The following options are not supported when diagnosing instance/sequential merge nonequivalence:

```
-SUMmary integer
-SOrt
-SUPport
```

For example, the compare result contains 3 sections:

```
================================================================
Compared points      PO     DFF       Total
----------------------------------------------------------------
Equivalent           1      3         4
================================================================
Compare results of merged compare points
================================================================
Compared points      DFF       Total
----------------------------------------------------------------
Non-equivalent       1         1
================================================================
Compare results of instance/output/pin equivalences and/or sequential merge
================================================================
Compared points      DFF       Total
----------------------------------------------------------------
Non-equivalent       1         1
================================================================
```

The second and third sections are comparison results for sequential merge and the `ADD INSTANCE` and `ADD PIN EQUIVALENCE` commands.

To diagnose nonequivalence shown in section `Compare results of merged compare points`, you must specify two compare points (the first in golden design and the second in revised design):

```
diagnose -merge 6 10
```

To diagnose nonequivalence shown in section `Compare results of instance/output/pin equivalences and/or sequential merge`, you must specify two compare points in the same design:

```
diagnose -revised 9 10
```

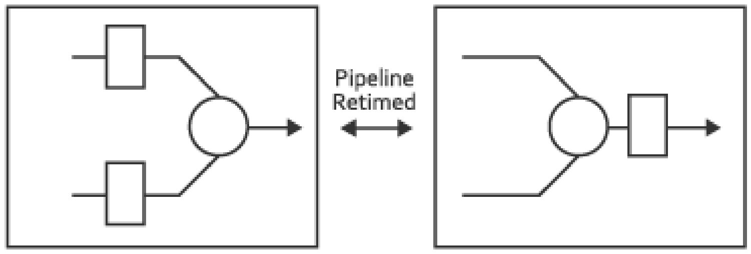
Note: You can use the `-golden` and `-revised` options to specify the design type.

To view the compare points information, use the `REPORT COMPARE DATA` command.

Retiming

Basic Pipeline Retiming

The Conformal software supports basic *pipeline retiming*, which is a method that retimes pipeline stages in an otherwise purely combinational design. Many times, the RTL models the pipeline by putting all of the registers at the output of the module, while the gate netlist is implemented with real pipeline, as shown in the following figure:



Guidelines

For basic pipeline retiming:

- All data must move from register stage to register stage
- All paths must have the same number of stages
- All pipeline registers must have the same clock
- Sequential loops or latches are prohibited
- Asynchronous set and reset should be disabled
- Stalls should be disabled

Procedure

To use basic pipeline retiming, use the ADD MODULE ATTRIBUTE `-pipeline_retime` command while in Setup mode (as shown in the following transcript):

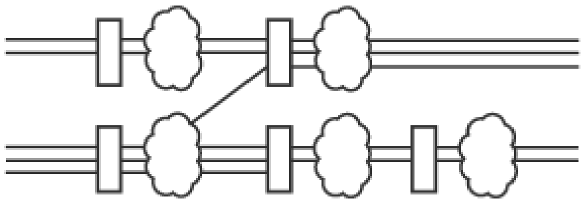
```
.
.
.
add module attribute ocd_cs_r4c -pipeline_retime -Golden
.
.
```

```
.
// Modeling Golden…
// Pipeline-retimed 50 DFF(s) as 51 DFF(s) in 3 stage
// Modeling Revised…
// Pipeline-retimed 59 DFF(s) as 51 DFF(s) in 3 stage
```

Advanced Pipeline Retiming

Conformal XL can handle *advanced pipeline retiming* circuits, where each part of the module can have different number of stages, as shown in the following figure.

Note: All of the vertical bars represent flip-flops or registers, and clouds represent combinational logic.



Unlike basic pipeline retiming, described in Basic Pipeline Retiming on page 274, advanced pipeline retiming supports:

- Some forms of set and reset
- Sequential feedback
- MUX enable, but not MUX stall
- Latches that are not retimed

Procedure

To use advanced pipeline retiming in Conformal XL, use the `ANALYZE RETIMING` command in LEC mode (as shown in the following transcript):

```
.
.
.
SETUP> set system mode lec -nomap
// Command: set system mode lec -nomap
LEC> analyze retiming
// Command: analyze retiming
LEC> map key points
// Command: map key points
// Mapping key points ...
================================================================================
Mapped points: SYSTEM class
--------------------------------------------------------------------------------
Mapped points     PI      PO      DFF     Z         Total
--------------------------------------------------------------------------------
Golden            5       4       4       1         14
--------------------------------------------------------------------------------
Revised           5       4       4       1         14
================================================================================
LEC> add compare points -all
// Command: add compare points -all
// 8 compared points added to compare list
LEC> compare
// Command: compare
================================================================================
Compared points      PO      DFF       Total
--------------------------------------------------------------------------------
Equivalent           4       4         8
================================================================================
```

Pipeline Retiming on a List of Specified Registers

By default, pipeline retiming moves all registers forward to the primary output side of the design as much as possible. If a list of registers is specified after the pipeline retiming option, retiming will only be performed on specified registers. Any register not in the list will not be pipeline retimed.

Run the `ANALYZE RETIMING -pipeline <identifier* ...>` command to specify a selected set of registers for pipeline retiming.

Pipeline Backward Retiming

Run the `ANALYZE RETIMING -pipeline -backward` command on the retimed design to move registers backward to the primary input side of the design as much as possible. As with pipeline forward retiming (the default), you can run pipeline backward retiming on either all registers or a selected set of registers.

Forward and backward pipeline retiming together provide the mechanism and the flexibility to debug retimed designs by selectively moving only a subset of the registers in the desired directions. You can run this multiple times with different set of registers and options, thus allowing the manual refinement of retiming steps.

Merging Equivalent Registers

Run the `ANALYZE RETIMING -merge` command on the retimed design to combine equivalent registers after registers are moved. This includes inverted-equivalent registers.

This can help to reduce unmapped register key points and the resulting false non-equivalences.

Note: This is on by default when running the `ANALYZE RETIMING` command.

Retiming Diagnosis

Run the `ANALYZE RETIMING -diagnosis <identifier>` command to check if the specified register can be retimed a step forward, or step backward to its fanout or fan-in gates, respectively. If the retime movement cannot succeed, the reason for the failure is reported. This option will not change the netlist—it only provides information about the specified retiming step.

Flattened Retiming Analysis

When comparing flattened designs with retiming modules, register moves are delimited by the retiming module boundaries. Specifically, for registers within the retiming modules, the movement of the registers by pipeline retiming is confined to the original retiming modules.

Registers that are outside the retiming modules will not be moved to the part of the design that has been subject to retiming. You can specify retiming modules by running the `ADD MODULE ATTRIBUTE -pipeline_retime` command.

Multithreading Process

Multithreaded comparison is best suited for large gate-to-gate comparisons, where the comparison can be distributed to multiple comparison threads. To possibly resolve more abort points and reduce the time spent on RTL-to-gate comparisons, the parallel analyze abort feature might be more effective.

The easiest method to invoke multithreaded comparisons is to use the `COMPARE` command's `-threads` option to specify the number of threads. For example, the following command starts the comparison using four parallel threads:

```
compare -threads 4
```

When a comparison is multithreaded, the number of executing threads are updated along with the comparison results. For example,

```
//  26% Comparing 4 out of 15 points, 0 Non-equivalent, 4 Threads
```

Multithreading Model

The multithreading model differs from other models in that the additional threads of processing are always run on the current machine instead of being launched through a server farm. This eliminates the complexity of setting up the multithreaded environment and instead uses multi-core, multi-CPU machines in a computing farm.

During a program's execution, the elapsed time, or wall-time, is the time measured using a wall clock (real time), whereas the CPU time is the amount of time that a processor used to work on the process. The goal of multithreading is to reduce the elapsed time, which does not necessarily reduce the total CPU time. Therefore, you can collect runtime statistics for multithreading features using the following command:

```
usage -elapse
```

The software spawns additional processes during multithreaded comparison. You can use the `ps` or `top` shell commands to show how many comparisons are running. The Conformal software also imposes a version match check between main software program and spawned software processes to maintain consistency.

Enabling Multithreaded Processing

Multithreaded processing is enabled when the `-threads` option is set to a value greater than 0, and it is disabled when the value is set to 0 (the default setting). Although it is valid to set the number of threads to 1, it might not yield any advantage over a non multithreaded run.

You can specify the number of threads in three different ways as follows, with decreasing order of precedence:

Note: You can only specify a maximum of 16 threads.

1. At the command. For example:
   ```
   compare -threads 4
   ```
2. As a global option to the command. For example:
   ```
   set compare options -threads 4
   ```
3. As a global option for all multithreaded comparison features. For example
   ```
   set parallel option -threads 4
   ```

For example, with either of the following commands, all subsequent `COMPARE` commands run with two computing threads:

```
set parallel option -threads 2
set compare option -threads 2
```

However, you can override this by running the following command to compare without multithreading:

```
compare -threads 0
```

Or you can compare with more computing threads using the following command:

```
compare -threads 4
```

Similarly, if you use the following sequence, all subsequent `COMPARE` commands will use two computing threads because the second command setting supersedes the first command setting:

```
set parallel option -threads 3
set compare options -threads 2
```

Setting Comparison Effort Levels

Use the `SET COMPARE EFFORT` command to set the effort level before running the `COMPARE -threads` command. For example, to applies greater effort to equivalency checking for each gate prior to running the multithreaded processing:

```
set compare effort medium
compare -threads 4
```

Setting Comparison Options

You can only run multithreaded comparison with the `COMPARE` command's `-noneq_stop`, `-abort_stop`, and `-gate_to_gate` options. Other `COMPARE` options will be ignored.

Number of Threads Recommendation

The wall-time can be estimated using the following equation:

$$\text{wall-time} = \frac{(\text{CPU-time + \#-of-threads x overhead})}{\text{MIN (\#-of-threads, \#-of-available-processors)}}$$

where `MIN()` returns the minimum of the number of threads and the number of available processors, and `overhead` is a constant factor incurred for each thread, respectively. The `SET PARALLEL OPTION -info` command shows you two types of overheads: spawned process latency and parallel compare overhead. The values displayed are the sum up overheads for all the threads, so you can roughly use the sum of these two values as '#-of-threads x overhead' as in the previous wall-time equation.

Example:

```
// Command: set parallel option -info
==========================================================================
Parallel processing information:
--------------------------------------------------------------------------
...
Spawned processes latency     : 10.47   sec
Parallel compare overhead     : 24.33   sec
...
==========================================================================
```

Because the wall-time speedup is bounded by the number of available processors, Cadence recommends that the number of threads be the same as the number of available processors (cores).

Running Jobs on Server Farms

Because wall-time speed up can be run only when there are multiple processors available, multiple cores must be reserved when submitting a parallel comparison job to a server farm. For example:

```
bsub -n 4 -R 'span[hosts=1]' lec
```

Note: Specifying too large an number (`-n`) could delay or prevent the job from being executed on the server farm depending on the resources and setup of the server farm.

You can configure the limit of the number of processes for the whole job with option `-p`. The default is no limit. Exceeding the limit causes the job to terminate. For example, the following sets the process limit to be 1:

```
bsub -p 1 lec
```

Tip

Make sure this option is not set or the number is set to be greater than the number of computing threads of parallel comparison.

Licensing Requirements

Each additional computing thread requires one additional license. These additional licenses are released when the multithreaded comparison is completed.

The multithreaded processing program supports one computing thread with no additional licensing requirement. If the software starts with a GXL license and comparison is performed with three threads (`compare -threads 3`), the licenses used are one GXL when the software starts and two additional XL when the parallel comparison starts, equaling three threads.

Note: Make sure to have enough licenses for multithreaded comparison. If you specify three threads (`compare -threads 3`), and only one license is available at that time, the software errors out and falls back to serial mode.

By default, the software attempts to check out additional XL licenses. If this fails, it attempts to check out the licenses with which the software started. For example, if the software starts with LP licenses, and comparison is performed with three threads, the software first attempts to check out two additional XL licenses. If this fails, it attempts to check out two additional LP licenses. After that, the tool will attempt to check out the license that is next on the default license list listed in the following table:

Table 11-4 Default License List

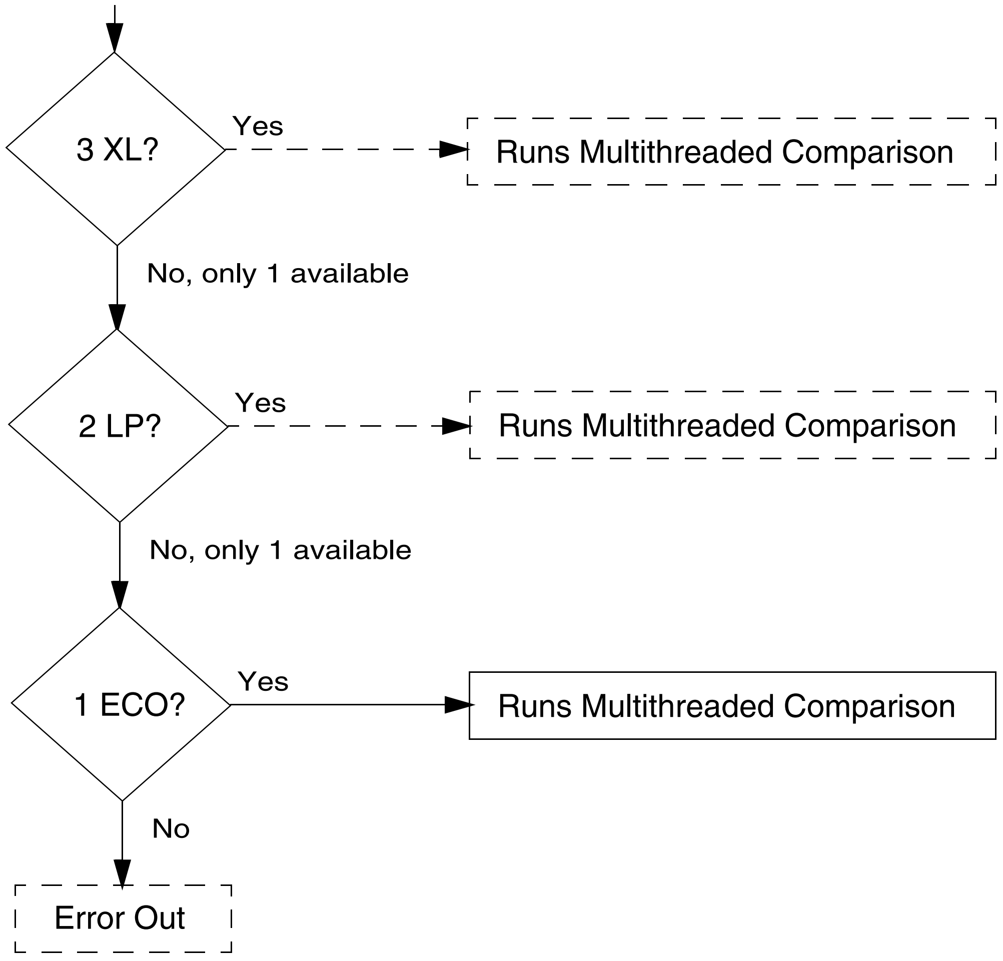
Starting License	**Default License List**
L	XL LP LPGXL ECO ECOGXL RCV
XL	XL LP LPGXL ECO ECOGXL RCV
GXL	XL LP LPGXL ECO ECOGXL RCV
LP	XL LP LPGXL ECO ECOGXL RCV
LPGXL	XL LPGXL LP ECO ECOGXL RCV
ECO	XL ECO LP LPGXL ECOGXL RCV
ECOGXL	XL ECOGXL LP LPGXL ECO RCV
RCV	XL RCV LP LPGXL ECO ECOGXL

Use the `SET PARALLEL OPTION -license <license_list>` command to explicitly specify and order the list of licenses to use for multithreaded processing. For example, if you have one LP and two ECO licenses, you can run multithreaded processing for up to four threads by using the following command:

```
set parallel option -license "lp eco"
```

The following shows a graphical example of the software flow as it attempts to check out licenses for multithreaded processing. For this run, the available additional licenses (to the one startup license) are one XL, one LP and one ECO.

```
set parallel option -license "XL LP ECO"
compare -threads 4
```


3 XL?
Yes
Runs Multithreaded Comparison
No, only 1 available
2 LP?
Yes
Runs Multithreaded Comparison
No, only 1 available
1 ECO?
Yes
Runs Multithreaded Comparison
No
Error Out

In the above example, if there were three XL licenses available, the software would have checked out the three XL licenses and run multithreaded comparison and disregarded the LP and ECO licenses.

Temporary Files and Directories

During its execution, the multithreading process feature creates and reads temporary files. By default, the software creates these files in a temporary directory in the current working directory. This directory is deleted automatically when the software exits normally. However, you can preserve this temporary directory after exiting with the `SET PARALLEL OPTION -keep_dir` command. The disk space requirement is approximately the same as the required amount when writing out the flattened Golden and Revised designs when running the `WRITE DESIGN` command.

Tip

Use the `/tmp` directory if your file system is slow. You can redefine the location of the temporary directory with the following environmental variable:

```
setenv RUN_REMOTE_TMPDIR_PREFIX /tmp
```

Multi-Threaded Functional Partitioning

The `RUN PARTITION_COMPARE` command offers multi-thread capability, which provides linear speed up due to the number of provided processor cores.

Multi-threaded functional partitioning offers the following features:

- Low multi-threading overhead
- Scalable with increased number of processor cores
- Automatic load balancing across all processor cores
- Easy to use command option, minimum change of existing dofiles
- Explicit progress report showing the percentage of completed tasks

There are two ways to set the number of threads for multi-threaded functional partitioning. Either call the following command:

```
run partition_compare -threads 2
```

Or, set the parallel option:

```
set parallel option -threads 2
```

followed by the following command:

```
run partition_compare
```

Either method executes multi-threaded functional partitioning with two active threads.

Note: The first method has higher priority than the second (in other words, if both methods are used, the number of threads specified in the first method takes priority).

Adding Partition Points

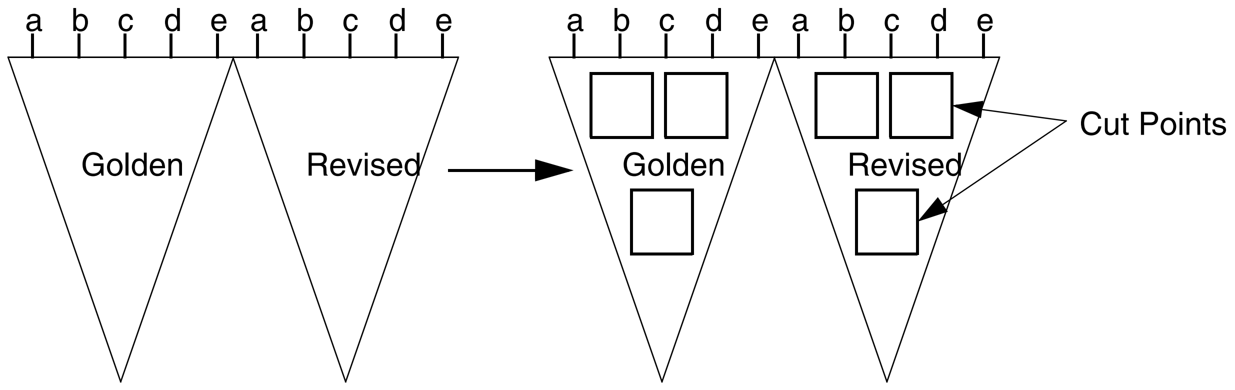
You can add partition points in LEC mode at specific pins, nets, gates, or at the boundaries of module instances and datapath operators. The partition points are added as physical CUT gates in the flattened netlists. The CUT gates, which serve as key points in the design, allow you to structurally partition the design. If the partition points are specified in only one design, the command will automatically deduce the corresponding partition points in the other design. Each of the partition pairs are first validated before adding them as physical CUT gates.

Adding Partition Points for Comparison

Using the ADD PARTITION POINTS command, you can specify any gate in the netlist to be a partition point according to the application. For example, the boundary gates of module instances or datapath operators are usually feasible places to add partition points for comparison.

The following command example and illustration shows how the partition (CUT) points are specified for the datapath operators in the Golden design, where the corresponding partition points (if any) are automatically deduced and added in the revised design:

```
add partition points -datapath
```



You can display these added partition points with the REPORT PARTITION POINTS command. You can delete the added partition points with the DELETE PARTITION POINTS command.

The following command example shows how the CUT points are specified for the module instance in the Golden design, where the corresponding partition points (if any) are automatically deduced and added in the Revised design:

```
add partition points -instance instance_name
```

You can also use ADD PARTITITON POINTS command for correspondence reporting only, instead of physically adding CUT gates. The following command example shows how you can specify a list of gate identifiers in the Revised design, and use the `-verbose` option to report the corresponding gates, if any, in the Golden design:

```
add partition points identifier1 identifier2   -revised -verbose -nocut
```

The `-nocut` option in the example ensures that the CUT gates are not physically added in the two designs.

After adding the CUT points, you can add all the compare points (including partition points) and run compare. If all the compare points (including the partition points) are equivalent, the designs are proved to be equivalent. However, if one of the compare point is non-equivalent, you will need to diagnose the non-equivalent points. If the partition points are the cause of non-equivalence, you can delete these at any time in LEC mode using the `DELETE PARTITION POINTS` command. You can add and delete partition points multiple times until you generate feasible partition points for comparison abort resolution.

Name-based Physical Partitioning

Use the `ADD PARTITION POINTS` command's `-name` option to add partition points around instances using a name-based algorithm. This is faster than the default algorithm, which adds partition points based on function.

To perform a flat run with hierarchical partitioning, add partition points to all module instances using the following command:

```
add partition points -instance * -name -input -output
```

To get better datapath quality, add partition points to all module instances containing datapath operators using the following command:

```
add partition points -datapath -name -input -output
```

Comparison with Functional Partitioning

You can use the `RUN PARTITION_COMPARE` command to run comparisons with functional partitioning. You can specify partition key points, or they can be selected automatically by the command.

For example, when abort points are encountered in comparison, you can run this command to do functional partitioning for the abort points.

Analyzing Non-Equivalence

Use the `ANALYZE NONEQUIVALENT` command to help identify the possible causes of non-equivalent compared points. The following schematics show examples of non-equivalent analysis:

Example Report

The following shows an example of a report when running the `ANALYZE NONEQUIVALENT` command. The lines in bold indicate the cause of the problems:

```
LEC> analyze noneq 213
//Command analyze noneq 213
Analyzing non-equivalent compared points:
  (G) + 213 DFF /wbs/hvlen_reg[28]
  (R) + 6277 DFF /wbs/hvlen_reg[28]/U$1
  The clock of DFF in Golden is not gated.
  The clock of DFF in Revised is gated.
Analysis of non-equivalent compared points:
  Gated clock of of DFF or DLAT. (Occurrence: 1)
  Unknown reason. (Occurrence: 1)
LEC> analyze noneq 170 -revised
//Command analyze noneq 170 -revised
Analyzing non-equivalent compared points:
  (G) + 167 PO /wbm_sel_o[0]
  (R) + 170 PO /wbm_sel_o[0]
  Following constraints may be necessary:
    Constant 1: (G) 1026 DFF /wbm/sel_o_reg[0]
Analysis of non-equivalent compared points:
  Sequential constant. (Occurrence: 1)
  Unknown reason. (Occurrence: 1)
```

Clock Gating

You can fix the first problem in the report:

```
  The clock of DFF in Golden is not gated.
  The clock of DFF in Revised is gated.
```

by running the following command in Setup mode:

```
set analyze option -auto
```

or the following command in LEC mode:

```
analyze setup
```

Sequential Constant

You can fix the second problem in the report:

```
Following constraints may be necessary:
  Constant 1: (G) 1026 DFF /wbm/sel_o_reg[0]
```

by running auto analysis in Setup mode with the following commands:

```
set analyze option -auto
set flatten model -seq_constant
```

or the following command in LEC mode:

```
remodel -seq_constant
```

Analyzing Implication Values

You can analyze implication values on the design with the `ANALYZE IMPLICATION` command. If you assign value(s) on certain gate(s), this command shows what the necessary values are on other gates to satisfy the assignment. It can also show if a gate has redundant fan-in and if a gate is a constant gate.

The results are displayed in the Schematic Viewer with the following colors:

- Blue: initial assignments
- Green: current implication values
- Red: gates on the conflict path
- Purple: location where conflict occurred

In the schematic view, you can also right click the gate and set a value. Holding the mouse pointer on a gate, an information box will show if this gate has redundant fan-in and if it is a constant gate.

Netlist Analysis

This feature provides another view of the flattened netlist, which could help datapath analysis, comparison, and diagnosis.

Currently, netlist analysis enables the following operations on a flattened netlist:

- Half adder and full adder cell extraction
- Library cell identification and optimization
- MUX logic extraction for DFFs

Extracting Half Adder and Full Adder Cells

Half and full adder cells are basic logic units widely used in synthesized datapath designs. For a datapath design with constant signals and/or equivalent signals, synthesis tools tend to optimize the adder cells based on these signals. These optimizations could make high quality datapath analysis difficult.

Extracting half and full adders from the synthesized netlist helps reduce the complications caused by optimizations, and improves datapath analysis and learning quality. To invoke the extraction, use the following command in LEC mode:

```
analyze netlist -abstract hfa
```

When using LEC, the synthesized netlist will most likely be taken as the revised netlist; therefore, use the command to invoke extractions on the synthesized netlist:

```
analyze netlist -abstract hfa -revised
```

This command modifies the specified netlist; these modifications cannot be revoked and can sometimes have an adverse effect for datapath analysis and comparison. Thus, the command is not invoked automatically and is recommended only for abort resolution. For datapath designs with aborts, if the datapath analysis quality is low, try this command before the `ANALYZE DATAPTH` command without `-MODULE` option.

Identifying and Optimizing Library Cells

In some designs, such as FPGA designs, library cells are not optimized for verification. Library cell optimization is provided to address this issue, and it can be invoked by the following command in LEC mode:

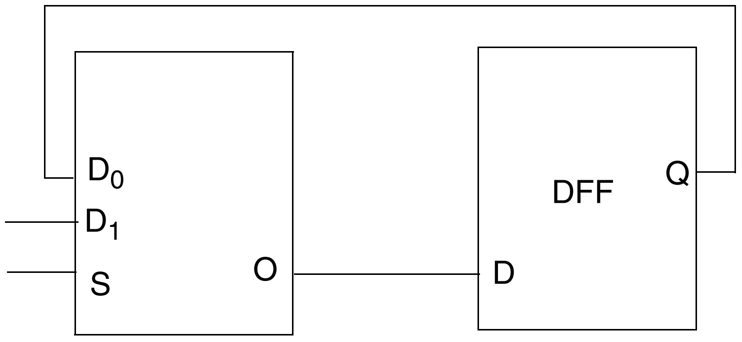
```
analyze netlist -abstract libcell
```

Extracting MUX Logic for DFFs

To invoke MUX logic extraction for DFFs, use the following command in LEC mode:

```
analyze netlist -abstract muxdff
```

This command has been integrated in the ANALYZE RETIMING command to facilitate retiming analysis, and can be helpful for diagnosing of retimed designs. For example, using this command, you can check if the input of a DFF can be implemented as in the following graph:



Sample Dofile

The following is an example of a Conformal dofile that includes the Conformal XL flow.

Note: Retiming can have impact on datapath learning. As a result, if the design has retiming, you should run the `ANALYZE RETIMING` command before running `ANALYZE DATAPATH`. Likewise, mapping will impact datapath learning and should be performed first.

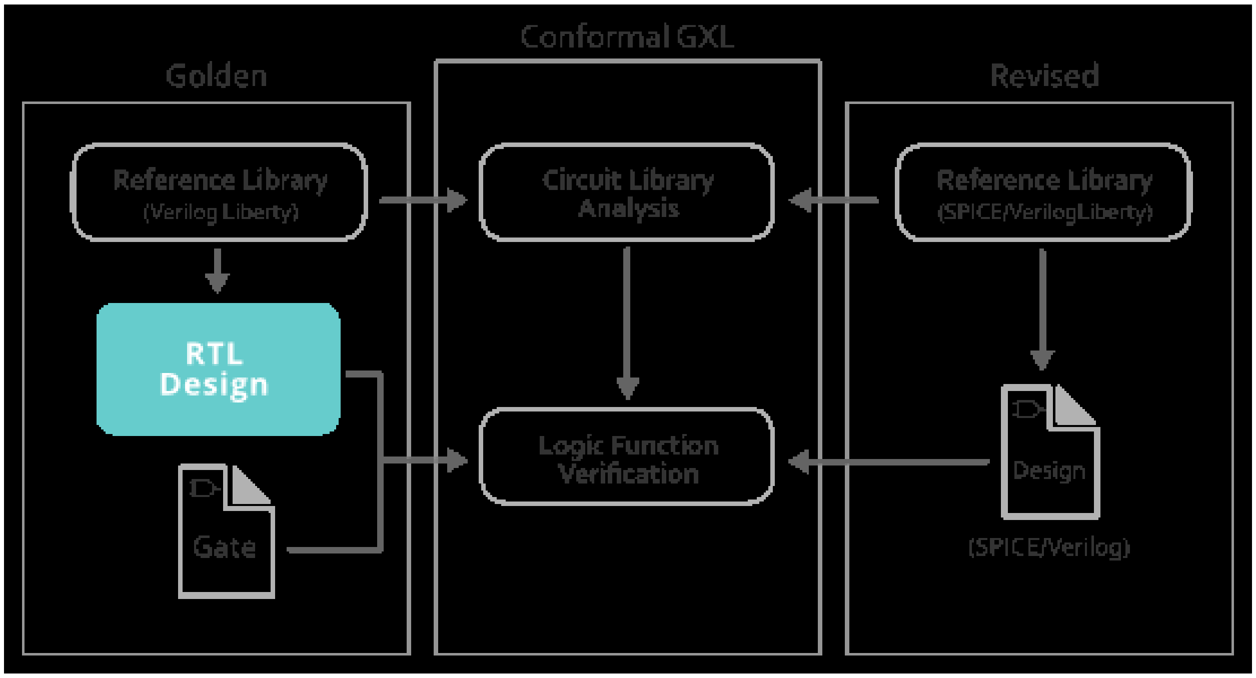
```
// To read in the RTL design and the synthesized gate
// netlist

read design rtl.v -Golden
read design netlist.v -revised

// To define all of the design constraints, specify pipeline
// retimed module and so forth

add pin constraint 0 SE -revised
add module attribute ocd_cs_r4c -pipeline_retime -Golden

// To automate modeling and mapping processes

set system mode lec

// To specify pipeline retiming, requires Conformal XL
// license

analyze retiming

// Map key points

map key points

// To specify datapath analysis, requires Conformal XL
// license

analyze datapath -merge

//To run key point comparison

add compare point -all

compare
```

12
Layout Versus Schematic

Overview

Designers use Conformal L to run logic function verification between an RTL model and a gate netlist , or between a gate netlist and another gate netlist. Subsequently, a SPICE netlist representing the circuit and a GDSII netlist representing the physical geometry of the design are created. Designers use Layout Versus Schematic (LVS), which is a physical verification tool to verify equivalence between the SPICE netlist and the GDSII netlist. LVS checks whether the connectivity of the circuit and the layout are equivalent. However, during this flow, neither the logic function of the SPICE netlist nor GDSII data is verified.

Conformal GXL operates within Conformal L to enable logic verification of the final design step: circuit implementation. After final place and route, the circuit design is implemented for tapeout. At this point, Conformal GXL enables functional verification on the final circuit design represented by the SPICE netlist used as reference to LVS verification. The following figure illustrates the Conformal GXL LVR process flow.


Conformal GXL
Golden
Revised
Reference Library (Verilog Liberty)
Circuit Library Analysis
Reference Library (SPICE/VerilogLiberty)
RTL Design
Logic Function Verification
Design
Gate
(SPICE/Verilog)

LVR Functionality

Conformal GXL LVR furnishes designers with two interdependent functions that allow complete verification from RTL or gate to GDSII: Circuit Library Analysis and Design Logic Function Verification.

Circuit Library Analysis

The first phase in the verification process is to isolate design and library problems. Conformal GXL LVR runs three different checks for circuit library analysis:

- Automatic Functional Analysis—LVR compares logic function of two libraries on a cell-by-cell basis. Every cell is automatically compared in this process.
- Cell Error Repairs—After the analysis, LVR generates error reports. LVR reports those cells that have errors that can prevent design logic comparisons and replaces those library cells with the reference logic to allow design verification.
- Phase Inversion Correction—LVR aligns state element phases between Golden and Revised library sequential elements to avoid phase inversion problems during the design level verification phase.

Design Logic Function Verification

After the Circuit Library Analysis phase completes, it initiates the Design Logic Function Verification phase. In this phase, Conformal GXL verifies full equivalence between two designs.

Starting Conformal GXL

To start the Conformal GXL software in graphical mode, run the following command:

```
lec -gxl
```

To start the ConformalGXL software in non-graphical mode, run the following command:

```
lec -gxl -nogui
```

LVR Flow

This section details the Conformal GXL LVR flow. As mentioned above, the LVR flow consists of two phases: Circuit Library Analysis and Design Logic Function Verification. Summaries of the objectives of these two phases of the flow are as follows:

- Circuit Library Analysis
    - **a.** Read Golden library (formats include: Verilog, VHDL, and Liberty)
    - **b.** Read Revised library (formats include: Verilog, VHDL, Liberty, and SPICE)
    - **c.** Run the `VALIDATE CIRCUIT` command to do the analysis

   d. Review warnings and errors

- Design Logic Function Verification

   a. Read the reference design model or netlist into the Golden design

   b. Define design constraints and run verification

Circuit Library Analysis

Conformal GXL LVR includes two phases; that is, two interdependent functions. In this first phase, complete the steps described below.

1. Read the Golden library.

   Read in the reference library view. Read the library into the *design space* using the following command:

   ```
   >read design <library view1> -Golden [-verilog | -liberty | -vhdl]
   ```

   Use the options noted above as follows.

`-verilog`	To specify that the reference library format is Verilog, use the `-verilog` switch. *This option is the default.*
`-liberty`	To specify that the reference library format is Liberty, use the `-liberty` option.
`-vhdl`	To specify that the reference library format is VHDL, use the `-vhdl` option.

2. Read the library to be analyzed.

   Read in the circuit library to be analyzed.

   Note: The `-spice` option is described below. All other applicable options are described above.

   ```
   >read design <library view2> -revised [-verilog|-liberty|-vhdl|-spice]
   ```

`-spice`	To specify that the reference library format is SPICE, use the `-spice` switch.

3. Validate the circuit.

   With both library views read into Conformal, run the circuit library analysis using the `VALIDATE CIRCUIT` command. LVR verifies all corresponding cells between the Golden and Revised databases during this step to isolate design and library problems, and then reports any errors or non-equivalent cells.

   ```
   >validate circuit [-revised |-Golden]
   ```

`-revised`	Use the `-revised` switch to specify that the Revised database will be validated. *This option is the default.*
`-Golden`	Use the `-Golden` switch to specify that the Golden database will be validated.

4. Check for library analysis problems.

   Check for any error or warning messages to determine if there are any library analysis problems.

Design Logic Function Verification

After LVR analyzes the circuit library logic in Phase I, it continues with Phase II, which is the Design Logic Function Verification. During this phase, LVR verifies the design netlist using the additional steps as follows.

Note: Ensure that Phase I completes before every Phase II design logic verification.

1. Read in the reference RTL model.

   For an RTL Model, use the following command. Use the `-replace` option when you do Phase I and Phase II in the same LVR session. This option ensures that the library data in the design space is replaced with design data.

   ```
   >read design <rtl model> -Golden -replace [-verilog | -vhdl]
   ```

2. Read in the gate netlist.

   For a Gate Netlist, use the following two commands.

   ❑ First, run the `READ LIBRARY` command below.

   Note: The file `library view1` is the simulation model library previously used as the reference for the validate circuit function above. See Read in the reference RTL model. on page 297

   ```
   >read library <library view1> [-verilog | -liberty | -vhdl]
   ```

- ❑ Run the `READ DESIGN` command. The `-replace` option lets the gate netlist replace the library data.

  ```
  >read design <gate netlist> -Golden -replace [-verilog | -vhdl]
  ```

3. Define the design constraints and run verification.

   This step includes setting up the verification with constraints, mapping key points, and comparing the two designs. For the command flow used to do this step, see Chapter 6, "Using the Setup Mode" and Chapter 7, "Using the LEC Mode".

LVR Implementation

Suggested Uses

Use Conformal GXL to analyze circuit libraries and compare designs.

Circuit Library Verification

Conformal analyzes different formats of a circuit library; for example, Verilog versus Liberty or Verilog versus SPICE. No manual modeling or setups are required. See LVR Flow on page 295 for library verification steps.

Design Comparison

Use Conformal GXL to compare a Verilog gate design netlist to a design SPICE netlist. Unlike comparison with an RTL model where scan function is disabled, using a gate design netlist as the reference enables Conformal to verify scan function.

Conformal Dofile Examples

The first dofile example below illustrates a typical Conformal session. Compare this dofile with the second dofile example to gain a better understanding of the relationship of the Conformal GXL flow to the general Conformal L flow.

The first example illustrates a typical Conformal dofile used to do an equivalence check on an RTL versus a gate netlist.

`lec.dofile`:

```
// Define embedded blocks for black boxing here
// For example: add notranslate module PLL* -both
```

```
read design rtl.v –Golden
read library library.v –revised –verilog
read design netlist.v –revised

// Define all design constraints and
// modeling options
// For example: add pin constraint 0 SE -both

set system mode lec
add compare points –all
compare
report compare data
```

As illustrated below, with the addition of the LVR flow into Conformal, the dofile requires very little modification.

`lvr.dofile`:

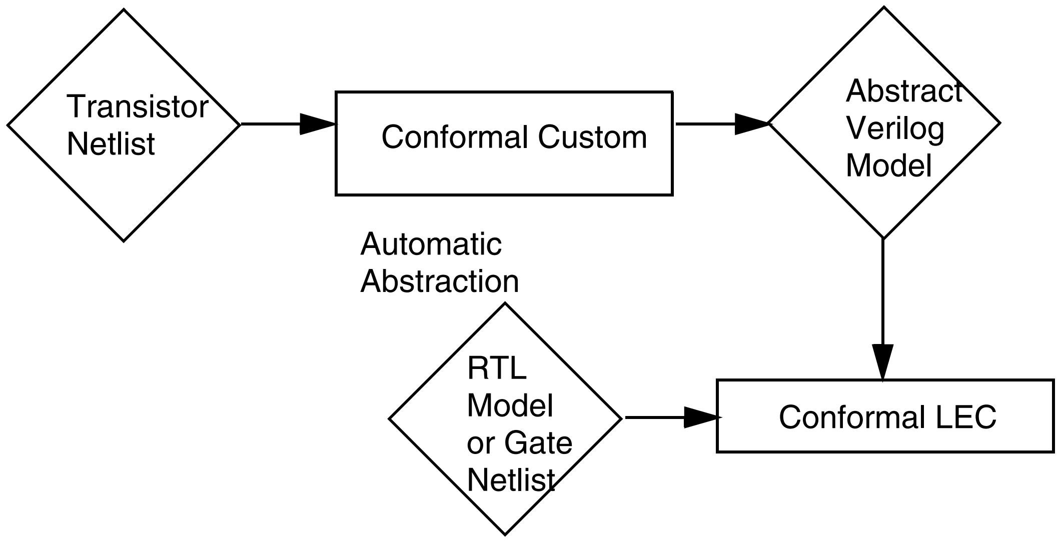
```
// Define embedded blocks for black
// boxing here
// For example: add notranslate module PLL* -both
// >>>> Circuit library analysis step

read design sim_library.v –Golden
read design netlist.sp –revised -spice
validate circuit –revised

// >>>> Netlist verification step

read design rtl.v -Golden –replace

// Define all design constraints and
// modeling options
// For example: add pin constraint 0 SE –both

set system mode lec
add compare points –all
compare
report compare data
```

13

Conformal Custom

Overview

This chapter describes the standard commands used in a typical Conformal Custom (also known as Conformal GXL) session and introduces you to the process flow, as shown in the following figure:



Conformal Custom supports transistor netlists with legal SPICE and some variants (`.cir`, `.sp`, `.spi`, and `.cdl`). Conformal Custom also supports Verilog switch-level netlists, but these netlists do not have the information required for Conformal Custom checking (such as MOS body connections).

The tool does not support DSPF netlists. A DSPF netlist is a SPICE format created for a GDS (layout) extraction tools and it includes RC networks. Instead, output the SPICE netlist used for cell-level LVS, which can be output from the library cell schematic view.

Note: When parsing a SPICE file, any net connected to a PMOS bulk node is assumed to be VDD. Any net connected to an NMOS bulk node is assumed to be ground. To override this setting, use the `SET SPICE OPTION` command with the `-NOBulk` option.

Important

If you use `SET SPICE OPTION -NOBulk`, you must use it before reading in the SPICE file.

You can also write out a Verilog gate-level netlist from the abstracted transistor netlist. This gate level netlist is used for simulation acceleration, emulation, and ATPG.

Custom Licensing

You must have the Conformal GXL license to use the transistor abstraction capability within Conformal. To check whether you are licensed to run transistor abstraction, inspect your license file and look for the following `FEATURE` lines:

```
FEATURE Conformal_Custom cdslmd 6.200 14-dec-2006 5…
```

Contact your Cadence sales representative if you want to obtain this feature.

Abstraction Methods

Conformal GXL automatically abstracts functionality from a transistor netlist using the following methods:

- Automatic Functional Analysis

  Conformal GXL automatically abstracts the Boolean function of Static CMOS, Pass-GATE, and Tristate logic. Conformal GXL recognizes MUXs, and abstracts transistors into latches and DFFs. It also determines signal directions through MOS transistors and module I/O.

- Pre-Charge Logic Abstraction

  Conformal GXL abstracts the logical function of a dynamic circuit in the evaluate phase. The tool requires you to identify the pre-charge (off-duty) clock.

- Pattern Matching

  Conformal GXL also recognizes transistor patterns that you define. If you specify the transistor netlist for the pattern along with its corresponding functional Verilog or VHDL model, Conformal GXL uses the functional model you supplied every time it encounters its corresponding pattern in the design during abstraction.

Starting Conformal GXL

To start the Conformal GXL software in graphical mode, run the following command:

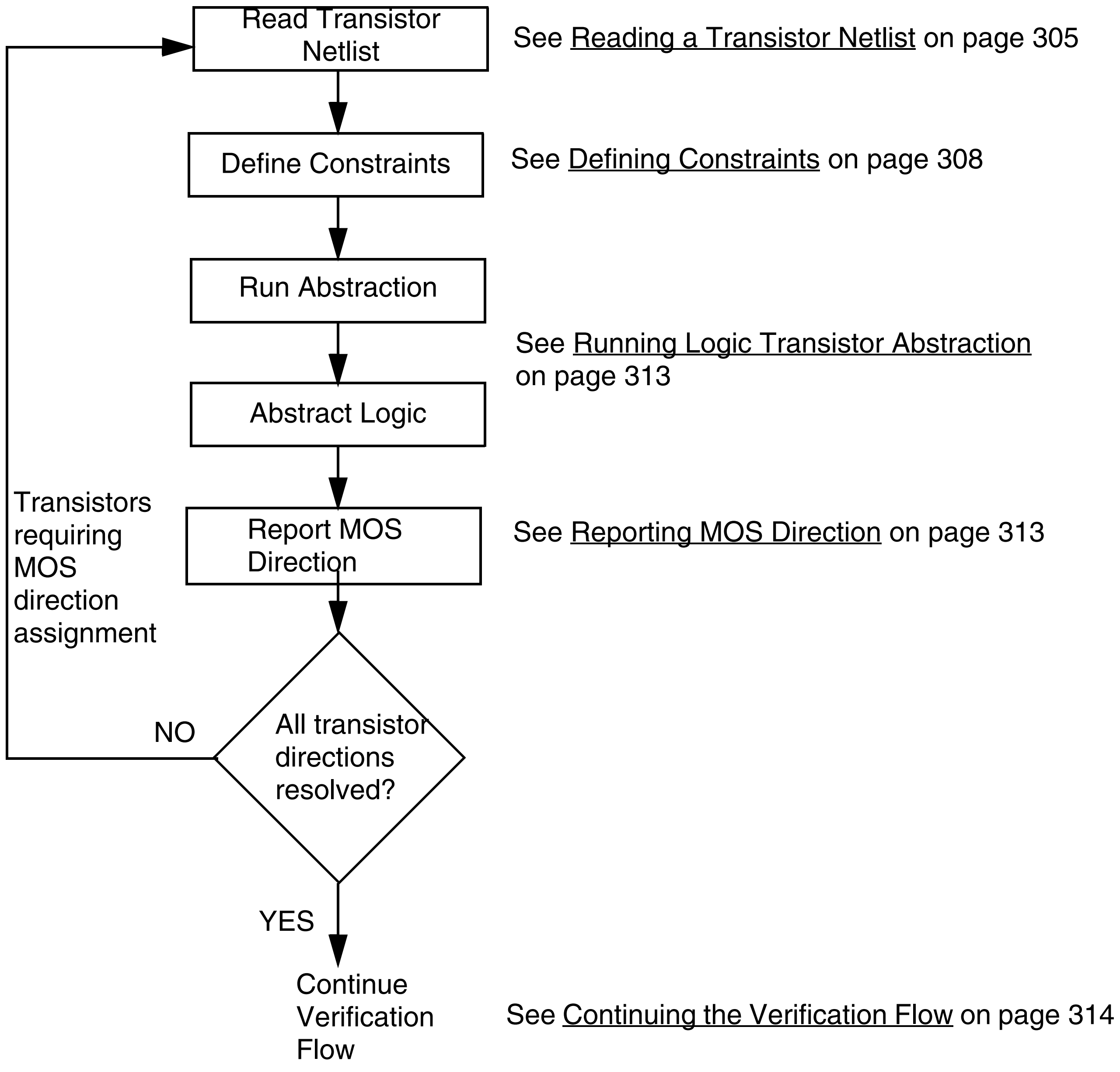
```
lec -gxl
```

To start the Conformal GXL software in non-graphical mode, run the following command:

```
lec -gxl -nogui
```

Conformal GXL Process Flow

As part of Conformal GXL, LTX is an extension to the basic Conformal process flow. The following figure shows the Conformal GXL Transistor Abstraction Flow:


Read Transistor Netlist
Define Constraints
Run Abstraction
Abstract Logic
Report MOS Direction
Transistors requiring MOS direction assignment
NO
All transistor directions resolved?
YES
Continue Verification Flow

See Reading a Transistor Netlist on page 305

See Defining Constraints on page 308

See Running Logic Transistor Abstraction on page 313

See Reporting MOS Direction on page 313

See Continuing the Verification Flow on page 314

Reading a Transistor Netlist

Your first task is to read in a compatible transistor netlist to Conformal. Conformal reads SPICE and Verilog transistor netlists. The following sections guide you as you:

- Prepare the netlist
- Read in a netlist and transistor description

Preparing to Read a SPICE Netlist

Before reading in a SPICE netlist to Conformal, inspect the netlist to determine the names of the N-channel and P-channel devices. Conformal automatically recognizes model names beginning with '`p`' and the model name '`up`' as a `PMOS` type, and recognizes model names beginning with '`n`' as an `NMOS` type. However, if other names are used, such as `UNAME1` or `UNAME2`, you must do one of the following:

- Insert a `.model` card at the top of the netlist.

  The `.model` card maps the device names `UNAME2` to `NMOS` and `UNAME1` to `PMOS`. For example:

  ```
  .model UNAME1 PMOS
  .model UNAME2 NMOS

  .SUBCKT K1 n0 n1 n2
  Mx0 n0 n1 n2 n3 UNAME1
  Mx1 n0 n1 n2 n3 UNAME2
  Xx2 n0 n1 n2 n3 n4 n5 n6 K2
  .ENDS K1
  ```

- Use the `SET MOS MODEL` command to define the MOS model names used in the SPICE netlist.

- Insert an `*.EQUIV` or `*.EQUIVALENCE` directive at the top of the netlist to replace user-defined model names with the model names that Conformal can automatically recognize. For example:

  ```
  *.EQUIV PMOS=UNAME1 NMOS=UNAME2
  SUBCKT K1 n0 n1 n2
  Mx0 n0 n1 n2 n3 UNAME1
  Mx1 n0 n1 n2 n3 UNAME2
  Xx2 n0 n1 n2 n3 n4 n5 n6 K2
  .ENDS K1
  ```

  Note: If the `*.EQUIV` statement stretches over multiple lines, you must use an asterisk and plus sign (`*+`) at the beginning of each additional line to indicate continuation.

Note: Conformal treats resistor definitions in the SPICE netlist as wires and it ignores capacitors.

SPICE ports and transistors are bidirectional. That is, all ports are defined as INOUT, and transistor source-drain is interchangeable. During abstraction, Conformal determines signal direction through the source-drain of a transistor. Additionally, it assigns port directions. In situations where Conformal cannot determine the signal direction through a transistor or a port, your assistance is required. Assign port directions in the SPICE netlist with the `.PININFO` directive as shown in the following example:

```
.SUBCKT K3 n0 n1 n2
*.PININFO n0:I n1:I n2:O
Mx0 n0 n1 n2 vdd PCH
Mx1 n0 n1 n2 gnd NCH
.ENDS K3
```

The following applies to .PININFO Directives:

- `<port>:I`

  Declares the port is an input (see `n0:I` in this example).

- `<port>:O`

  Declares the port is an output (see `n2:O` in this example).

- `<port>:B`

  Declares the port is bidirectional (not used in this example).

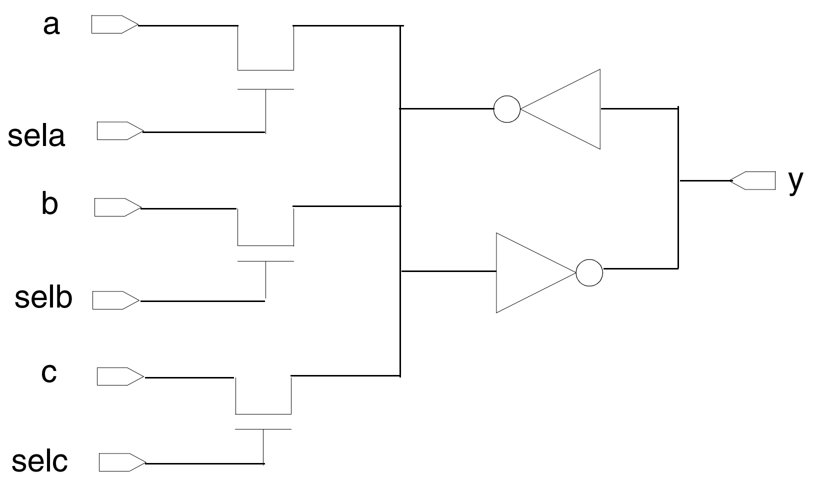
Reading Netlists and Descriptions

To read a transistor netlist into the design space of Conformal, use the `READ DESIGN` command with either the `-Golden` or `-revised` switch.

Reading a Transistor Description

In addition to reading netlists, read a transistor description for pattern-matching purposes. See the READ PATTERN command and refer to the next figure:

The design shown in the following figure can be interpreted as a latch or as a 3-to-1 multiplexer. For this example, the circuit is treated as a multiplexer.



The following example lists the Verilog transistor netlist (pattern) for the previous design.

```
module mux3x1 (a, b, c, sela, selb, selc, y);
input a, b, c, sela, selb, selc;
output y;

wire s;
tranif1 t0 (s, a, sela);
tranif1 t1 (s, b, selb);
tranif1 t2 (s, c, selc);
inv     d0(y,s);
inv      fdbk(s, y);
endmodule
```

Additionally, the following example lists the substitute Verilog model for this pattern.

```
module mux3x1 (a, b, c, sela, selb, selc, y);
input a, b, c, sela, selb, selc;
output y;

wire y_;
assign y_ = sela ? a : 1'bz;
assign y_ = selb ? b : 1'bz;
assign y_ = selc ? c : 1'bz;
assign y = ~y_;
endmodule
```

The following is an example of a command file (dofile) for Conformal GXL LTX that demonstrates how to execute the multiplexer as shown in the previous example:

```
read pattern pattern.v -verilog
// reads the mux3x1 transistor netlist or pattern

read library lib.v -verilog
// reads the verilog substitute model for the pattern

read design Golden.v
// reads the Golden transistor netlist that
// includes mux3x1 along with other circuitry

abstract logic
// calls up LTX abstraction routine
```

Defining Constraints

The `ABSTRACT LOGIC` command is included in the dofile shown in the previous example. However, before you use this command, you must consider issues that can impact the abstraction.

Resolving Global Net Names

Conformal GXL automatically recognizes VDD, GND, and VSS global nets in SPICE. If other global net names are used for power and ground in the netlist, use the `ADD TIED SIGNALS` command before transistor abstraction. For example:

```
add tied signals 0 cds.global.gnd_ -all -Golden
add tied signals 1 cds.global.vdd_ -all -Golden
```

Adding Net Attributes

Use the `ADD NET ATTRIBUTE` command to identify an internal net as a pre-charge clock to a dynamic circuit. This case arises when the pre-charge clock is generated internally and is not available as an external design pin. For example:

```
add net attribute CLOCK0 net134 -module bitpre -Golden
```

In this example, `CLOCK0` defines the off-state of the pre-charge logic. That is, a zero value on the net `(net134)` puts the circuit in pre-charge or off-state mode, while a one value on the same net puts the circuit in duty-state or evaluate mode.

Resolving Modules in the Design Hierarchy

Conformal GXL automatically resolves modules in simple cases; that is, modules having few inputs, outputs, and instances. However, in more complex cases that include unnecessary levels of design hierarchy, use the RESOLVE command to *manually* resolve modules in the design hierarchy.

This command un-groups a module and raises its contents one level. For example, when a flip-flop module instantiates two latches, a master and a slave latch, use the RESOLVE command to resolve the latch hierarchy. Then, the Conformal GXL abstraction engine combines the master and slave latch circuitry to create a single flip-flop.

After resolving the hierarchy, the internal database in Conformal is modified. You *cannot undo* the changes in the hierarchy.

Important

The hierarchy in the designs should be maintained to the extent possible to simplify abstraction and diagnosis.

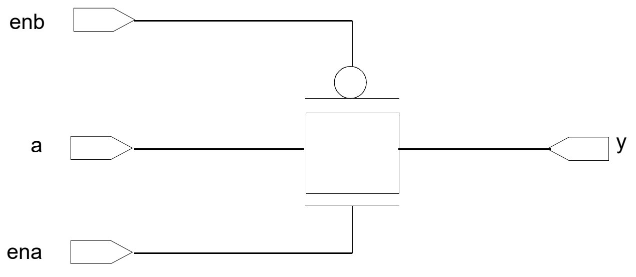
When the design is relatively small, and submodule abstraction fails because of many hierarchical relationships between pins, use the FLATTEN command to force Conformal to resolve all submodules. This command can sometimes help improve abstraction. For example:

```
flatten -module chip -force -revised
```

Adding Pin Equivalences

Pin equivalences are used to define the relationship between two or more pins in a module. When two pins are equivalent through inversion, as shown in the following pass-gate example, use the ADD PIN EQUIVALENCES command.

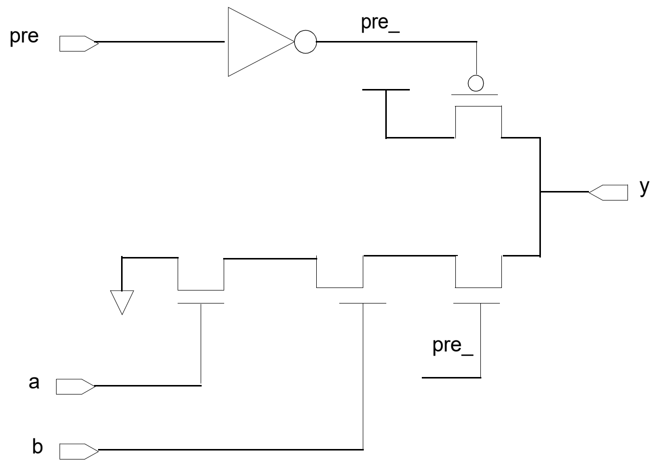
Note: If you use the `-both` option, every primary input pin you list must exist in both designs (Golden and Revised). If they do not, Conformal returns an error message.



Disabling the Pre-Charge Clock in a Dynamic Circuit

When a design has pre-charge logic, Conformal GXL requires you to identify the pre-charge clock and define its off-state. Use the `ADD CLOCK` command to declare a pre-charge clock.

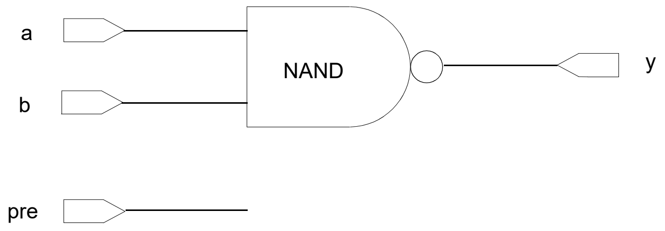
The following figure illustrates a pre-charge NAND gate. In this example, pin `pre` in module `NAND` represents the pre-charge clock.



To define the off-state for this circuit, specify the following constraint:

```
add clock 1 pre -Golden
```

As a result of the `ADD CLOCK` command, Conformal GXL models the design as a static NAND gate, as shown in the following figure:



Specifying Conditions for Abstracting Logic

The `SET ABSTRACT MODEL` command specifies certain conditions for abstracting transistor logic. The command's `-pre_charge_keep_clock` option includes the defined pre-charge clock in the abstracted logic function (the default behavior removes the defined pre-charge clock from the abstracted logic). This is indicated when you define a precharge clock with one of the following commands:

```
add net attribute CLOCK0 | CLOCK1
add clock 0 | 1
```

When you use `-pre_charge_keep_clock`, the resulting logic is equivalent to RTL that explicitly models the pre-charge condition, rather than RTL that models only the evaluate function. In the latter, the output function is not defined during precharging. (See the following examples.)

Explicit modeling:

```
always @(clock or a or b) begin
if (clk) y = a ! b;
else y = 1'b1; // precharge
end

Evaluate function:
assign y = a ! b;
```

Note: Designers employ explicit modeling when the precharge function output is latched during pre-charge. Thus, the resulting functionality of either method is equivalent. It is strictly a matter of coding style. Both methods are acceptable.

Reporting Conditions for Abstracting Logic

If you ran the `SET ABSTRACT MODEL` command to abstract transistor logic from particular modules, use the `REPORT ABSTRACT MODEL` command and Conformal GXL reports their abstraction conditions.

Resetting Conditions for Abstracting Logic

If you ran the `SET ABSTRACT MODEL` command to abstract transistor logic from particular modules, use the `RESET ABSTRACT MODEL` command to reset the abstraction conditions to their original state.

Assigning Pin Direction

Although Conformal GXL is capable of determining pin and signal direction, some cases require manual assistance. For transistor netlists that lack pin direction information, use the `ASSIGN PIN DIRECTION` command to make a manual assignment. For example:

```
assign pin direction in mux2p in0 -revised
```

Copying Modules

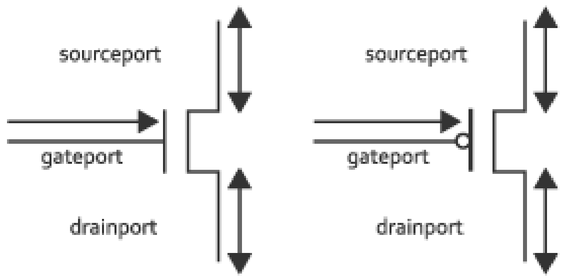
You can copy a module's logic or pin direction from the RTL to assist in abstraction. Use the `COPY MODULE` command:

Assigning Transistor Direction

For Conformal GXL to successfully abstract a logical gate-level model of a transistor netlist, it must determine signal directions through all transistors in the design. There are three possible directions a signal can flow in an N channel or P channel transistor:

- From drain to source
- From source to drain
- Both ways (bidirectional)

Note: In SPICE, MOS transistors are all bidirectional. Thus, the source and drain ports are interchangeable. However, signals always flow into the gate of a transistor, and the burden falls on Conformal GXL to determine the signal directions. The following figure illustrates this concept.



In Verilog the task of determining signal direction is simplified because Verilog supports two transistor definitions:

- One definition is a unidirectional transistor declaration for which the signal flows from source to drain, or from drain to source, but not both directions.
- The other is a bidirectional N or P transistor declaration, where signals can flow in both directions.

The following illustrates the two Verilog transistor definition types:

- `nmos ins0 (net1, net2, net3)`
  - Unidirectional N channel transistor
  - `net1` is output and `net2` is input
  - Similarly for P transistors, `pmos ins1` (`net1`, `net2`, `net3`)
- `tranif1 ins0 (net1, net2, net3)`
  - Bidirectional N channel transistor

- ❑ `net1` and `net2` are inout
- ❑ `net3` is gate port
- ❑ Similarly for P transistors, `tranif0 ins1` (`net1`, `net2`, `net3`)

Assigning MOS Direction

If Conformal GXL cannot determine the direction of a signal through a transistor device, manually assign the direction using the `ADD MOS DIRECTION` command. For example:

```
add mos direction phgx10 M30/N12 ixMM -Golden
```

The following is a transistor definition example:

```
transistor-type name output(drain) data(source) control(gate)
nmos            n1   i1            vss          a
```

For this example shown above, use the `-all` option as follows to assign direction from `vss` to `i1`:

```
-all -from_source
```

In your designs, use `-all -from_source` to apply MOS direction from source to drain on all of the transistor-MOS instances in your design.

Likewise, `-all -from_drain` assigns MOS direction from drain to source on all of the transistor-MOS instances in your design.

Running Logic Transistor Abstraction

After reading a transistor design and defining constraints as needed, perform logical transistor abstraction, which is the process of abstracting a functional gate model from the transistor netlist. Use the `ABSTRACT LOGIC` command for this step.

Note: If neither the `-all` or `-module` option is specified, Conformal abstracts the current root module and any modules that are instantiated under it.

Reporting MOS Direction

After reading in a SPICE or Verilog transistor netlist, applying constraints, and running transistor abstraction, you can use the `REPORT MOS DIRECTION` command to check if the abstraction is complete.

If the resulting abstraction is not complete, Conformal GXL lists the modules and the number of transistors you must constrain. The following is an example of what the command returns:

```
// Command: abstract logic -Golden
Module ‘mux4x2’ has 4 bi-directional MOS
Module ‘cluster’ has 2 bi-directional MOS
```

Continuing the Verification Flow

When you have successfully abstracted a transistor netlist using Conformal GXL, use Conformal to run logic equivalency checking against another RTL, GATE, or Transistor netlist.

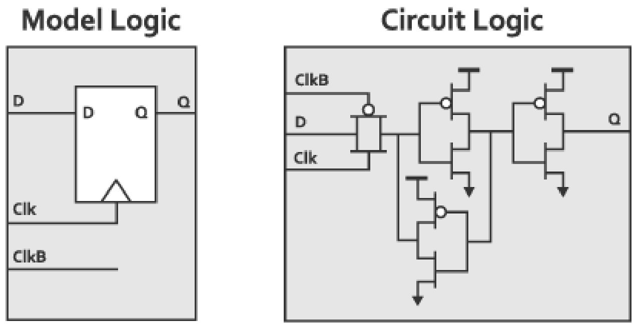
Specifying Conditions for Abstracting Logic

This section describes some of the issues that can occur when specifying conditions for abstracting transistor logic for latch modeling.

Pin Equivalences

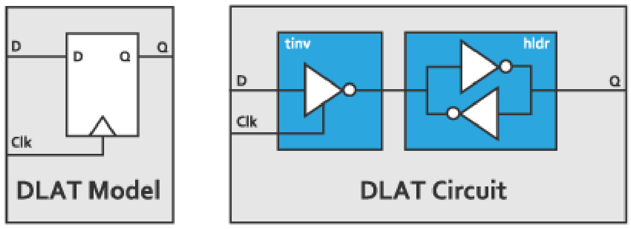
In this example, an RTL model and circuit are not equivalent except when `Clk` and `ClkB` are inverted. Before abstracting circuits, use the following command:

```
add pin equivalence Clk -inv Clkb
```



Flatten or Resolve

Abstraction works hierarchically but not across hierarchy except for constraint or pin equivalence propagation. Functions across hierarchy are not abstracted. In the following illustration of a circuit, the right `tinv` and `hldr` cells are parts of the D Latch.



Use the following command to pop these cells up one level:

```
resolve tinv hldr -revised
```

Or you can use the following command to remove all hierarchy:

```
flatten -module DLAT -revised
```

Abstraction will complete and result compares to model.

Abstraction Options

Abstraction options affect how certain circuits are abstracted. Use these options to transform special circuits, such as:

- Domino (pre-charge) logic modeling
- RAM pre-charge, sense amp, and pulse clock modeling
- Custom circuit weak pull-up and level restorer modeling

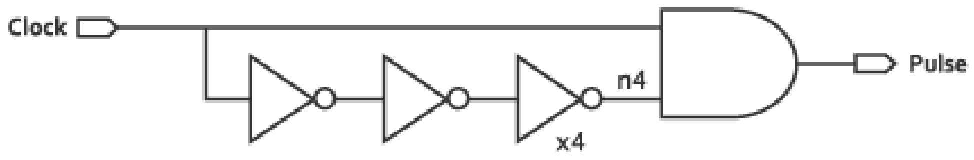
Before abstracting, use the `SET ABSTRACT MODEL` command. To limit scope of the abstraction, use the command's `-module` option.

Transient Pulse Generators

Pulse generators are commonly used in memories. The start of the pulse initiates an access. The end of the pulse provides a signal to sample the data for reading. Pulse generators have transient function and are not supported by traditional abstraction or static formal verification equivalence checking technique.

In the following example, transient pulse generators abstract correctly but cannot be used for logic verification:

```
Pulse = Clock && !Clock
```

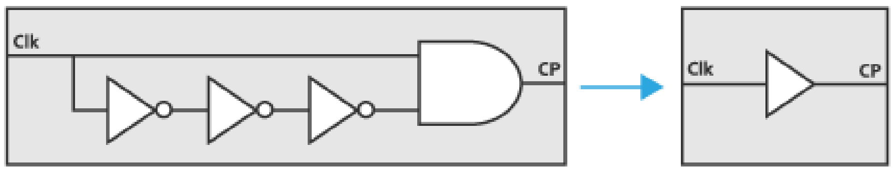


To fix this, manually disable the trigger to turn the pulse off:

```
remove x4
add tied signal 1 n4
```

Or use the abstraction model capability:

```
set abstract model -transform_pulse_generator_on
```



Analyzing Switch and Primitive Drive Strength

Use the SET XC command to analyze switch and primitive drive strength to achieve the most accurate logic function result. This technique can be applied to complex custom macros such as RAM and ROM and is essential to accurate verification of circuits with complex layer switch nets.

Diagram 1 shows an example of a layered switch network of a ROM function. This is the same logic function as the logic primitive network in Diagram 2. The `SET XC` command provides the analysis capability needed to successfully compare these functions.

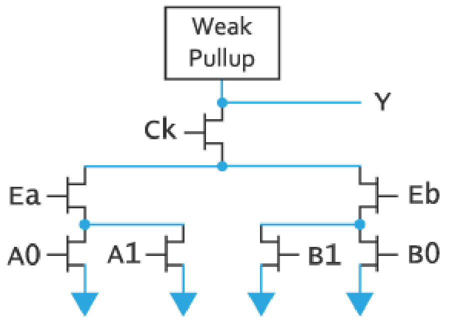


Diagram 1

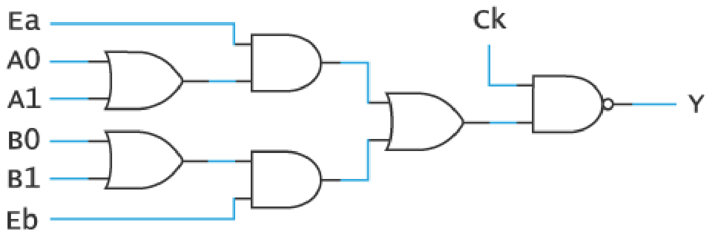


Diagram 2

Custom Menu

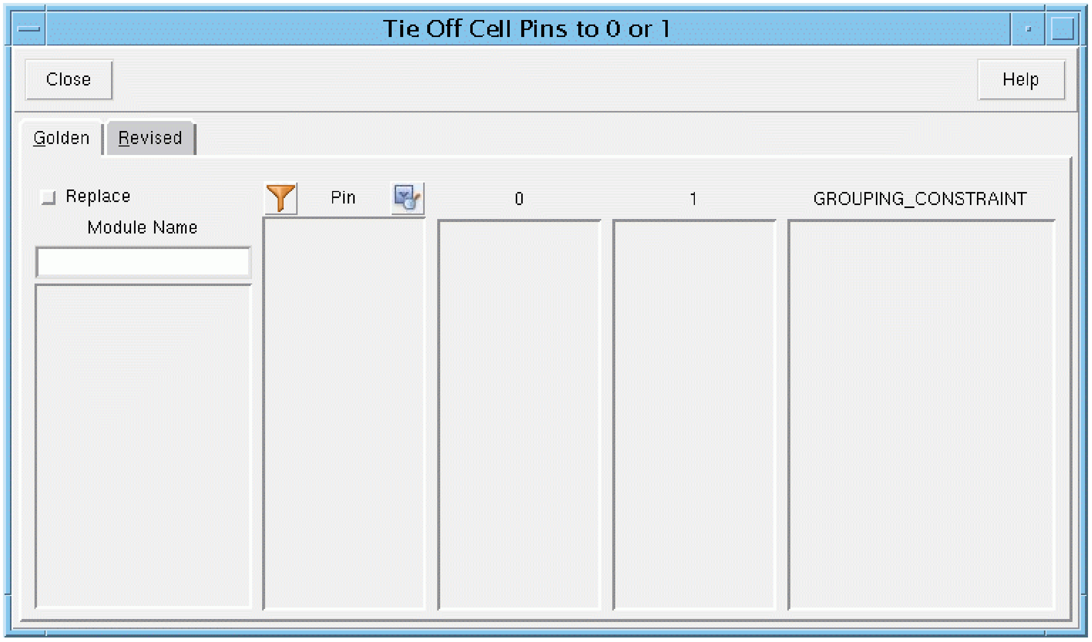
This section describes forms are accessible from the *Custom* menu. This menu contains the following sub-menus:

Note: These features require a Conformal GXL license.

General Setup

Tie Off Cell Pins to 0 or 1

Use the Tie Off Cell Pins to 0 or 1 form to add and delete pin constraints to primary input pins. To open this form, choose *Custom – General Setup – Tie Off Cell Pins to 0 or 1.*



For each list there are four columns with the headings: *Pin*, *0*, *1*, and *GROUPING_CONSTRAINT*. The primary input list is shown in the Pin column. Each primary input is either a system class primary input (*S:* `name`) or a user-defined class primary input (*U:* `name`).

Selecting Primary Inputs

In the following procedures, you are asked to select primary inputs. Use any of the following procedures to select primary inputs:

- Click a primary input to select it.
- Click and drag the mouse over a group of adjacent primary inputs to select them.
- Click the first primary input in a group, press and hold the Shift key, and click the final primary input in a group to select the entire group.

- Press the `Ctrl`-key and click a primary input to add it to the selected group.

The following procedures explain how to add and delete pin constraints.

Adding a Pin Constraint to a Primary Input

Use the following procedure to add a constraint to a single primary input using the Pin Constraints form:

1. In the *Pin* column, click a primary input to select it.
2. Right-click and choose the *Constraint 0* or *Constraint 1* constraint from the pop-up menu.

   The selected primary input appears in the appropriate column.

Adding a Constraint to a Group of Primary Inputs

Use the following procedure to add a constraint to a group of primary inputs using the Pin Constraints form:

1. In the *Pin* column, select multiple pins with one of the methods described in "Selecting Primary Inputs" on page 319.
2. Right-click to open the pop-up menu and choose a constraint.

Deleting Pin Constraints

Use the following procedure to delete one or all constraints using the Pin Constraints form:

1. Click a primary input in the *0*, *1*, or *GROUPING_CONSTRAINT* column to select it.
2. Right-click and choose one of the following from the pop-up menu:

   To delete a constraint from the selected pin, choose *Delete Pin Constraint*. Conformal removes the pin constraint. And in the case of *GROUPING_CONSTRAINT*, Conformal deletes the entire group.

   To delete all constraints, choose *Delete All Pin Constraints*. Conformal deletes all constraints from all columns.

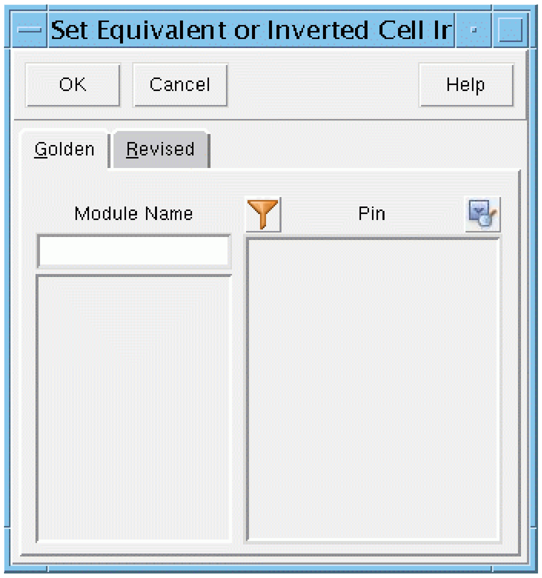
Sorting Pin Lists

You can alphabetically sort the primary input lists.

1. Right-click in the *Pin*, *0*, or *1* column.
2. Choose *Sort* from the pop-up menu.

Set Equivalent or Inverted Cell Input Pins

Use the Set Equivalent or Inverted Cell Input Pins form to add and delete pin equivalences. To open this form, choose *Custom – General Setup – Set Equivalent or Inverted Cell Input Pins*.



Module Name	Specifies the module that needs to be updated pins of that module are converted to a bus.
Pin	Specifies the pins of that module are converted to a bus.

The primary input lists for each of the Golden and the Revised designs are displayed in their respective columns. Conformal displays added pin equivalences below the target primary input with a connecting line. Inverted pin equivalences are denoted with (-) following the primary input name.

Adding a Pin Equivalence

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

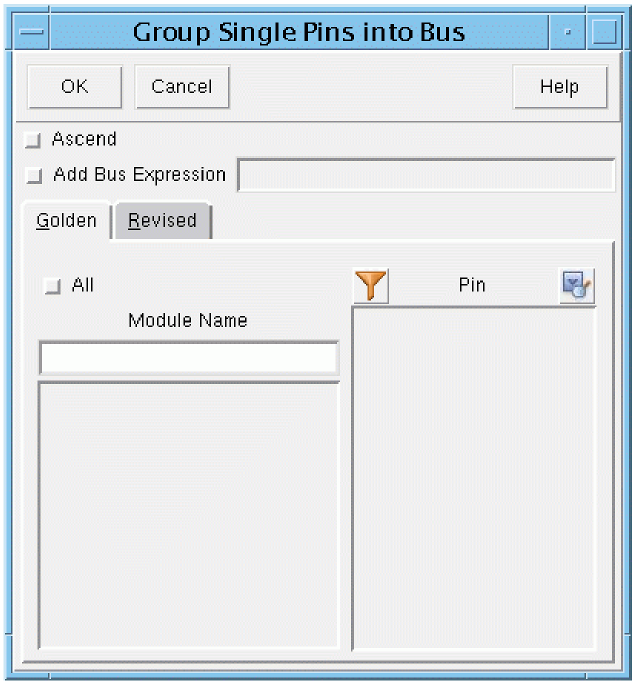
   The font color of the selected primary input changes to red to show that it is the target primary input.
3. Click the second primary input (in the same column) that must be equivalent to the target primary input.
4. Right-click and choose *Add Pin Equivalence* or *Add Invert Pin Equivalence* from the pop-up menu.

Sorting the Primary Input Lists

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

Group Single Pins into Bus

Use the Group Single Pins into Bus form, or the `PIN GROUP` command, to combine a group of single nets or pins into a bus. To open the Pin Group form, choose *Custom – General Setup – Group Single Pins into Bus.*



The Conformal software uses the following two default patterns to group pins or nets into busses:

- `Name[#]`
- `Name<#>`

For example, nets `blb[3] blb[4] blb[5]` will be grouped into bus `blb[5:3`], and pins `wladd<1> wladd<2> wladd<3>` will be grouped into bus `wladd[3:1`].

Ascend	Defines the bus in ascending numerical order. By default, buses are defined in descending numerical order.

Add Bus Expression	Specifies expression(s) for rules on signals to bus. You can specify your own renaming mapping of specific names to two default patterns, so that it recognizes those names as buses also.
	For example:
	`"mybus_%d_bar" "mybus_bar[@1]"`
	maps the following names into the first default bus name:
	`mybus_12_bar mybus_13_bar mybus_14_bar => mybus_bar[12] mybus_bar[13] mybus_bar[14]`
	then the renamed names will be further grouped into bus `mybus_bar[14:12]`

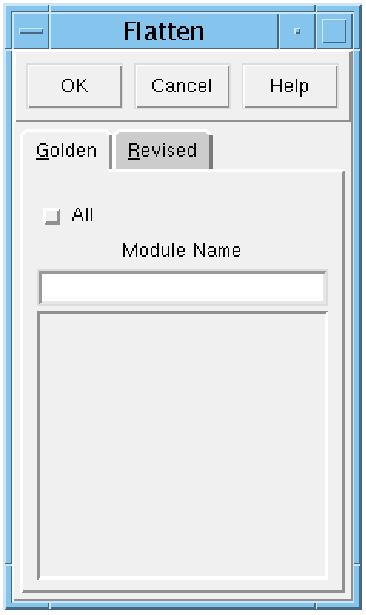
All	Specifies that when pins of a module are converted to a bus, all instantiations of that module need to be updated.
Module Name	Specifies the module that needs to be updated pins of that module are converted to a bus.
Pin	Specifies the single nets or pins to be grouped into a bus. Right-click and choose *Group Pin* from the pop-up menu to open the form to create a bus.

Flatten

Use the Flatten form, or the `FLATTEN` command, to remove all hierarchy on a specified module or for all modules in the database. If you do not specify one or all modules, Conformal flattens the root module by default. Thus, this expands all of the gate primitive or transistor primitive devices into the cell that is being flattened.

To open the Flatten form, do the following:

➤ Choose *Custom – General Setup – Flatten*

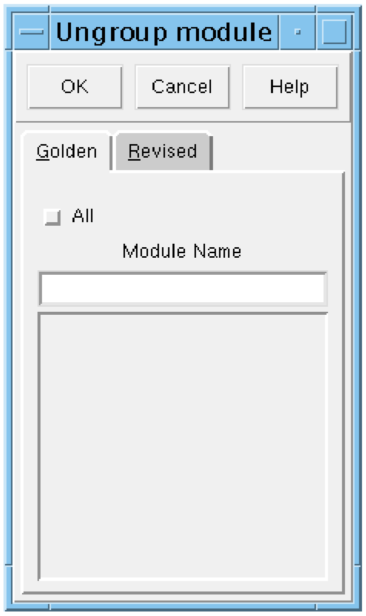


All	Flattens all modules within the given defaults.
Module Name	Specifies the module to flatten. By default, the root module is flattened. In the module list, right-click on a name and choose *Flatten* from the pop-up menu to flatten the module.

Ungroup Module

Use the Ungroup Module form, or the `RESOLVE` command, to ungroup a module in the Golden or Revised design hierarchy. Resolving or ungrouping is the process of eliminating a module and promoting its content up one level of the hierarchy.

To open the Ungroup Module form, choose *Custom – General Setup – Ungroup Module.*

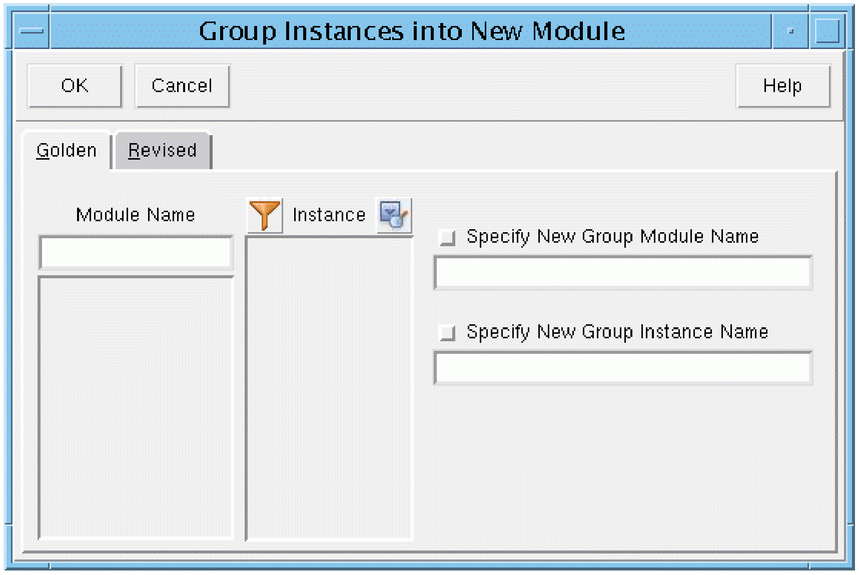


All	Resolves all modules within all hierarchies of the specified design.
Module Name	Specifies the module for which to resolve its hierarchy. In the module list, right-click on a name and choose *Resolve* from the pop-up menu to ungroup the module in the design hierarchy.

Group Instances into New Module

Use the Group Instances into New Module form, or `GROUP` command, to group instances together so that they become a new submodule. This applies to submodules, latches, registers, gates, and transistors.

To open the Group Instances form, choose *Custom – General Setup – Group Instances into New Module*

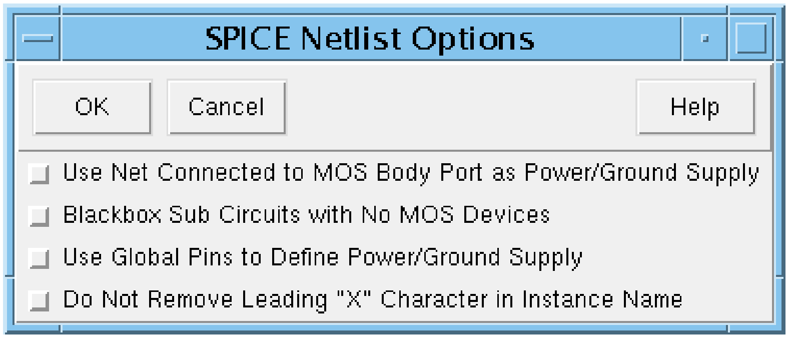

Group Instances into New Module
OK
Cancel
Help
Golden
Revised
Module Name
Instance
Specify New Group Module Name
Specify New Group Instance Name

Module Name	Specifies a module for which to apply the grouping.
Instance	Specifies the instances to group. Right-click on an instance name and choose *Group Instance* from the pop-up menu to group instances into a new submodule.
Specify New Group Module Name	Specifies the name of the new module.
Specify New Group Instance Name	Specifies the instance name for the new module.

Custom Setup

SPICE Netlist Options

Use the SPICE Netlist Options form, or the `SET SPICE OPTION` command, to specify options for reading the SPICE netlist design. To open this form, choose *Custom – Custom Setup – SPICE Netlist Options.*



- [ ] Use Net Connected to MOS Body Port as Power/Ground Supply
- [ ] Blackbox Sub Circuits with No MOS Devices
- [ ] Use Global Pins to Define Power/Ground Supply
- [ ] Do Not Remove Leading "X" Character in Instance Name

Use Net Connected to MOS Body Port as Power/Ground Supply

Identifies nets connected to PMOS bulk terminals as power and nets connected to NMOS bulk terminals as ground.

Blackbox Sub Circuits with No MOS Devices

Specifies that `SUBCKT` contains no transistors and will be treated as a blackbox.

Use Global Pins to Define Power/Ground Supply

Specifies that extra ports for `GLOBAL` signals will be created for `SUBCKT`.

Do Not Remove Leading "X" Character in Instance Name

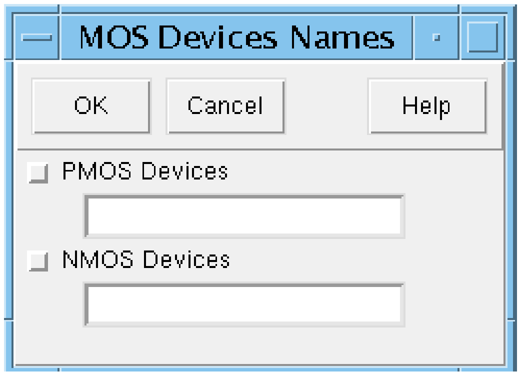
Specifies that the first character ’X’ of the name of instance will not be retained.

MOS Devices Name

Use the MOS Devices Names form, or the `SET MOS MODEL` command, to specify the MOS model names used in SPICE. You can then re-read the SPICE netlist.

When reading in SPICE netlists, the parser automatically identifies transistor model names as PMOS and NMOS types. However, if you have models that were not defined using `.MODEL` statements, the parser identifies them as `ERROR`. Instead of altering your SPICE file, you can use this form.

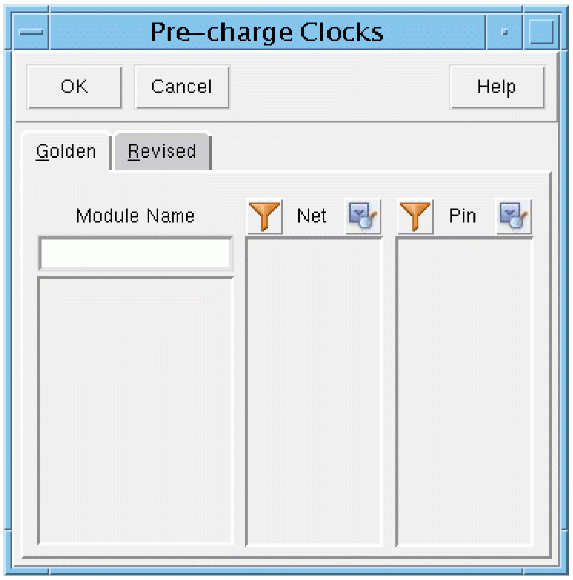
To open the MOS Devices Name form, choose *Custom – Custom Setup – MOS Devices Names*.



PMOS Devices	Defines the model name as a P-Channel device.
NMOS Devices	Defines the model name as an N-Channel device.

Pre-charge Clocks

Use the Pre-charge Clocks form, or the `ADD NET ATTRIBUTE` and `ADD CLOCK` commands, to specify or delete primary inputs as clocks to transistor-MOS. To open this form, choose *Custom – Custom Setup – Pre-charge Clocks*.



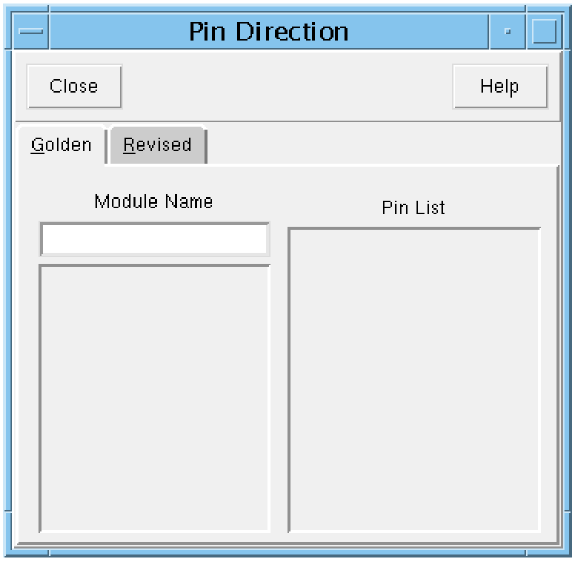
Click the *Golden* or *Revised* tab to switch between the two lists. The primary input list is shown in the *Pin* column. Each primary input is either a system class primary input (S: name) or a user-defined class primary input (U: name).

To add a clock to a primary input, click a primary input under the *Net* or *Pin* column to select it, then right-click to open the pop-up menu and select *Add Clock 0* or *Add Clock 1*.

To alphabetically sort the primary input lists, right-click in one of the columns and choose *Sort* from the pop-up menu.

Module Pin Direction

Use the Pin Direction form to assign direction to module boundary pins in the Golden or Revised design. To open this form, choose *Custom – Custom Setup – Module Pin Direction*.



Click the *Golden* or *Revised* tab to switch between the two pages, where for each page, there is a *Module Name* and *Pin List* column. All of the design's modules are listed in the *Module Name* column. All of the selected module's boundary pins are displayed with their current direction (*IN*, *OUT*, or *IO*) in the *Pin List* column.

To sort the displayed names, right-click in the column you want to sort to open the pop-up menu and choose *Sort*.

To assign pin direction to a module, do the following:

1. Double-click a module name in the *Module Name* column.

   The pin names appear in the *Pin List* column.

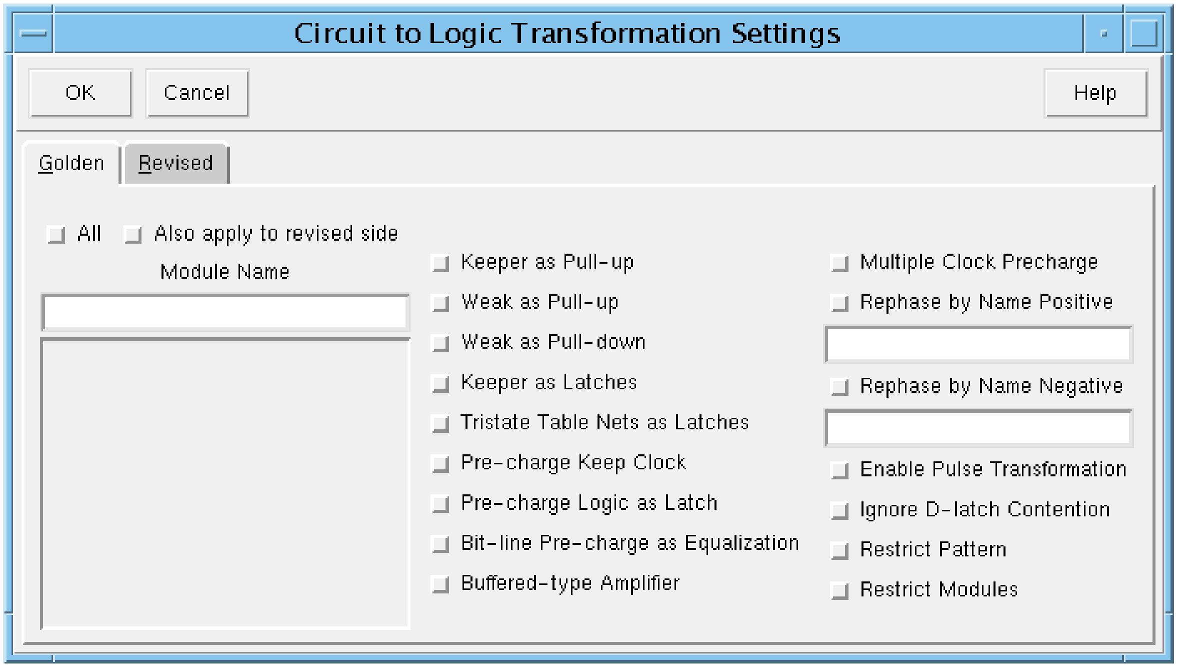
2. Click a pin in the *Pin List* column to select it.

3. Right-click to open the pop-up menu and choose *IN*, *OUT*, or *IO* to assign a new pin direction.

   The *Pin List* column updates with the new pin direction assignment.

Circuit to Logic Transformation Settings

Use the Circuit to Logic Transformation Settings form, or the `SET ABSTRACT MODEL` command options, to specify certain conditions for abstracting transistor logic. To open this form, choose *Custom – Custom Setup – Circuit to Logic Transformation Setting*.



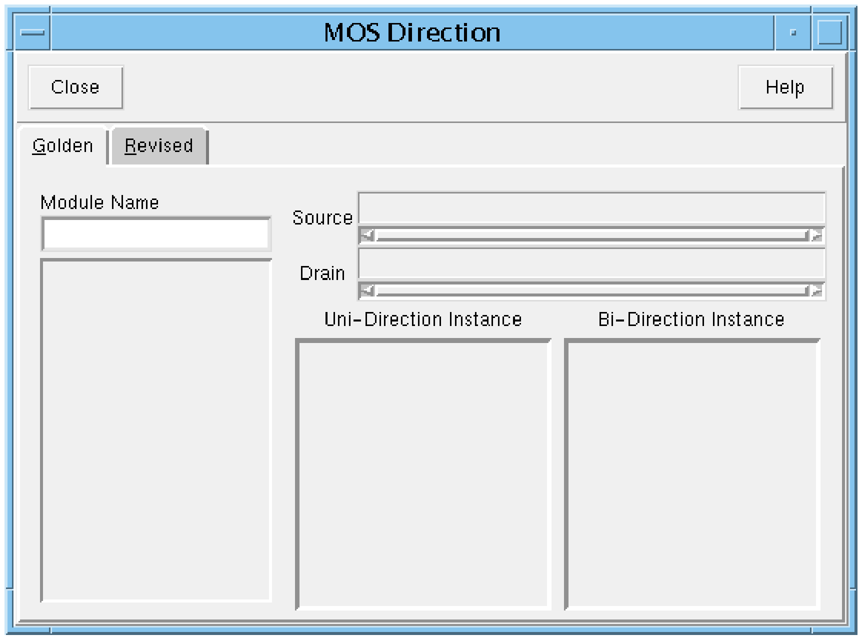
All	Abstracts transistor logic from all modules within the given defaults.
Also Apply to Golden/Revised Side	Applies the same abstraction conditions for the Golden or Revised design.
Module Name	Specifies the modules to which to abstracts transistor logic.
Keeper as Pull-up	Regards charge keepers as weak pull-up devices.
Weak as Pull-up	Regards devices that are tied to PMOS as weak devices.
Weak as Pull-down	Regards devices that are tied to NMOS as weak devices.
Keeper as Latches	Regards charge keepers as latches.
Tristate Table Nets as Latches	Regards tristate table nets as latches.

Pre-charge Keep Clock	For domino logic, regards pre-charge clocks as part of the logic function. This option includes the defined pre-charge clock in the abstracted logic function (the default behavior removes the defined pre-charge clock from the abstracted logic). When you use this option, the resulting logic is equivalent to RTL that explicitly models the pre-charge condition, rather than RTL that models only the evaluate function. In the latter, the output function is not defined during pre-charging.
Pre-charge Logical as Latch	Abstracts pre-charge logic functions as a latch. This assumes that data input is stable in active clocks.
Bit-line Pre-Charge as Equalization	Handles circuits that include bit-line pre-charge, and equalization.
Buffered-type Amplifier	Handles the following portions of a circuit: buffered-type sense amplifiers, level shifters, pre-charge, and equalization.
Multiple Clock Precharge	Propagates clocks through logic gates which have more than one clock input.
Rephrase by Name Positive	Gives logic abstraction hint about the desired phase of state elements, such as D-latches and DFFs. When abstracting state elements, logic abstraction will choose a phase for each state element, where its name specifies the net which will be driven by the 'Q' pin, if possible. If this is not possible, then abstraction will try to choose a net for the 'Qn' pin which has a name specified by the *Rephrase by Name Negative* option. **Note:** The `SET MAPPING METHOD` command's `-phase` option will allow mapping and comparison of state elements with different phases in the Golden and Revised designs. Consider running `SET MAPPING METHOD -phase` before using this option, as it requires less effort.
Rephrase by Name Negative	Specifies the net which will be driven by the 'Qn' pin, if possible.
Enable Pulse Transformation	Enables pulse transformation.

Ignore D-latch Contention	Continues to form the D-Latch, even if contention on a net is detected. By default, Circuit Logic Transformation stops the execution of abstraction of latches and flip-flips (state elements) when a power to ground through a stack of active ON transistors is possible. Use this option to report the short and continue to abstract the state element.
Restrict Pattern	Restricts the pattern to the specified modules specified and restricts the specified modules to use only those patterns. Other unspecified modules can be abstracted using other patterns not associated with that or any other module. By default, the pattern and module linkage and abstraction uses only patterns for that module, but still can use that pattern and any pattern for modules that are not defined.
Restrict Modules	Restricts the pattern matching to the specified modules.

MOS Direction

Use the MOS Direction form to add and delete unidirection to bidirectional MOS devices. To open this form, choose *Custom – Custom Setup – Transistor Logic Direction Settings*.



Click the *Golden* or *Revised* tab to switch between the two lists.

Module Name	Specifies the module to adds unidirection to all bidirectional MOS devices. All of the design's modules are shown in the *Module Name* column.
Source	Displays the source pin name for the MOS direction.
Drain	Displays the drain pin name for the MOS direction.
Uni-Direction Instance	Displays all unidirection MOS devices.
Bi-Direction Instance	Displays all bidirectional MOS devices.

To add a transistor-MOS instance direction, do the following:

1. Double-click a module name to show the name in the *Module Name* field.
2. In the *Bi-Direction Instance* column, double-click an instance to select it.

This shows the source pin and drain pin in the respective *Source* and *Drain* fields.

3. Right-click in the *Bi-Direction Instance* display area to open the pop-up menu and choose *Add Direction from Source to Drain* or *Add Direction from Drain to Source*.

   The selected instance moves to the *Uni-Direction Instance* column.

To delete a transistor-MOS instance direction, do the following:

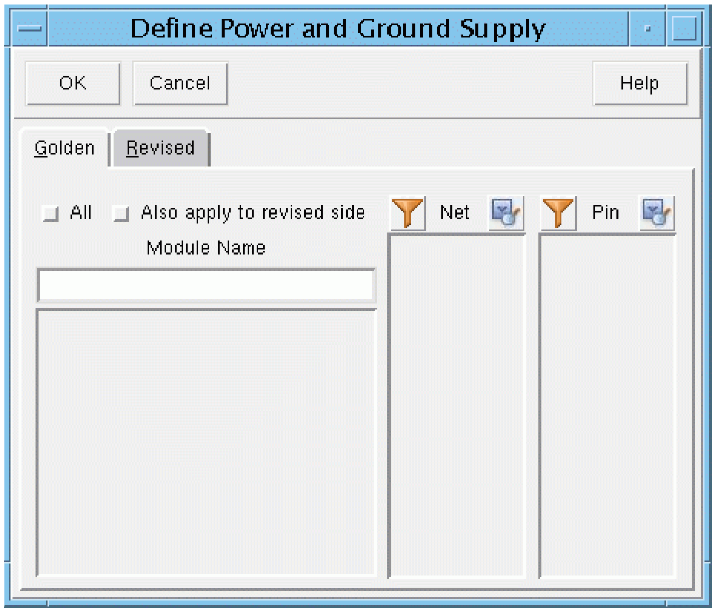
1. Double-click a module name to show the name in the *Module Name* field.
2. Click an instance name in the *Uni-Direction Instance* column to select it.
3. Right-click to open the pop-up menu and choose *Delete MOS Direction*.

   This removes the transistor-MOS direction and moves the instance name to the *Bi-Direction Instance* display list.

To alphabetically sort module and MOS direction instance lists, right-click in the appropriate column to open the pop-up menu and choose *Sort*.

Define Power and Ground Supply

Use the Define Power and Ground Supply form, or the `ADD SUPPLY` command, to define power and ground ports of a module or the global power and ground signals for the entire design. To open this form, choose *Custom – Custom Setup – Define Power and Ground Supply*.



Click the *Golden* or *Revised* tab to switch between the two lists.

All	Defines power and ground ports for all modules within the given defaults.
Also Apply to Golden/Revised Side	Defines power and ground ports for all modules in the Golden or Revised design.
Module Name	Specifies the module to apply the attribute setting.
Net and *Pin*	Specifies the module to apply the attribute setting.

To add a power or ground attribute, click a name in the *Net* or *Pin* column to select it, then right-click to open the pop-up menu and select *Add Power Attribute* or *Add Power Attribute*.

Data Entry Menu

Design

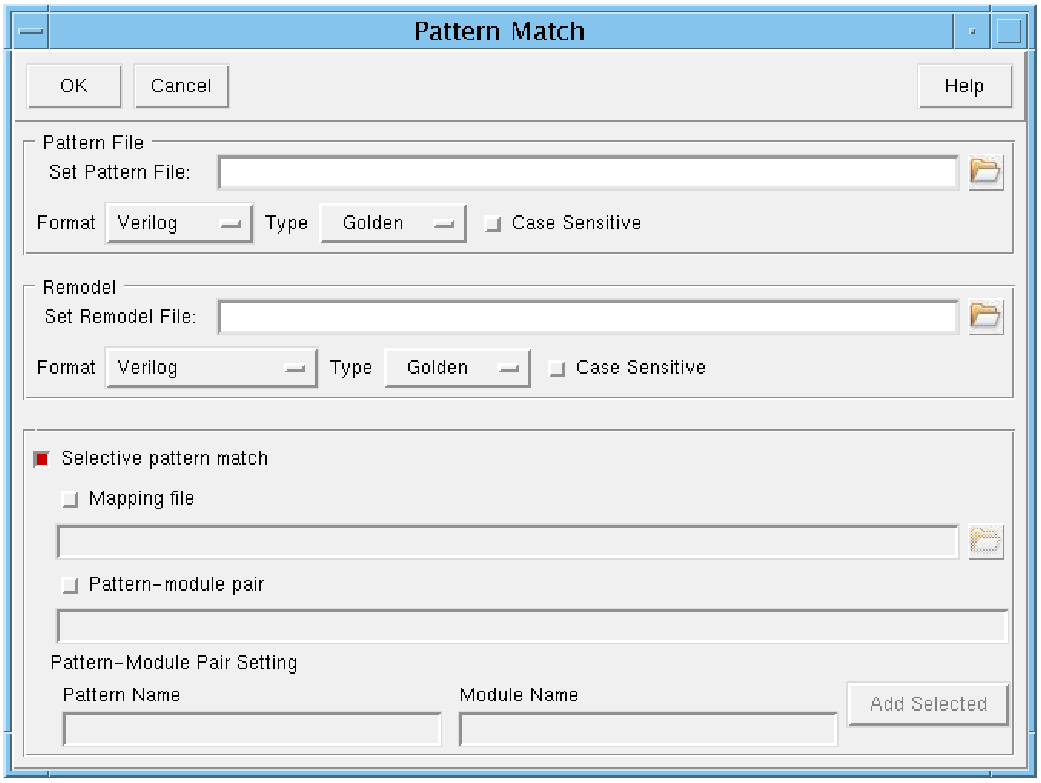
Use the Read Design form, or the `READ DESIGN` command, to select and configure the format of the design(s) to read in.

➤ Choose *Custom – Data Entry – Design*.

For more information, see Read Design Form on page 109.

Pattern Match

Use the Pattern Match form, or the `SET PATTERN MATCH` command, to set the pattern matching modules. To open this form, choose *Custom – Data Entry – Pattern Match*.



Set Pattern File:
Format: Verilog Type: Golden Case Sensitive

Set Remodel File:
Format: Verilog Type: Golden Case Sensitive

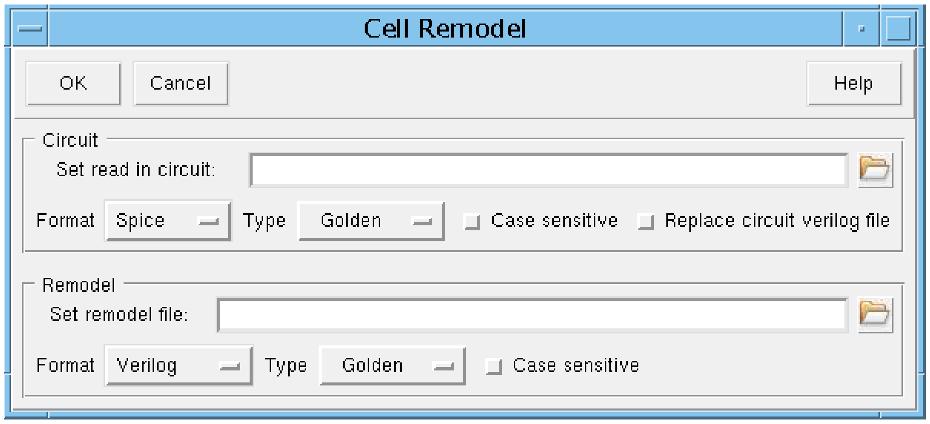
Selective pattern match
Mapping file
Pattern-module pair
Pattern Name
Module Name

Set Pattern File	Specifies that the name of the pattern (that was read in with the `READ PATTERN` command) that will apply to the module(s), as well as its format, type, and whether the files are handled as case-sensitive. You can enter the name of the file, or click the *Browser* icon select a file from the Log File browser window.
Set Remodel File	Specifies the name of the remodel file and its format, type, and whether the files are handled as case-sensitive. You can enter the name of the file, or click the *Browser* icon select a file from the Log File browser window.

Selective Pattern Match	Enables pattern matching. *Mapping File* specifies the file that contains the pairs of patterns and modules for pattern matching. The format of the mapping file is `<pattern_name> <module_name>`. You can enter the name of the file, or click the *Browser* icon select a file from the Log File browser window.
Pattern Module Pair Setting	Specifies a pattern module pair. Specify the pattern name and module name and select *Add Selected* to apply the setting.

Cell Remodel

Use the Cell Remodel form to read in a circuit and write it out to a Verilog format. To open this form, choose *Custom – Data Entry – Cell Remodel*.



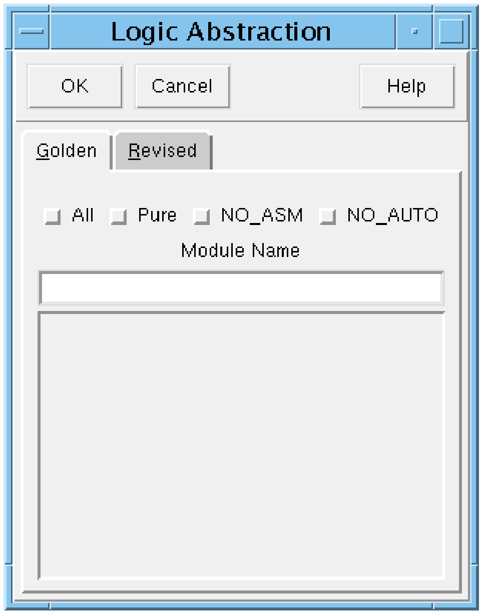
Set Read In Circuit	Specifies the name of the circuit to be remodeled, including its format, type (*Golden* or *Revised*), and whether the files are handled as case-sensitive. You can also select the *Replace Circuit Verilog File* option to write out the Golden or Revised design in Verilog format and replace the existing file's contents. You can enter the name of the file, or click the *Browser* icon select a file from the Design File browser window.
Set Remodel File	Specifies the name of the remodel version netlist file. You can enter the name of the file, or click the *Browser* icon select a file from the Design File browser window.

Application Menu

Logic Abstraction

Use the Logic Abstraction form to run functional analysis on circuit netlists, which can contain different devices, including transistors, gates, and state elements. The analysis abstracts a logically-correct gate and a state primitive model. Use the logic model and compare it to the RTL model for complete functional verification. You can also write out the logic model and use it during high-performance simulation or fault grading.

➤ Choose *Custom – Application – Logic Abstraction*.



The *Golden Module* and *Revised Module* columns list all of the relevant design's modules, which you can specify for transistor abstraction.

All	Abstracts logic information from all cells in the database, including cells that are not used by the current root module.
Pure	Performs basic gate abstraction, which is useful for debugging.
NO_ASM	Disables the Advanced State-element Modeling (ASM) algorithm. By default, Logic Abstraction enables the Advanced State-element Modeling (ASM) algorithm to analyze loop structure to produce better modeling of state elements, such as D-Latch, DFF, and bus-keeping I/O logic.

NO_AUTO	Does not invoke hierarchical analysis. By default, Logic Abstraction enables propagation of constants, pin constraints, non-inverted and inverted pin relationships across module boundaries.
Module Name	Specifies the module and its hierarchy for which to abstract logic information. Type the name in the field and press Enter to add it to the list.

To post abstraction information to the Transcript window, click the module name, right-click to open the pop-up menu, and choose *Abstract*.

To display the schematic view, click a module name to select it, right-click to open the pop-up menu, and choose *Schematic View*.

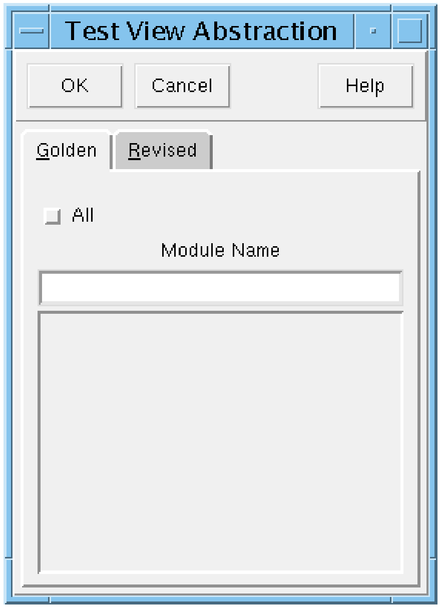
To alphabetically sort the module lists, right-click in the column to open the pop-up menu, and choose *Sort*.

Click *Refresh* to return the displayed modules to their original numerical order or to update the displayed hierarchy after you read in a design.

Test View Abstraction

Use the Test View Abstraction form to run structurally accurate abstraction. With this feature, only limited boolean simplification is done for abstraction. As a result, the gate-level structure of the original logic is preserved as much as possible after abstraction.

➤ Choose *Custom – Application – Test View Abstraction*.



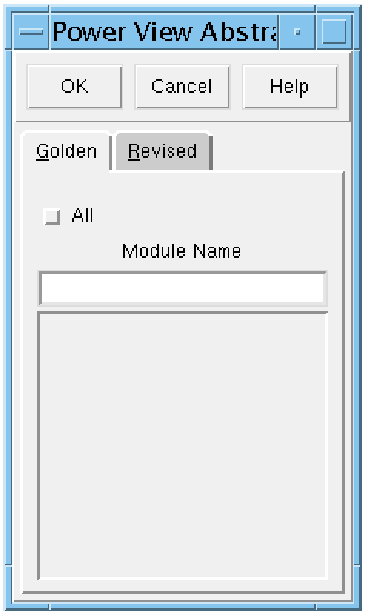
The *Golden Module* and *Revised Module* columns list all of the relevant design's modules, which you can specify for transistor abstraction.

All	Abstracts test view information from all cells in the database, including cells that are not used by the current root module.
Module Name	Specifies the module to abstract test view information. double left-click on the name in list to add it to this field.

Power View Abstraction

Use the Power View Abstraction form, or the `ABSTRACT LOGIC -POWER_VIEW` command, to run power-aware abstraction. With this feature, the connectivity of the power and ground pins are retained. Only limited boolean simplification is run to ensure that the abstraction results are as similar as possible to the original switch-level netlist.

➤ Choose *Custom – Application – Power View Abstraction*.



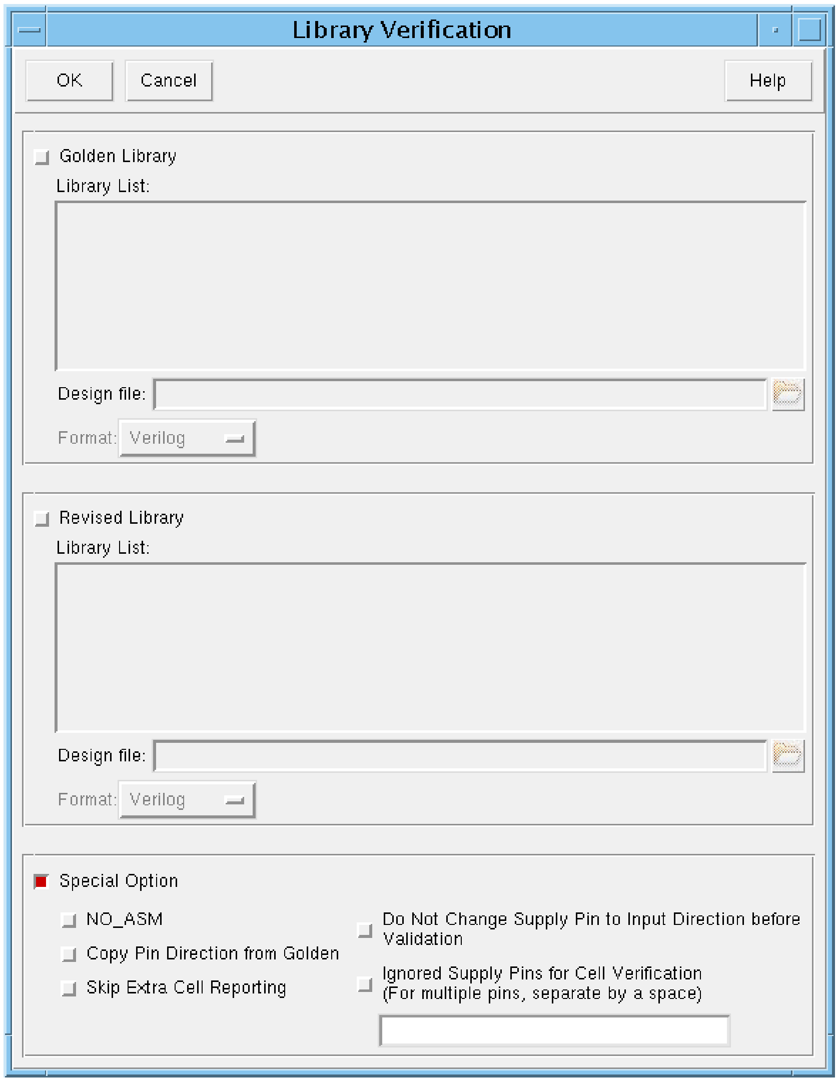
The *Golden Module* and *Revised Module* columns list all of the relevant design's modules, which you can specify for power-aware abstraction.

All	Abstracts power-aware information from all cells in the database, including cells that are not used by the current root module.
Module Name	Specifies the module and its hierarchy for which to abstract power-aware information. Type the name in the field and press Enter to add it to the list.

Library Verification

Use the Library Verification form, or the `VALIDATE LIBRARY` command, to compare all top-level cells with matching names. Conformal abstracts the modules on the SPICE side before comparison. This application is for library verification during library design.

➤ Choose *Custom – Application – Library Verification*.



Golden Library
Library List:
Design file:
Format: Verilog

Revised Library
Library List:
Design file:
Format: Verilog

Special Option
NO_ASM
Copy Pin Direction from Golden
Skip Extra Cell Reporting
Do Not Change Supply Pin to Input Direction before Validation
Ignored Supply Pins for Cell Verification (For multiple pins, separate by a space)

The *Golden Module* and *Revised Module* columns list all of the relevant design's modules, which you can specify for power-aware abstraction.

Golden Library	Validates the Golden database.

Revised Library	Validates the Revised database.
No ASM	Disables the Advanced State-element Modeling (ASM) algorithm. ASM (the default) helps to analyze loop structure to produce better modeling of state elements, such as D-Latch, DFF, and bus-keeping I/O logic.
Copy Pin Direction From Golden	Copies the pin directions from the Golden design to the Revised design. This is for all pins within the library cells being validated.
Skip Extra Cell Reporting	Skips reporting the cells that only exist in the Golden or Revised design.

RAM Primitive

Use the RAM Primitive window to generate Verilog RAM primitive models.

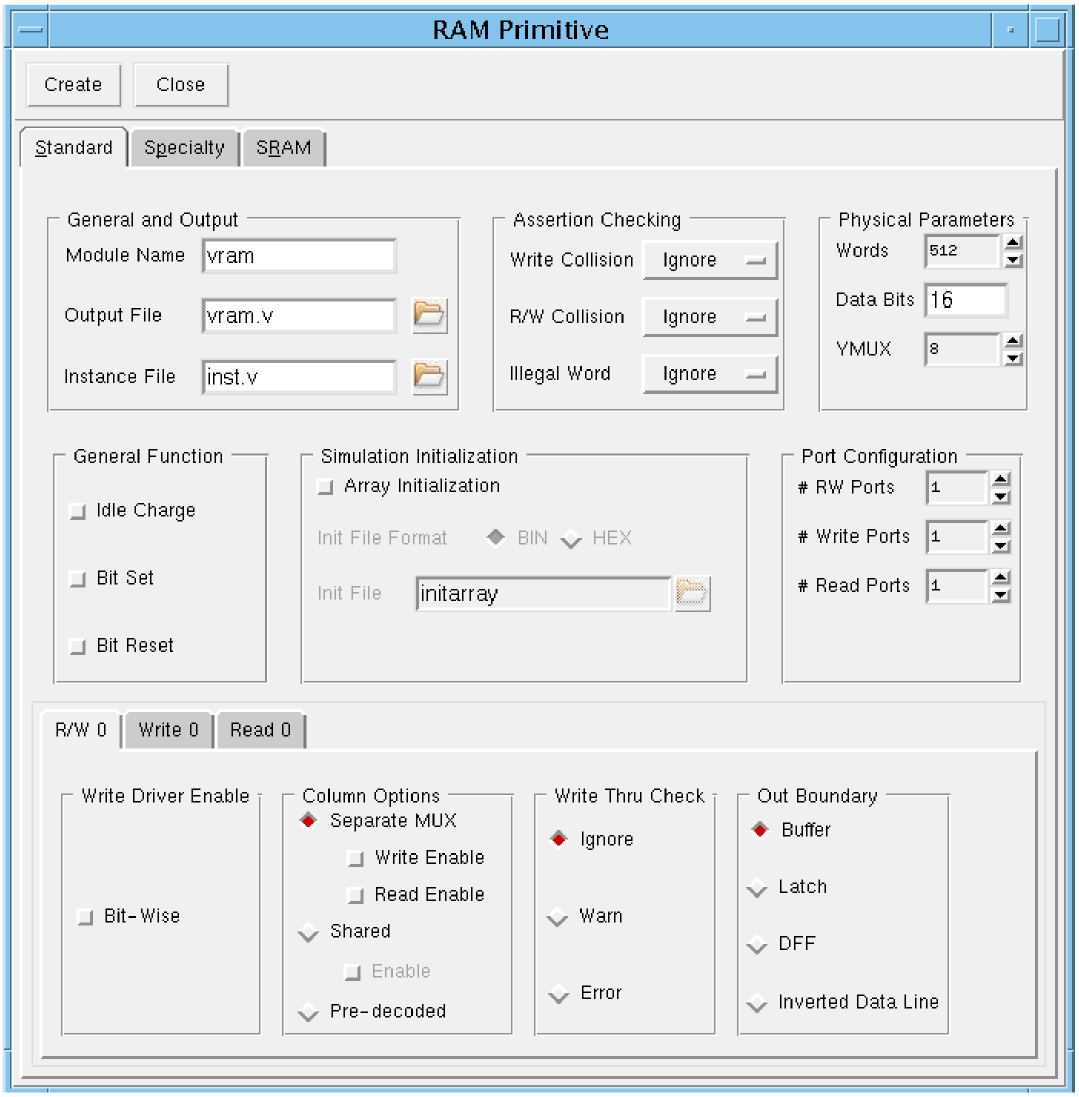
The Place form contains the following two pages:

- RAM Primitive - Standard on page 348
- RAM Primitive - Specialty on page 349
- RAM Primitive - SRAM on page 350
- ROM Primitive on page 355

RAM Primitive - Standard

Use the RAM Primitive *Standard* page to create standard memory primitives.

➤ Choose *Custom – RAM Primitive*, and select the *Standard* tab.



General and Output
Module Name: vram
Output File: vram.v
Instance File: inst.v

Assertion Checking
Write Collision: Ignore
R/W Collision: Ignore
Illegal Word: Ignore

Physical Parameters
Words: 512
Data Bits: 16
YMUX: 8

General Function
☐ Idle Charge
☐ Bit Set
☐ Bit Reset

Simulation Initialization
☐ Array Initialization
Init File Format: [✓ BIN] HEX
Init File: initarray

Port Configuration
RW Ports: 1
Write Ports: 1
Read Ports: 1

R/W 0 | Write 0 | Read 0

Write Driver Enable
☐ Bit-Wise

Column Options
[✓ Separate MUX]
☐ Write Enable
☐ Read Enable
Shared
☐ Enable
Pre-decoded

Write Thru Check
[✓ Ignore]
Warn
Error

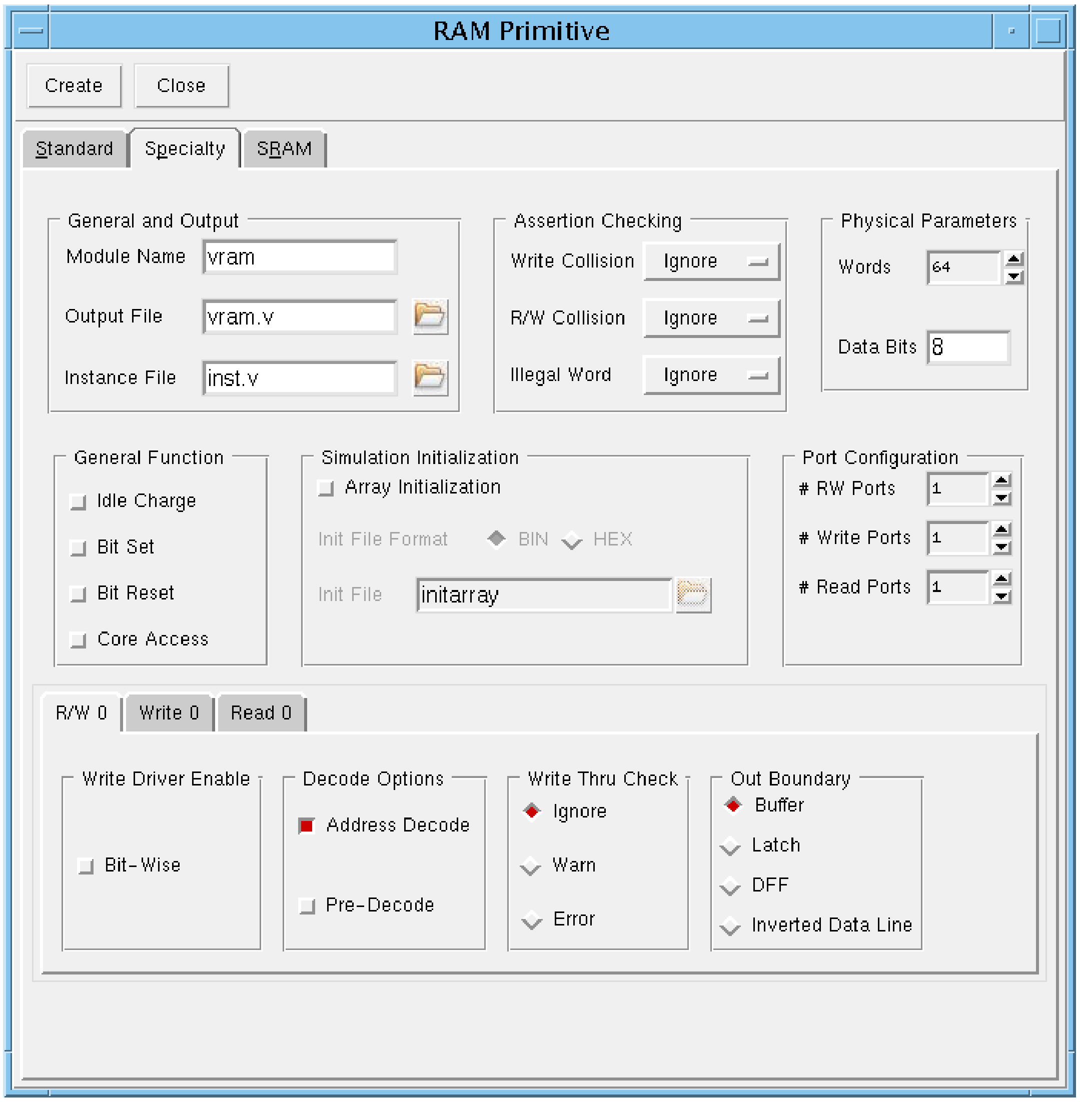
Out Boundary
[✓ Buffer]
Latch
DFF
Inverted Data Line

Most of the RAM Primitive forms's Speciality page options are the same as in the *Specialty* and *SRAM* pages. See RAM Primitive Form Fields and Options on page 351 for more information. Options that are unique to the Standard page are identified in the descriptions.

RAM Primitive - Specialty

Use the RAM Primitive *Specialty* page to create specialty memory primitives, such as CAMs, using the RAM Primitive window. Unlike standard memory primitives, with specialty memory primitives there is no column muxing, you can make the memory array core directly accessible, and pre-address decode provides direct access to word lines

➤ Choose *Custom – RAM Primitive*, and select the *Specialty* tab.



General and Output
Module Name: vram
Output File: vram.v
Instance File: inst.v

Assertion Checking
Write Collision: Ignore
R/W Collision: Ignore
Illegal Word: Ignore

Physical Parameters
Words: 64
Data Bits: 8

General Function
☐ Idle Charge
☐ Bit Set
☐ Bit Reset
☐ Core Access

Simulation Initialization
☐ Array Initialization
Init File Format: [✓ BIN] HEX
Init File: initarray

Port Configuration
RW Ports: 1
Write Ports: 1
Read Ports: 1

Write Driver Enable
☐ Bit-Wise

Decode Options
☑ Address Decode
☐ Pre-Decode

Write Thru Check
[✓ Ignore] Warn Error

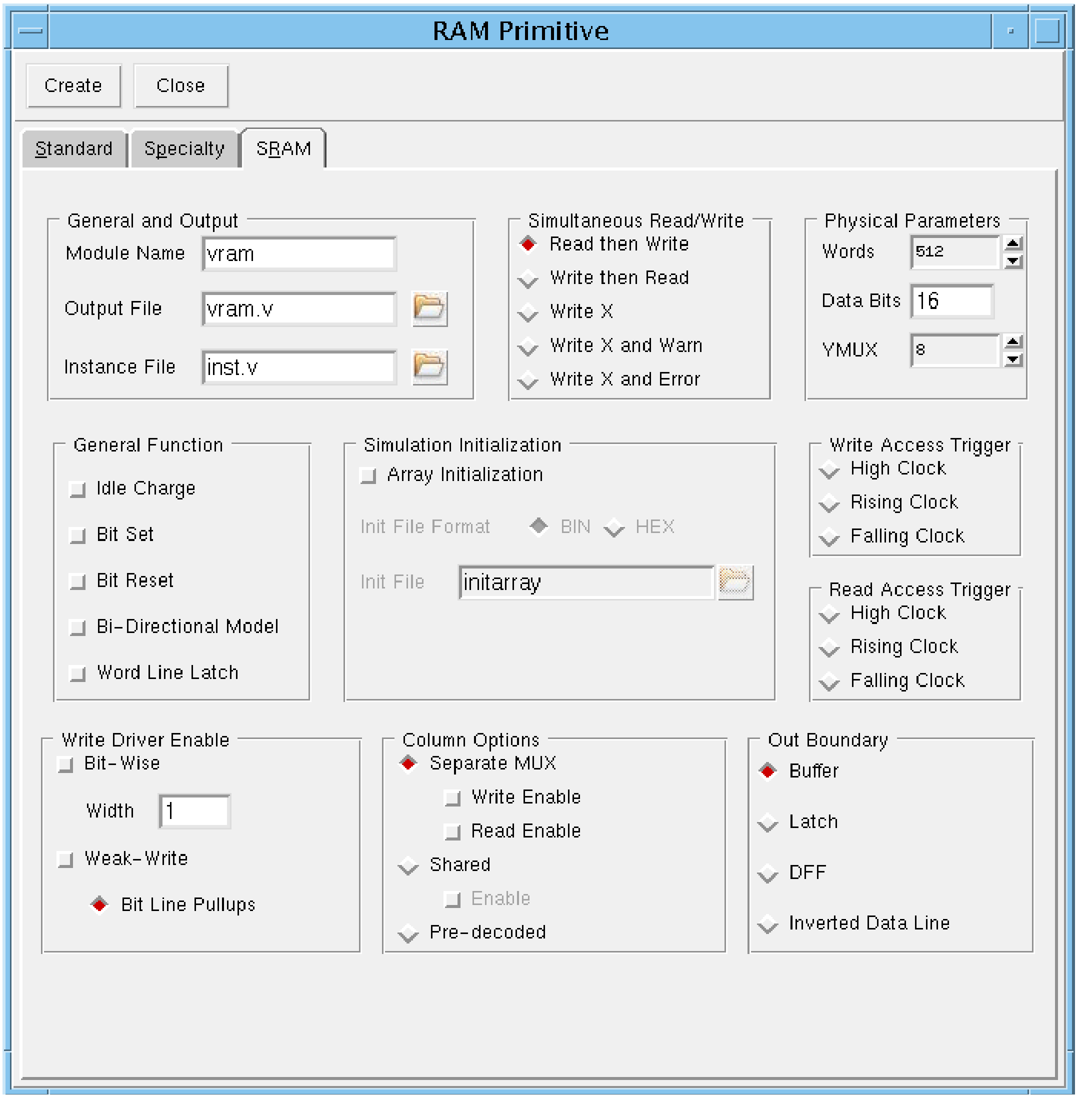
Out Boundary
[✓ Buffer] Latch DFF Inverted Data Line

Most of the RAM Primitive forms's Speciality page options are the same as in the *Standard* and *SRAM* pages. See RAM Primitive Form Fields and Options on page 351 for more information. Options that are unique to the Speciality page are identified in the descriptions.

RAM Primitive - SRAM

Use the RAM Primitive *SRAM* page to create static random access memory primitives. An SRAM family is a traditional single port RAM with custom read and write access to support high speed applications and multiple access in a single clock cycle (read/write or write/read). Bi-directional memory bit modeling support is available with this type of memory.

➤ Choose *Custom – RAM Primitive*, and select the *SRAM* tab.



RAM Primitive

Create | Close

Standard | Specialty | SRAM

General and Output
- Module Name: vram
- Output File: vram.v
- Instance File: inst.v

Simultaneous Read/Write
- [✓ Read then Write]
- Write then Read
- Write X
- Write X and Warn
- Write X and Error

Physical Parameters
- Words: 512
- Data Bits: 16
- YMUX: 8

General Function
- Idle Charge
- Bit Set
- Bit Reset
- Bi-Directional Model
- Word Line Latch

Simulation Initialization
- Array Initialization
- Init File Format: [✓ BIN] HEX
- Init File: initarray

Write Access Trigger
- High Clock
- Rising Clock
- Falling Clock

Read Access Trigger
- High Clock
- Rising Clock
- Falling Clock

Write Driver Enable
- Bit-Wise
- Width: 1
- Weak-Write
- [✓ Bit Line Pullups]

Column Options
- [✓ Separate MUX]
  - Write Enable
  - Read Enable
- Shared
  - Enable
- Pre-decoded

Out Boundary
- [✓ Buffer]
- Latch
- DFF
- Inverted Data Line

Most of the RAM Primitive forms's Speciality page options are the same as in the *Standard* and *Specialty* pages. See RAM Primitive Form Fields and Options on page 351 and R/W Sub-Page Fields and Options on page 353 for more information. Options that are unique to the SRAM page are identified in the descriptions.

RAM Primitive Form Fields and Options

The following describes all fields and options for all pages of the RAM Primitive form. The differences in each are noted.

General and Output

Module Name specifies the name of the module.

Output File specifies the name of the output file.

Instance File specifies the name of the instance file.

Assertion Checking (for the *Standard* and *Specialty* pages)

During Verilog simulation, this specifies the type of assertions to trigger for the occurrences: *Ignore* (the default), *Warning*, or *Error*. You can select one or more of the following:

- *Write Collision* for multiple write or red/write ports.
- *R/W Collision* between different read and write or read and read/write ports.
- *Illegal Word* does not use the entire address space.

Simultaneous Read/Write (for the *SRAM* page)

Specifies the simultaneous read/write behavior. The default is *Read then Write*.

Physical Parameters

Sets default values for the following physical parameters, which you can override during instantiation:

- *Words* specifies the value for the number of words (default is 512).
- *Data Bits* specifies the value for the number of bits per word (default is 16).
- *YMUX* default is 8. (This is for standard and SRAM primitives only.)

General Function	Specifies your memory primitive’s general functionality. You can select one or more of the following: ■ *Idle Charge* preserves pre-charge. Tristate bit lines are modeled as high value (1). ■ *Bit Set* specifies asynchronous set for one or more bits in the array for every address. ■ *Bit Reset* specifies asynchronous reset for one or more bits in the array for every address. ■ *Core Access* (for specialty primitives only) specifies access to every state of a pin. ■ *Bi-Directional Model* (for SRAM primitives only) enables bi-directional bit cell model during synthesis. ■ *Word Line Match* (for SRAM primitives only) specifies that latches are at the word line address inputs of the simulation model and at the word lines of the synthesized primitive.
Simulation Initialization	■ *Array Initialization* sets a simulation initialization of array. This is required to support Verilog simulation. ■ *Init File Format* specifies the format of your initialization file: *BIN* (the default) or *HEX*. ■ *Init File* specifies the name of the initialization file.
Write/Read Access Trigger (for the *SRAM* page)	Specifies in the simulation model how the read and write access is initiated: high, rising, or falling clock edge.
Port Configuration	Configures the number of ports. The maximum number for each port type is 8. This section controls the number pages located at the bottom of the RAM Primitive form. ■ *# RW Ports* specifies the number of read/write ports. This corresponds to the R/W # sub-tab. ■ *# Write Ports* specifies the number of write-only ports. This corresponds to the Write # sub-tab. ■ *# Read Ports* specifies the number of read-only ports. This corresponds to the Read # sub-tab.

Depending on what you specified in the *Port Configuration* section, Conformal displays sub-pages (tabs) for each port. Use these to configure each of your ports.

R/W Sub-Page Fields and Options

Write Driver Enable	*Bit-Wise* specifies a bit-wise driver write enable.
	For the *SRAM* page, select *Width* or *Weak-Write*.
	Bit Line Pullups generates memory primitive when -sram option is used also ??
Column Options (for the *Standard* page).	*Separate MUX* specifies that your memory writes and reads bit line data through separate column-select circuitry. Use the *Write Enable* or *Read Enable* options to specify a column decode enable input, either `<port>_wclk` or `<port>_rclk`, that can disable all column selection.
	Shared specifies that your memory writes and reads bit line data through the same column-select circuitry. Use the *Enable* sub-option to specify a column decode enable input, `<port>_colclk`, that can disable all column selection.
	Pre-decoded specifies that your memory writes and reads bit line data were previously decoded.
Decode Options (for the *Specialty* page).	*Address Decode* specifies word lines that ere decoded internally.
	Pre-Decode specifies word lines that were previously decoded.
Write Thru Checks	Specifies how to treat write thru checks. Select *Ignore*, *Warn*, or *Error*.
Out Boundary	Specifies whether the out boundary should be a *buffer*, D-latch flip-flop (*DFF*), or an *Inverted Data Line*.
	For memories that do not have column MUXes, the *Inverted Data Line* option brings the bit-line bar directly out to the output and calls it `<>port_doutB`. For memories with YMUXes, this option brings the bit-line bar out and calls it `<>port_doutB`.

Write Sub-Page Fields and Options

Write Driver Enable	Choose *Global* or *Bit-Wise*.
Write Driver Types	Choose *Buffer*, *Tristate*, *Precharge*, or *Discharge*.
Column Option (for the *Standard* page).	*Column Clock* specifies an enable clock for the column decoder.
Decode Options (for the *Specialty* page).	*Address Decode* specifies word lines that ere decoded internally.
	Pre-Decode specifies word lines that were previously decoded.

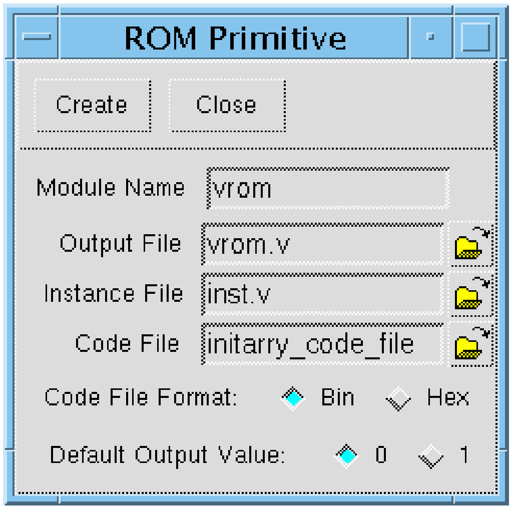
Read Sub-Page Fields and Options

X Decode Option (for the *Standard* page).	*Row Clock* specifies a clock enable for the row decoder.
Decode Options (for the *Specialty* page).	*Address Decode* specifies word lines that ere decoded internally.
	Pre-Decode specifies word lines that were previously decoded.
Column Option	*Column Clock* specifies a clock enable for the column decoder.
Bit Line Option *Bit Lines*	Choose *Differential*, *Single Wired-AND* or *Single Wired-OR*.
Out Boundary	Specify whether the out boundary should be a *buffer*, latch D flip-flop (*DFF*), or an *Inverted Data Line*.
	For memories that do not have column MUXes, the *Inverted Data Line* option brings the bit-line bar directly out to the output and calls it `<>port_doutB`. For memories with YMUXes, this option brings the bit-line bar out and calls it `<>port_doutB`.

ROM Primitive

Use the ROM Primitive window to generate Verilog ROM primitive models.

➤ Choose *Custom – ROM Primitive*.



Module Name: vrom
Output File: vrom.v
Instance File: inst.v
Code File: initarry_code_file
Code File Format: [✓ Bin] Hex
Default Output Value: [✓ 0] 1

Click *Create* to generate the memory primitive, or *Close* to cancel your settings.

Module Name	Specifies the name of the module.
Output File	Specifies the name of the output file. You can type the name, or use the *Open File* button to select an output file.
Instance File	Specifies the name of the instance file. You can type the name, or use the *Open File* button to select an output file.
Code File	Specifies the name of the code file. You can type the name, or use the *Open File* button to select an instance file.
Code File Format	Specifies a file format of *Bin* (the default) or *Hex*.
Default Output Value	Specifies an output value of 0 (the default) or 1.

14

Conformal ECO Designer Functionality and Methodology

Note: This feature requires a Conformal ECO XL or GXL license.

An Engineering Change Order (ECO) is a change to a design after it has already been processed, typically after place and route. There are two different types of ECOs: functional and non-functional. Functional ECOs change the functionality of the design. Non-functional ECOs do not change the design functionality and normally deal with timing, design rule, or signal integrity.

For information on the Conformal ECO methodology, refer to the *Conformal ECO User Guide*.

15

FPGA Capabilities and Process Flow

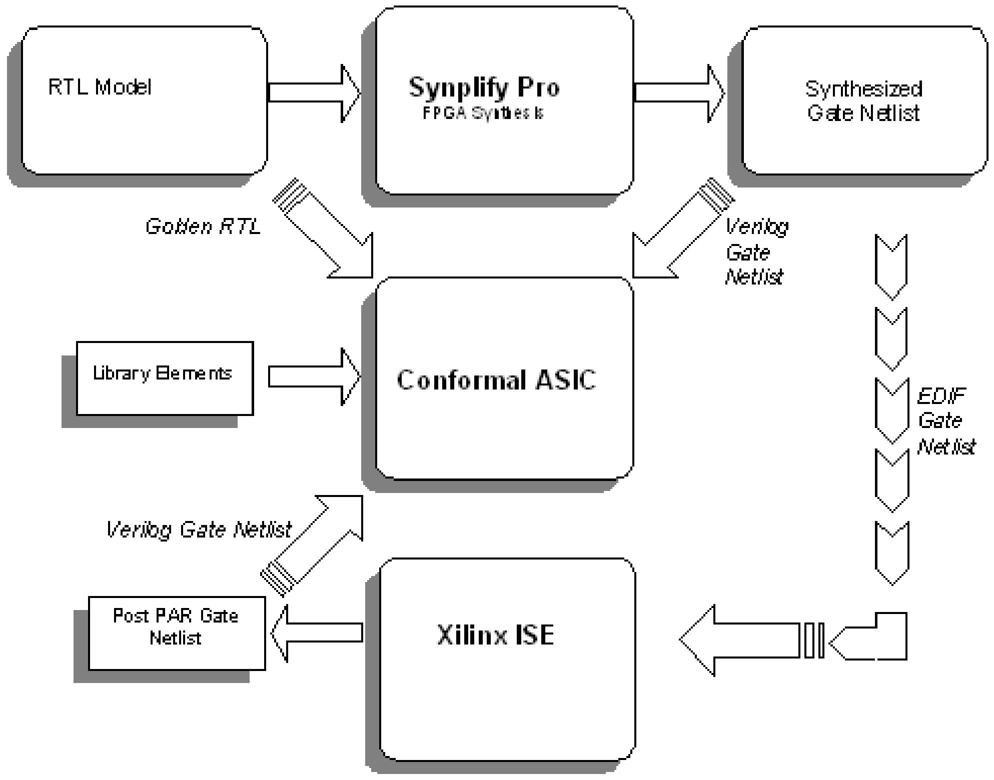
Overview

More and more of today’s designers are using programmable logic devices in their designs. FPGA designs have been closing the gap on their ASIC counterparts because of technological advancements in density and performance.

As the size and complexity for FPGA devices increase, conventional verification approaches that many FPGA designers have been using are no longer adequate. FPGA synthesis involves steps that are unique to FPGA devices, such as FSM recoding, aggressive sequential optimization, memory inference, and mapping to embedded on-chip functions. The Conformal Equivalence Checker’s FPGA feature uses FPGA-specific modeling and analysis techniques to check functional equivalencies during the FPGA implementation process.

The FPGA feature is targeted at designs created by the Synplicity FPGA synthesis tool, Synplify Pro. Better integration with FPGA synthesis means that FPGA runs verification smoother and more efficiently. The FPGA feature reads in the setup files created by Synplify Pro, thus creating a correct verification environment for the *front end*. Additionally, the FPGA feature addresses the *back end* by allowing you to verify gate netlists that have been through place and route (PAR) changes within the Integrated Synthesis Environment (ISE) from Xilinx. That is, with the FPGA feature you will compare the post-synthesis gate netlists against the post-PAR gate netlists, thereby allowing functional closure on your designs.

The following figure illustrates an overview of the FPGA flow.



FPGA Front-End Verification

This section details the FPGA front-end verification. During front-end verification, the FPGA feature compares the Golden RTL design with the post-synthesis Verilog gate netlist.

Requirements and Licensing

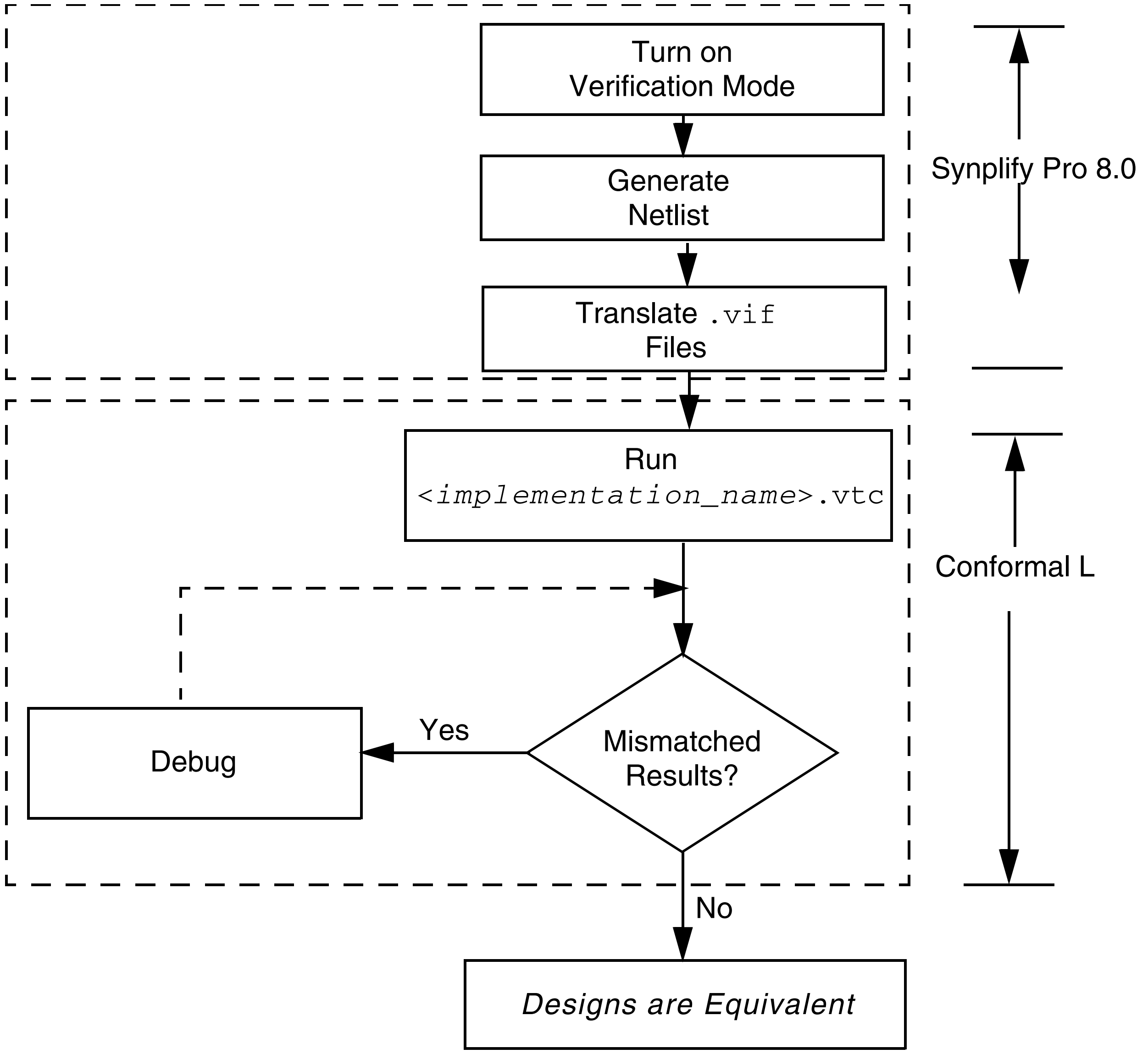
The FPGA flow requires Synplify Pro 8.0, Alliance™ Xilinx ISE 6.2i (or later), and Conformal L 5.0 (or later).

Front-End Verification Flow

A summary of the front-end process flow is as follows, shown in the following figure.

- Generate a post-synthesis gate netlist using Synplify Pro with the Verification Mode on.
- Translate the `.vif` files into Conformal command files.

- Run Conformal using the `/verif/`*`<implementation_name>`*`.vtc` dofile.
- Debug the designs if there are miscompares.



Generating a Gate Netlist

The first step in the FPGA flow is to generate a gate netlist using the Verification Mode in Synplify Pro.

To access the Verification Mode from Synplify Pro, do one of the following:

- Use the Synplify Pro GUI:

a. Choose *Project – Implementation Options*, or click the *Impl Options* button from the Synplify Pro main window.

The Options for Implementation window appears.

b. Select the *Device* tab.

c. Under Technology, select one of the following supported technologies:

- ❍ Xilinx Spartan, Spartan-II, Spartan-IIE, Spartan-3, Virtex, Virtex-II, Virtex-II Pro, Virtex-E.

d. Under *Device Mapping Options*, select *Verification Mode*.

e. Click *OK*.

- Add the following line to your synthesis project file:

```
set_option -verification_mode 1
```

Continue synthesizing the design after turning on the verification mode.

Comparing RTL Versus Post Synthesis Verilog Gate Netlist

Synplify Pro operates in a verification mode in which it generates scripts and gate netlists compatible with Conformal. This product records the optimizations it has performed during synthesis in a verification interface format (VIF) file. The interface is triggered by enabling the variables: `Impl Options...Verification Mode`, and `Impl Options...Implementation results...Write Verification Interface Format (VIF) file` in the GUI, or by inserting the following options in the synthesis project file:

```
set_option -verification_mode 1
set_option -write_vif 1
```

Because Conformal does not read the VIF file directly, Synplicity provides a Tcl script called `vif2conformal.tcl` to translate the content of this file to Conformal command file format. You can run this script automatically after logic synthesis using a new mechanism available in Synplify Pro v8.0. This mechanism provides a method to configure user-defined Tcl commands at different times during the Synthesis process.

You can specify Tcl commands in a file called `synhooks.tcl`, and set the environment variable `SYN_TCL_HOOKS` as follows:

```
SYN_TCL_HOOKS =<some path>/synhooks.tcl
```

The `synhooks.tcl` file template is available under the Synplify installation and can be modified as required.

The following portion from the `synhooks.tcl` file shows how to generate Conformal-specific side files automatically. The `vif2conformal` translator is available in the `<Synplify_install>/lib/` directory (accessible through the `$LIB` variable).

```
##################################################
proc syn_on_end_run {runName run_dir implName} {

# runName: Name of the run Ex: compile, synthesis
# run_dir: Current run directory.
# implName: Implementation Name Ex:rev_1
puts "*** syn_on_end_run called. Options: $runName, $run_dir $implName"
# TODO: Add your custom code here
global LIB
#cd to the verif directory under current implementation
cd $run_dir/verif
#Source vif2conformal script:
source $LIB/vif2conformal.tcl
#Set the variable "vif_file" to the .vif file
set vif_file [glob *.vif]
#Execute the vif2conformal on the .vif file
vif2conformal $vif_file
}
##################################################
```

To run Synplify Pro FPGA synthesis in batch mode:

```
% synplify_pro -batch <project>.prj
```

Once logic Synthesis is run, Synplify Pro generates a Verification Interface File `<design>.vif` in `<project directory>/verif` and a Verilog gate netlist `<design>.vm` in the `<project directory>`.

The vif2conformal translation script will generate Conformal command files. For a list of these files, see Synplify Pro Generated Setup Files on page 365.

To run verification on the generated gate netlist:

```
% cd <syn_project_dir>/verif
% lec -dofile <design>.vtc
```

Note: Make sure the `$XILINX` environment variable is defined. This environment variable points to the Xilinx software tree where the formal verification libraries are stored.

Note: If Conformal encounters issues with any the side files, user can manually edit the Conformal command files to fix the problem. Issues encountered in the past revolve around register naming differences between Conformal and Synplify Pro.

Continuing the Conformal Verification Flow

After running the Conformal dofile, proceed to debug the designs if there are any miscompares. For more information see Chapter 8, "Debugging".

Synplify Pro Generated Setup Files

Setup files are generated when you turn on the Verification Mode for Synplify Pro (see Generating a Gate Netlist on page 362). The files are saved in the implementation directory that contains the output files, under `/verif`. Conformal uses the setup files during verification, thus you must run Conformal from the `/verif` directory:

- `vtc`—Synplify Pro generates this file to run Conformal verification.
- `vlc`—This file links Conformal to simulation libraries for the FPGA devices. A sample `vlc` file contains something similar to the following:

  ```
  -y $XILINX/verilog/verplex/unisims
  -y $XILINX/verilog/verplex/simprims
  ```

  You must set the correct path to `$XILINX` before running the verification.
- `vfc`, `vmc`, `vsc`—These files give Conformal FSM encoding information, mapping information, and setup constraints.
- `vif`—Verification interface in ASCII format and it contains all of the Synplify Pro information necessary to perform verification. The `vtc`, `vlc`, `vfc`, `vmc`, and `vsc` files are generated automatically using the `vif` file.
- `vsq`—This file contains sequential constant information. The run script does not utilize this file because Conformal performs its own sequential constant learning.

Current Capabilities

This section describes FPGA capabilities for front-end verification. The FPGA feature supports the following:

- Flattened comparisons
- Verilog, VHDL, and mixed Verilog/VHDL
- Output files from Synplify Pro 8.0 on a Sun Solaris, Linux, or Hewlett-Packard HP-UX platform, but not output files from a Microsoft Windows platform

- Xilinx Virtex and Spartan family of FPGA devices (Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan, Spartan-II, Spartan-IIE, and Spartan-3)
- Xilinx Alliance series (ISE 6.2 or later)

Support for Synthesis Features

The FPGA feature supports a wide range of synthesis features:

- `full_case`
- `parallel_case`
- `syn_direct_enable`
- `syn_encoding`
- `syn_hier`
- `syn_keep`
- `syn_maxfan`
- `syn_multstyle`
- `syn_noclockbuf`
- `syn_replicate`
- `syn_sharing`
- `syn_srlstyle`
- `syn_state_machine`
- `syn_useenables`
- `syn_useioff`
- `translate_on/off`

However, FPGA has limited support for the following features:

- `syn_romstyle` (supports `select_rom` and logic)
- `syn_ramstyle` (supports `select_ram` and registers)

The features that are not yet supported are:

- `xc_pullup/xc_pulldown`

- `syn_pipeline`
- `syn_allow_retiming`
- `syn_tristatomux.`

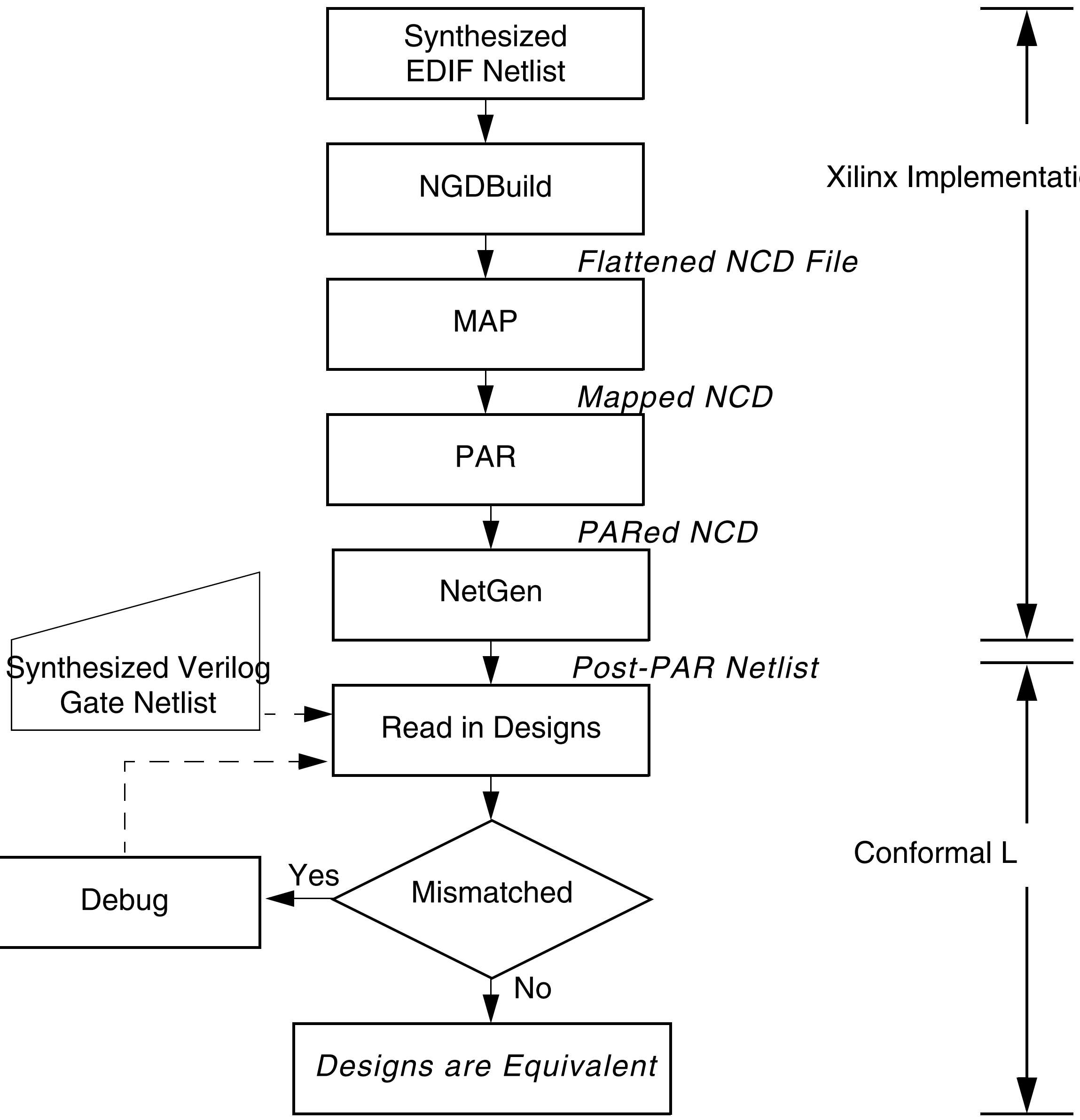
FPGA Back-End Verification Flow

During back-end verification, Conformal compares the post-synthesis Verilog gate netlist generated by Synplify Pro with the post-PAR Verilog gate netlist generated by Xilinx ISE 6.2 or later.

A summary of the back-end verification flow is as follows:

- Generate a post-PAR gate netlist using Xilinx ISE 6.2 or later.
- Read the post-synthesis gate netlist and the processed post-PAR gate netlist into Conformal and compare the designs.
- Debug the designs if there are miscompares.

The following figure shows the process flow, which is described in detail in the sections that follow.



Generating a Post-PAR Gate Netlist

Using the EDIF gate netlist produced by Synplify Pro, generate the post-PAR gate netlist. The EDIF gate netlist is located in the synthesis implementation directory. There are two ways to generate the gate netlist: with the GUI or the command terminal.

- From the Xilinx ISE GUI
- From the command line

To generate the post-PAR gate netlist from the Xilinx ISE GUI:

1. Launch the Xilinx software and create a Xilinx ISE project using the EDIF gate netlist.
2. Create a post-PAR Verilog gate netlist using the Xilinx ISE tools.

Historically, Xilinx generated flat post-PAR netlists. However, in Xilinx ISE 5.i releases (or later), you can maintain hierarchy levels. This allows for better control over blackboxing, and running gate-to-gate Conformal hierarchically.

To maintain a hierarchy levels, create a file called `<filename>.ucf` that contains the following directive:

```
INST <instance_path> KEEP_HIERARCHY = TRUE;
```

Then, run the Xilinx flow as follows:

1. Process EDIF
   ```
   % ngdbuild -uc <filename>.ucf <filename>.edf <filename>.ngd
   ```
2. Run MAP
   ```
   % map <filename>.ngd -o <mapped>.ncd
   ```
3. Run PAR
   ```
   % par -w <mapped>.ncd <par>.ncd <pcffile>.pcf
   ```
4. Create Post-PAR Verilog file for equivalence checking
   ```
   % netgen -ecn conformal -mhf -ngm <mapped>.ngm <par>.ncd <design_par_ecn>.v
   ```

Note: `Netgen` generates extra Verilog files that represent the hierarchical models that you want to keep.

Comparing the Designs

To run verification, read the synthesized Verilog gate netlist from Synplify Pro and the processed post PAR Verilog gate into Conformal. The following is a sample dofile for the verification:

```
set log file lec.log –replace
add notranslate module RAM* X_RAM* -library –both


read design -f $XILINX/verilog/Verplex/verilog.vc \
<post_synthesis>.vm -verilog -Golden
read design -f $XILINX/verilog/Verplex/verilog.vc <post_PAR>.v -verilog -revised
```

```
set flatten model -seq_constant
set flatten model –all_seq_merge
set flatten model –self_seq_merge

//Connect GSR and GTS to 0(GND)
add tied signal 0 glbl.GSR -rev
add tied signal 0 glbl.GTS –rev
add renaming rule r1 "_Z" "" -Golden
set mapping method -nets

// Flat comparison
set system mode lec
add compared points -all
compare
```

Continuing the Conformal Verification Flow

After comparing the two netlists, proceed to debugging the designs if there are any miscompares.

Tips for the FPGA Flow

This section discusses topics that can be helpful when using Conformal to formally verify FPGA designs that are synthesized using Synplify Pro and implemented using Xilinx tools and libraries.

Xilinx Tips

This section discusses tips for using the Xilinx tools and libraries.

Xilinx Formal Verification Libraries

Xilinx distributes two verification libraries for Conformal, which you can download and copy into your Xilinx ISE 6.2i installation:

- UNISIMS Xilinx library—A front-end Verilog library referenced by the Synplify Pro netlist.

  Note: Designers can instantiate UNISIMS models directly into their RTL code. See Instantiating UNISIM Models Directly into RTL on page 371.

- SIMPRIM—A back-end Verilog library for designs that have gone through map or place-and-route.

You can download the Cadence Conformal verification libraries at http://www.xilinx.com/ise/partner_libraries. To download these libraries, you must have Xilinx ISE 6.2i (or higher) installed on your system. After you download these libraries, copy them into the following directories in your Xilinx ISE 6.2i installation:

```
$XILINX/verilog/verplex/unisims
$XILINX/verilog/verplex/simprims
```

Note: `$XILINX` is an environment variable that points to your Xilinx installation directory.

Instantiating UNISIM Models Directly into RTL

To instantiate UNISIM models (such as DCM, clock buffers, LVDS input buffers, and differential receivers) directly into RTL code, blackbox these instantiated blocks during the RTL to gate comparison:

➤ Edit the Conformal command file called `<design>.vtc` by adding the `ADD NOTRANSLATE MODULE` command before the `READ DESIGN` commands. For example:

```
add notranslate module DCM IBUFGDS IBUFDS -both
add notranslate module RAMB* -both
     .
     .
```

.

If you are instantiating UNISIM models from the Synplify Pro software tree under `$SYNPLIFY_HOME/lib/xilinx/{virtex.v, virtex2.v, virtexe.v, unisim.v etc.,}`, always reference these libraries in the Synplicity Project file, as follows:

```
add_file -verilog "$LIB/xilinx/unisim.v"
```

This ensures that, when Conformal files are created from `vif`, the Conformal run script `<design>.vtc` makes references to the Xilinx FV libraries—not the Synplicity models. Reason being, the UNISIM models under the Synplify Pro software tree are defined as blackboxes and, in some cases, models should not be blackboxed during the Conformal run. This way, you have control over what gets blackboxed in the Golden and Revised design netlists.

Xilinx CORE Generator

The Xilinx CORE Generator has a catalog of ready-made parameterized functions that range from simple arithmetic operators (like adders, accumulators, and multipliers) to system-level building blocks (like filters, transforms, and memory resources).

Xilinx does not provide synthesizable models for these cores, which is a prerequisite for Conformal. Instead, Xilinx provides:

- `<`*`block`*`>.edn`—A tailored, Xilinx implementation netlist with complete relative placement information to guarantee performance
- `<`*`block`*`>.vho` or `<`*`block`*`>.veo`—VHDL or Verilog instantiation code
- `<`*`block`*`>.vhd` or `<`*`block`*`>.v`—VHDL or Verilog wrappers for simulation support
- A schematic symbol

The lack of synthesizable models directly affects the equivalency checking flow, because the implementation netlist generated by the Xilinx CORE generator cannot be verified against a higher-level RTL Golden model. The assumption is that the generator creates a netlist that is constructed correctly and that can map directly to a particular Xilinx device.

To work around this:

- For RTL to post-FPGA synthesis comparisons:

  Blackbox the Xilinx CORE Generator blocks for the Golden and Revised sides. Add the following directive to the Conformal dofile, for each block generated by the Xilinx CORE generator:

  ```
  add notranslate module <module_name> -both
  ```

- For post-FPGA synthesis to post-PAR comparisons, either:
  - Blackbox the Xilinx CORE generator blocks using the HRC verification capability in Xilinx ISE 6.2i.
  - Or, translate the EDIF netlist for blocks into Verilog, and instantiate it in both gate netlists for comparison.

  Xilinx provides a Perl script that runs the commands necessary to translate the EDIF netlist into Verilog:

  ```
  $XILINX/verilog/bin/<platform>/core2formal.pl
  ```

  To run these commands, set the following Xilinx environment:

  ```
  % xilperl $XILINX/verilog/bin/<platform>/core2formal.pl -verplex \
  -<family> <block>.edn
  ```

  Where:

  - `<family>` can be `virtex`, `virtexe`, `virtex2`, `virtex2p`, `spartan2`, `spartan2e`, or `spartan3`.
  - `<platform>` can be:
    - `sol`—Solaris UNIX workstations
    - `lin`— for Linux workstations
    - `nt` for PC workstations

Synplify Pro Tips

This section discusses tips for using Synplify Pro with Conformal.

Synplify Pro Limitations

Conformal does not support

- Synplify Pro Pipelining or Register Retiming options.
- RAM inference (not fully supported, see RAM Modeling on page 374)
- MUX transformations

Hierarchical and Blackboxing Options

By default, Synplify Pro FPGA Synthesis performs cross-hierarchical boundary optimizations. Even though the hierarchy in `<design>.vm`, which is the gate-level netlist that Synplify Pro generates, matches the RTL hierarchy, the module interfaces (ports) do not. Thus, you cannot run the HRC flow in Conformal—you can only run the flat (default) runs.

If you want to blackbox a portion of the hierarchy, the module boundaries/pin-out must match in both netlists. You can specify in Synplify Pro that you want to maintain the module-interface boundaries using the following directive:

```
Syn_hier = "hard"
```

Without this directive, blackboxing results in non-equivalencies when you run Conformal.

General FPGA Tips

This section describes some general tips for using the FPGA feature in Conformal.

RAM Modeling

If you do not use the Xilinx CORE Generator to create RAM memories, you can infer the memory in your RTL code. Synplify Pro provides directives and coding styles to infer memories.

For RTL-to-gate comparisons, Conformal can compare distributed/select RAM memories and single-port block RAM. Inferred dual-port block RAMs are harder to compare, and should be blackboxed.

Distributed/select ROM implementation is supported in this flow, but mapping a ROM function to a block RAM is not supported. To work around this, use the Xilinx CORE generator to create the block RAM/ROM and then blackbox it for equivalence checking.

Handling Multipliers

If the RTL code contains multiplication(s), then Synplify Pro usually map them into Xilinx `MULT18X18` blocks (for `Virtex2` devices). Consider the following RTL example:

```
module mult_11x13 (z, a, b);
output [23:0] z;
input [10:0] a;
input [12:0] b;

assign z = a * b;

endmodule
```

The synthesized netlist will instantiate a `MULT18X18` block, as follows:

```
MULT18X18 \z.I_3  (
.P({\z.bmult.multxx_genmulta.0.genmultb.genmultb.0.multx_prod [35],
\z.bmult.multxx_genmulta.0.genmultb.genmultb.0.multx_prod [34],
\z.bmult.multxx_pbuf_0 [33], \z.bmult.multxx_pbuf_0 [32], \z.bmult.multxx_pbuf_0 [31],
\z.bmult.multxx_pbuf_0 [30], \z.bmult.multxx_pbuf_0 [29], \z.bmult.multxx_pbuf_0 [28],
\z.bmult.multxx_pbuf_0 [27], \z.bmult.multxx_pbuf_0 [26], \z.bmult.multxx_pbuf_0 [25],
\z.bmult.multxx_pbuf_0 [24], z_c[23], z_c[22], z_c[21], z_c[20], z_c[19],
z_c[18], z_c[17], z_c[16], z_c[15], z_c[14], z_c[13], z_c[12], z_c[11],
z_c[10], z_c[9], z_c[8], z_c[7], z_c[6], z_c[5], z_c[4], z_c[3], z_c[2],
z_c[1], z_c[0]}),
    .A({GND, GND, GND, GND, GND, b_c[12], b_c[11], b_c[10], b_c[9], b_c[8],
b_c[7], b_c[6], b_c[5], b_c[4], b_c[3], b_c[2], b_c[1], b_c[0]}),
    .B({GND, GND, GND, GND, GND, GND, GND, a_c[10], a_c[9], a_c[8], a_c[7],
a_c[6], a_c[5], a_c[4], a_c[3], a_c[2], a_c[1], a_c[0]})
);
```

Notice that the operands of the multiplier were swapped in the gate netlist. To successfully compare the RTL versus the gate, add the following commands to your Conformal dofile:

```
> set multiplier implementation csa -Golden
> set multiplier implementation csa -swap -revised
```

Conformal picks the `csa` multiplier architecture and swaps the operands on the Revised netlist. If the swapping fails, Conformal aborts during the comparison.

For multipliers larger than 18x18 (such as 24X24), Synplify Pro uses more than one Xilinx multiplier block because it cannot fit in a single `MULT18X18` block. In this case, comparing the RTL versus post-synthesis will generate aborts because the structure of the two multipliers is very different. In such cases, you can either:

- Use the Xilinx CORE generator to generate the desired large multiplier, rather than infer it in the RTL. This way, it can be blackboxed during the RTL-to-gate comparison.
- Or, isolate the multiplier to its own module and then blackbox it during the RTL-to-gate comparison.

Using the Verilog Always Statement with Mixed Register Types

The following coding style can cause a mismatch between Conformal and Synplify Pro, because of the way the code is interpreted. (Conformal is in line with simulation behavior.)

Consider the following example:

```
module test (clk, rst_l, a, b, q1, q2);
   input  clk, rst_l, a, b;
   output q1, q2;
   reg  q1, q2;

    always @(posedge clk or negedge rst_l)
   begin
         if (!rst_l)
            q1 <= 1'b0;
         else
      begin
        q1 <= a;
        q2 <= b;
      end
   end
endmodule
```

In the code above:

- Synplify Pro implements `q2` with a simple DFF (without reset) with `b` connected to the `D` pin.
- Conformal implements the `q2` register as a DFF with a MUX at the `D` pin. Where the MUX is selected by reset and the selection is between input signal `b` and `q2` (hold condition).

To work around this inconsistency, recode your RTL as follows (using the example above):

```
module test (clk, rst_l, a, b, q1, q2);
   input  clk, rst_l, a, b;
    output q1, q2;
    reg  q1, q2;

    always @(posedge clk or negedge rst_l)
    begin
       if (!rst_l)
             q1 <= 1'b0;
      else
      begin
             q1 <= a;
     end
   end
   always @(posedge clk)
   begin
            q2 <= b;
   end
endmodule
```

The following reset coding style is not supported in Conformal. Consider the test case:

```
module test(in, clk, reset, out);
input clk, in, reset;
output out;

reg rx_reset;
reg out;

always @(posedge clk or posedge reset) begin
    if (reset) rx_reset = 1'b0;
    else rx_reset = 1'b1;
end

always @(posedge clk or posedge reset) begin
    if (reset || rx_reset) out = 1'b0;  // Problem Section
    else
     out = in;
end

endmodule
```

Conformal does not support conditional expressions with asynchronous and synchronous signals, in this case `if (reset || rx_reset)`. (Commented in bold.)

Instantiating Virtex/Virtex2 Startup Models Directly into RTL

For designs that instantiate start-up blocks, such as `STARTUP_VIRTEX2`, when a reset signal other than GSR or GTS is connected to the start-up block, the synthesis tool (in the gate-level netlist), connects this reset signal to all registers in the design. If the same behavior is not described in the RTL model, Conformal treats the RTL and the gate-level netlist as non-equivalent, unless the reset pin is constrained (on both designs) to 0. Consider the following simple RTL example:

```
module top (in, clk, rst, out);
input in, clk, rst;
output out;
reg out;

STARTUP_VIRTEX2 STARTUP_I (.GSR(rst));

always @(posedge clk)
begin
    out <= in;
end
endmodule
```

Since the asynchronous reset signal `rst` is not explicitly specified in the sensitivity list of the `always` statement, the register synthesized on the RTL does not have a reset connection. This causes a mis-compare in Conformal. To work around this, add a pin constraint:

```
> add pin constraint 0 rst -both
```

Or, modify the RTL code to behave like the actual implementation:

```
    always @(posedge clk or posedge rst) begin
        if  (rst)   out <= 1'b0;
        else
            out <= in;
    end
```

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 385.

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 389.
Configurations:	configuration declaration	supported
	block configuration	supported
	use	supported
	attribute specifications	ignored
	component configurations	supported* See Component Configuration on page 390.
	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 393.
	variable	supported
	shared variable	supported* See Shared Variable on page 393.
	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 394.
	predefined attributes	supported* See Predefined Attributes on page 395.
	user-defined attributes	supported* See User-Defined Attributes on page 399.
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 399.
	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 400.
	assertion	ignored
	report	ignored
	guarded signal assignment	supported* See VHDL GUARDED Block Support on page 402.
	transport / after	ignored
	signal assignment	supported* See Signal Assignment on page 402.
	variable assignment	supported
	procedure call	supported* See Procedure Calls on page 404.
	if statement	supported
	case statement	supported
	for loop statement	supported* See For Loops on page 404.
	while loop statement	supported* See While Loops on page 406.
	next statement	supported
	exit statement	supported
	return statement	supported
	null statement	supported
Concurrent Statements:	block guard	supported* See VHDL GUARDED Block Support on page 402.
	block	supported

Support Status for RTL VHDL Synthesis Subset Constructs		
	process	supported
	sensitivity list	ignored
	concurrent procedure call	supported
	concurrent assertion	ignored
	concurrent signal assignment	supported* See Signal Assignment on page 402.
	guarded concurrent signal assignment	supported* See VHDL GUARDED Block Support on page 402.
	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`

VHDL 2008 Support

Types			
	Unconstrained Elements	Array	supported
		Records	not supported.
Declarations			
	PSL Declarations		not supported
	Reading of Output Ports		supported (might issue rule violation in VHDL 93)
	Non-static Expressions in Port Map		supported.
	Aliases of Multidimensional Arrays		supported
Expressions			
	Unary Logical Operators	for loop statement	supported
	Matching Relational Operators	?=, ?/=, ?<, ?<=, ?>, ?>=	supported.
	Conditional Operators	??	supported
External Names			
	(only supported for parsing and skipping the information)		
		Relative Paths	not supported
		Absolute Paths	not supported
		Package Paths	not supported
Concurrent Statements			
	PSL Directives		not supported
	Process All		supported
Lexical			
	Comments	C Style Comments	supported (some limitation in handling of pragma)
	Literals	Enhanced Bit String Literals	supported

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
5.   gen_label_1: FOR idx IN 0 TO 255 GENERATE
6.     comp_a (…);
```

```
  END FOR;
  gen_label_2: FOR idx IN 0 TO 255 GENERATE
    comp_b (…);
  END FOR;
END

CONFIGURATION real_config OF design IS
  USE work.all;
  FOR rtl_arch
     -- using generate statement label and indices
     FOR gen_label_1(255 DOWNTO 1)
      USE CONFIGURATION my_lib.comp_a_config;
     END FOR;
     FOR gen_label_1(0)
       USE CONFIGURATION my_lib.comp_a_config_2;
     END FOR;
     -- using generate statement label w/o indices
     FOR gen_label_2
      USE CONFIGURATION my_lib.comp_b_config;
    END FOR;
  END FOR;
END;
```

Nested Configurations

The Conformal software supports nested configurations and configurations with more than one level of hierarchy. It allows multiple architectures of an entity to exist by creating new modules with composite names created from the entity and architecture names. The Conformal software also allows the same architecture to be configured differently internally for different instances by creating a unique composite name for each such differing sub-configuration.

The following is an example:

```
-- entity e1 has two architectures e1a0 and e1a1
entity e1 is
    port (e1out : out BIT);
end e1;

architecture e1a0 of e1 is
begin
    e1out <= '0';
end e1a0;

architecture e1a1 of e1 is
begin
    e1out <= '1';
end e1a1;

-- entity e2 has an architecture e2arch which has a component C1
entity e2 is
    port (e2out : out BIT);
end e2;

architecture e2arch of e2 is
    component C1 is
        port (e1out : out BIT);
    end component C1;
begin
    e2i1 : C1 port map (e1out => e2out);
end e2arch;

-- a configuration for e2 which binds component C1 to
entity/architecture e1(e1a0)
use work.e1;
configuration e2conf of e2 is
    for e2arch
        for e2i1 : C1 use entity e1(e1a0);
        end for;
    end for;
end configuration e2conf;

-- entity e3
entity e3 is
    port (e3out1, e3out2 : out BIT);
end e3;

architecture e3arch of e3 is
    component C2 is
        port (e2out : out BIT);
    end component C2;
begin
    e3i1 : C2 port map (e2out => e3out1);
    e3i2 : C2 port map (e2out => e3out2);
end e3arch;

configuration e3conf of e3 is
    for e3arch
        for e3i1 : C2
            use configuration work.e2conf;  -- nested configuration
        end for;
        for e3i2 : C2
            use entity work.e2(e2arch); -- further configuring sub-hierarchy
            for e2arch
                for e2i1 : C1 use entity work.e1(e1a1);
                end for;
            end for;
        end for;
    end for;
end configuration e3conf;
```

Declarations

Initial Value

The Conformal software supports Initial Value variables or signals with the `READ DESIGN` command's `-initial_value` option.

Example

In the following example, signal `out1` will get the initial value of high, which is '`1`'. This initial value '`1`' will be discarded if the variable `high` is assigned.

```
proc1 : PROCESS is
    VARIABLE high : BIT := '1';
    BEGIN
        out1 <= high;
END PROCESS proc1;
```

Shared Variable

Restriction

The Conformal software does not support Shared Variables when they are declared inside a package.

Example

In the following example, the Conformal software does not support the `SHARED VARIABLE counter` because it is declared inside a package. See line 5, in bold.

```
1.  LIBRARY IEEE;
2.  USE IEEE.STD_LOGIC_1164.ALL;
3.
4.  PACKAGE pack1 IS
5.  SHARED VARIABLE counter: INTEGER RANGE 0 TO 99 := 0;
6.  END PACKAGE pack1;
```

Names

Sliced Names

Restriction

The Conformal software does not support Sliced Names when their ranges are not computable.

Example

In the following example, the Conformal software does not support Sliced Name `in0 (idx DOWNTO 0)` because input `idx` is not computable. See line 15, in bold.

```
1.  ENTITY test IS
2.    PORT (
3.      clk : IN BIT;
4.      idx : IN INTEGER RANGE 0 TO 3;
5.      in0 : IN BIT_VECTOR(3 DOWNTO 0);
6.      out0 : OUT BIT_VECTOR(3 DOWNTO 0)
```

```
  );
end test;

ARCHITECTURE arch OF test IS
BEGIN
  proc1 : PROCESS (clk)
    BEGIN
      IF clk‘EVENT AND clk=‘1’ THEN
        out0 <= in0(idx DOWNTO 0);
      END IF;
    END PROCESS;
END arch;
```

Predefined Attributes

The Conformal software only supports the following Predefined Attributes:

- LEFT
- RIGHT
- HIGH
- LOW
- RANGE
- REVERSE_RANGE
- LENGTH
- ASCENDING
- LEFTOF
- RIGHTOF
- PRED
- SUCC
- POS
- VAL
- BASE
- EVENT*
- STABLE*
- LAST_VALUE*
- TRANSACTION*

*See Restriction 2 on page 396.

Restriction 1

The return value of a Predefined Attribute must be globally computable.

Example 1

In the following example, the Conformal software does not support the Predefined Attribute `RANGE` because the variable `tmp` is not computable. See line 13, in bold.

```
 1.    ENTITY attributes3 IS
 2.      PORT ( input : IN BIT_VECTOR(7 DOWNTO 0);
 3.             output1 : OUT BIT_VECTOR(7 DOWNTO 0);
 4.             idx : In INTEGER RANGE 0 TO 7
 5.           );
 6.    END attributes3;
 7.
 8.    ARCHITECTURE arch OF attributes3 IS
 9.    BEGIN
10.      PROCESS ( input )
11.      VARIABLE tmp: BIT_VECTOR(idx DOWNTO 0);
12.      BEGIN
13.         FOR i IN tmp'RANGE LOOP
14.           output1(i) <= input(i) XOR '1';
15.         END LOOP;
16.      END PROCESS;
17.    END arch;
```

Restriction 2

The Conformal software supports the asterisked (*) predefined attributes (shown above), but only when they are used with synthesizable clock expressions.

Example 2a

In this example, the Conformal software supports the Predefined Attribute `STABLE` because it is used in a synthesizable clock expression on line 3 (in bold).

```
1.    PROCESS
2.     BEGIN
3.       IF NOT clk'STABLE AND clk = '1' THEN
4.    ...
```

Example 2b

In this example, the Conformal software does not support the Predefined Attribute `EVENT` because the design uses it in a non-synthesizable clock expression on line 3 (in bold).

```
1.  PROCESS
2.   BEGIN
3.     IF clk'EVENT THEN
4.  …
```

Out-of-Range Handling

For the following attributes:

- LEFTOF
- RIGHTOF
- PRED
- SUCC
- VAL

If variable x is out-of-range, the Conformal software has two choices to interpret the attribute value:

- If -RANGECONSTRAINT is specified in the READ DESIGN command, (or `set hdl compiler rangeconstraint`), Conformal will result dont care for attributes when 'x' if out-of-range

- If -RANGECONSTRAINT is not specified, the Conformal software will result a value for attribute as following:

`T'VAL(x)`	x itself
`T'SUCC(x)`	`T'RIGHT` if T is ascending, `T'LEFT` is T is descending
`T'PRED(x)`	`T'LEFT` if T is ascending, `T'RIGHT` is T is descending
`T'RIGHTOF(x)`	`T'RIGHT`
`T'LEFTOF(x)`	`T'LEFT`

For example:

```
type T is (e0, e1, e2, e3, e4, e5);
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of T: type is "1100 0110 1000 0110 0100 0001";
subtype ST is T range e4 downto e1;
```

```
-- p = 0
ST'val(p) = 0000
-- x = e0 (1100)
ST'succ(x) = e4 (0100)
ST'pred(x) = e1 (0110)
ST'rightof(x) = e1 (0110)
ST'leftof(x) = e4 (0100)
-- x = e1 (0110)
ST'pred(x) = ST'rightof(x) = e1 (0110)
-- x = e4 (0100)
ST'succ(x) = ST'leftof(x) = e4 (0100)
```

Note: The default values of the attributes are only for the scenarios where x is a variable. If the argument of these attributes is a constant which is out-of-range, the Conformal software will error it out.

If you use the `READ DESIGN` command with the `-architecture`, `-configuration`, `-rootconfig`, and `-lastmod` options, the Conformal software links the entity/architecture based on the following priorities:

1. `-rootconfig` has the highest priority.
2. `-configuration` has the second priority.
3. `-architecture` has the third priority.
4. `-lastmod` is the fourth priority.

User-Defined Attributes

Restriction

Conformal ignores all User-Defined Attributes except when they are used in one of the following forms:

- Form A:

```
TYPE state_type IS (Init, State1, State2, State3);
ATTRIBUTE enum_encoding : STRING;
ATTRIBUTE enum_encoding OF state_type :
     TYPE IS "0001 0010 0100 1000";
SIGNAL current_state, next_state: state_type;
```

- Form B:

```
TYPE state_type IS (Init, State1, State2, State3);
ATTRIBUTE enum_encoding : STRING;
ATTRIBUTE enum_encoding OF state_type :
     TYPE IS      "Init=0001,State1=0010,State2=0100,State3=1000";
SIGNAL current_state, next_state: state_type;
```

Expressions

Function Calls

Restriction

The Conformal software does not support a Function Call when the function includes a `WAIT` construct or clock signal.

Example

In the following example, the Conformal software does not support `FUNCTION func1` because it contains a clock expression on line 3 (in bold).

```
1.     FUNCTION func1 (in0, clk : IN STD_LOGIC) RETURN STD_LOGIC IS
2.     BEGIN
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;
```

While Loops

Restriction

Conformal does not support while loops that have loop control statements.

Example

In the following example, the while loop in line 20 is not supported because it contains a loop control statement.

```
1.  LIBRARY IEEE;
2.  USE IEEE.STD_LOGIC_1164.ALL;
3.
4.  ENTITY while_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 while_loop;
10.
11. ARCHITECTURE rtl OF while_loop IS
12. BEGIN
13. PROCESS (clk, count)
14.   VARIABLE i: integer;
15.   VARIABLE data_temp : STD_LOGIC_VECTOR(3 DOWNTO 0);
16. BEGIN
17. IF (CLK'EVENT AND CLK = '1') THEN
18.   i := 0;
19.   WHILE(i<4) LOOP
20.     NEXT WHEN i = 1;
21.     data_temp(i) := data(i);
22.     i := i + 1;
23.   END LOOP;
24. END IF;
25. data_out <= data_temp;
26. END PROCESS;
27. END rtl;
```

Concurrent Statements

Signal Assignment

See Signal Assignment on page 402.

B

Verilog Support

Verilog Configurations

A configuration is an explicit set of rules that specify the exact source description to be used to represent each instance in a design. There could be more than one model describing the same module if they are at different levels of abstraction, such as behavioral, synthesis, and simulation. A configuration allows you to specify which model is to be used for each instance (or selected instances) in the design.

The Conformal software supports a subset of Verilog configurations, as defined in the *Verilog 2005 Language Reference Manual*. In particular, 'hierarchical use configurations,' liblist ordering, and cell configurations are not supported. The namespace mapping is supported through the `READ DESIGN` command's `-map` and `-mapfile` options. The Conformal software supports the configuration of several levels of hierarchy through instance configurations.

The Conformal software allows Verilog modules read during `READ DESIGN` to be stored in a user defined design namespace using the `-map` or `-mapfile` option. If either option is not specified, the Verilog modules are stored in the default design namespace called '`work`'. The verilog modules read in during `READ LIBRARY` are automatically stored in a default library namespace also called '`work`'. Thus, '`work`' is the name of two default namespaces in Conformal: default design namespace and default library namespace. For Liberty library cells, the name of the library is itself the name of the namespace.

If a module is specified in the configuration along with a library name, for example, `clock_lib.clock_mod_1`, then the module `clock_mod_1` is searched only in the namespace `clock_lib`. However, if a module `work.mod_2` or simply `mod_2` was specified, then the design namespace '`work`' is first searched for module `mod_2`. Only if the module is not found, is the library space '`work`' searched.

Supported Constructs

The Conformal software supports the configuration of several levels of hierarchy through instance configurations. For example, the following instance configurations is the supported constructs:

```
config cfg;
  design work.top;
  instance top.i1 use lib1.mybuf;
// the instance i1 of module top is bound to the module mybuf in library lib1
endconfig
```

Synthesizable UDPs

Conformal tools focus on synthesizable user-defined primitives (UDPs). The behavior, however, might be different in simulation tools. For example, the following UDPs are synthesized as the same `OR` gate in Conformal, but in simulation tools they might have different behavior.

```
table
//A1, A2: O
1  0: 1;
0  1: 1;
1  1: 1;
0  0: 0;
endtable

table
//A1, A2: O
1  ?: 1;
?  1: 1;
0  0: 0;
endtable
```

Unsupported Constructs and Workaround Solutions

`liblist` ordering is not supported. For example, the module `top` has two instances `i1` and `i2` in `top.v`.

There is one module `mybuf` in `mybuf.v`. The following commands will map the module `mybuf` to user defined design namespace `lib1/lib2` and map the module `top` to user defined design namespace `mywork`.

```
read design -root top -noelab  \
  -mapfile lib1 lib1_src/mybuf.v \
  -mapfile lib2 lib2_src/mybuf.v \
  -mapfile mywork mywork_src/top.v
```

The following construct binds instance `top.i1` to module `mybuf` in `lib1`, and binds instance `top.i2` to module `mybuf` in `lib2` using the `liblist` constructs.

```
config cfg;
  design mywork.top;
  instance top.i1 liblist lib1;
  instance top.i2 liblist lib2;
endconfig
```

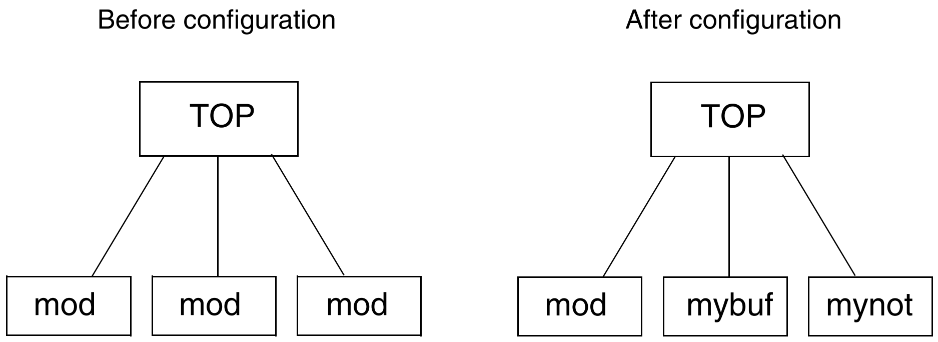
Since the `liblist` is an unsupported construct, the workaround solution is to use instance configuration as the following:

```
config cfg;
  design mywork.top;
  instance top.i1 use lib1.mybuf;
  instance top.i2 use lib2.mybuf;
endconfig
```

Instance Configuration Examples

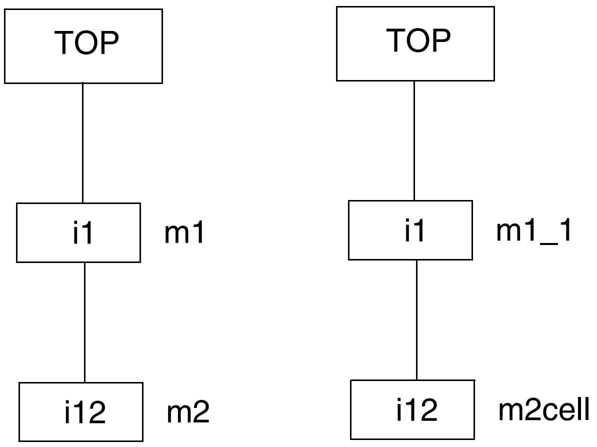
For example, if a design whose top module top has three instances `i1`, `i2` and `i3` of module `mod1`, and you read in the configuration `cfg1`, the following shows the design hierarchy before and after applying the configuration:

```
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

SystemVerilog Support

SystemVerilog Support Tables

The following tables are sorted by category.

Literals

IEEE 1800-2005		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 1800-2005	IEEE 1800-2009		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	6.16	string data type	Unsupported
4.7	6.16	Parameters and localparams of strings	Unsupported
4.7	6.16	string data arrays	Unsupported

IEEE 1800-2005	IEEE 1800-2009		Status
4.7	6.16	string operator: !=	Unsupported
4.7	6.16	string operator: ==	Unsupported
4.7	6.16	string operator: <	Unsupported
4.7	6.16	string operator: <=	Unsupported
4.7	6.16	string operator: >	Unsupported
4.7	6.16	string operator: >=	Unsupported
4.7	6.16	string operator: concat {s1,s2}	Unsupported
4.7	6.16	string operator: {multiplier{s1}}	Unsupported
4.7	6.16	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

IEEE 1800-2005	IEEE 1800-2009		Status
4.8		event data type	Unsupported
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

IEEE 1800-2005	IEEE 1800-2009		Status
4.11		Unpacked structure data type (OOMR's to members)	Supported
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 in 1800-2005, or 7.10 in 1800-2009))	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 1800-2005	IEEE 1800-2009		Status
5.6, 9, 14, and 4.7	7.5, 6.16	Public access to QDAs and strings	Unsupported
5.6, 9, 14, and 4.7	7.5, 6.16	Local and protected access to QDAs and strings	Unsupported

IEEE 1800-2005	IEEE 1800-2009		Status
5.6, 9, 14, and 4.7	7.5, 6.16	QDAs as local variables in tasks/functions/ methods	Unsupported
5.6	7.5	Dynamic arrays of strings	Unsupported
5.9, 14, and 4.7	7.8, 6.16	Queues and associative arrays of strings	Unsupported
5.6, 9, and 14	7.5	Hierarchical (OOMR) refereces to QDAs	Unsupported
5.2		array of mailboxes	Unsupported
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	7.11	array query functions	Supported
5.6	7.5	dynamic arrays (details below)	Unsupported
5.6	7.5	dynamic arrays of classes	Unsupported
5.6	7.5	dynamic arrays in classes (public/local)	Unsupported
5.6	7.5	dynamic arrays in packages	Unsupported
5.6	7.5	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	7.8	associative arrays	Unsupported
5.9	7.8	associative arrays of classes	Unsupported

IEEE 1800-2005	IEEE 1800-2009		Status
5.9	7.8	associative arrays in classes (public/local)	Unsupported
5.9	7.8	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
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	7.10	queues	Unsupported
5.14	7.10	queues of classes	Unsupported
5.14	7.10	queues in classes (public/local)	Unsupported
5.14	7.10	queues in packages	Unsupported
5.15		array manipulation methods	Unsupported

Data Declarations

IEEE 1800-2005	IEEE 1800-2009		Status
6.3		constants (in classes)	Unsupported

IEEE 1800-2005	IEEE 1800-2009		Status
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
6.10	6.23	Type operator	Supported

Attributes

IEEE 1800-2005		Status
	Default attribute type	Unsupported

Operators & Expressions

IEEE 1800-2005	IEEE 1800-2009		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

IEEE 1800-2005	IEEE 1800-2009		Status
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
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
	11.4.7	Logical operators - logical implication (->) and logical equivalence (<->)	Supported
	11.13	let construct	Supported

Procedural Statements

IEEE 1800-2005	IEEE 1800-2009		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
	12.4	Conditional if-else statement - unique0-if	Supported
	12.5	Case statement - unique0-case	Supported

Processes

IEEE 1800-2005		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 1800-2005	IEEE 1800-2009		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 1800-2005	IEEE 1800-2009		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
	13.4.4	Background processes spawned by function calls - Nonblocking assignment inside a function	Supported

Classes

IEEE 1800-2005		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

IEEE 1800-2005		Status
7.5	Object properties - dynamic arrays	Unsupported
7.5	Object properties - queues	Unsupported
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

IEEE 1800-2005		Status
7.18	Constant class properties - parsing support	Unsupported
7.19	Abstract classes - can use empty virtual methods	Limited Support

Only virtual (parameterized) classes with (static) function declarations are supported. For example:

```
package ParamFuncPkg;
 virtual class Functions #(parameter SIZE=32);
 static function [SIZE-1:0] GetParity (input
 [SIZE-1:0] a);
 return ^a;
endfunction
endclass
endpackage

module Test (
 input    logic [7:0]  Addr,
 output   logic        Parity
 );
always_comb
 begin
 Parity = (ParamFuncPkg::Functions
 #($size(Addr))::GetParity(Addr));
end
endmodule
```

IEEE 1800-2005		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 1800-2005		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

IEEE 1800-2005		Status
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
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

IEEE 1800-2005		Status
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
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

IEEE 1800-2005		Status
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
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 1800-2005		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 1800-2005		Status
9.3	Stratified event scheduler	Unsupported

Clocking Blocks

IEEE 1800-2005	IEEE 1800-2009		Status
	14.14	Global clocking and $global_clock system function	Supported
15.2		Clocking blocks (in generate loops)	Unsupported
15.3		Input/output skews	Unsupported

IEEE 1800-2005	IEEE 1800-2009		Status
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 1800-2005		Status
16.2-06	Program Blocks	Unsupported

Assertions

For more information, see System Verilog Assertions (SVA) on page 446.

IEEE 1800-2005	IEEE 1800-2009		Status
17.2	16.3	Immediate Assertions	Supported
17.4		Boolean Assertions	Supported
17.5		Sequences (see specifics under 17.6 and 17.7)	Unsupported
17.5	16.7	Sequences cycle delay range	Supported

IEEE 1800-2005	IEEE 1800-2009		Status
	16.9.4	Global clocking past and future sampled value functions ($past_gclk, $rose_gclk, $fell_gclk, $stable_gclk, and $changed_gclk)	Supported
	16.9.4	Global clocking future sampled value functions ($future_gclk, $rising_gclk, $falling_gclk, $steady_gclk, and $changing_gclk)	Supported
17.6		Declaring sequences	Unsupported
17.6.1		Typed formal argument in sequences	Unsupported
17.7.1		Sequence operator precedence	Unsupported
17.7.2		Sequence repetition in sequences	Unsupported
17.7.3		Sequence sampled value functions ($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 1800-2005	**IEEE 1800-2009**		**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 1800-2005		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

Modules and Hierarchy

IEEE 1364-2005		Status
12.4.1	Generate support in if-generates	Supported
12.4.2	Generate support in for-generates	Supported

IEEE 1800-2005		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 1800-2005		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

IEEE 1800-2005	IEEE 1800-2009		Status
	23.2.2.4	Default port values	Supported

Interfaces

IEEE 1800-2005		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

IEEE 1800-2005		Status
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
20.6	Modports - clocking keyword on Modports	Supported
20.6	Modports - task/function export	Unsupported
20.6	Extern fork/join tasks	Unsupported
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

Packages

IEEE 1800-2005	IEEE 1800-2009		Status
	26.6	Exporting imported names from packages - package export	Supported

Configuration Libraries

IEEE 1800-2005		Status
21.2	config support for interfaces	Unsupported

System Tasks and Functions

IEEE 1800-2005	IEEE 1800-2009		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
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 1800-2005		Status
24.0	Mapping SV types to VCD format	Unsupported

Macros and Compiler Directives

IEEE 1800-2005		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 1800-2005 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 1800-2005		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

IEEE 1800-2005		Status
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
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

Configuring the Contents of a Design

IEEE 1800-2005	IEEE 1800-2009		Status
	33.4.3	Local parameter declaration	Supported
	33.4.3	Named parameter assignment	Supported

Annexes

IEEE 1800-2005		Status
C	Std Package	Unsupported
D	Link Lists	Unsupported

Non-std

IEEE 1800-2005		Status
	$psprintf	Unsupported

System Verilog Assertions (SVA)

The Conformal software accepts all syntactically correct SystemVerilog Assertion (SVA) constructs, including property and sequence declarations. However, only Boolean-level constraints are supported by Conformal. Other SVA constructs are ignored. Conformal supports 'assert' and 'assume' properties for Boolean expressions, and treats them as constraints. The software considers 'assert' and 'assume' as interchangeable, and treats both as constraints.

To read in SVA constructs in the design, run the `READ DESIGN` command with the `-sva` option. The following example shows a command sequence for reading in the SVA constructs in the `barrel_shifter.v` file:

```
read design -Golden barrel_shifter.v -root barrel_shifter -sva
read design -revised shifter.v
set system mode lec
add compare point -all
compare
```

The following brief describes what the Conformal software supports for System Verilog:

- Conformal will read all SVA constructs including property and sequence declarations without erroring out during parsing.
- Conformal will support assert/assume property for boolean expressions. These assertions will be treated as constraints.
- Simple named property instantiations are also supported. Properties can be referred to by name inside another assertion. However, formal/actual argument passing in the named property is not currently supported.
- Assertions can also be embedded inside clocked always blocks.
- Simple default clocking block is supported.
- Sampled value functions are supported including `$rose`, `$past`, `$fell`, `$onehot`, `$stable`, and so on.
- Property operators are supported including negation, disjunction, conjunction, implication, and if-else.
- Binding properties are supported for properties declared inside/outside module declaration, and in a separate file.

The following brief describes what the Conformal software does not support for System Verilog:

- Complex sequential assertions that span over time
- Immediate assertions
- Conformal ignores cover statements
- Sequence operators

Supported SVA System Functions

The following system functions are supported by Conformal, but they are not sampled functions because no sampled clocks are used.

```
$onehot (<expression>) returns true if only 1 bit of the expression is high.
$onehot0 (<expression>) returns true if at most 1 bit of the expression is high.
$countones (<expression>) returns the number of ones in a bit vector expression.
```

For more details, see SV-1800-2005 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-1800-2005 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

Supported Directives

Supported Vendors

The following lists the supported vendors for use with the `SET DIRECTIVE` command to enable or disable the specified synthesis directives when they are used with the specified `<vendor_name>` prefix:

- `ambit`
- `cadence`
- `conformal`
- `pragma`
- `quickturn`
- `synopsys`
- `synthesis`

Supported Directives

The following lists the supported directives for use with the `SET DIRECTIVE` command to enable or disable the effects of the specified synthesis directives when reading in a Verilog or VHDL file:

- `assertion_library`
- `async_set_reset`
- `black_box`
- `built_in`
- `clock_hold`
- `compile_off`
- `compile_on`
- `dc_script_begin`
- `dc_script_end`
- `divider`
- `enum`
- `fsm_state`

- `full_case`
- `implementation`
- `infer_latch`
- `is_isolation_cell` (`quickturn` not supported)
- `is_level_shifter`
- `mem_rowselect`
- `multi_port`
- `multiplier`
- `operand`
- `parallel_case`
- `power_gating_cell`
- `set_implementation`
- `pragma`
- `state_vector`
- `synthesis_off`
- `synthesis_on`
- `synthesis` (`ambit` only)
- `synthesis off` (`ambit` only)
- `synthesis on` (`ambit` only)
- `template`
- `translate_off`
- `translate_on`

Conformal Directive Examples

The following includes short descriptions and examples of supported Conformal directives:

- `clock_hold “<name> ...”`

  This directive instructs Conformal to synthesize latch arrays so that the array address is placed into the clock cone of the synthesized logic.

  Example:

  ```
  // conformal clock_hold “memory_array”
  always @(clk or we or addr or din) begin
  if (clk && we) memory_array[addr] = din;
  end
  ```

  Without this directive, the above `always` process results in a latch array with both `clk` and `we` in the clock logic. And `addr` is used to mux between `din` and the old state. Thus, with this directive, we move the `addr` into the clock logic of the array. This directive is useful for register files and memory arrays.

- `infer_latch`

  This directive instructs Conformal to use a D-latch instead of a DFF when there is an `always` statement with an edge-triggered clock. The default is to use a DFF.

  Example:

  ```
  always @(posedge clk) begin // conformal infer_latch
  qstate = din;
  end
  ```

  In this example, the `infer_latch` directive tells Conformal to synthesize a latch enabled with a high clock (rather than a D flip-flop with a positive edge triggered clock) for the `always` process. It is similar to writing the following RTL:

  ```
  always @(clk or din) begin
  if (clk) qstate = din;
  end
  ```

- `multi_port`

  This directive instructs Conformal to synthesize a multi-port latch or register when multiple, simultaneous definitions exist for the same state variable.

  Example:

  ```
  always @(clk1 or din1) if (clk1) qstate = din1;
  always @(clk2 or din2) if (clk2) qstate = din2;
  …
  always @(clkn or dinn) if (clkn) qstate = dinn;
  ```

This sample case results in `n` number of latches, each with separate clocks and data inputs and all outputs wired together. However, the implementation of a multiport cannot be compared with an `n` port latch. Thus, you would use the `// conformal multi_port "qstate"` directive to synthesize an `n` port latch with one `Q` output. Internally, a primitive UDP model represents the valid function. If a simultaneous write occurs on multiple ports and the input data on those ports is not equal, the state becomes an `X`. This directive is generally used for multi-port memory arrays and custom designs.

- `mem_rowselect`

  This directive supersedes the `clock_hold` directive. It guides memory array RTL model synthesis so that it includes the same logic in the clock and data cones as in the implementation. Thus, Conformal can complete equivalence checking. For example:

  ```
  // conformal mem_rowselect "mem clk addr[7:5] addr[2:0]"
  always @(clk or we or addr or din) begin
  if (clk && we) mem[addr] = din;
  end
  ```

  The synthesized result creates a row decoder with address bits `7, 6, 5,` and `2, 1, 0,` and used `clk` as an enable. The `addr` bits `3` and `4` are used to column multiplex input data when `we` is active. However, when `we` is not active or a column is selected, the array data input is in a high `Z` state, which is representative of memory implementation.

Note: The wildcard (*) represents any zero or more characters in filenames.

Enabling One Directive

When you employ the `SET DIRECTIVE` command and you do not specify a directive, the command applies to all directives. In the following example, the objective is to enable only the `parallel_case` directive. To do so, first disable all directives, then enable the specified directive (`parallel_case`).

```
//disable all directives
set directive off
//enable parallel_case
set directive on parallel_case
```

Disabling All Directives for One Vendor

In the following example, the objective is to disable all Synopsys directives (`synopsys translate_off`, `synopsys translate_on`, `synopsys full_case ...`).

```
//disable all synopsys directives
set directive off synopsys
```

Disabling Specified Directives for One Vendor

In the following example, the objective is to disable `synopsys translate_off` and `synopsys translate_on`. This command has no effect on `conformal translate_off` and `conformal translate_on`.

```
//disable synopsys translate_off and synopsys translate_on
set directive off synopsys translate_off translate_on
```

Enabling a List of Directives from an RTL File

In the following examples, we have 2 RTL files: `test.v` and `test1.v`.

- In the following command, the synthesis directive `parallel_case` is on (enabled) in file `test.v`:

  ```
  set directive on parallel_case -file test.v
  ```

- In the following command, the synthesis directive `parallel_case` is on (enabled) in file `test.v` and `test1.v`:

  ```
  set directive on parallel_case -file *.v
  ```

E

Conformal Sample Test Case

This appendix uses a small test case to familiarize you with the command flow of the GUI windows. It includes step-by-step procedures to compare a Verilog gate-level non-scan design (Golden) with the same design after inserting scan (Revised).

The step-by-step procedure illustrates the following tasks:

1. Starting Conformal on page 460
2. Reading the Library on page 460
3. Read the Designs on page 460
4. Changing to the LEC System Mode on page 462
5. Viewing Unmapped and Mapped Points on page 462
6. Running a Comparison on page 463
7. Diagnosing a Non-Equivalent Point on page 464
8. Opening the Schematic Viewer on page 467
9. Adding Pin Constraints on page 470
10. Rerunning the Comparison on page 470
11. Exiting on page 471

There is also a dofile example at the end of this appendix. See Standard Dofile Example on page 472.

Starting Conformal

Start the Conformal software at the UNIX system prompt in the `demo/doc_testcase` directory:

```
demo/doc_testcase% lec
```

This action opens the main Conformal CD GUI window.

Reading the Library

The Verilog designs use a Verilog simulation library. To read in the Verilog simulation library, use the procedure described below. First, open the Read Library window. Then, use the window to prepare Conformal to read the Library.

1. To open the Read Library form, choose *File – Read Library* or click the *Read Library* icon.

   For a description of the Read Library form, see Read Library Form on page 105.
2. When the Read Library window opens, keep the defaults for *Format*, which is set to *Verilog,* and for *Type*, which is set to *Both*.
3. Select the Verilog library file `(lib.v)`.
4. Click *Add Selected*.

   The filename appears in the *File list* display.
5. Click *OK* to read in the Verilog library.

   The Read Library form closes and the Transcript window of the main window notes that Conformal has successfully read the Verilog library. For the purposes of this test case, ignore warning messages noted in the transcript window as Conformal reads the library.

Read the Designs

The Verilog gate-level, *non*-scan design is the Golden design. The Verilog gate-level, *scan* design is the Revised design. To read the Golden and Revised designs, use the following procedure, which directs you to open the Read Design window and prepare Conformal to read the designs.

First, read in the Golden design as follows:

1. To open the Read Design form, choose *File – Read Design* or click the *Read Design* icon.

   For a description of the Read Design form, see Read Design Form on page 109.
2. Click *Format –Verilog* and *Type – Golden*.
3. Click the Golden Verilog design, `Golden.v`.
4. Click *Add Selected* to display the filename in the *File list* display.
5. Click *OK* to read in the Golden Verilog design.

   The Read Design form closes and the Transcript window of the main CONFORMAL-LEC window notes that Conformal has successfully read the Verilog design.

   For the purposes of this test case, ignore warning messages noted in the transcript window as Conformal reads the designs.

Repeat the procedure to read in the Revised Verilog design as noted below.

1. To open the Read Design form, choose *File – Read Design* or click the *Read Design* icon.
2. Click *Format –Verilog* and *Type – Revised*.
3. Click the Revised Verilog design, `revised.v`.
4. Click *Add Selected* to display the filename in the *File list* display.
5. Click *OK* to read in the Revised Verilog design.

The Read Design window closes and the Transcript window of the main window notes that Conformal has successfully read the Verilog design.

For the purposes of this test case, ignore warning messages noted in the transcript window as Conformal reads the designs.

Changing to the LEC System Mode

For this small test case, assume there are no design constraints to specify. Therefore, you are ready to change system modes to do automatic key point mapping:

➤ Click the *LEC* mode icon. LEC

When Conformal completes the key point mapping, it displays the following in the Transcript window of the main CONFORMAL-LEC window:

- Warning messages
- Summary table of the mapped and unmapped points

In the Diagnosing a Non-Equivalent Point step, you will use the Conformal tools to examine the unmapped and mapped points.

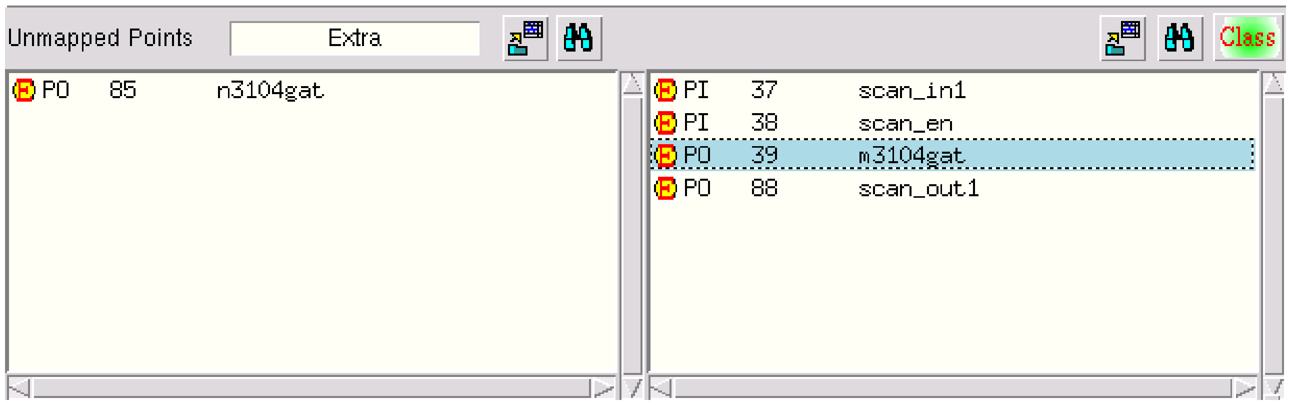
Viewing Unmapped and Mapped Points

To view details about the key points information that is displayed in the summary, choose *Tools – Mapping Manager* to open the Mapping Manager, or click the *Mapping Manager* icon.

As you view the Unmapped Points section of the Mapping Manager, note the following:

- The *Extra* icon flags extra unmapped points.
- The Revised design includes three extra scan pins, which is acceptable because the Golden design is a non-scan design.
- Both the Golden and Revised designs have an extra primary output pin because the names are not the same The Golden pin name is `n3104gat`, and the Revised pin name

is `m3104gat`. Do the following to manually map the unmapped points as mapped key points.


Unmapped Points Extra

```
PO  85    n3104gat
```

```
PI  37    scan_in1
PI  38    scan_en
PO  39    m3104gat
PO  88    scan_out1
```

Do the following to manually map key points:

1. Click the primary output pin in the Golden design to highlight it.
2. Right-click and choose *Set Target Mapping Point* from the pop-up menu.

   The text for the Golden unmapped point changes color from black to red.
3. Click the primary output pin in the Revised design to highlight it.
4. Right-click and choose *Add Mapping Point – Non-invert* from the pop-up menu.

   Conformal removes the two key points from the Unmapped Points section and they appear at the bottom of the list in the Mapped Points section.

In the Running a Comparison step, you will run a comparison on all of the mapped points to determine whether they are equivalent or non-equivalent.

Running a Comparison

1. With the Mapping Manager still open, add all of the mapped points as compare points for the comparison by clicking the *Add All Compare Points* icon in the *Mapped Points* section.

ADD

The compare points are now displayed in the *Compared Points* section. The question mark (?) preceding each compare point tells that Conformal has not compared these points.

2. Compare points in the *Compared Points* section:

   Click the *Compare* icon.

   In the main window, the status bar shows the comparison's progress (shown as a percentage). On completion, each pair of equivalent points is denoted with a green circle. The non-equivalent compared points are denoted with red circles. (You can specify Check Mark indicators, if you prefer.)

3. Scroll down the *Compared Points* list to find the non-equivalent points, or click the *Class* icon in the *Compared Points* section and deselect the *Equivalent* check box on the pop-up menu.

   Class

Diagnosing a Non-Equivalent Point

In this step, you will diagnose one of the non-equivalent compared points to find out why it is non-equivalent.

1. In the Mapping Manager, click any non-equivalent compared point. (For example: `Golden: U100/DF`.)
2. Right-click and choose *Diagnose* from the pop-up menu to open the Diagnosis Manager.

   The Diagnosis Manager displays information related to the diagnosis point you selected in the Running a Comparison step. The information is organized in two columns: Golden and Revised. In the Diagnosis Manager, find the following fields and windows:

   ❑ Non-Equivalent Compared Point
   You selected this point in the Running a Comparison step.

   ❑ Diagnosis Point (active)
   This point is also shown in the Diagnosis Points (inputs) section.

   The simulation values for the Golden and Revised designs are displayed in the

Diagnosis Point (active) section. Notice that the simulation values for the Golden and Revised designs, which are shown in parenthesis ( ), are not the same. The simulation value for the Revised design is one, while the value for the Golden design is zero.

- ❑ Corresponding Support
  This section displays the *corresponding* support key point of the diagnosis point for both the Golden and Revised designs.
- ❑ Non-corresponding Support
  This section displays the *non*-corresponding support key point of the diagnosis point for both the Golden and Revised designs.

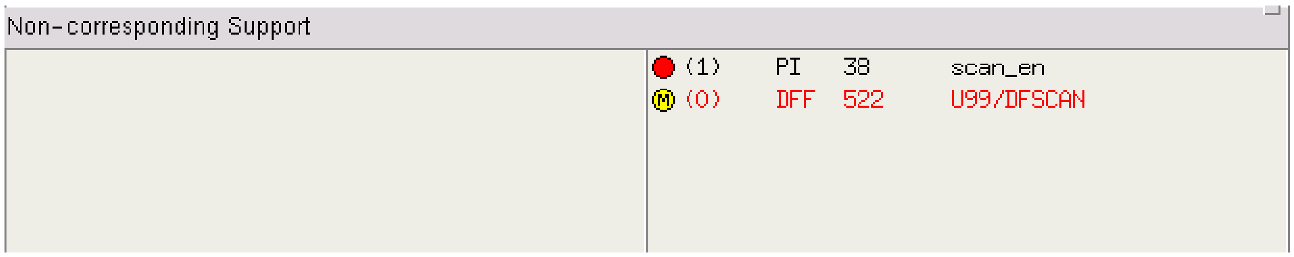
The corresponding and non-corresponding support key points are the logic fan-in cone key points of the diagnosis point.

Support points are color coded as follows:

- ❑ Red—The support point is a non-equivalent compare point
- ❑ Green—The support point is an equivalent compare point
- ❑ Black—The support point either has not been compared yet or cannot be compared (for example, PI)

Non-Corresponding Support Section

This section lists key points in the Revised DFF logic cone that are not in the Golden DFF logic cone. The `scan_en` pin is an unmapped key point in the Revised DFF logic cone. The non-corresponding DFF in the Revised logic cone is a mapped key point, but its Golden corresponding mapped key point is not in the Golden DFF logic cone.

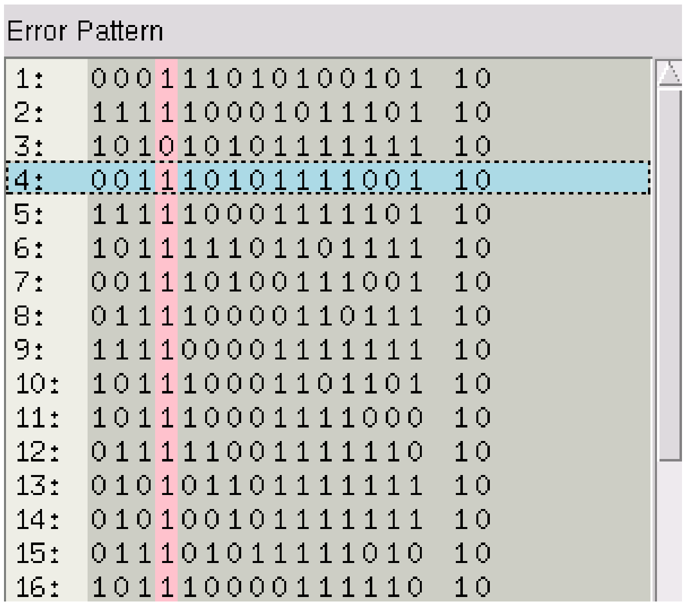


```
● (1)   PI    38    scan_en
Ⓜ (0)   DFF   522   U99/DFSCAN
```

Error Pattern Section

This section lists error patterns that prove the difference in simulation values between the Golden and Revised diagnosis point. These simulation values for each error pattern correlate to the Support key points (Corresponding and Non-corresponding) in the Revised design.

Note: When you select a non-corresponding support point, Conformal applies a pink highlight to the associated column in the test vector set. Refer to the Error Pattern section.



Error Pattern

```
1:   000111010100101  10
2:   111110001011101  10
3:   101001010111111  10
4:   001111010111001  10
5:   111110001111101  10
6:   101111101101111  10
7:   001111010011001  10
8:   011110000110111  10
9:   111110000111111  10
10:  101110001101101  10
11:  101110001111000  10
12:  011111001111110  10
13:  010101101111111  10
14:  010100101111111  10
15:  011110101111010  10
16:  101110000111110  10
```

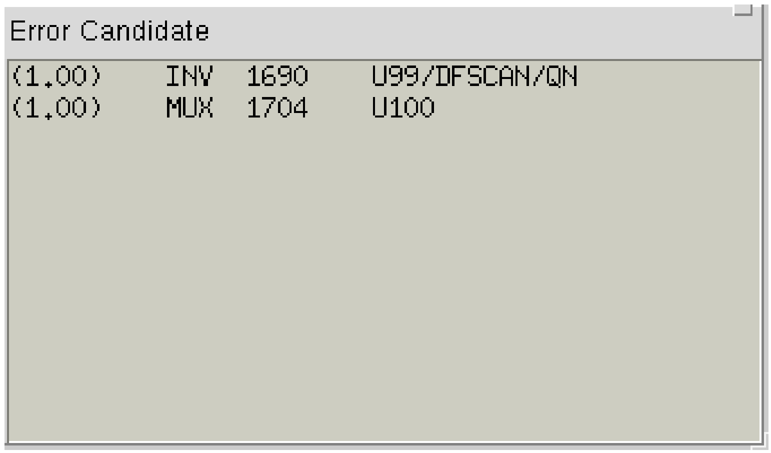
Do the following to see the difference in simulation values for the diagnosis points.

1. Click any error pattern in the list (for example, Error Pattern 5).
2. Click another error pattern and examine the change in simulation values listed in the *Corresponding Support* section. (Values are shown in parenthesis.)

Next, you will examine the *Error Candidate* section to try to determine the cause of this difference.

Error Candidate Section

This section lists gates with a weighted percentage representing likely causes for the difference in simulation values in the Golden and Revised designs.



```
(1.00)    INV  1690    U99/DFSCAN/QN
(1.00)    MUX  1704    U100
```

Opening the Schematic Viewer continues the diagnosis procedure. It describes how you will access the schematic viewer to examine a schematic representation of the diagnosis point.

Opening the Schematic Viewer

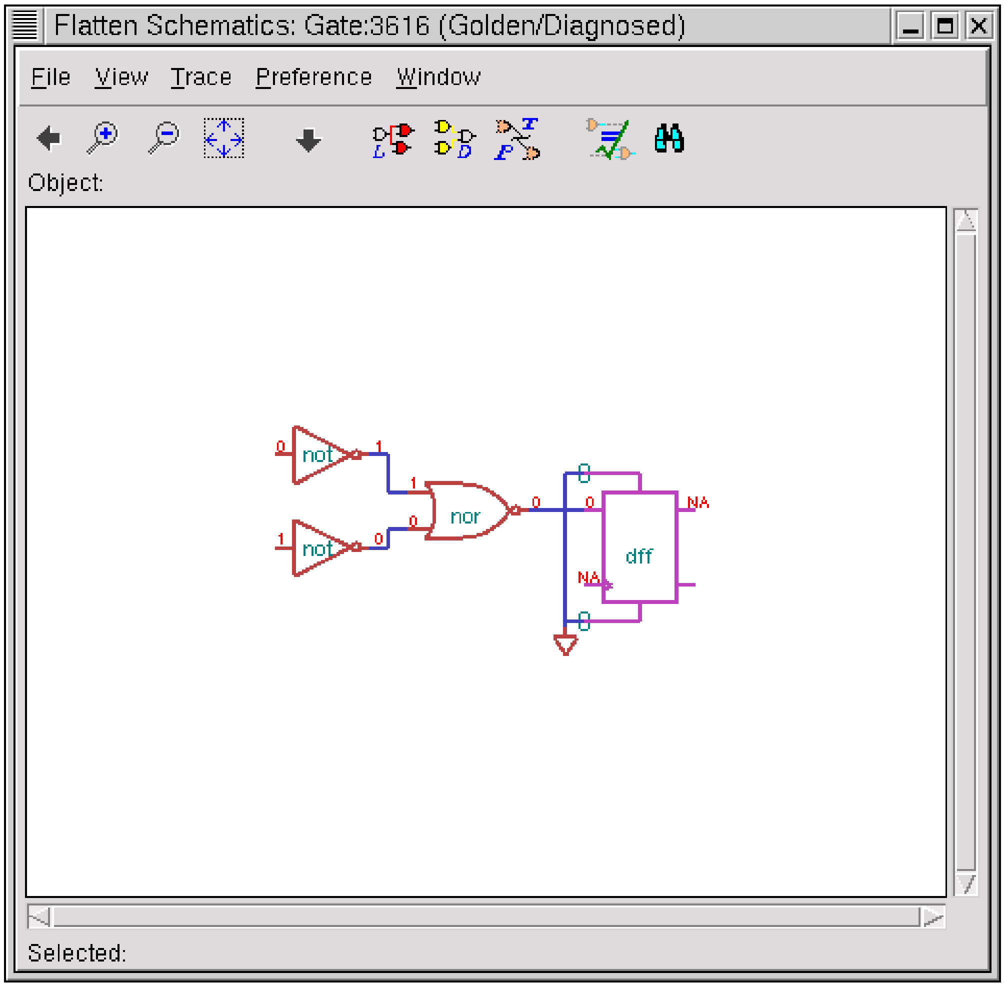
For optimal viewing of schematic viewer figures, access the PDF version of this *Conformal Equivalence Checking User Guide* on screen with a PDF reader, or print this section in color.

To view a schematic representation of the diagnosis point along with the logic fan-in cone for both the Golden and Revised designs, click on the *Schematic* icon on the Diagnosis Manager menu bar.

The schematic viewer opens with two separate windows for the Golden and Revised DFF logic cones.

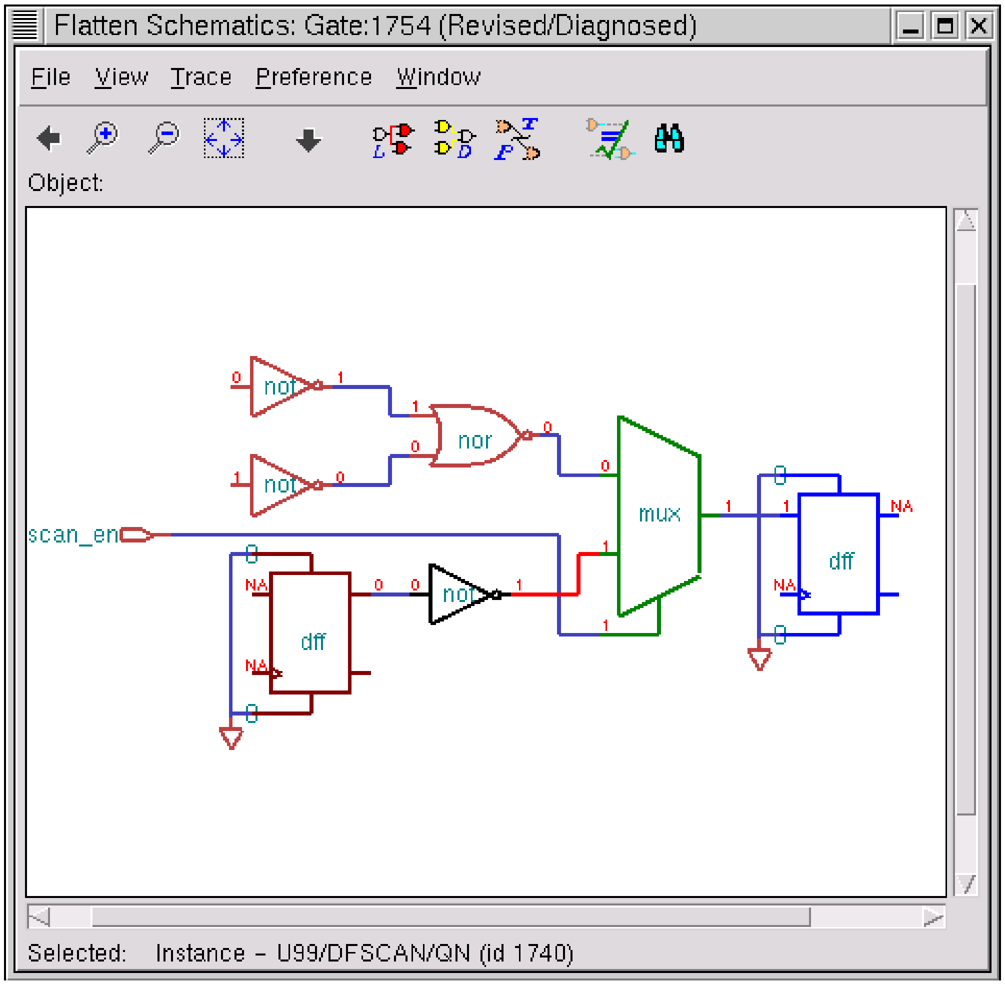
In the Golden schematic representation (Figure E-1 on page 468), the NOR gate leads to the D input of the D flip-flop. In the Revised schematic representation (Figure E-2 on page 469), the NOR gate also leads to the D input of the MUX-DFF gate. Notice the simulation value differences.

Figure E-1 Golden Fan-In Cone


Flatten Schematics: Gate:3616 (Golden/Diagnosed)
File View Trace Preference Window
Object:
not
not
nor
dff
0
1
1
0
1
0
0
0
NA
NA
Selected:

In the Revised schematic representation, the select line (`scan_en`) of the MUX gate is selecting the scan path instead of the functional path. To see the select line, click the net of the select line of the MUX gate to highlight the `scan_en` signal.

Figure E-2 Revised Fan-In Cone


Flatten Schematics: Gate:1754 (Revised/Diagnosed)
File View Trace Preference Window
Object:
not
nor
not
mux
scan_en
dff
not
dff
NA
Selected: Instance – U99/DFSCAN/QN (id 1740)

From your examination of the design, you can see that you must constrain the `scan_en` pin to the functional path of the MUX-DFF gate. With this constraint, the non-equivalent point becomes equivalent. (Also notice that the `scan_en` pin was listed as a Non-corresponding Support point in the Revised column of the Diagnosis Manager.)

In the Adding Pin Constraints,step, you will add the pin constraint that disable the scan logic that was introduced into the Revised design through the and flow. Before proceeding this step, use the following information to close the Diagnosis Manager and the Schematic Viewer.

1. Click *Cancel* in the Diagnosis Manager.

2. Use a schematic viewer *File* drop-down menu to close one schematic window, and its complementary window closes as well.

Adding Pin Constraints

Add pin constraints in the Setup system mode. Do the following to switch the system mode and add a pin constraint to the scan_en pin of the Revised design.

1. Click the *Setup* mode icon at the right end of the toolbar.

   Setup

2. Choose *Setup – Pin Constraints* to open the Pin Constraints window.
3. Click the *Revised* tab.
4. Left-drag to scroll down the *Pin* list, and then click the `scan_en` pin to highlight it.
5. Right-click and choose *Add Constraint 0* from the pop-up menu.

   The (scan_en) pin appears in the *0* column.

6. Click *Close*.

In Rerunning the Comparison, Conformal will compare with the newly added pin constraint.

Rerunning the Comparison

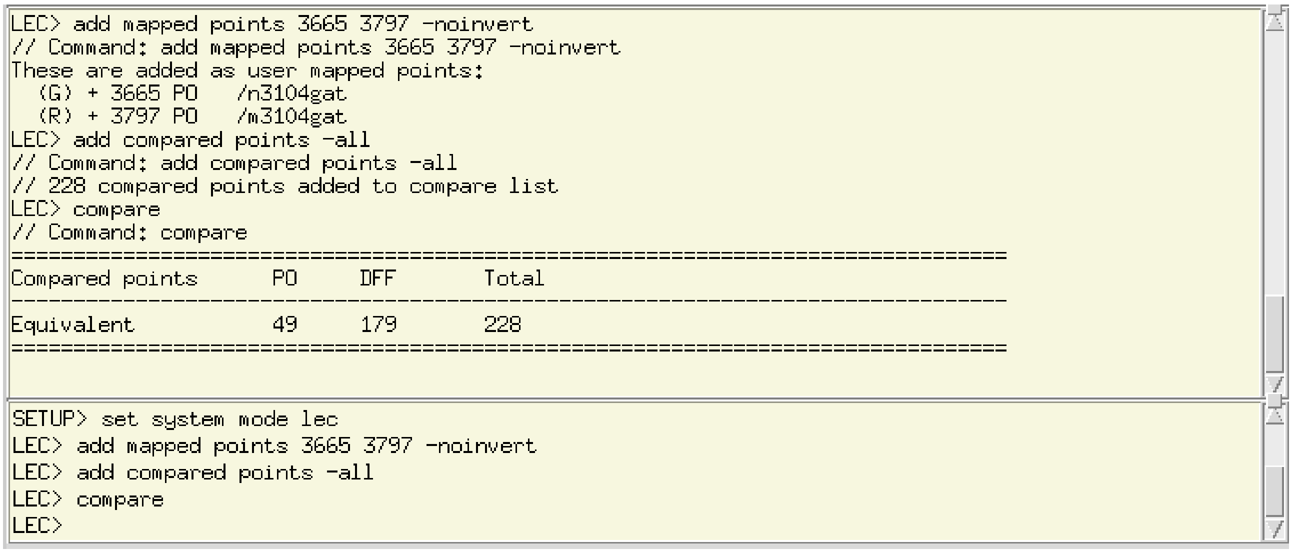
Repeat the following steps to view the new comparison results:

- Step 4, Changing to the LEC System Mode on page 462
- Step 5, Viewing Unmapped and Mapped Points on page 462

  Remember to add the two unmapped primary output pins as mapped points.

- Step 6, Running a Comparison on page 463

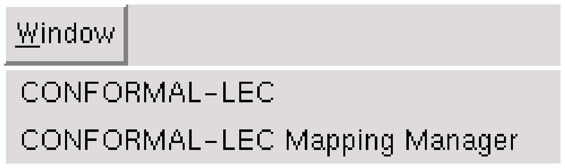
When Conformal completes the comparison, the summary table displayed in the Transcript window of the main CONFORMAL-LEC window shows that all compared points are equivalent. Do the following to see that Conformal also displays this information in the Mapping Manager.



```
LEC> add mapped points 3665 3797 -noinvert
// Command: add mapped points 3665 3797 -noinvert
These are added as user mapped points:
  (G) + 3665 PO   /n3104gat
  (R) + 3797 PO   /m3104gat
LEC> add compared points -all
// Command: add compared points -all
// 228 compared points added to compare list
LEC> compare
// Command: compare
================================================================================
Compared points      PO     DFF       Total
--------------------------------------------------------------------------------
Equivalent           49     179       228
================================================================================
```

```
SETUP> set system mode lec
LEC> add mapped points 3665 3797 -noinvert
LEC> add compared points -all
LEC> compare
LEC>
```

1. Choose the *Window – Mapping Manager.*



2. In the Mapping Manager, scroll down the Compared Points section to confirm that each compared point is preceded by a green circle.

●

Note: When you added the pin constraint you remedied all of the non-equivalent points.

Exiting

Now that you have verified that the Golden and Revised designs are equivalent, exit Conformal by doing the following:

1. Choose *File – Exit*.

   The exit confirmation dialog box opens to confirm the exit.

2. Click *Yes*.

Standard Dofile Example

The following is an example of a standard flat (non-hierarchical) design dofile:

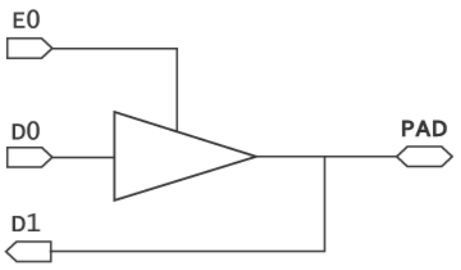
```
set log file lec.log -replace
read library ...
read design -Golden ...
read design -revised ...
report design data
report black box
report module
<apply design constraints>
<apply modeling options>
set system mode lec
add compare point -all
compare
usage
```

F

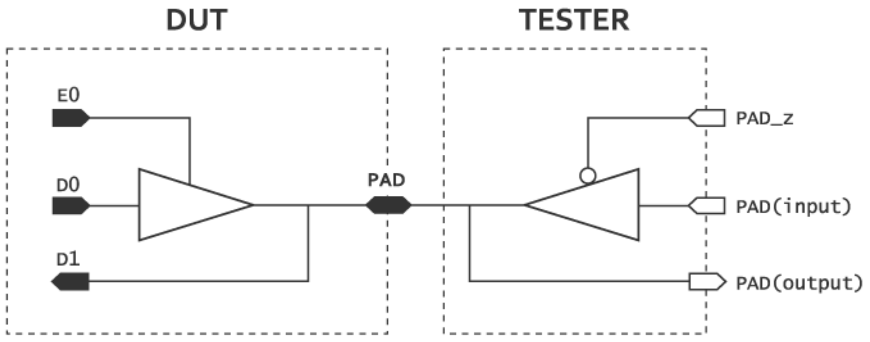
Top-level IO Port Modeling

When a top-level port is declared as an inout (IO) port, the Conformal software must verify it as both an input port and an output port. In static equivalency checking, all possible modes of operation are verified regardless of how the module was internally configured. Each top-level IO port is also replaced by three ports (two inputs and one output) in the flattened design.

The following figure shows a typical IO port configuration where `EO`, `DO`, `DI` are internal signals and `PAD` is the top-level IO port. In actual operations, `EO` is either (`i`) HIGH and `PAD` acts as an output port for `DO`, or (`ii`) LOW and `PAD` acts as an input port to `DI`. In proper operations, `PAD` must not be driven when it is operating as an output port.

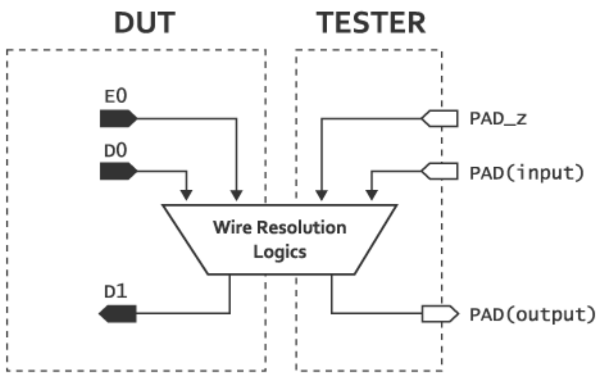


In Conformal equivalency checking, because `PAD` is an IO port, it will be modeled as follows:

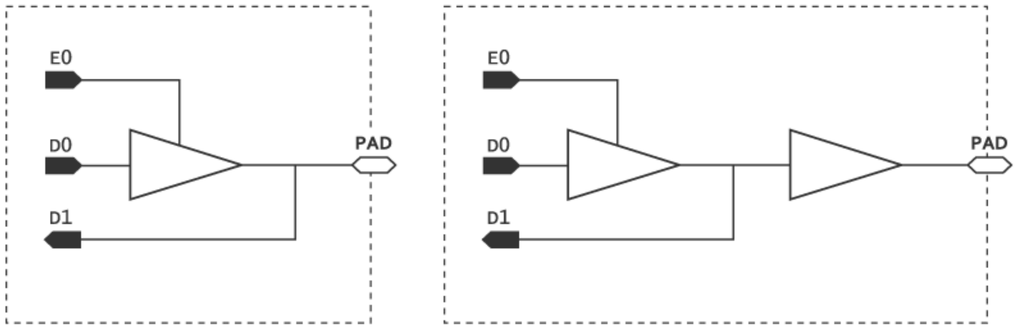


The original PAD port is now an internal signal. The software adds separate input and output ports (with the same name) that correspond to the inout PAD port. The software also adds an input port PAD_z which tristates the testers output when it is HIGH.

Because the internal PAD signal is now connected to two output devices, it becomes a wire resolution node. The software creates wire resolution logics which combine the signals EO, DO, PAD(input) and PAD_z, and produces the two signals driving DI and PAD(output).



Because the buffer in the circuit on the right blocks the tristate output of the tristate driver, DI and PAD(output) are not the same signals. In particular, PAD(output) is subjected to output tristate (output-Z) verification so that the following circuits are non-equivalent:



G

Conformal Primitive Gate Types

The following lists the primitive gate types that are used in the Conformal software.

Note: Although the primitives are listed below in all uppercase, Conformal primitives are case insensitive.

Cell Type	Description
`ADD`	word-level addition primitive
`AND`	AND gate
`BUF`	buffer
`BUFIF0`	buffer if output is zero
`BUFIF1`	buffer if output is one
`CD`	2-input AND gate. First input is control, second input is data.
`CMOS`	complementary-symmetry metaloxidesemiconductor
`DFF`	delay flip-flop
`DIV`	divider
`DLAT`	delay latch
`EQ`	logical equality
`GE`	greater than or equal
`GT`	greater than
`INV`	inverter
`LE`	less than or equal
`LT`	less than
`MODULUS`	modulus
`MUX`	multiplexer

Cell Type	Description
`MULT`	multiplier
`NAND`	NAND gate
`NE`	logical not equal
`NMOS`	n-type metal-oxide-semiconductor
`NOR`	NOR gate
`NOTIF0`	NOT if output is zero
`NOTIF1`	NOT if output is one
`ONECOLD`	One-cold condition
`ONECOLD0`	Zero-one-cold condition
`ONEHOT`	One-hot condition
`ONEHOT0`	Zero-one-hot condition
`OR`	OR gate
`PMOS`	p-type metal-oxide-semiconductor
`PULLDOWN`	pull-down resistor
`PULLUP`	pull-up resistor
`RCMOS`	primitive which is same as Verilog's rcmos primitive gate
`REM`	remainder
`RNMOS`	primitive which is same as Verilog's rnmos primitive gate
`ROL`	rotate left
`ROR`	rotate right
`RPMOS`	same as Verilog's rpmos primitive gate
`RTRAN`	same as Verilog's rtran primitive gate
`RTRANIF0`	same as Verilog's rtranif0 primitive gate
`RTRANIF1`	same as Verilog's rtranif1 primitive gate
`SLA`	shifter left arithmetic
`SLL`	shifter left logical
`SRA`	shifter right arithmetic
`SRL`	shifter right logical

Cell Type	Description
`SUBTRACT`	subtractor
`TIE0`	constant 1’b0
`TIE1`	constant 1’b1
`TIEX`	constant 1’bx
`TIEZ`	constant 1’bz
`TRAN`	transistor
`TRANIF0`	transistor if output is zero
`TRANIF1`	transistor if output is one
`WAND`	word-level AND
`WBUF`	word-level buffer
`WBUFIF0`	word-level bufif0
`WBUFIF1`	word-level bufif1
`WCD`	word-level CD, m-bit Data, 1-bit Control
`WDC`	word-level DC, m-bit Data, 1-bit Control
`WDFF`	word-level D Flop
`WDLAT`	word-level D Latch
`WINV`	word-level inverter
`WMUX`	word-level MUX
`WNAND`	word-level NAND
`WNOR`	word-level NOR
`WOR`	word-level OR
`WSEL`	word-level selector
`WXNOR`	word-level XNOR
`WXOR`	word-level XOR
`XNOR`	XNOR gate
`XOR`	XOR gate