PrimeTime® GCA User Guide

Version L-2016.06, June 2016

SYNOPSYS®

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

Contents

Preface

This preface includes the following sections:

- About This User Guide
- Customer Support

About This User Guide

PrimeTime GCA is a gate-level tool for constraint analysis and debugging of full-chip or block-level netlists. This document describes how you can use PrimeTime GCA to debug constraint problems found in Synopsys tools such as the Design Compiler, IC Compiler, and PrimeTime tools.

Audience

This document has been developed for designers who are

- Responsible for constraint development, analysis, or editing
- Familiar with the Synopsys tools, such as PrimeTime, that use the constraints being debugged by PrimeTime GCA

Related Publications

For additional information about the PrimeTime GCA tool, see the documentation on the Synopsys SolvNet® online support site at the following address:

https://solvnet.synopsys.com/DocsOnWeb

You might also want to see the documentation for the following related Synopsys products:

- PrimeTime®
- Design Compiler®
- IC Compiler™

Release Notes

Information about new features, enhancements, changes, known limitations, and resolved Synopsys Technical Action Requests (STARs) is available in the *PrimeTime Suite Release Notes* on the SolvNet site.

To see the *PrimeTime Suite Release Notes*,

1. Go to the SolvNet Download Center located at the following address:

   https://solvnet.synopsys.com/DownloadCenter

2. Select PrimeTime Suite, and then select a release in the list that appears.

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
`Courier`	Indicates syntax, such as `write_file`.
`Courier italic`	Indicates a user-defined value in syntax, such as `write_file` *`design_list`*.
`Courier bold`	Indicates user input—text you type verbatim—in examples, such as `prompt>` **`write_file top`**
[ ]	Denotes optional arguments in syntax, such as `write_file [-format` *`fmt`*`]`
...	Indicates that arguments can be repeated as many times as needed, such as *`pin1 pin2 ... pinN`*
|	Indicates a choice among alternatives, such as `low | medium | high`
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing C.
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys Technical Support Center.

Accessing SolvNet

The SolvNet site includes a knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. The SolvNet site also gives you access to a wide range of Synopsys online services including software downloads, documentation, and technical support.

To access the SolvNet site, go to the following address:

https://solvnet.synopsys.com

If prompted, enter your user name and password. If you do not have a Synopsys user name and password, follow the instructions to sign up for an account.

If you need help using the SolvNet site, click HELP in the top-right menu bar.

Contacting the Synopsys Technical Support Center

If you have problems, questions, or suggestions, you can contact the Synopsys Technical Support Center in the following ways:

- Open a support case to your local support center online by signing in to the SolvNet site at https://solvnet.synopsys.com, clicking Support, and then clicking “Open A Support Case.”
- Send an e-mail message to your local support center.
  - ❍ E-mail support_center@synopsys.com from within North America.
  - ❍ Find other local support center e-mail addresses at

    http://www.synopsys.com/Support/GlobalSupportCenters/Pages
- Telephone your local support center.
  - ❍ Call (800) 245-8005 from within North America.
  - ❍ Find other local support center telephone numbers at

    http://www.synopsys.com/Support/GlobalSupportCenters/Pages

1

PrimeTime GCA Overview

The PrimeTime GCA tool is a tool for constraint analysis and debugging of full-chip or block-level designs. Design analysis, based on rule checking, identifies many types of potential constraint problems. This includes constraints that are missing, invalid, unnecessary, inefficient, or conflicting. The PrimeTime GCA tool provides a very comprehensive and intuitive graphical user interface (GUI) where further debugging can be performed if needed. The individual violations in the design can be accessed through a violation browser, which provides suggestions about how to identify the root cause of a constraint problem and how to fix the problem.

This chapter describes the PrimeTime GCA startup in the following sections:

Constraints in the Design Flow

Correct and complete constraints are essential for timing-driven design implementation steps such as logic synthesis, placement, and routing. Constraints describe the operating environment and other information essential for valid static timing analysis, whether the analysis is stand alone or embedded in an optimization tool. Timing constraints are typically specified as Synopsys Design Constraints (SDC) format files or as more general Tcl scripts. Incorrect or incomplete constraints can lead to wasted design iterations or even silicon failure. Designers can minimize risk and improve productivity by finding and fixing constraint problems as early as possible.

The PrimeTime GCA tool analyzes constraints at the gate level to determine where and how clock signals, data signals, and constants propagate through the design. Constraints are often changed at various steps. For example, pre-layout constraints can be modified to obtain constraints for placement, clock tree synthesis, routing, and chip assembly. Whenever the constraints are created or modified, they should be analyzed for correctness and consistency.

The PrimeTime GCA tool is a standalone tool that analyzes a design and its associated constraints for correctness, completeness, and consistency. You read in and link a top-level or block-level design in Verilog format and apply the constraints using the same SDC commands as you would in the Design Compiler or PrimeTime tools. The PrimeTime GCA tool applies the SDC constraints and ignores any tool-specific, non-SDC commands in the constraint script.

After you read, link, and constrain the design, you can apply one or more of the PrimeTime GCA commands to analyze the design. The first step is to run the `analyze_design` command, which checks the design and its constraints and generates a summary report that you view in a violation browser.

The violation browser is a GUI tool that makes it easy to view and examine the causes of potential errors in the design and its constraints. From the summary list, you can click a particular violation to get more detailed information about that violation. From the detailed violation report, you can click links to find the corresponding constraint command in the SDC script, to view the relevant part of the schematic, or to get suggestions for fixing the violation.

The following types of violations are detected:

- Incorrect capacitance values
- Incorrectly specified clocks
- Incorrectly specified generated clocks
- Incorrectly specified clock latency and transition times
- Incorrect or missing driving cells

- Incorrectly specified timing exceptions
- Incorrectly specified input and output delays
- Netlist problems such as conflicting drivers
- Conflicting case analysis values

To get more detailed information about the source of a problem, additional commands are available: `analyze_paths`, `analyze_clock_networks`, `analyze_unclocked_pins`, and `report_case_details`. These commands can help you fix problems such as blocked timing paths and missing clock definitions. The `report_analysis_coverage`, `report_exceptions`, and `compare_block_to_top` commands can help you debug multi-scenario constraints, multiple timing exceptions, and conflicting constraints in hierarchical designs.

PrimeTime GCA Debugging Features

The PrimeTime GCA tool has numerous unique capabilities. Some of the important features of the tool include:

- A set of rules that covers a broad set of potential problems with constraints
- Support for user-defined rules
- Ability to analyze multiple scenarios, such as operating modes, in a single process
- Capacity and performance to efficiently analyze designs containing many millions of cell instances
- Powerful diagnostics and detailed analysis commands that help users find the root cause of reported problems and understand the impact of constraints
- A rich set of collection commands and object attributes that are provided to enable debugging and custom scripting
- Rule checkers that are fully compliant with SDC version 1.9 to identify syntax errors for invalid objects
- An interactive, Tcl-based shell that supports most SDC commands as well as loops, variables, conditional constructs, and collections
- GUI output that is easy to understand and interpret. Most constraint references include detailed information such as the source file name and line number where the constraint was defined
- Excellent consistency with PrimeTime and other Synopsys tools. This includes constraint interpretation, propagation of clocks, signals and constants, user interface, and more

Analysis and Debugging Methodology

The PrimeTime GCA tool accepts the following input data:

- Verilog netlists
- Cell libraries in Synopsys .db format
- Design constraints in SDC or PrimeTime Tcl format
- Tool control scripts in Tcl format

The PrimeTime GCA tool produces the following output data:

- Text reports
- Query commands and object attributes for interactive debugging, custom reporting, or conditional scripting

The steps in a typical PrimeTime GCA constraint analysis flow are as follows:

1. Start the PrimeTime GCA tool by entering `gca_shell` at a UNIX prompt.

   See “Starting a Session” on page 1-6 for more information.

2. Read in your design using the following commands:

   ```
   set_app_var search_path ...
   set_app_var link_path ...
   read_verilog ...
   ```

   See “Reading the Design” on page 1-8 for more information.

3. Link the design using the following commands:

   ```
   current_design design_name
   link_design
   ```

4. Apply constraints for each scenario.

   For Tcl scripts, use the `source` *`constraint_file`* command.

   For SDC commands, use the `read_sdc` *`constraint_file`* command.

   See Chapter 2, “Rule Checking” for more information about supported constraints.

5. Analyze your design using the following command:

   ```
   analyze_design
   ```

   See Chapter 2, “Rule Checking” for more information.

6. Diagnose problems.

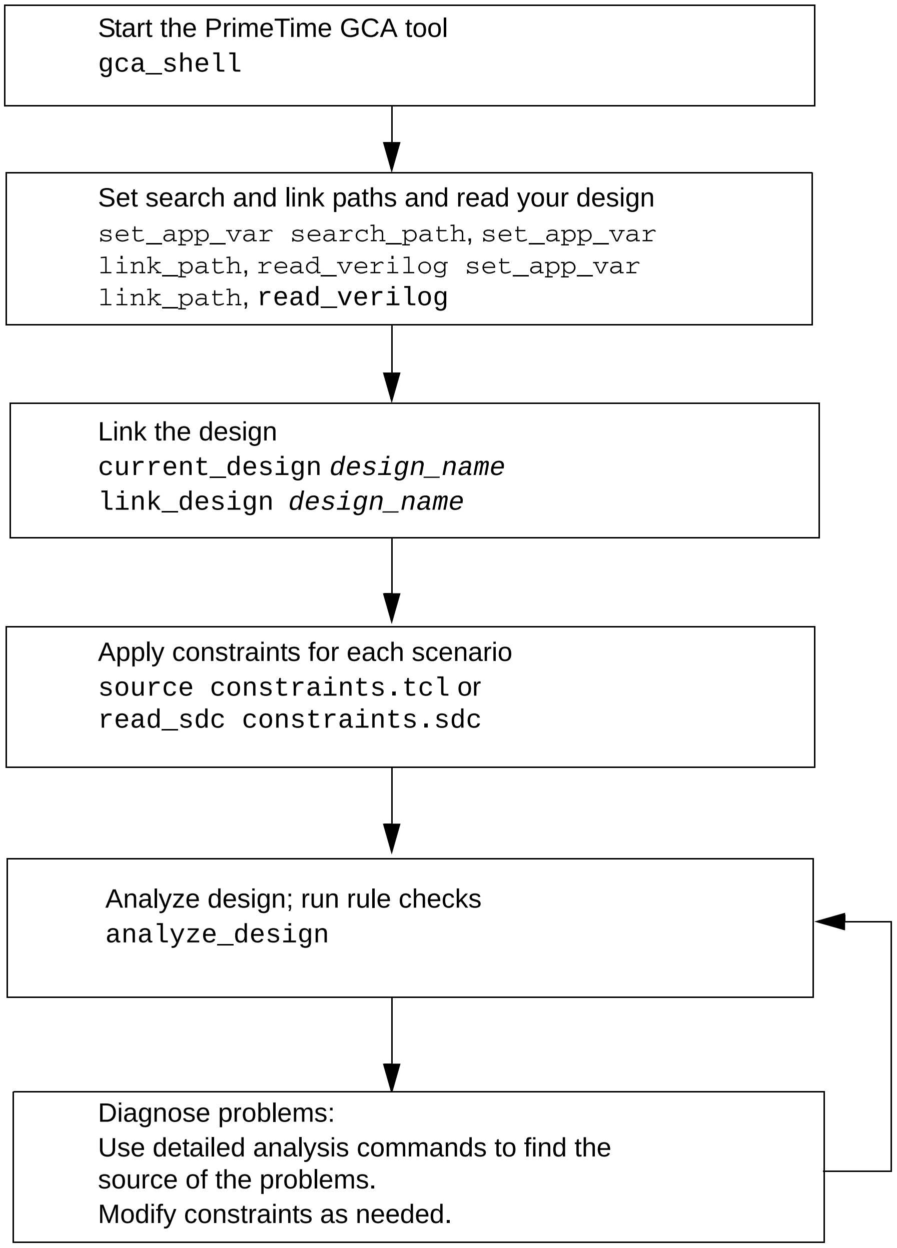
You can diagnose issues with a set of dedicated commands.

See “Additional Analysis Features” in Chapter 3 for more information.

7. Repeat the analysis and diagnosis steps as needed.

Figure 1-1 shows a typical flow for constraint analysis.

Figure 1-1 Typical PrimeTime GCA Flow


Start the PrimeTime GCA tool
gca_shell
Set search and link paths and read your design
set_app_var search_path, set_app_var link_path, read_verilog set_app_var link_path, read_verilog
Link the design
current_design design_name
link_design design_name
Apply constraints for each scenario
source constraints.tcl or
read_sdc constraints.sdc
Analyze design; run rule checks
analyze_design
Diagnose problems:
Use detailed analysis commands to find the source of the problems.
Modify constraints as needed.

Starting a PrimeTime GCA Session

PrimeTime GCA is a GUI-driven tool that runs under the Linux operating system. Before you can use the PrimeTime GCA tool, the software must be installed and licensed at your site. For information about installation and licensing, see the documentation that comes with the software release.

This section covers the following topics:

- Starting a Session
- License
- Setup Files
- Command Log File

Starting a Session

The PrimeTime GCA executable is called `gca_shell`. To start the tool, enter the following at the operating system prompt:

```
% gca_shell
```

The PrimeTime GCA tool automatically checks out a license, starts the GUI, and displays the initial message and the `gca_shell` prompt, similar to the following message.

```
PrimeTime (R) GCA
Version ... for RHEL64 -- ...
Copyright (c) 1988-2014 by Synopsys, Inc
ALL RIGHTS RESERVED
This program is proprietary and confidential ...
gca_shell>
```

If the PrimeTime GCA tool fails to start, verify that

- The PrimeTime GCA tool is correctly installed
- The PrimeTime installation path is included in your path definition
- The Synopsys license server is running
- Your Synopsys license key file is available and current, and that it includes a PrimeTime GCA license

If you need assistance, ask your system administrator or consult the installation and licensing documentation.

To end a PrimeTime GCA session, enter the `quit` or `exit` command at the PrimeTime GCA prompt:

```
gca_shell> exit
Maximum memory usage for this session: 0.72 MB
CPU usage for this session: 0 seconds
Diagnostics summary: 2 errors
Thank you for using gca_shell!
%
```

License

You need a PrimeTime GCA license to start `gca_shell`. The PrimeTime GCA tool automatically checks out a license when you start the tool. When you exit the PrimeTime GCA tool, the license is automatically checked in, allowing others at your site to use it. In addition, the PrimeTime GCA tool requires that you have a PrimeTime license available on the same server. The PrimeTime license is not checked out, but the license must be present for the PrimeTime GCA tool to start.

Setup Files

Each time you start a PrimeTime GCA session, the tool runs the commands contained in a set of setup files. You can put commands into a setup file to set variables, to specify the design environment, and to select your preferred working options.

The name of each setup file is .synopsys_gca.setup. The tool checks for the presence of the file in the following directories:

1. The Synopsys installation setup directory at admin/setup. For example, if the installation directory is /usr/synopsys, the setup file name is /usr/synopsys/admin/setup/.synopsys_gca.setup.
2. Your home directory.
3. The current working directory from which you started the tool.

If more than one of these directories contains a .synopsys_gca.setup file, the files are executed in the order shown previously: first in the Synopsys setup directory, then in your home directory, and finally in the current working directory. Typically, the file in the Synopsys setup directory contains setup information for all users at your site; the one in your home directory sets your personal preferred working configuration; and the one in your current working directory sets the environment for the current project.

To suppress execution of any .synopsys_gca.setup files, use the `-no_init` option when you start the tool.

Command Log File

The PrimeTime GCA tool saves the session history in a file called the command log file. This file contains all the commands executed during the session and serves as a record of your work. You can repeat the whole session later by running the file as a script using the `source` command.

The PrimeTime GCA tool creates the log file in the current working directory and names it gca_shell_command.log. Any existing log file with the same name is overwritten. Before you start a new PrimeTime GCA session, be sure to rename any log file that you want to keep.

You can specify a different name for the command log file by setting the `sh_command_log_file` variable in your setup file. You cannot change this variable during a working session.

Reading the Design

The first step in the PrimeTime GCA methodology is to read in the gate-level design description and associated technology library information. Use the `read_verilog` command to read in your designs. After you read in a set of design files, use the `link_design` command to resolve all references between the design and the timing library.

You can set variables with the `set_app_var` command in the PrimeTime GCA tool. The `search_path` variable specifies a list of directories from which to search the design and library files so that you do not need to specify a full path each time you read in a file. The `link_path` variable specifies where and in what order the PrimeTime GCA tool looks for design files and library files for linking the design. For example:

```
gca_shell> set_app_var search_path ". /u/proj/design /u/proj/lib"
gca_shell> set_app_var link_path "* STDLIB.db"
```

The tool searches for design and library files in the current directory, then the “/u/proj/design” directory, and lastly the “/u/proj/lib” directory. The "*" entry in the link path means to first search through all designs loaded into memory for library elements.

Reading Designs With Incomplete or Mismatched Netlists

During early design development, design data is updated often. For example, a block recently updated by a block-level designer might have fewer pins than the same block that was used by a top-level designer. This causes a design mismatch. To link designs with incomplete or mismatched netlists, set the `link_allow_design_mismatch` variable to true. The default is false. When mismatched pins, ports, cells, or nets are linked, the tool issues DMM warning messages.

If an object is affected by a mismatch, the `is_design_mismatch` attribute of the object is set to true. The default is false. The `is_design_mismatch` attribute is viewable in the Properties dialog box in the GUI.

When the `link_allow_design_mismatch` variable is true, the tool links designs with the following types of netlist problems:

- Pins exist in higher-level hierarchies that do not exist at lower-level hierarchies

  If the higher-level hierarchies have pins that do not exist at the lower level, these extra pins are ignored and do not exist in the linked netlist. Nets connected to unconnected higher-level pins are left dangling. Constraints are not applied on these pins.

  The PrimeTime GCA tool also links netlists with anchor cells inserted on the dangling nets. In this case, constraints related to mismatched pins in the in-memory linked design are preserved.

- The tool ignores extra bits on the bus. If the width of a bus varies for different hierarchical blocks, the bits of the bus are connected beginning with the least significant bits. If the lower-level block has additional bus bits not present in the next higher level, the bits are unconnected. If the higher-level block has additional bits that are not present in the lower-level block, the pins are ignored.

- Library and netlist cells contain different power and ground (PG) information

  If there is no PG pin information in the library, but the information is in the netlist, then the mismatch is treated as if pins exist in higher level hierarchies of the design, but not at the lower level.

- Cells missing from the library

  By default, the linker automatically creates black boxes for unresolved references, which enables the tool to continue linking. The tool issues a LNK-005 message for this condition.

  When the `link_allow_design_mismatch` variable is set to true, the tool issues a DMM-905 message for unresolved references and continues linking.

To view the mismatches located while linking a design, use the `report_design_mismatch` command. Example 1-1 shows a report example.

Example 1-1 Reporting Mismatches

```
gca_shell> report_design_mismatch
****************************************
Report : design_mismatch
Design : top
...
****************************************
Message   Design     Object     Type       Count  Mismatch Description
---------------------------------------------------------------------------
  DMM-038 design     object     type       1      Pin "%s" of cell "%s" of reference
                                                  "%s" in design "%s" shows
                                                  inconsistency between master and
                                                  instantiation because %s.
                                                  Ignoring this pin.
          top        u_block/reset pin    1 of 1  Pin "reset" of cell "u_block" of
                                                  reference "top" in design
                                                  "block" shows inconsistency between
                                                  master and instantiation becuase it
                                                  does not exist in the reference
                                                  reference block. Ignoring this pin.

Message   Design     Object     Type       Count     Mismatch Description
---------------------------------------------------------------------------
  DMM-905 design     object     type       48        Cannot find the design '%s' in
                                                     design libraries.
          top        BUFFD1     design     1 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     2 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     3 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     4 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     5 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     6 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     7 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     8 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     9 of 48   Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     10 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     11 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     12 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     13 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     14 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     15 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top        BUFFD1     design     16 of 48  Cannot find the design 'BUFFD1'
                                                     in design libraries.
          top         BUFFD1      design      17 of 48  Cannot find the design 'BUFFD1'
                                                        in design libraries.
          top         BUFFD1      design      18 of 48  Cannot find the design 'BUFFD1'
                                                        in design libraries.
          top         BUFFD1      design      19 of 48  Cannot find the design 'BUFFD1'
                                                        in design libraries.
          top         BUFFD1      design      20 of 48  Cannot find the design 'BUFFD1'
                                                        in design libraries.
Information: Reaching the message display limit of report_design_mismatch. (UIC-063)
1
```

Saving and Restoring Sessions

The `save_session` and `restore_session` commands allow you to save the current state of a session and restore the same session later. Using the `restore_session` command takes you to the same point in the analysis by using only a small fraction of the original runtime. This command removes any existing design data and library data in memory before it restores the saved session.

A saved and restored session retains the following information:

- Libraries
- Netlists
- Constraints
- Results of the `analyze_design`, `compare_block_to_top` and `compare_constraints` commands
- Application variables, simple Tcl variables, and netlist collections
- User-defined rules, violation waivers, and rule sets

The platforms used for saving and restoring the session can be different but the version of the tool must be the same.

You must specify a session directory name when you save or restore a session. For example,

```
gca_shell> save_session my_session1A
...
gca_shell> restore_session my_session1A
```

The session is saved in a new subdirectory created in the current directory.

Using the Shell Interface

The PrimeTime GCA command-line editor allows you to work interactively with `gca_shell` by using key sequences, much as you would work in a UNIX shell. The command-line editor supports the gcatcl command language. The PrimeTime GCA tool has the same Tcl command-line interface as the Design Compiler, IC Compiler, and PrimeTime tools.

For more information about the shell interface, see the following sections:

- Changing the Settings of the Command-Line Editor
- Listing Key Mappings
- Setting the Key Bindings
- Navigating the Command Line
- Completing Commands, Variables, and File Names
- Searching the Command History

Changing the Settings of the Command-Line Editor

The command-line editor is enabled by default. You can disable the command-line feature by setting the `sh_enable_line_editing` variable to false in the .synopsys_gca.setup file. You can use the `set_cle_options` command to change the default settings. Use options to this command as described in Table 1-1:

Table 1-1 Using the set_cle_options Command Options

To do this	Use this option
Set the key bindings (default is emacs editing mode)	`-mode vi | emacs`
Set the terminal beep (default is off)	`-beep on | off`
Specify default settings	`-default`

If you enter the `set_cle_options` command without any options, the current settings are displayed.

Listing Key Mappings

The command-line editor allows you to access any of the last 1000 commands by using a combination of keys. In addition, you can manipulate text on the command line and cut and paste text. Cutting text deletes text from the current line and saves it for later use. Pasting text reinserts text into the line.

These features are available in both vi and emacs mode.

Note:

In the key mappings displayed when you use the `sh_list_key_bindings` command, the text Ctrl-K is the character that results when you hold down the Ctrl key and press k.

Meta-K is the character that results when you hold down the meta key and press k. On many keyboards, the meta key is labeled Alt. On keyboards with two Alt keys, the one on the left of the Space bar is generally set as the meta key. The Alt key on the right of the Space bar might also be configured as the meta key or some other modifier, such as Compose, which is used to enter accented characters.

If your keyboard does not have a meta or Alt key or any other key configured as a meta key, press Esc followed by the k key (for meta-K). This is known as “metafying” the k key.

Setting the Key Bindings

By default, the key bindings are set to emacs editing mode. To change the key bindings to vi mode, use the `-mode` option of the `set_cle_options` command. You can also use the `sh_line_editing_mode` variable to change the key bindings to vi mode. You can set this variable either in the .synopsys_gca.setup file or at the shell prompt.

Examples

```
gca_shell> set_cle_options -mode emacs
Information: Command line editor mode is set to emacs
successfully. (CLE-01)

gca_shell> set sh_line_editing_mode vi
Information: Command line editor mode is set to vi
successfully. (CLE-01)
vi

gca_shell> set sh_line_editing_mode abc
Error: Command line editor mode cannot be set to ’abc’.
Proceeding with vi mode. (CLE-02)
vi
```

Navigating the Command Line

Use the keys listed in Table 1-2 to navigate the command line in both vi and emacs mode.

Table 1-2 Command-Line Navigation Keys

Key	Action
Down	Moves the cursor down to the next command
Up	Moves the cursor up to the previous command
Left	Moves the cursor to the previous character
Right	Moves the cursor to the next character
Home	Moves the cursor to the start of the current line
End	Moves the cursor to the end of the line

Completing Commands, Variables, and File Names

You can press the Tab key to complete commands automatically, including nested commands and their options, variables, and file names. Additionally, you can use the Tab key to automatically complete aliases (short forms for the commands you commonly use, defined with the `alias` command). When removing alias definitions by using the `unalias` command, you can use the command completion feature to list alias definitions.

In all these cases, the results are sorted alphabetically. If the command-line editor cannot find a matching string, it lists all closely matching strings.

Table 1-3 lists the results of pressing the Tab key within different contexts.

Table 1-3 Result of Pressing the Tab Key Within Different Contexts

Context	Action taken by the command-line editor
Command is not entered fully	Completes the command
Command is followed by a hyphen (-)	Completes the command argument
After a > or | command	Completes the file name
After a set, unset, or printvar command	Completes the variable
After a dollar sign ($)	Completes the variable

Table 1-3 Result of Pressing the Tab Key Within Different Contexts (Continued)

Context	Action taken by the command-line editor
After the help command	Completes the command
After the man command	Completes the command or variable
In all other contexts	Completes file names

Searching the Command History

The command-line editor provides an incremental search capability. You can search the command history in both vi and emacs mode by pressing Ctrl+R. A secondary prompt appears below the command prompt. The command-line editor searches the history as you enter each character of the search string and displays the first matching command. You can continue to add characters to the search string if the matching command is not the one you are searching for. As long as the search string is valid, a colon (:) appears in the secondary search prompt; otherwise a question mark (?) appears in the secondary search prompt and the command-line editor retains the last successful match on the prompt. After you find the command that you are searching for, you can press the Return key to run the command.

Using Threaded Multicore Analysis

Threaded multicore analysis enables the PrimeTime GCA tool to improve performance by running multiple threads on the cores available on that server. Threaded analysis is enabled by default, with up to four cores that can be used for each license. When using threaded multicore analysis, the tool executes many time-consuming algorithms in parallel to yield a faster run without requiring any script changes.

The following commands use threaded multicore technology:

- `compare_block_to_top`
- `compare_constraints`
- `save_session`

You control the number of cores the tool uses by setting the `-max_cores` option of the `set_host_options` command. You can set this option to `1` if only a single core is expected to be used. A number higher than 4 requires more then one license. For example, if the `-max_cores` option is set from 5 and 8, two licenses are used.

In addition to the number of cores specified by the `set_host_options` command, the tool gives a warning and limits the used cores to the physical number of cores available on the server. For example, in a dual core server, threads are launched to use only the two available cores.

2

Rule Checking

This chapter describes rule-checking capabilities in the following sections:

Rules and Violations

The PrimeTime GCA tool checks a loaded, linked, and constrained design for a variety of design-related and constraint-related conditions. Each condition that is checked is called a rule. An occurrence of something in the design that fails to meet a checked condition is called a rule violation. There can be multiple violations of a given rule in different places within the design. A violation can have a severity level of Information, Warning, or Error, depending on the type of rule.

Each rule has a designated code consisting of three or four letters, an underscore, and four digits. For example, the code "CLK_0003" represents a clock rule, "Generated clock clock_name is not expanded because it has no clock reaching its master source pin_name." If a generated clock is defined in the design, but the `-source` option of the `create_generated_clock` command designates a pin that does not have a clock defined on it, the tool detects and reports this improper constraint specification as a CLK_0003 violation.

The `analyze_design` command checks a standard set of rules. The `compare_block_to_top` command checks another, different set of standard rules, and the `compare_constraints` command checks yet another set of standard rules. You can disable any given rule so that it is not checked. Some rules are disabled by default to save runtime; you can enable these rules when you want them to be checked.

You can skip checking of a particular rule for a given set of conditions, such as checking for a particular block in the design or checking of a particular clock. A specified condition that prevents checking of a rule is called a waiver.

In addition to the standard rules, you can create your own rules to find and report conditions that are important for your design. These are called user-defined rules.

Built-In Rules

When you start the PrimeTime GCA tool, it automatically loads the set of built-in rules shown in Table 2-1. These rules check your design for potential timing constraint and library-related problems.

Table 2-1 Built-In Rules

Boundary Conditions	CAP_xxxx – capacitance values
	DRV_xxxx – drive constraints
	EXD_xxxx – external delays

Table 2-1 Built-In Rules (Continued)

Constraints/Exception Analysis	CAS_xxxx – case analysis
	EXC_xxxx – timing exceptions
Clocks	CGR_xxxx – clock groups
	CLK_xxxx – clock properties
	CNL_xxxx – network latencies
	CTR_xxxx – clock transitions
	CSL_xxxx – clock source latencies
	UNC_xxxx – clock uncertainties
General	CMP_xxxx – tools compatibility
	DES_xxxx – design constraints
	HIER_xxxx – hierarchical
	LOOP_xxxx – timing loops
	PRF_xxxx – performance
	NTL_xxxx – netlists
	UNT_xxxx – library units

See the PrimeTime GCA Online Help for detailed rule descriptions.

Supported Constraints

The PrimeTime GCA tool supports all constraints in SDC format and Tcl format.

For more detailed information about SDC, see the *Using the Synopsys Design Constraints Format Application Note*.

For more detailed information about Tcl, see the *Using Tcl With Synopsys Tools* documentation.

The PrimeTime GCA tool supports the `source` command to read in constraints in Tcl scripting format. Example 2-1 shows a Tcl constraint script fragment. You need to use the `source` command to load these constraints, as shown in Example 2-2.

Example 2-1 Tcl Constraints File constraints.tcl

```
...
foreach_in_collection i [all_clocks] {
        echo [format "Clock : '%s'"  [get_attribute $i full_name]]
        set_clock_uncertainty 1.5 $i
}
...
```

Example 2-2 Using the Source Command to Load Tcl Scripted Constraints

```
set_app_var sh_continue_on_error true

set design_dir ./

set MyDesign b_top
current_design $MyDesign
link_design

create_scenario system
create_scenario test

current_scenario system
source constraints.tcl
...
```

The PrimeTime GCA tool supports the `read_sdc` command to read constraints in SDC format.

The PrimeTime GCA tool supports multiple scenarios. To create constraints for each scenario, first define the scenarios:

```
create_scenario system
create_scenario myscenario_1
create_scenario myscenario_2
```

You can use the script in Example 2-3 to apply both common and scenario-specific Tcl constraints.

Example 2-3 Tcl Script for Loading Common and Unique Constraints

```
set all_scenarios “system myscenario_1 myscenario_2”
foreach scenario $all_scenarios {
  create_scenario ${scenario}
  source ./scripts/common.tcl
  source ./scripts/${scenario}_constraints.tcl
}
```

Reporting Constraint Acceptance

You can use the `statistics` keyword with the `-include` option of the `report_constraint_analysis` command to report the constraints accepted for analysis by the tool. By using the `statistics` keyword, you can quickly see if the constraints are correctly sourced into the PrimeTime GCA tool. Example 2-4 shows the statistics report.

Example 2-4 Statistics Report

```
gca_shell> report_constraint_analysis -include statistics
****************************************

Report : report_constraint_analysis
         -include {statistics }
    -style {full}
         ...
Date   : ...
****************************************
Constraint Processing Statistics

Constraint                          Accepted     Rejected     Total
--------------------------------------------------------------------------------
create_clock                        7            0            7
set_clock_uncertainty               1            0            1
set_input_delay                     6            0            6
set_output_delay                    6            0            6
1
```

Exception Order of Precedence

If different timing exception commands are in conflict for a particular path, the exception the tool uses for that path depends on the exception types or the path specification methods used in the conflicting commands. A set of rules establishes the order of priority for different exception-setting situations.

Note that the tool applies the exception precedence rules independently on each path (not each command). For example, suppose that you use the following commands:

```
gca_shell> set_max_delay -from A 5.1
gca_shell> set_false_path -to B
```

The `set_false_path` command has priority over the `set_max_delay` command, so any paths that begin at A and end at B are false paths. However, the `set_max_delay` command still applies to paths that begin at A but do not end at B.

Exception Type Priority

The following pairs of timing exception types are not considered to be in conflict, so both settings can be valid for a path:

- Two `set_false_path` command settings
- `set_min_delay` and `set_max_delay` command settings
- `set_multicycle_path -setup` and `-hold` command and option settings

In case of conflicting exceptions for a particular path, the timing exception types have the following order of priority, from highest to lowest:

1. `set_false_path`
2. `set_max_delay` and `set_min_delay`
3. `set_multicycle_path`

For example, if you declare a path to be false and also set its maximum delay to some value, the false path declaration has priority. The maximum delay setting is ignored. You can list ignored timing exceptions by using the `report_exceptions -ignored` command.

Path Specification Priority

If you apply the same type of timing exception using commands with different path specifications, the more specific command has priority over the more general one. Exceptions of any type on more specific objects, such as pins or ports, take precedence over exceptions applied to more general objects, such as clocks. For example,

```
gca_shell> set_max_delay 12 -from [get_clocks CLK1]
gca_shell> set_max_delay 15 -from [get_clocks CLK1] \
          -to [get_clocks CLK2]
```

The first command sets the maximum delay of all paths starting from CLK1. However, the second command is more specific, so it overrides the first command for paths starting at CLK1 and ending at CLK2. The remaining paths starting from CLK1 are still controlled by the first command.

The various `-from` and `-to` path specification methods have the following order of priority, from highest to lowest:

1. `-from` *`pin`*, `-rise_from` *`pin`*, `-fall_from` *`pin`*
2. `-to` *`pin`*, `-rise_to` *`pin`*, `-fall_to` *`pin`*
3. `-through`, `-rise_through`, `-fall_through`

4. `-from` *`clock`*, `-rise_from` *`clock`*, `-fall_from` *`clock`*
5. `-to` *`clock`*, `-rise_to` *`clock`*, `-fall_to` *`clock`*

Use the preceding list to determine which of two conflicting timing exception commands has priority (for example, two `set_max_delay` commands). Starting from the top of the list:

1. A command containing `-from` *`pin`*, `-rise_from` *`pin`*, or `-fall_from` *`pin`* has priority over a command that does not contain `-from` *`pin`*, `-rise_from` *`pin`*, or `-fall_from` *`pin`*.
2. A command containing `-to` *`pin`*, `-rise_to` *`pin`*, or `-fall_to` *`pin`* has priority over a command that does not contain `-to` *`pin`*, `-rise_to` *`pin`*, or `-fall_to` *`pin`*. And, so on down the list until the priority is resolved.

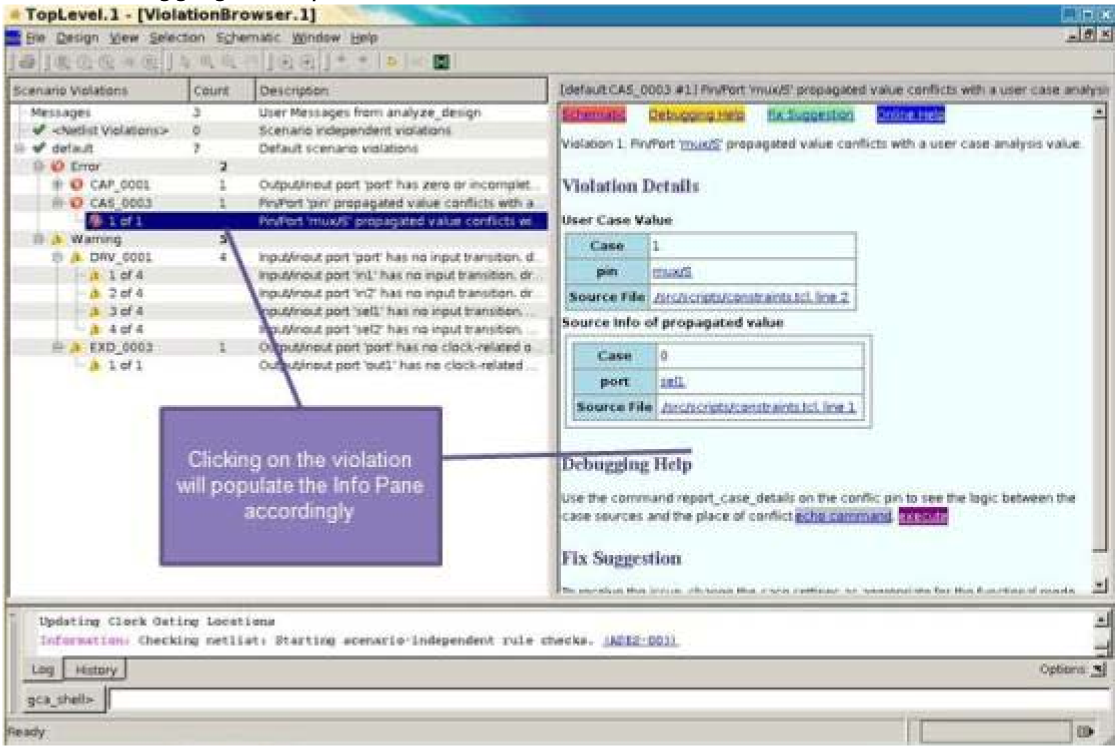
Here are some possible path specification combinations, listed in order of priority from highest to lowest, according to the preceding priority rules:

1. `-from` *`pin`* `-to` *`pin`*
2. `-from` *`pin`* `-to` *`clock`*
3. `-from` *`pin`*
4. `-from` *`clock`* `-to` *`pin`*
5. `-to` *`pin`*
6. `-from` *`clock`* `-to` *`clock`*
7. `-from` *`clock`*
8. `-to` *`clock`*

Analyzing a Violation

You can debug constraint problems by using the GUI. In the violation browser window, select the violation you want to analyze, and view the details of the violation that are shown in the information pane. Figure 2-1 and Figure 2-2 show the violation tree and the violation details in the information pane.

Figure 2-1 Debugging Example - Violation Browser Provides Access to Violation Details


TopLevel.1 - [ViolationBrowser.1]
File Design View Selection Schematic Window Help
Schematic
Debugging Help
Fix Suggestion
Online Help
Clicking on the violation will populate the Info Pane accordingly
Log
History
Options
gca_shell>
Ready

Scenario Violations	Count	Description
Messages	3	User Messages from analyze_design
<Netlist Violations>	0	Scenario independent violations
default	7	Default scenario violations
Error	2	
CAP_0001	1	Output/inout port 'port' has zero or incomplet...
CAS_0003	1	Pin/Port 'pin' propagated value conflicts with a
1 of 1		Pin/Port 'mux/S' propagated value conflicts wi
Warning	5	
DRV_0001	4	Input/inout port 'port' has no input transition, d...
1 of 4		Input/inout port 'in1' has no input transition, dr...
2 of 4		Input/inout port 'in2' has no input transition, dr
3 of 4		Input/inout port 'sel1' has no input transition, ...
4 of 4		Input/inout port 'sel2' has no input transition, ...
EXD_0003	1	Output/inout port 'port' has no clock-related o...
1 of 1		Output/inout port 'out1' has no clock-related ...

[default:CAS_0003 #1] Pin/Port 'mux/S' propagated value conflicts with a user case analysi

Violation 1. Pin/Port 'mux/S' propagated value conflicts with a user case analysis value.

Violation Details

User Case Value

Case	1
pin	mux/S
Source File	/src/scripts/constraints.tcl, line 2

Source Info of propagated value

Case	0
port	sel1
Source File	/src/scripts/constraints.tcl, line 1

Debugging Help

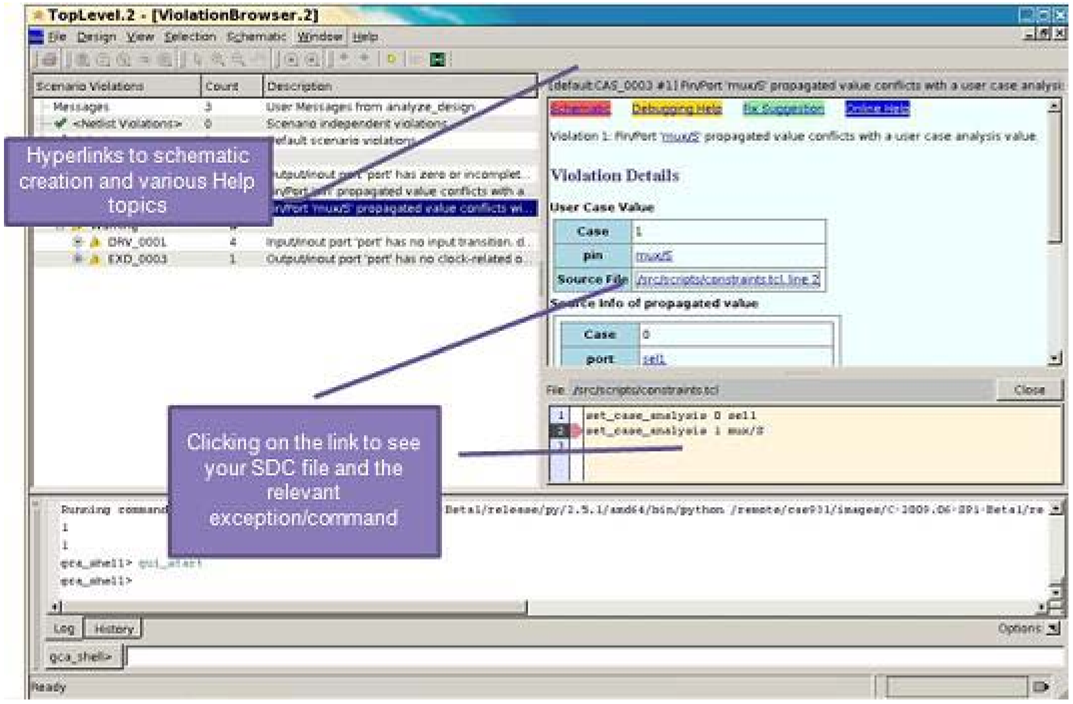
Use the command report_case_details on the conflic pin to see the logic between the case sources and the place of conflict echo command, execute

Fix Suggestion

```
Updating Clock Gating Locations
Information: Checking netlist: Starting scenario-independent rule checks. (AE02-001)
```

Figure 2-2 labels the various hyperlinks that you can click to access the SDC file and the relevant exception or command.

Figure 2-2 Debugging Example - Information Pane Links to Violation Details


TopLevel.2 - [ViolationBrowser.2]
File Design View Selection Schematic Window Help
Hyperlinks to schematic creation and various Help topics
Clicking on the link to see your SDC file and the relevant exception/command
Schematic
Debugging Help
Fix Suggestion
Online Help
Close
Log
History
Options

Scenario Violations	Count	Description
Messages	3	User Messages from analyze_design
<Netlist Violations>	0	Scenario independent violations
DRV_0001	4	Input/inout port 'port' has no input transition, d...
EXD_0003	1	Output/inout port 'port' has no clock-related o...

[default:CAS_0003 #1] Pin/Port 'mux/S' propagated value conflicts with a user case analysis

Violation 1: Pin/Port 'mux/S' propagated value conflicts with a user case analysis value

Violation Details

User Case Value

Case	1
pin	mux/S
Source File	/src/scripts/constraints.tcl, line 2

Source info of propagated value

Case	0
port	sel1

File: /src/scripts/constraints.tcl

```
set_case_analysis 0 sel1
set_case_analysis 1 mux/S
```

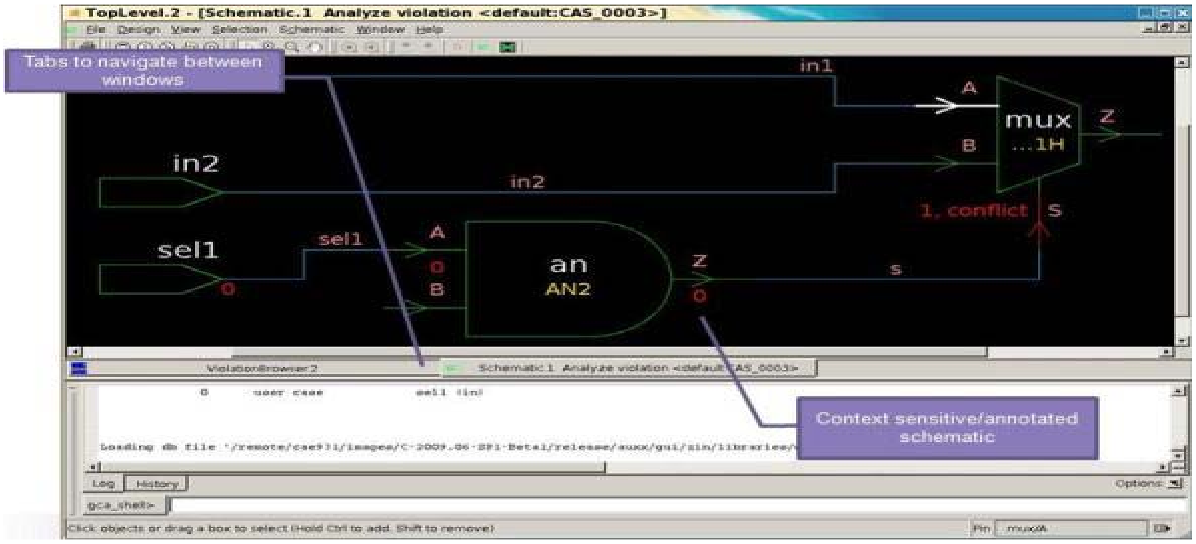
```
Running command ... Beta1/release/py/2.5.1/amd64/bin/python /remote/cae931/images/C-2009.06-SP1-Beta1/re
1
1
gca_shell> gui_start
gca_shell>
```

gca_shell>

Ready

Figure 2-3 shows the relevant schematic area of the problem constraint. The schematic is accessed from the schematic link in the information pane.

Figure 2-3 Schematic Viewer Shows Violation Circuit Path

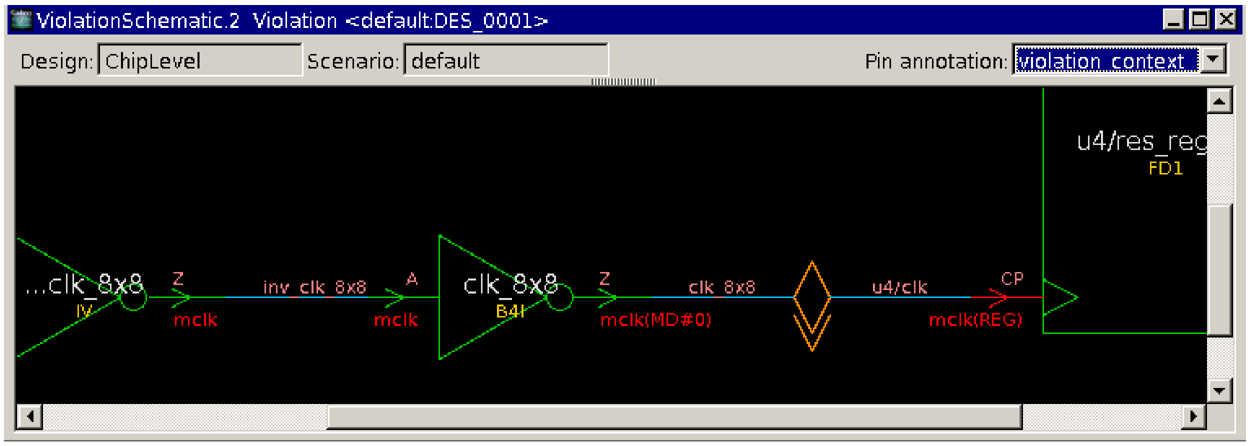

TopLevel.2 - [Schematic.1 Analyze violation <default:CAS_0003>]
File Design View Selection Schematic Window Help
Tabs to navigate between windows
in1
A
mux
...1H
Z
B
in2
in2
1. conflict
S
sel1
sel1
A
0
B
an
AN2
Z
0
s
0
ViolationBrowser.2
Schematic.1 Analyze violation <default:CAS_0003>
Context sensitive/annotated schematic
Log
History
Options
gca_shell>
Pin mux/A

```
          0      user case          sel1 (in)

Loading db file '/remote/cae931/images/C-2009.06-SP1-Beta1/release/auxx/gui/sin/libraries/
```

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)

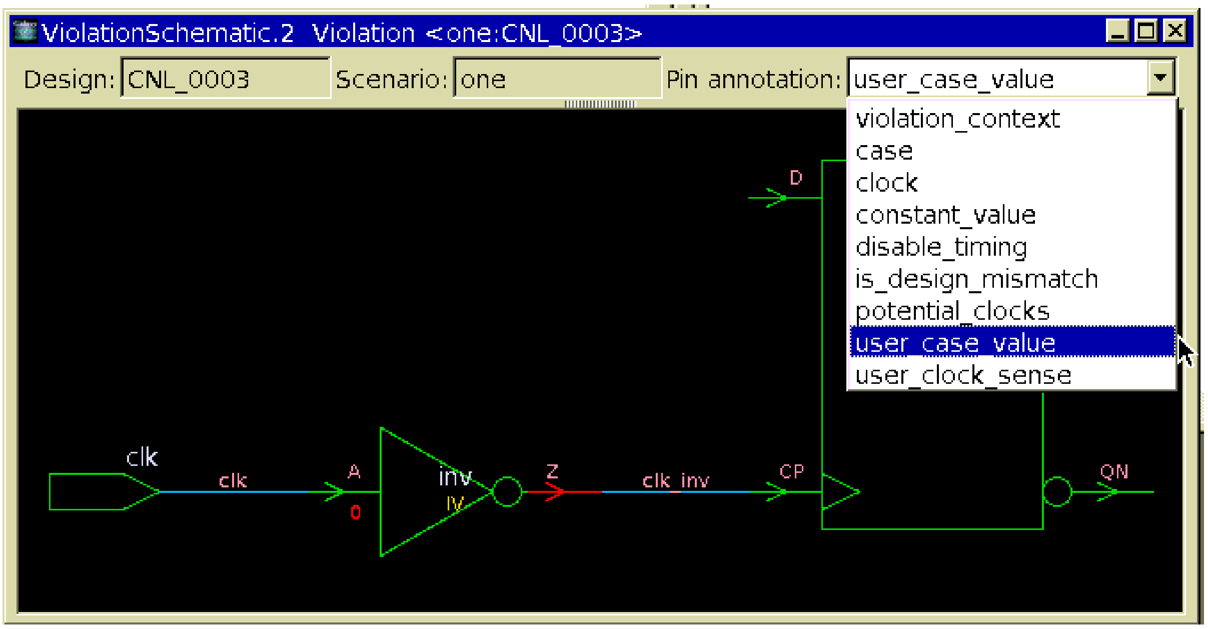
To help debug constraint problems, you can view clock and case annotations in the schematic view using the Pin annotation list, as shown in Figure 2-4. The information related to a violation is displayed by default.

Figure 2-4 Schematic Viewer With Full Violation Context



The Pin annotation list is expanded to provide a visualization of any of the annotations in the schematic, as shown in Figure 2-5.

Figure 2-5 Schematic Viewer With Complete Pin Annotation List



If further debugging is required, use the Debugging Help link, the Fix Suggestion link, or investigate the constraint in the SDC file by selecting the source file link in the Violation Details section of the information pane. In addition to those sources of help, you can consult the rule reference for this violation in the online Help.

Suppressing Violations

The PrimeTime GCA tool allows you to suppress rule violations. You can either disable a rule completely or suppress a specific instance of a rule violation, also known as waiving a violation. For example, you might not want the PrimeTime GCA tool to check if your design violates the capacitance-related rules; in this case you would disable the CAP_xxxx rules. As another example, you might want the PrimeTime GCA tool to analyze your design for capacitance rule violations on all but one output port; in this case you would waive violations of the CAP_xxxx rules on the specific output port.

The PrimeTime GCA tool includes commands for creating, reporting, removing, and writing out waivers. The GUI allows you to easily create and remove waivers.

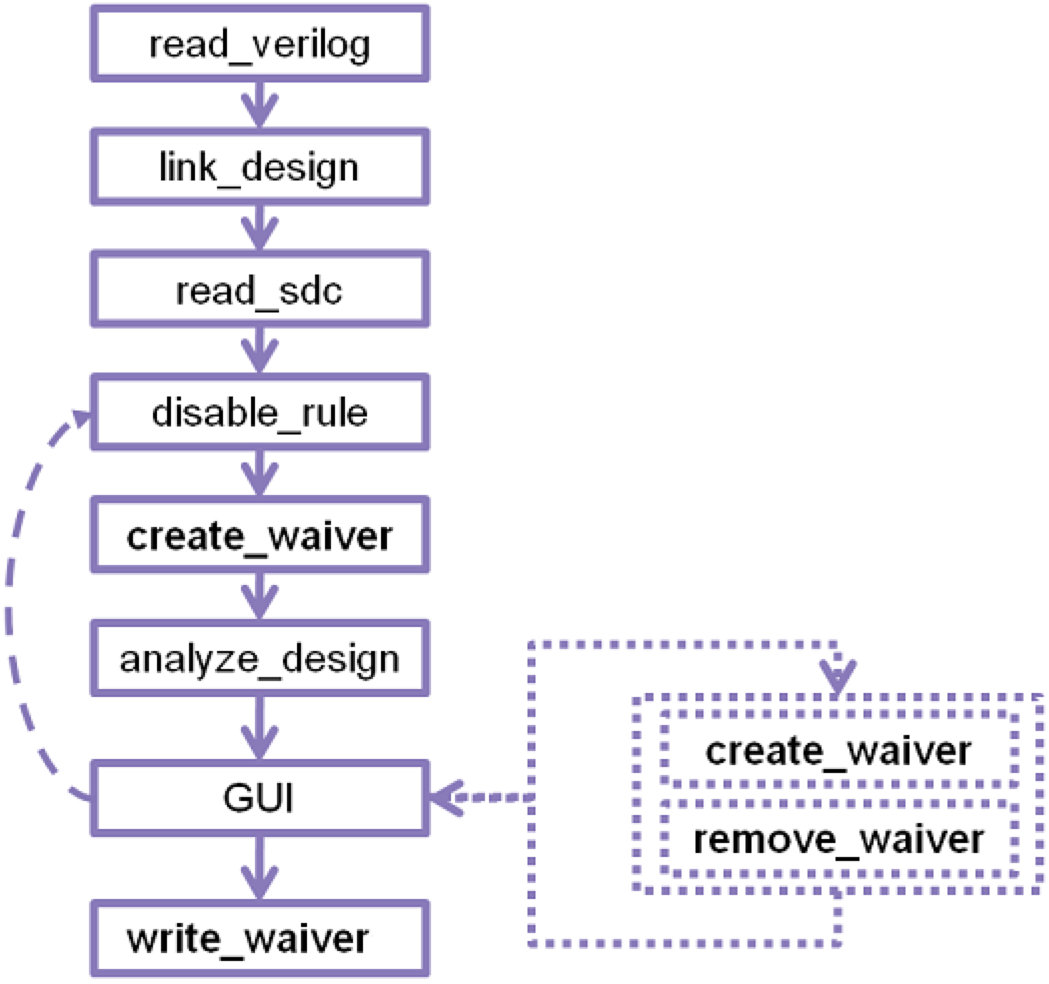
Violation suppression is discussed in the following sections:

- Overview of Violation Suppression Flow
- Disabling Rules
- Waiving Specific Violations of a Rule
- Using the create_waiver Command
- Creating Instance-Level Waivers
- Usage Guidelines
- Modifying Waivers
- Reporting Waivers
- Removing Waivers
- Writing Waivers to a File

Overview of Violation Suppression Flow

The violation suppression usage model is shown in Figure 2-6.

Figure 2-6 Violation Suppression Usage Model



As shown in Figure 2-6, you can suppress violations either before or after running the `analyze_design` command. If you suppress a violation after an analysis, the violation browser shows the violation marked it with a special icon to indicate that it has been suppressed. If you suppress a violation before analysis, a rule that has been disabled completely is omitted entirely from the report, whereas a rule violation that has been waived for certain instances is still reported, but shown with the special icon.

If a violation is already listed in the violation browser, its status is immediately updated when you waive it. This allows you to see the impact of the violation suppression on the current set of violations. The next time you run the `analyze_design` command, the suppressed violations are listed with a special icon in the violation browser.

Disabling Rules

If you do not want the PrimeTime GCA tool to check for violations of a rule, you can disable the rule by using the violation browser, the Waiver Configuration dialog box, or the `disable_rule` command.

To disable a rule using the violation browser,

1. Right-click the rule you want to disable.
2. Choose Disable rule.

   When you choose Disable rule, the PrimeTime GCA tool automatically generates the `disable_rule` command to disable the selected rule. The disabled rule has a disabled icon next to it in the violation browser.

To disable a rule using the Waiver Configuration dialog box,

1. Choose Design > Waiver Configuration.

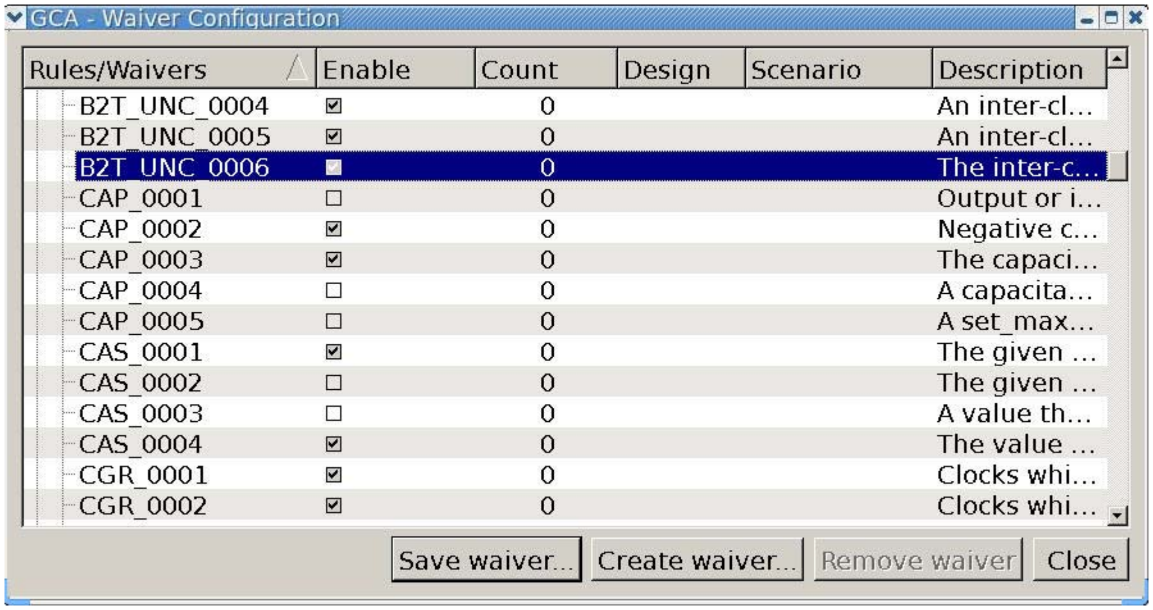
   The Waiver Configuration dialog box appears, as shown in Figure 2-7.

2. Uncheck the box next to the rule you want to disable.

   The rule is not checked or reported the next time you run the `analyze_design` command.

For information about disabling rules and reporting disabled rules, see “Suppressing Violations” on page 2-11 and “Reporting Waivers” on page 2-23.

Figure 2-7 Waiver Configuration Dialog Box



Rules/Waivers	Enable	Count	Design	Scenario	Description
B2T_UNC_0004	☑	0			An inter-cl...
B2T_UNC_0005	☑	0			An inter-cl...
B2T_UNC_0006	☑	0			The inter-c...
CAP_0001	☐	0			Output or i...
CAP_0002	☑	0			Negative c...
CAP_0003	☑	0			The capaci...
CAP_0004	☐	0			A capacita...
CAP_0005	☐	0			A set_max...
CAS_0001	☑	0			The given ...
CAS_0002	☐	0			The given ...
CAS_0003	☐	0			A value th...
CAS_0004	☑	0			The value ...
CGR_0001	☑	0			Clocks whi...
CGR_0002	☑	0			Clocks whi...

Waiving Specific Violations of a Rule

You can waive specific violations of a rule by using the violation browser, the Waiver Configuration dialog box, or the `create_waiver` command.

To waive a specific violation of a rule with the violation browser,

1. Right-click the violation.

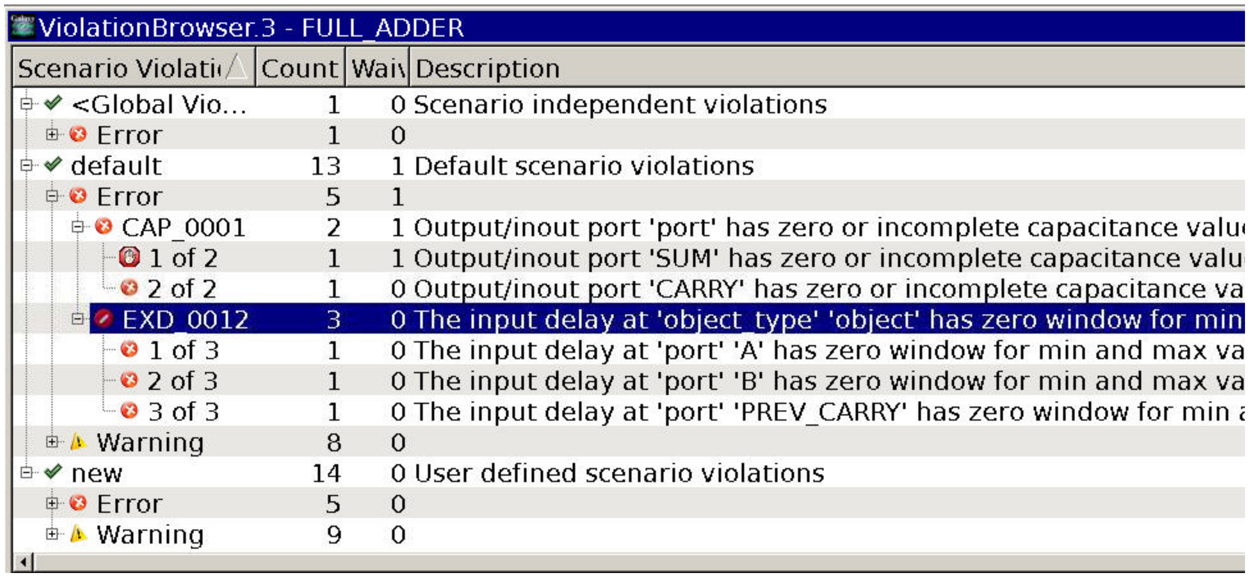
   You can select multiple violations. However, the “Undo last waiver” selection cannot undo multiple waivers; it will undo only the last waiver in the group.

2. Choose Waive violation.

   The tool generates a `create_waiver` command with complete arguments filled in to suppress the selected violation. The waived violation has a “waived” icon next to it in the violation browser, as shown in Figure 2-8.

   The waived violation and the icon disappear from the violation browser after you run the `analyze_design` command.

Figure 2-8 Waived Violation in the Violation Browser



ViolationBrowser.3 - FULL_ADDER

Scenario Violati	Count	Waiv	Description
<Global Vio...	1	0	Scenario independent violations
Error	1	0	
default	13	1	Default scenario violations
Error	5	1	
CAP_0001	2	1	Output/inout port 'port' has zero or incomplete capacitance valu
1 of 2	1	1	Output/inout port 'SUM' has zero or incomplete capacitance valu
2 of 2	1	0	Output/inout port 'CARRY' has zero or incomplete capacitance va
EXD_0012	3	0	The input delay at 'object_type' 'object' has zero window for min
1 of 3	1	0	The input delay at 'port' 'A' has zero window for min and max va
2 of 3	1	0	The input delay at 'port' 'B' has zero window for min and max va
3 of 3	1	0	The input delay at 'port' 'PREV_CARRY' has zero window for min
Warning	8	0	
new	14	0	User defined scenario violations
Error	5	0	
Warning	9	0	

To waive a specific violation of a rule with the Waiver Configuration dialog box,

1. Choose Design > Waiver Configuration.

   The Waiver Configuration dialog box appears.

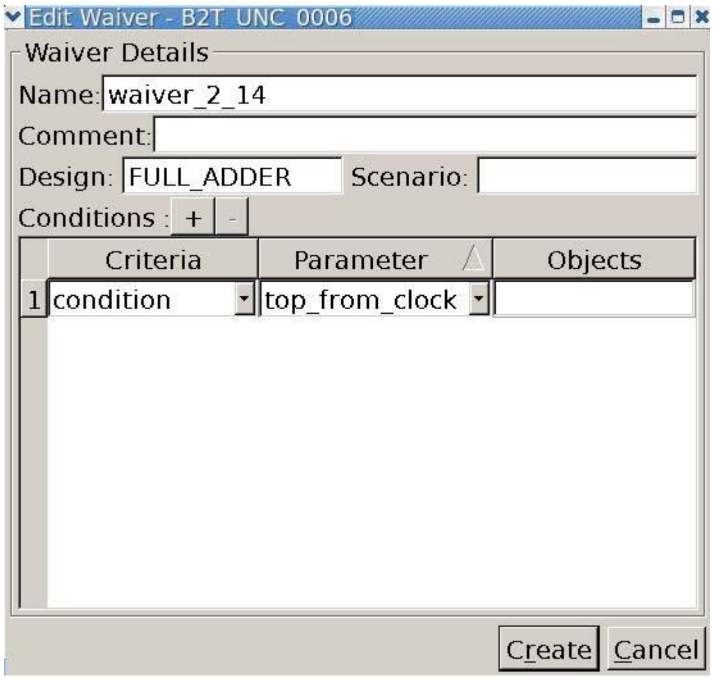
2. Click “Create waiver.”

   The Edit Waiver dialog box appears. Figure 2-9 shows the editor window.

3. Enter the waiver specification.

To disable a specific violation of a rule using the `create_waiver` command, see the "Using the create_waiver Command" on page 2-16.

Figure 2-9 Edit Waiver Dialog Box



Edit Waiver - B2T_UNC_0006

Waiver Details

Name: waiver_2_14

Comment:

Design: FULL_ADDER Scenario:

Conditions : + -

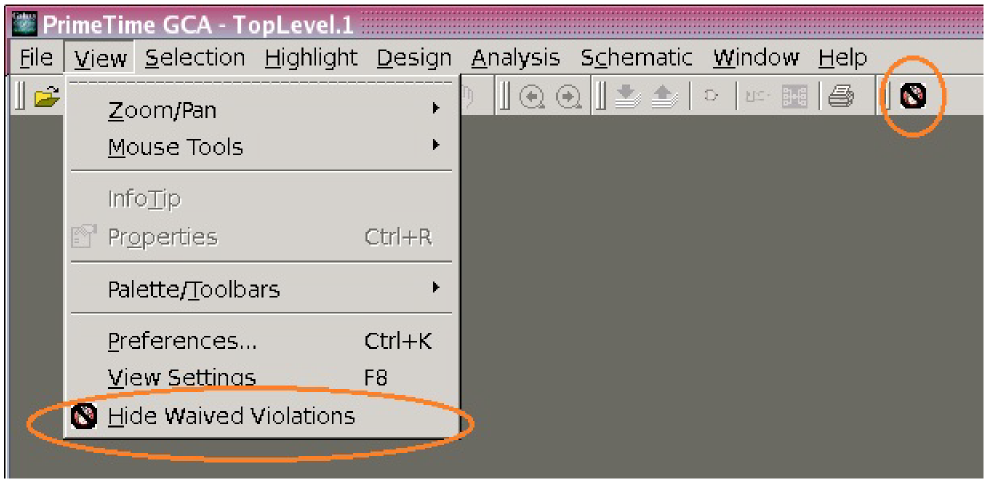
	Criteria	Parameter	Objects
1	condition	top_from_clock	

Create Cancel

For more information about disabling rules, see "Suppressing Violations" on page 2-11.

When you set the `hide_waived_violations` variable to false, the Hide Waived Violations command is toggled on the View menu, as shown in Figure 2-10.

Figure 2-10 Hide Waived Violations



Using the create_waiver Command

You can create waivers by using the `create_waiver` command.

In the `create_waiver` command, you can specify multiple `-condition` and `-not_condition` options. A violation is suppressed only if all of the `-condition` and `-not_condition` options are satisfied. You can specify collections of netlist objects with `-condition` options.

The argument to the `-condition` or `-not_condition` option must be a list consisting of two items: the name of the rule parameter, and a collection of netlist objects or constraints. To create the collection, you need to use a syntax such as `[get_clocks CLK]`.

When working with multiple designs, use the `-design` and `-scenario` options to assign waivers to specific designs. See the waivers in the following examples:

Example 1

Waiver using the `-condition` option

To waive violations of the CLK_0026 rule on clocks that start with CLK, use the following command:

```
gca_shell> create_waiver -rule CLK_0026 -condition [list "clock" \
              [get_clocks CLK*]]
```

Example 2

Waiver using the `-not_condition` option

To waive violations of the CLK_0026 rule on all clocks except CLK3, use the following command:

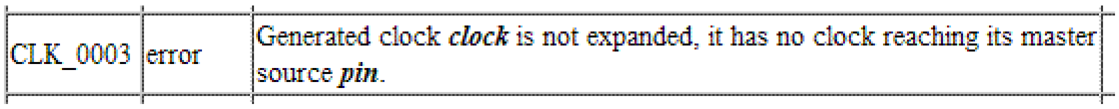
```
gca_shell> create_waiver -rule CLK_0026 \
              -not_condition [list "clock" [get_clocks CLK3]]
```

Example 3

Waiver using multiple parameters

The CLK_0003 rule checks multiple parameters: a clock and a pin.

Figure 2-11 Multiple Parameters



CLK_0003	error	Generated clock ***clock*** is not expanded, it has no clock reaching its master source ***pin***.

To create a waiver for this rule, you can specify conditions for both the clock and the pin. For example, to waive CLK_0003 violations if the clock is CLK1 or CLK2 and the source pin is not U1/A and not U1/B, use the following command:

```
gca_shell> create_waiver -rule CLK_0003 \
             -condition [list "clock" [get_clocks {CLK1 CLK2}] \
             -not_condition [list "pin" [get_pins U1/A] [get_pins U1/B]]
```

Example 4

Multiple waivers of the same rule

To waive violations of the CLK_0003 rule if the clock is CLK1 and the source pin is U1/A or the clock is CLK2 and the source pin is U1/B, you need to create two separate waivers, as shown in the following example:

```
gca_shell> create_waiver -rule CLK_0003 \
             -condition [list "clock" [get_clocks CLK1]] \
             -condition [list "pin" [get_pins U1/A]] \
gca_shell> create_waiver -rule CLK_0003 \
             -condition [list "clock" [get_clocks CLK2]] \
             -condition [list "pin" [get_pins U1/B]]
```

Example 5

Waiver using an exception parameter

To waive violations of the EXC_0001 rule with an exception specified from `U1/A` to `U2/Z` and the `rise_or_fall` parameter set to `rise`, use the following command:

```
gca_shell> create_waiver -rule EXC_0001 \
     -condition [list "exception" [get_exceptions -from U1/A -to U2/Z]] \
     -condition [list "rise_or_fall" "rise"]
```

If the rule on which the waiver is applied is scenario-dependent, the waiver is added to current scenario. If the rule is scenario-independent, the waiver applies to scenario-independent checks.

Netlist objects are interpreted with respect to the current instance. For example, the following two sets of commands produce equivalent waivers:

```
current_instance U1
gca_shell> create_waiver -name "my_waiver4" -rule CAS_0003 \
             -condition [list "pin" [get_pins U2/U3/A]]

current_instance U1/U2
gca_shell> create_waiver -name "my_waiver4" -rule CAS_0003 \
             -condition [list "pin" [get_pins U3/A]]
```

The waiver suppresses violations based on violation parameters that you specify in the waiver conditions. Parameter names can be retrieved using the `report_rule -parameters` *`rule_name`* command.

For example, to report the parameters available for the CLK_0026 rule, run the following command:

```
gca_shell> report_rule -parameters CLK_0026
****************************************
Report : rule
Version: ...
Date   : ...
****************************************

Rule          Severity  Status    Message
-------------------------------------------------------------------------
CLK_0026      warning   enabled   Clock 'clock' is used as data. One or
                                  more sources of the clock fans out to a
                                  register data pin or to a constrained
                                  primary output or inout port.

CLK_0026 parameters:
Name          Value Types
----------------------------------------
clock         clock
```

Example 6

Creating a waiver for a Block-to-Top (B2T) rule

When you create a waiver for a Block-to-Top rule, you work with two contexts: the TOP design and the BLOCK design. In the condition statement, before you can retrieve an object, you have to define from which context to retrieve. Depending on the violation you are waiving, you might also need to define the parameters in your violation such as “top_object_type” and “block_object_type.” The parameters are listed in the B2T rules which are described in the online Help. The waiver needs to apply to a particular design and scenario pair and an instance of the block.

```
gca_shell> create_waiver -rule B2T_CLK_0012 \
            -condition [list "blk_object" [current_design BLOCK ;
            get_pins zGateA]] \
            -condition [list "blk_object_type" "pin"] \
            -condition [list "top_object" [current_design TOP ; get_pins
gca_shell> b1/zGate/A]] \
            -condition[list  "top_object_type" "pin"] \
            -design1 TOP \
            -scenario1 default \
            -design2 BLOCK \
            -scenario2 default \
            -inst2 b1
```

Example 7

Creating a waiver for an SDC-to-SDC rule

When you create a waiver for an SDC-to-SDC rule, you work with two contexts: the first SDC definition and the second SDC definition. In the condition statement, before you can retrieve an object, you have to define from which context to retrieve. You also need to define the parameters in your violation. The parameters are described in the SDC-to-SDC rules which are available in the online Help. The waiver needs to apply to a particular design and scenario pair.

```
gca_shell> create_waiver -rule S2S_DIS_0001 \
     -condition [list "object" [current_design S2S_CLK_0001 ; get_pins
     ck2_i2/]] \
     -condition [list "object_type" "pin"] \
     -condition [list "scenario1" "set2"] \
     -condition [list "scenario2" "set1"] \
     -design1 zDesign \
     -scenario1 set1 \
     -design2 zDesign \
     -scenario2 set2
```

You can waive rules for specific cells or design instances by using the GUI or the command line. All related waiver commands, such as `remove_waiver`, `write_waiver`, and `report_waiver` support this feature.

You can create an instance-level waiver from the hierarchy browser or from the Waiver Configuration dialog box.

Creating Instance-Level Waivers

To create an instance-level waiver from the hierarchy browser,

1. Open the hierarchy browser.

   Select Design > Hierarchy Browser.

2. Click the cell instance.
3. Right-click the cell instance to open the pop-up menu. See Figure 2-12.
4. Choose Waive Instance.

   The instance waiver icon is displayed in the violation browser next to all the cell instances waived for that rule. See Figure 2-13.

To create an instance-level waiver from the GUI from the Waiver Configuration dialog box,

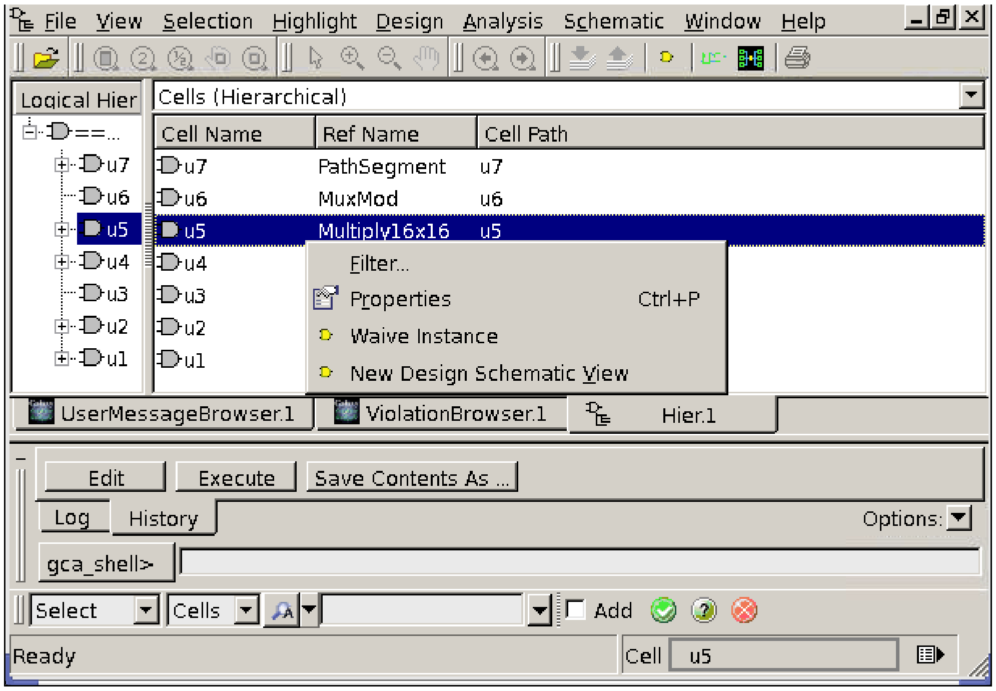
1. Open the Waiver Configuration dialog box.

   Choose Design > Waiver Configuration.

2. Click Instance Waivers. See Figure 2-14.
3. Click the Create Waiver button.

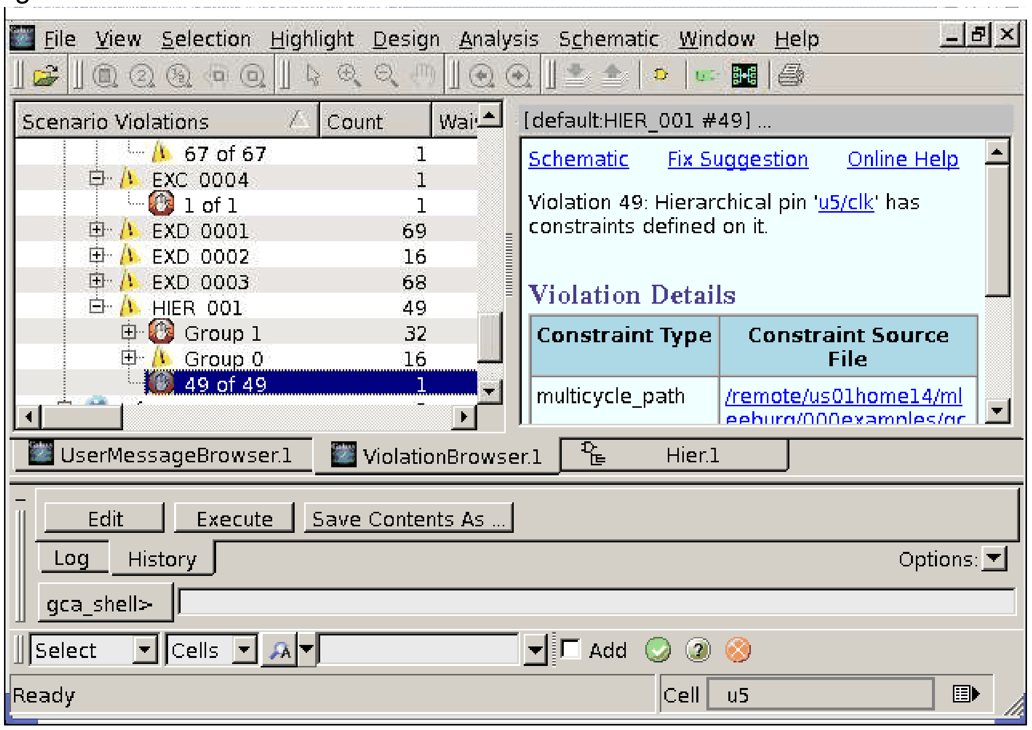
   The Edit Waiver Instance dialog box appears. See Figure 2-15.
4. Enter your instance waiver details.
5. Click the Create button.

Figure 2-12 Using the Hierarchy Browser to Create Instance Waivers



Cell Name	Ref Name	Cell Path
u7	PathSegment	u7
u6	MuxMod	u6
u5	Multiply16x16	u5
u4		
u3		
u2		
u1		

Figure 2-13 Waiver Icon in Violation Browser


File View Selection Highlight Design Analysis Schematic Window Help
Schematic Fix Suggestion Online Help
UserMessageBrowser.1 ViolationBrowser.1 Hier.1
Edit Execute Save Contents As ...
Log History Options:
gca_shell>
Select Cells Add
Ready Cell u5

Scenario Violations	Count	Wai
67 of 67	1	
EXC_0004	1	
1 of 1	1	
EXD_0001	69	
EXD_0002	16	
EXD_0003	68	
HIER_001	49	
Group 1	32	
Group 0	16	
49 of 49	1	

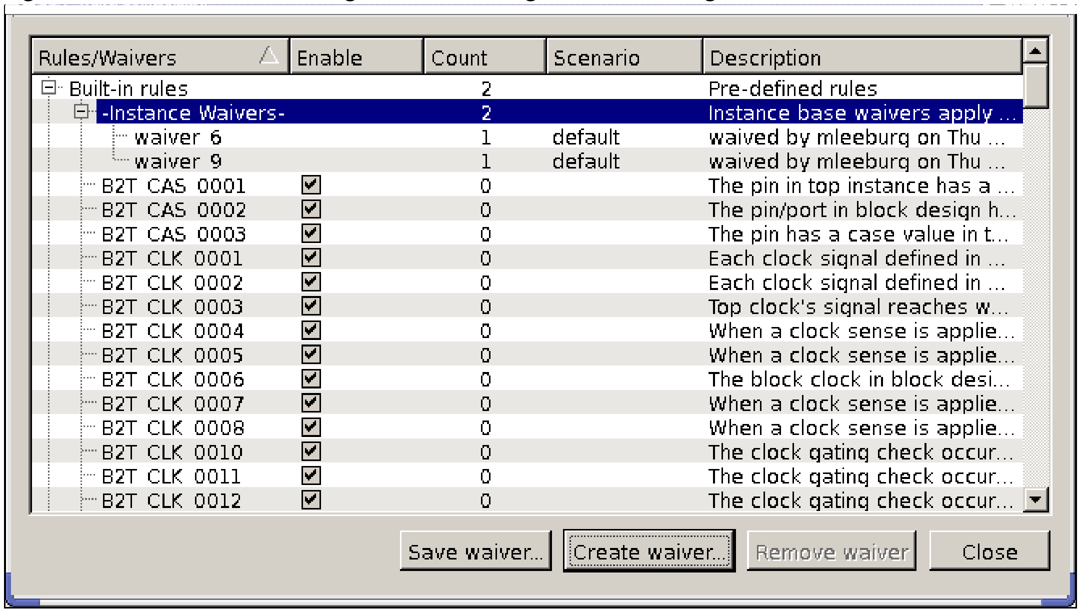
[default:HIER_001 #49] ...

Violation 49: Hierarchical pin 'u5/clk' has constraints defined on it.

Violation Details

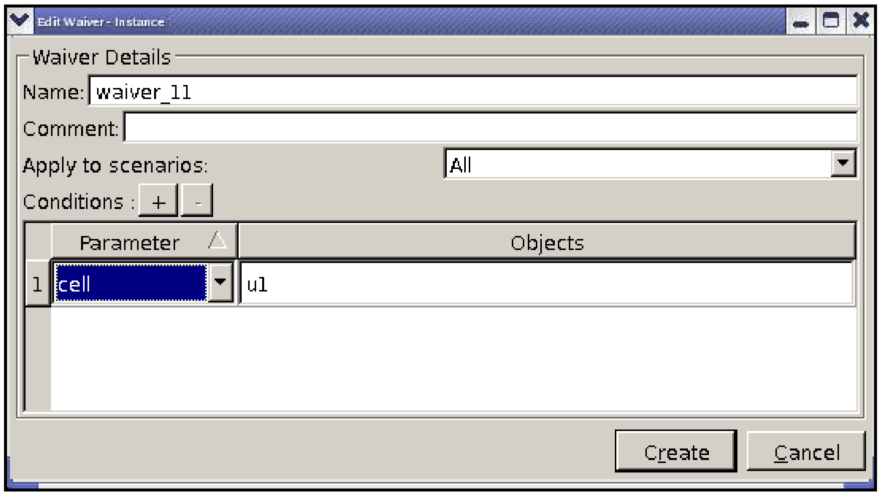
Constraint Type	Constraint Source File
multicycle_path	/remote/us01home14/ml eeburg/000examples/gc

Figure 2-14 Waiver Configuration Dialog Box Showing Instance Waivers



Rules/Waivers	Enable	Count	Scenario	Description
Built-in rules		2		Pre-defined rules
-Instance Waivers-		2		Instance base waivers apply ...
waiver 6		1	default	waived by mleeburg on Thu ...
waiver 9		1	default	waived by mleeburg on Thu ...
B2T_CAS_0001	✓	0		The pin in top instance has a ...
B2T_CAS_0002	✓	0		The pin/port in block design h...
B2T_CAS_0003	✓	0		The pin has a case value in t...
B2T_CLK_0001	✓	0		Each clock signal defined in ...
B2T_CLK_0002	✓	0		Each clock signal defined in ...
B2T_CLK_0003	✓	0		Top clock's signal reaches w...
B2T_CLK_0004	✓	0		When a clock sense is applie...
B2T_CLK_0005	✓	0		When a clock sense is applie...
B2T_CLK_0006	✓	0		The block clock in block desi...
B2T_CLK_0007	✓	0		When a clock sense is applie...
B2T_CLK_0008	✓	0		When a clock sense is applie...
B2T_CLK_0010	✓	0		The clock gating check occur...
B2T_CLK_0011	✓	0		The clock gating check occur...
B2T_CLK_0012	✓	0		The clock gating check occur...

Figure 2-15 Creating an Instance Waiver From the Edit Waiver Dialog Box



Name: waiver_11
Comment:
Apply to scenarios: All
Conditions : + -

	Parameter	Objects
1	cell	u1

To create an instance-level waiver from the command line, use the `-cells` option with the `create_waiver` command, as shown in Example 2-5.

In Example 2-5, my_waiver_1 is applied to cell U1/U1 and my_waiver_2 is applied to cells U1/U2 and U1/U3.

Example 2-5 Waiving Rules for Cells

```
gca_shell> create_waiver -name my_waiver_1  -cells [get_cell U1/U1]
gca_shell> create_waiver -name my_waiver_2  -cells [get_cell {U1/U2 U1/U3}]

gca_shell> report_waiver
****************************************
Report : report_waiver
Version: ...
Date   : ...
****************************************
Name           Rule  Scenario  Condition               Comment
-------------------------------------------------------------------------------------
my_waiver_1                    -cells {U1/U1}          waived by Jane on ...
my_waiver_2                    -cells {U1/U2 U1/U3}    waived by Jane on ...
```

Usage Guidelines

When you create an instance-level waiver, the tool applies it to the current scenario. To apply it across all scenarios of a design, use the `-all_scenarios` option.

Because an instance-level waiver is based on leaf cells or the netlist objects contained in an instance, a violation is waived only if the objects related to the violation are entirely contained within the specified cell or instance. Therefore, some general rules cannot be waived with this type of waiver. Custom rules cannot be waived with instance-level waivers.

Modifying Waivers

Use the following procedure to modify an existing waiver:

1. Open the Waiver Configuration dialog box (choose Design > Waiver Configuration).
2. Double-click the waiver you want to modify.

   This opens the Edit Waiver dialog box. The various fields are set to match the waiver you have selected.
3. Make any modifications.
4. Click the Create button.

   A message asks you to confirm that you want to replace the existing waiver.
5. Click OK. Your waiver is modified.

Reporting Waivers

To report violation waivers, use the `report_waiver` command. By default, the `report_waiver` command reports all the waivers for each design in the session.

Use the `-design` option to report waivers for specific designs. Example 2-6 shows the waivers reported for the TOP and HALF_ADDER designs.

Example 2-6 The report_waiver Output

```
gca_shell> report_waiver -design TOP HALF_ADDER
****************************************
Report : report_waiver
Version: ...
Date   : ...
****************************************
Name          Rule      Scenario    Condition                         Comment
------------------------------------------------------------------------------
Design: TOP

waiver_0_0    UNC_0003  scn1        -condition [list clock1  [get_clocks {TOP_VCLK}]]
                                                                  waived by Tom on ...
waiver_0_1    CAP_0001  scn3        -condition [list port [get_ports {PLUS_CARRY}]]
                                                                  waived by Dick on ...
Design: HALF_ADDER
waiver_1_0    DRV_0001  default     -condition [list port [get_ports {A}]]
                                                                  waived by John on ...
```

To report only the waivers for a specific rule, use the `-rules` option, for example

```
gca_shell> report_waiver -rules CAP_0001
```

Removing Waivers

To remove waivers, use the violation browser, the Waiver Configuration dialog box, or the `remove_waiver` command. Select the waiver and click “Remove waiver” as shown in Figure 2-16.

To remove a waiver using the violation browser,

1. Right-click the waiver you want to remove.
2. Choose Undo created waiver.

To remove a waiver using the Waiver Configuration dialog box,

1. Choose Design > Waiver Configuration.

   The Waiver Configuration dialog box appears, as shown in Figure 2-16.

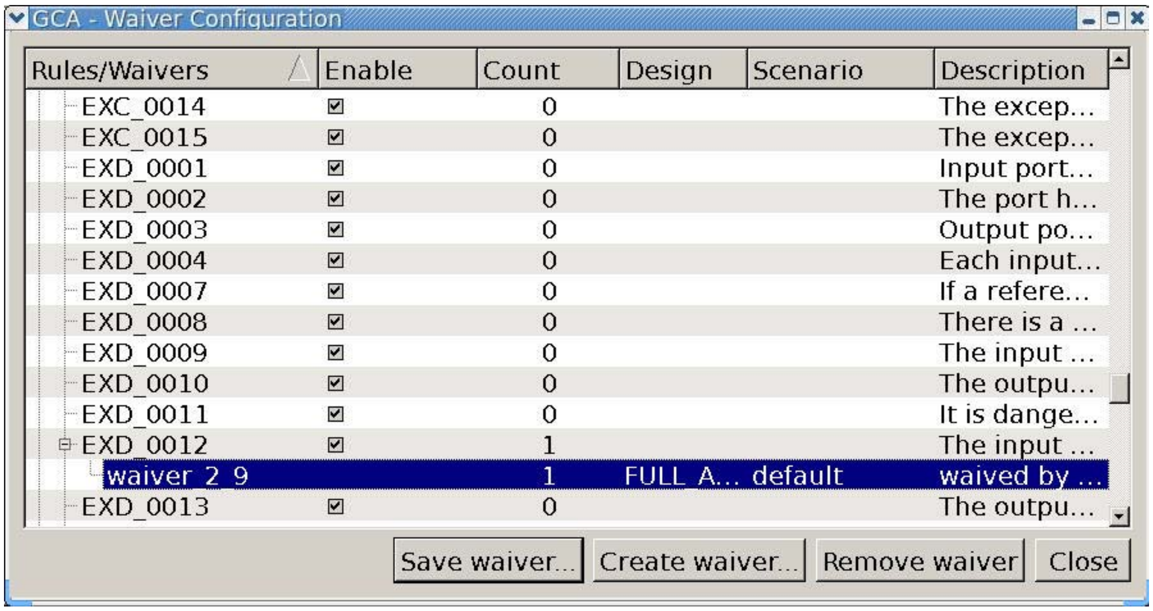
2. Select the waiver you want to remove.

   The rule is not checked or reported the next time you run the `analyze_design` command.

3. Choose Remove waiver.

See “Suppressing Violations” on page 2-11 and the `create_waiver`, `report_waiver, and remove_waiver` man pages for more information about using waivers.

Figure 2-16 Removing a Waiver With the Waiver Configuration Dialog Box



Rules/Waivers	Enable	Count	Design	Scenario	Description
EXC_0014	✓	0			The excep...
EXC_0015	✓	0			The excep...
EXD_0001	✓	0			Input port...
EXD_0002	✓	0			The port h...
EXD_0003	✓	0			Output po...
EXD_0004	✓	0			Each input...
EXD_0007	✓	0			If a refere...
EXD_0008	✓	0			There is a ...
EXD_0009	✓	0			The input ...
EXD_0010	✓	0			The outpu...
EXD_0011	✓	0			It is dange...
EXD_0012	✓	1			The input ...
waiver_2_9		1	FULL_A...	default	waived by ...
EXD_0013	✓	0			The outpu...

Writing Waivers to a File

You can use the `write_waiver` command to write waivers to a Tcl file, which can be sourced later to restore all the waivers. The syntax of the `write_waiver` command is

```
write_waiver -output output_file [-force]
```

Example 2-7 shows output of the `write_waiver` command:

Example 2-7 The write_waiver Output

```
if {[current_scenario] == "func"} {
     create_waiver -name "my_waiver1" -rule CLK_0026 -condition \
     [list "clock" [get_clocks {clk1 clk2}]] -comment "test"
     create_waiver -name "my_waiver2" -rule CLK_0003 -not_condition \
     [list "clock" [get_clocks clk3]] -comment "test"
}
if {[current_scenario] == "test"} {
     create_waiver -name "my_waiver3" -rule CLK_0003 -condition \
     [list "clock" [get_clocks {clk1 clk2}]] -not_condition \
     [list "pin" [get_pins {U1/A U1/B U1/Z}] [get_pins {U3/A U3/Z}]] \
     -comment "test"
}
```

When waivers are written to the waiver file, full paths of netlist objects are used, even if the original waiver specified an object name relative to the current instance.

Although the output of the `write_waiver` command is an accurate description of the active set of waivers, the waiver file might not be as readable as the original `create_waiver` commands. Also, the waiver output file can be large, particularly if you used wildcards in your original `create_waiver` commands.

Reporting Rule Violations With the report_constraint_analysis Command

To create a text report that summarizes violations in a design, use the `report_constraint_analysis` command. This command reports rule violations, user messages, and information about the defined rules. The results are sorted by analysis type, design, and scenario. Narrow the report by providing the following options:

- The `-include` option specifies the content to be included in the report. Use the `statistics` keyword with the `-include` option to report the constraints accepted for analysis.
- The `-style` option specifies whether a full or a summary report is generated.

You can tailor the output of the `report_constraint_analysis` command by using the `-rules` or `-rule_types` options. Using these options eliminates extraneous information from the report.

The report can be customized by using the `-format` option to generate the report in either a comma-separated values (CSV) file or a plain text file. The CSV format is useful for adding the report data to spreadsheets for data analysis.

Example 2-8 shows the `report_constraint_analysis` command results based on the script in Example 2-9.

Example 2-8 Analysis Report for Multiple Designs

```
gca_shell> report_constraint_analysis -include violations -style full

****************************************
Report : report_constraint_analysis
        -include {violations }
        -style {full}
Version: ...
Date   : ...
****************************************

Scenario Violations       Count Waived Description
------------------------------------------------------------------------------
Design: FULL_ADDER
<Global Violations>          1    0    Scenario independent violations
  error                      1    0
    UNT_0002                 1    0    Library 'library' has incomplete units defined.
      1 of 1                      0    Library '...' has incomplete units defined.
default                      13   0    Default scenario violations
  error                      5    0
    CAP_0001                 2    0    Output/inout port 'port' has zero or incomplete
                                       ...
      1 of 2                      0    Output/inout port 'SUM' has zero or incomplete
                                       ...
      2 of 2                      0    Output/inout port 'CARRY' has zero or
                                       incomplete ...
    EXD_0012                 3    0    The input delay at 'object_type' 'object' has
                                       zero ...
      1 of 3                      0    The input delay at 'port' 'A' has zero window
                                       for ...
      2 of 3                      0    The input delay at 'port' 'B' has zero window
                                       ...
      3 of 3                      0    The input delay at 'port' 'PREV_CARRY' has zero
                                       ...
  warning                    8    0
   ...


Design: HALF_ADDER
<Global Violations>          1    0    Scenario independent violations
  error                      1    0
    UNT_0002                 1    0    Library 'library' has incomplete units defined.
      1 of 1                      0    Library ...
default                      12   0    Default scenario violations
  error                      4    0
```

```
    CAP_0001                2    0    Output/inout port 'port' has zero ...
      1 of 2                     0    Output/inout port 'SUM' has zero ...
      2 of 2                     0    Output/inout port 'CARRY' has zero ...
        ...

Design: TOP
<Global Violations>         1    0    Scenario independent violations
  error                     1    0
    UNT_0002                1    0    Library 'library' has incomplete units defined.
      1 of 1                     0    Library...
scn1                        33   0    User-defined scenario violations
  error                     18   0
    CAP_0001                10   0    Output/inout port 'port' has zero or ...
      1 of 10                    0    Output/inout port 'PLUS1' has zero ...
      2 of 10                    0    Output/inout port 'PLUS2' has zero ...
    EXD_0012                8    0    The input delay at 'object_type' 'object' has
 zero window
for min and max values.
    CLK_0021                1    0    Clock 'clock' is not used in this scenario.
      1 of 1                     0    Clock 'TOP_VCLK2' is not used in this scenario.
...

scn2                        32   0    User-defined scenario violations
  error                     18   0
    CAP_0001                10   0    Output/inout port 'port' has zero ...
    EXD_0012                8    0    The input delay at 'object_type' 'object' has
                                      zero window for min and max values.

      1 of 8                     0    The input delay at 'port' 'A1' has zero window
                                      for min and max values.

      2 of 8                     0    The input delay at 'port' 'A2' has ...
...


Top/Block Scenario        Count Waived Description
-------------------------------------------------------------------------------
Designs: TOP/HALF_ADDER
scn1/default                2    0
  U1/U1/U2                  2    0    Block Instance
    error                   2    0
      B2T_CAS_0001          2    0    Pin 'top_pin' in top instance has case value
                                      that is missing in block design

        1 of 2                   0    Pin 'U1/U1/U2/U1/B' in top instance has case
                                      value that is missing in block design

        2 of 2                   0    Pin 'U1/U1/U2/U2/B' in top instance has case
                                      value that is missing in block design

scn2/new                    2    0
  U1/U1/U2                  2    0    Block Instance
    error                   2    0
      B2T_CAS_0001          2    0    Pin 'top_pin' in top instance has case value
                                      that is missing in block design

        1 of 2                   0    Pin 'U1/U1/U2/U1/B' in top instance has case
                                      value that is missing in block design
        2 of 2                    0    Pin 'U1/U1/U2/U2/B' in top instance has case
                                       value that is missing in block design

...
-------------------------------------------------------------------------------
Total Error Messages         89   0
Total Warning Messages       61   0
Total Info Messages          1    0
Report : report_constraint_analysis
```

Example 2-9 Script for Running Basic Constraint Analysis and Block-to-Top Constraint Analysis on Multiple Designs

```
link_design TOP
link_design -add HALF_ADDER
link_design -add FULL_ADDER

current_design TOP
create_scenario scn1
create_clock -period 10 -name TOP_VCLK
create_clock -period 10 -name TOP_VCLK2
set_input_delay 2 -clock TOP_VCLK [all_inputs]
set_output_delay 2 -clock TOP_VCLK [all_outputs]

create_scenario scn2
create_clock -period 20 -name TOP_VCLK
set_input_delay 3 -clock TOP_VCLK [all_inputs]
set_output_delay 3 -clock TOP_VCLK [all_outputs]
analyze_design -scenarios "scn1 scn2"

current_design HALF_ADDER

create_clock -period 10 -name HA_VCLK
set_input_delay 3 -clock HA_VCLK [all_inputs]
set_output_delay 1 -clock HA_VCLK [all_outputs]

create_scenario new
create_clock -period 20 -name HA_VCLK
set_input_delay 2 -clock HA_VCLK [all_inputs]
set_output_delay 2 -clock HA_VCLK [all_outputs]

analyze_design

report_constraint_analysis -include violations
```

Using Attributes

The following sections describe attributes and how to use them:

- Overview
- Setting, Listing, and Reporting Attributes
- Attribute Groups
- Using Arcs to Generate Custom Reports
- Creating a Collection of Library Timing Arcs

Overview

An attribute is a string or value associated with an object in the design that carries some information about that object. For example, the `number_of_pins` attribute attached to a cell indicates the number of pins in the cell. You can write programs in Tcl to get attribute information from the design database.

Setting, Listing, and Reporting Attributes

The PrimeTime GCA tool provides a set of commands for setting, listing, and reporting attributes as summarized in Table 2-2.

Table 2-2 Attribute Commands

Commands	Description
`list_attributes`	Shows the attributes defined for each object class or a specified object class; optionally shows application attributes.
`get_attribute`	Retrieves the value of any attribute from a single object.
`report_attribute`	Displays the value of all attributes on one or more objects; optionally shows application attributes.

The `get_attribute` and `report_attribute` commands accept an object specification, which can be a collection or a list of collections. The object specification for the `get_attribute` command is limited to one collection containing one object.

Attribute Groups

The PrimeTime GCA tool supports the attribute groups listed in Table 2-3. The properties attached to each attribute group are described in the PrimeTime GCA Online Help.

To see a list of all attributes available for a class of objects, use the `list_attributes -application` command. For example, to see the attributes associated with the net class of attributes, use the following command:

```
gca_shell> list_attributes -application -class net
```

Attributes are read-only unless otherwise specified.

Table 2-3 Attribute Groups

Type of attribute group	Class name
Cell Attributes	cell
Clock Attributes	clock
Clock Group Attributes	clock_group
Clock Group Group Attributes	clock_group_group
Design Attributes	design
Exception Attributes	exception
Exception Group Attributes	exception_group
Input Delay Attributes	input_delay
Library Attributes	lib
Library Cell Attributes	lib_cell
Library Pin Attributes	lib_pin
Library Timing Arc Attributes	lib_timing_arc
Net Attributes	net
Output Delay Attributes	output_delay
Pin Attributes	pin
Port Attributes	port

Table 2-3 Attribute Groups (Continued)

Type of attribute group	Class name
Rule Attributes	rule
Rule Violation Attributes	rule_violation
Ruleset Attributes	ruleset
Scenario Attributes	scenario
Timing Arc Attributes	timing_arc

Using Arcs to Generate Custom Reports

To create a collection of timing arcs for custom reporting and other processing, use the `get_timing_arcs` command. You can assign these timing arcs to a variable and get the needed attributes for further processing.

Use the `foreach_in_collection` command to iterate among the arcs in the collection. Use the `get_attribute` command to obtain information about the arcs. However, you cannot copy, sort, or index the collection of arcs. You can also use the `-filter` option of `get_timing_arcs` to obtain just the arcs that satisfy specified conditions, but you cannot use the `filter_collection` command to filter an already generated collection of arcs.

The `from_pin` is an attribute of a timing arc and is the pin or port where the timing arc begins. The `to_pin` attribute is the pin or port where the timing arc ends. To determine the value of an attribute, use the `get_attribute` command.

Use the script in Example 2-10 to show if any of the positive unate timing arcs of the U1 cell are disabled.

Example 2-10 Finding Disabled Timing Arcs of Specified Cells

```
gca_shell> set arcs [get_timing_arcs -of_objects U1 -filter \
             "sense == positive_unate"]
_sel3
gca_shell> foreach_in_collection arc $arcs { \
             echo [get_attribute $arc is_disabled] \
             }
true
false
```

Creating a Collection of Library Timing Arcs

To create a collection of library timing arcs for custom reporting and other processing, use the `get_lib_timing_arcs` command. You can assign these library arcs to a variable and get the needed attributes for further processing.

Use the `foreach_in_collection` command to iterate among the library arcs in the collection. Use the `get_attribute` command to obtain information about the arcs. However, you cannot copy, sort, or index a collection of library arcs. Use the `-filter` option of the `get_lib_timing_arcs` command to obtain only the library arcs that satisfy specified conditions. You cannot use the `filter_collection` command to filter a collection of library arcs. For a list of supported library timing arc object class attributes, see the PrimeTime GCA online Help.

The `from_lib_pin` is an attribute of a library timing arc, and is the library pin or port where the timing arc begins. The `to_lib_pin` attribute is the library pin or port where the timing arc ends. To determine the value of an attribute, use the `get_attribute` command.

Use the script in Example 2-11 to list the senses of timing arcs starting from the clock pin of a flip-flop library cell.

Example 2-11 Listing the Senses of Timing Arcs

```
gca_shell> set larcs [get_lib_timing_arcs -from class/FD1/CP]
_sel5
gca_shell> foreach_in_collection larc $larcs \
           { echo [get_attribute $larc sense] }
hold_clk_rise
setup_clk_rise
rising_edge
rising_edge
```

Customizing Rules and Reports

This section describes rule customizing in the following sections:

- Enabling and Disabling Rules
- Modifying Rules
- Using and Creating Rule Sets
- Using Tcl Scripts to Report Violation Data
- Opening the SDC Browser From Custom Rules

Enabling and Disabling Rules

The PrimeTime GCA tool supports an extensive set of built-in rules. Not all of the built-in rules are enabled by default. For a list of all built-in rules and their default status, use the `report_rule *` command. When a rule is enabled, the PrimeTime GCA tool checks the design attributes against the rule and reports cases where the design violated the rule.

To enable a rule, use the `enable_rule` command, as shown in Example 2-12.

Example 2-12 Enabling the EXD_0012 Rule

```
gca_shell> enable_rule [list EXD_0012]
```

To disable a rule, use the `disable_rule` command. Example 2-13 disables the DES_0001 and EXC_0002 rules.

Example 2-13 Disabling the DES_0001 and EXC_0002 Rules

```
gca_shell> disable_rule [list DES_0001 EXC_0002]
```

Some of the rules are disabled by default because they inherently conflict with the enabled rules. For example, pre-layout rules are different from post-layout rules and can cause conflicts. Other rules are disabled by default because they are runtime intensive.

Modifying Rules

The PrimeTime GCA tool enables you to customize any part of a rule. You can customize the message, the description, or the severity by using the `set_rule_message`, `set_rule_description`, and `set_rule_severity` commands.

For example, to change the severity of rule DES_0001 from warning level to error level, use the `set_rule_severity` command, as shown in Example 2-14.

Example 2-14 Changing the Rule Severity

```
gca_shell> set_rule_severity error [get_rules DES_0001]
```

To change the message of rule DES_0001, use the `set_rule_message` command, as shown in Example 2-15.

Example 2-15 Changing the Rule Message

```
gca_shell> set_rule_message [list "The pin of the register" \
            "didn’t receive any clock"] [get_rules DES_0001]
```

Using and Creating Rule Sets

The PrimeTime GCA rule sets are groups of rules designed for a specific function, such as signal integrity. Rule sets are designed to simplify constraint analysis and debugging by providing rules for specific areas in the design flow.

The PrimeTime GCA tool supports the following built-in rule sets:

- `compatibility`
- `postlayout`
- `prelayout`
- `primetime_si`
- `hierarchical`

To determine the rules in the rule sets, use the `report_rule` command, as shown in Example 2-16.

Example 2-16 Report the Rules in the primetime_si Rule Set

```
gca_shell> report_rules [get_rulesets primetime_si]
```

To analyze your design using only the built-in `primetime_si` rule set, use the `analyze_design` command with the `-rules primetime_si` option, as shown in Example 2-17.

Example 2-17 Analyze the Design Using Only the primetime_si Rule Set

```
gca_shell> analyze_design -rules primetime_si
```

In addition to built-in rule sets, the PrimeTime GCA tool supports user-created rule sets with the `create_ruleset` command. This feature helps you debug issues specific to your design. For example, if you only need to analyze your design for two rules, such as DES_0001 and DES_0003, you would create a rule set containing only DES_0001 and DES_0003, as shown in Example 2-18.

Example 2-18 Create a Rule Set Containing DES_0001 and DES_0003 Rules

```
## Create a ruleset targeting 2 rules only
gca_shell> create_ruleset -name my_ruleset [list DES_0001 DES_0003]
```

To analyze your design with the user-defined rule set, my_ruleset, use the `analyze_design` command with the `-rules my_ruleset` option, as shown in Example 2-19.

Example 2-19 Analyze the Design Using a Rule Set

```
gca_shell> analyze_design -rules my_ruleset ...
```

To create a rule set containing only EXC type rules, use the `create_ruleset` command, as shown in Example 2-20.

Example 2-20 Create a Rule Set for EXC Rules

```
gca_shell> create_ruleset -name myExcRuleset [get_rules EXC*]
```

To create a rule set containing all built-in rules without the EXD rules, use the `create_ruleset` command, as shown in Example 2-21.

Example 2-21 Create an Exclusion Rule Set

```
gca_shell> set allRules [get_rules *]
gca_shell> set EXDRules [get_rules EXD*]
gca_shell> set myRules [remove_from_collection $allRules $EXDRules]
gca_shell> create_ruleset -name myRuleset $myRules
```

Using Tcl Scripts to Report Violation Data

Although violation information is available in the GUI, you can use Tcl scripting to retrieve violation data. The `get_rule_violations` command allows you to obtain violation data in collection object form, so that you can use Tcl collection commands to manipulate the violation data.

Use the `get_rule_violations` command to create a collection of violation objects that meet a specified criteria. For example, if you wanted a collection of all clock rule violations in your design, use the following command:

```
gca_shell> get_rule_violations -of_objects [get_rules CLK_*]
```

You can also report the value of the attributes associated with the violation object. In the following example, the value of the `message` attribute for the first rule violation in a collection of rule violations is reported:

```
gca_shell> get_attribute [index_collection [get_rule_violations \
                          -of_objects CLK_0020] 0] message
Generated clock 'Gen_CLK' has edge relationships with its master clock
'MCLK' that cannot be satisfied. Only paths with 'negative' sense exist
from the master clock to source pin 'buf1/Z'. A 'positive' sense is
expected.
```

The `get_violation_info` command allows you to query the value of a rule parameter for a violation or the value of a violation details attribute (if applicable) for a rule violation.

To determine if a rule has `violation_details` attributes that can be queried with the `get_violation_info` command, query the `violation_details` attribute for that rule using the `get_attribute` command. For example,

```
gca_shell> get_attribute [get_rule B2T_CLK_0001] violation_details \
                          top_missing_at_pin
```

The `get_attribute` command returns `top_missing_at_pin`. This indicates that the B2T_CLK_0001 rule has the `top_missing_at_pin` attribute, which you can query with the `get_violation_info` command.

You can also query the `object_class` of the attributes associated with the violation details of a rule. Each violation details object has all of the attributes for the object class it belongs to. For example,

```
gca_shell> set top_clk_pins [get_violation_info \
           -attribute top_missing_at_pins [index_collection \
           [get_rule_violations -of_objects [get_rule B2T_CLK_0001]] 0]]
{"u7/ff1/CP", "u7/ff2/CP"}
gca_shell> foreach_in_collection pin $top_clk_pins { \
           query_object -verbose $pin
           }
{"pin:u7/ff1/CP"}
{"pin:u7/ff2/CP"}
gca_shell> get_attribute [get_pins u7/ff1/CP] clocks
{"clk2"}
gca_shell> get_attribute [get_pins u7/ff2/CP] clocks
{"clk2"}
```

The first rule violation for the B2T_CLK_0001 rule has two pins at the top level for which the clk2 clock is reaching the register at the top level.

Opening the SDC Browser From Custom Rules

You can add hyperlinks that open the SDC browser custom rules.

To create a hyperlink that opens the SDC browser and passes the SDC file name and the line number of the violating constraint, use HTML tags and the `src:filename#line` syntax when you create your violation. The following example creates a hyperlink called File:constraints.sdc, Line 12, within the custom rule Information Pane:

```
<a href="src:constraints.sdc#12"> File:constraints.sdc, Line 12</a>
```

When this hyperlink is selected, the SDC browser opens the constraints.sdc file and points to line 12.

Creating and Using User-Defined Rules

The PrimeTime GCA tool has many built-in rules which can detect a wide variety of issues. However, your design flow might have specific requirements that are not covered by the built-in rules. For this situation, the PrimeTime GCA tool allows you to create your own rules. These rules are checked automatically when you run the `analyze_design` command, and they are reported with the built-in rules in the violation browser.

This section includes the following sections:

- Creating a Tcl Rule-Checking Procedure
- Registering a User-Defined Rule
- Using User-Defined Rules
- Viewing Violations of User-Defined Rules
- User-Defined Rule Example

Creating a Tcl Rule-Checking Procedure

Using Tcl procedures, you can define a rule that tests for specific conditions on objects or attributes in the design. If an unwanted condition is detected, a violation of your rule is reported. The Synopsys Tcl interface provides full access to these objects and attributes.

To determine which attributes exist on any given object class, use the `list_attributes` command. For example, to list all the attributes that exist on the `rule` class, use the `list_attributes -class rule -application` command.

The Tcl procedure defining your custom rule contains

- Tcl code to inspect the objects and attributes in your design
- A `create_rule_violation` command that generates a violation entry in the violation browser

Your Tcl code tests your design for the presence of an unwanted condition. When the condition occurs, the Tcl code calls the `create_rule_violation` command, which reports a violation to the PrimeTime GCA tool.

A single Tcl procedure can create violations for multiple rules; you do not need a separate procedure for each rule. This allows you to create efficient Tcl code because collections of objects can be shared between rules.

If a large collection is used multiple times in a Tcl procedure, such as `[get_nets -hier *]`, it is more efficient to create this collection only one time and store the result in a variable.

The Tcl procedures in Example 2-22 check that no output delay exists on a hierarchical pin. If any hierarchical pin has an output delay, the PrimeTime GCA tool reports a violation of a rule named UDEF_0001.

Example 2-22   Creating the Rule-Checking Procedure

```
proc get_details_for_output_delays {output_delays} {
  set text "<b>Relative Clocks</b><ul>"
  foreach_in_collection out_del $output_delays {
    append text "<li>" [get_attribute $out_del clock_name]
  }
  append text "</ul>"
  return $text
}

proc UDEF_0001 {} {
  set hier_cells [get_cells * -filter "is_hierarchical==true"]
  set hier_pins [get_pins -of_objects $hier_cells]
  foreach_in_collection pin $hier_pins {
    set output_delays [get_output_delays -of_objects $pin -quiet]
    if {[sizeof_collection $output_delays] > 0} {
      set pin_name [get_attribute $pin full_name]
      create_rule_violation -rule UDEF_0001 -parameter_values $pin_name \
       -details [get_details_for_output_delays $output_delays]
    }
  }
}
```

The `create_rule_violation` command reports a violation for the rule specified with the `-rule` option. You can use the `-parameter_values` option to pass parameters to the message that is displayed when a rule violation occurs. Parameters are passed by position. The order of the parameters needs to match the order in which they are declared with the `create_rule` command.

Extra text can be passed to any violation by using the `-details` option. The text appears in the Violation Details section. The GUI can display HTML formatting; you can use HTML tags to format the text. Example 2-25 on page 2-40 shows an example of a user-defined rule.

Registering a User-Defined Rule

After you create a Tcl procedure to check for a design violation, you need to register the rule by using the `create_rule` command, as shown in Example 2-23.

Example 2-23 Using the create_rule to Create a New Rule Named UDEF_0001

```
gca_shell> create_rule -name UDEF_0001 \
            -severity warning \
            -description {"Team XYZ does not allow output delays to be \
specified on hierarchical pins"} \
            -message {"Output delay is specified on hierarchical pin '" \
"'."} \
            -parameters {"pin"} \
            -checker_proc UDEF_0001_checks
```

This command registers the new rule with the PrimeTime GCA tool, specifies parameters such as the severity of the violation, and identifies the name of the Tcl procedure that checks for the violation.

Each rule must have a unique name. You must start your rule name with “UDEF_” to avoid conflict with the built-in rule names. The severity can be adjusted with the `-severity` option; legal values are `info`, `warning`, `error`, and `fatal`.

A rule also needs to have a message. The message is displayed whenever a violation occurs. The message field is composed of fields of fixed text and optional parameters that are defined with the `-parameters` option. The parameters are passed to the message from the `create_rule_violation` command. The parameters are inserted between the message strings in the order they have been defined.

For example, to report the name of the clock and number of clock sources causing a violation, you could use the following parameter definition:

```
gca_shell> create_rule -name UDEF_0001 -severity warning \
  -message {"The clock "  " has "  "source(s) on inout ports."} \
  -parameters {"clock"  "count"} \
  -description "Clock defined on an inout port" \
  -checker_proc clk_inout_checker
```

Use the `-checker_proc` option to name the Tcl procedure that performs the check. The procedure that you specify with this option needs to exist before you run the `create_rule` command.

Use the following guidelines to define when to run user-defined rules:

- Global rules – This type of rule depends on the structure of the design or libraries. It is not related to the timing constraints. This type of rule is run only one time for the whole design. To define a global rule, use the `-global` switch.
- Scenario-related rules – This type of rule depends on the current set of constraints applied to the design. It is run one time for each scenario.

A Tcl checking procedure can define several custom rules. However, all the rules in a Tcl procedure must be the same type, either global or scenario. The first rule declared in the checker code sets the type, global or scenario. Subsequent rules defined in the same checker code must be the same type.

Using User-Defined Rules

After you create and register your rule, the PrimeTime GCA tool automatically checks the rule when you run the `analyze_design` command. Violations of your rule are displayed in the violation browser along with the built-in rule violations.

Messages written in the log file give you information about the status of your rules, as shown in Example 2-24.

Example 2-24 Messages Produced by the analyze_design Command for a User-Defined Rule

```
gca_shell> analyze_design
Information: Checking scenario 'scen1': Starting rule checks (ADES-002)
Information: Checking scenario 'scen1': Starting user-defined rule checks
(ADES-002)
Information: Running user-defined checker : UDEF_Rule_6. (ADES-021)
Information: Running user-defined checker : check_UDEF_CLK_0026.
(ADES-021)
Information: Running user-defined checker : check_UDEF_SID_SOD_virtual.
(ADES-021)
Information: Checking netlist: Starting scenario-independent rule checks.
(ADES-003)
```

Viewing Violations of User-Defined Rules

The violation browser shows violations of all rules. Violations of user-defined rules are displayed with the violations of the built-in rules. See "Violation Browser" on page 4-9 for detailed information about the violation browser.

User-Defined Rule Example

Example 2-25 shows Tcl code that creates the user-defined UDEF_rule_6 rule.

Example 2-25 Tcl Code for a User-Defined Rule

```
## Here is a global variable used to store the latency data
global latencyData

proc getHTMLExampleRule_6 {} {

    set HTML_page ""
    append HTML_page "<P>"
    append HTML_page "<tr><th align=\"left\">gca_shell> report_clocks \[get_clocks
CLK_virtual\] -skew<\/td><\/tr>"
    append HTML_page "<tr><th
align=\"left\">****************************************<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">Report : clock_skew<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">Design : design<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">Scenario: default<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">Version: ...<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">Date    : ...<\/td><\/tr>"
    append HTML_page "<tr><th
align=\"left\">****************************************<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\"><\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">               Min Condition Source
Latency      Max Condition Source Latency<\/td><\/tr>"
    append HTML_page "<tr><th
align=\"left\">----------------------------------------------------------------------
---------<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">Object         Early_r Early_f  Late_r
Late_f   Early_r Early_f  Late_r  Late_f    Rel_clk<\/td><\/tr>"
```

```
    append HTML_page "<tr><th
align=\"left\">-----------------------------------------------------------------------------
---------<\/td><\/tr>"
    append HTML_page "<tr><th align=\"left\">CLK_virtual     2.53    2.53    3.01
3.01        -       -    3.23    3.23         --<\/td><\/tr>"
   append HTML_page "<\/P>"
     return ${HTML_page}
}

proc createHTMLPage { clk } {
    global latencyData
    set HTML ""

    append HTML "<style>"
    append HTML " P { font-face : \"Courier New\" }"
    append HTML "<\/style>"

    append HTML "<table border=\"1\" width=\"500\" cellspacing=\"0\"
cellpadding=\"3\">"

    ## Create the Top 2 rows of the table containing labels
    ## First Row
    append HTML "<tr><td><\/td>"
    append HTML "<th colspan=\"2\" align=\"center\" bgcolor=\"lightblue\">Early<\/th>"
    append HTML "<th colspan=\"2\" align=\"center\" bgcolor=\"lightblue\">Late<\/th>"
    append HTML "<\/tr>"
    # Second row
    append HTML "<tr><td><th bgcolor=\"lightblue\" align=\"center\">Min<\/th>"
    append HTML "<th align=\"center\" bgcolor=\"lightblue\">Max<\/th>"
    append HTML "<th align=\"center\" bgcolor=\"lightblue\">Min<\/th>"
    append HTML "<th align=\"center\" bgcolor=\"lightblue\">Max<\/th>"
    append HTML "<\/tr>"

    ## Create the next two rows containing latency data
    # RISE row
    append HTML "<tr><th bgcolor=\"lightblue\" align=\"center\">Rise<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_early_rise_min)<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_early_rise_max)<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_late_rise_min)<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_late_rise_max)<\/th>"
    append HTML "<\/tr>"
    # FALL row
    append HTML "<tr><th bgcolor=\"lightblue\" align=\"center\">Fall<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_early_fall_min)<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_early_fall_max)<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_late_fall_min)<\/th>"
    append HTML "<th
align=\"center\">$latencyData(clock_source_latency_late_fall_max)<\/th>"
    append HTML "<\/tr>"

    append HTML "<\/table>"
    # Some spacing to ease reading...
    append HTML "<tr><td>"
    append HTML "<tr><td>"

    ## Display where the clock was created
    append HTML "<table border=\"1\" width=\"400\" cellspacing=\"0\"
cellpadding=\"3\">"

    ## File location is a String containing the filename and lines of operation on
that clock
    ## The first pair is where the clock has been created
    ## Other should notably match the eventual set_clock_latency commands
    ## Other commands can affect this clock and thus be added in this String as
well...
    set fileLocation [get_attribute -quiet [get_clocks ${clk}] file_line_info]
    set fileAttributes [split $fileLocation " "]

    append HTML "<tr><th align=\"left\" bgcolor=\"lightblue\">${clk} definition<\/th>"
    set name [string trim [lindex ${fileAttributes} 0] "{}"]
    set line [string trim [lindex ${fileAttributes} 1] "{}"]
    append HTML "<th align=\"left\">File ${name}, line ${line}<\/th>"
    append HTML "<\/tr>"
    append HTML "<tr><td>"

    ## variable i starts at 2 because we already printed the file/line for clock
creation
    for {set i 2} {$i < [expr [length ${fileAttributes}]]} {set i [expr ${i} +2]} {
       set name [string trim [lindex ${fileAttributes} ${i}] "{}"]
       set line [string trim [lindex ${fileAttributes} [expr ${i} + 1]] "{}"]
       append HTML "<tr><th align=\"left\" bgcolor=\"lightblue\"> Other related
exceptions<\/th>"
       append HTML "<th align=\"left\">File ${name}, line ${line}<\/th>"
       append HTML "<\/tr>"
    }
    append HTML "<\/table>"

    ## Suggest some help.
    # Some spacing to ease reading...
    append HTML "<tr><td>"
    append HTML "<tr><th align=\"left\"> <font size=\"4\"> <font
color=\"DarkSlateBlue\">Debug Suggestions<\/th><\/tr>"
    append HTML "<tr><th align=\"left\"> You can get more information about your
particular issue <\/th><\/tr>"
    append HTML "<tr><th align=\"left\">using the following command. Cut and paste it
in your console <\/th><\/tr>"
    append HTML "<tr><td>"
    append HTML "<tr><th align=\"left\">report_clocks \[get_clocks ${clk}\] -skew<\/
th><\/tr>"

    ## Add an example to complete the page
    append HTML "<tr><td>"
     append HTML "<tr><th align=\"left\"> <font size=\"4\"> <font
color=\"DarkSlateBlue\">Example<\/th><\/tr>"
    append HTML [getHTMLExampleRule_6]
    return ${HTML}
}
```

```
proc UDEF_Rule_6 {} {
    global latencyData
    ## I am going to reuse this variable to simplify the scripting below
    set sourceLatAttrList [list \
                clock_source_latency_early_fall_min \
                clock_source_latency_early_fall_max \
                clock_source_latency_early_rise_min \
                clock_source_latency_early_rise_max \
                clock_source_latency_late_fall_min \
                clock_source_latency_late_fall_max \
                clock_source_latency_late_rise_min \
                clock_source_latency_late_rise_max \
               ]

    ## Start from all the clocks and only keep the virtual ones.
    set zVirtualClocks [filter_collection [all_clocks] -regexp { undefined(sources) }]

    ## Iterate through all the virtual clocks
    ## We have a violation if the latency is incompletely defined
    ## Incompletely defined = at leat one attribute missing or
    ##                        one of the latency value is '0'
    foreach_in_collection itr ${zVirtualClocks} {
   set incompleteLatency false
   ## At the same time we check the attributes, we will also fill a hash
   ## table with the relevant values. This will save time when printing the
   ## values since we won't have to query again.
   foreach att ${sourceLatAttrList} {
       set zAttValue [get_attribute -quiet [get_clocks [get_object_name ${itr}]]
${att}]
       if {(${zAttValue} == "") || (${zAttValue} == 0)} {
      set incompleteLatency true
       }
       if {${zAttValue} == ""} {
      set latencyData(${att}) "--" } else {
          set latencyData(${att}) ${zAttValue} }
   }
   ## If we do have a violation, we need to create the proper display
   if {${incompleteLatency} == true} {
       create_rule_violation -rule UDEF_Rule_6 -parameter_values [get_object_name
${itr}] \
      -details [createHTMLPage [get_object_name ${itr}]]
   }

    }
}

create_rule -name UDEF_Rule_6 -severity warning \
    -message {"Clock latency is not fully defined for virtual clock '" "'."} \
    -parameters {"clock"} \
     -description {"The clock source latency is missing or incompletely defined for a
virtual clock" } \
    -checker_proc  UDEF_Rule_6
```

Rule-Related Commands

The rule-related commands include:

- `get_rule_property`

  This command returns the property of a single rule.
- `get_rules`

  This command creates a collection of rules.
- `get_rulesets`

  This command creates a collection of rule sets.
- `remove_ruleset`

  This command removes specified user-defined rule sets. Built-in rule sets cannot be removed.
- `report_rules`

  This command reports detailed rule information.
- `set_rule_property`

  This command enables you to set rule properties.

3

Additional Analysis Features

This chapter describes additional features that help you to diagnose constraint problems and to understand the impact of constraints. These features are described in the following sections:

Analyzing Timing Paths Using analyze_paths

You can debug most constraint problems using the `analyze_paths` command. However, some problems require detailed timing path analysis. When the `report_timing` command in the PrimeTime, Design Compiler, or IC Compiler tools does not show you certain timing paths, it is hard to find out what is causing a problem, and whether it is an expected condition or a constraint problem. In such cases, use the `analyze_paths` command to find out what is blocking propagation of particular timing paths and the cause of the blockage.

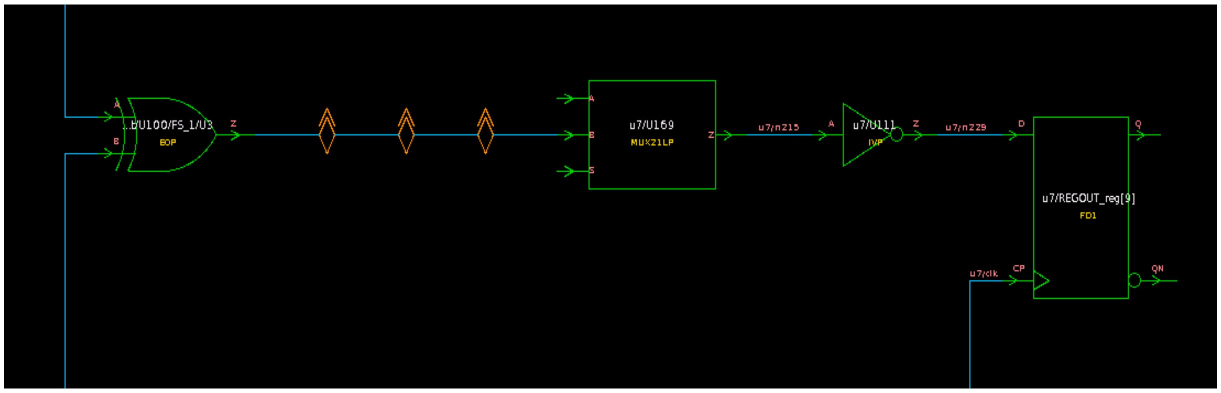
The `analyze_paths` command helps you to

- Identify timing exceptions and constraints on particular timing paths in a design
- Find out how many timing paths are covered, given a particular condition of rise/fall and from/through/to for an object set

The `analyze_paths` command identifies pins involved in specified paths and ranks the information based on the number of timing paths affected by various constraints.

If the GUI is open and you use the `-path_type full` argument, the `analyze_paths` command generates both a text report and a schematic showing the paths that satisfy the specifications you provide as arguments to the command. An example of a schematic is shown in Figure 3-1.

Figure 3-1 Paths Identified by the analyze_paths Command



You can use the `-max_endpoints` option to specify the maximum number of paths displayed in the schematic.

If the GUI is not open or you use the `-text_only` option, the `analyze_paths` command generates only a text report. Examples of text reports are shown in the remainder of this section.

The analyze_paths Default Output Report

You can use the `analyze_paths` command to see how many paths there are from, through, or to a particular set of objects and the corresponding clock interaction. Example 3-1 shows a default output report.

Example 3-1 Default Report Generated by the analyze_paths Command

```
gca_shell> analyze_paths -to Tranx/pay_count__reg[1]/D
Updating the exception annotations on the graph
****************************************
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
****************************************
Path Endpoints:
                                                   Path      Clocks
Endpoint                               Constraints  Count  Launch/Capture
--------------------------------------------------------------------------------
Tranx/pay_count__reg[1]/D                           10     (set 0)/(set 0)
Tranx/pay_count__reg[1]/D                           8      sysclk/(set 0)

Sets of Launching and Capturing Clocks:
  set 0 (2 clocks):
    falling sys_clk
    falling sysclk
1
```

The analyze_paths Detailed Output Report

You can use the `analyze_paths -path_type full` command to see how many paths there are from, through, or to a particular set of objects and the corresponding clock interaction with the breakdown of startpoints and endpoints. In addition, the report includes detailed logic levels and through points with possible startpoints and endpoints. See Example 3-2.

When timing exceptions are set on these paths, as in Example 3-3 on page 3-5, the report shows

- The name of the object that the exception is set on
- The name of the timing exception applied
- The script name and line location of the exception

Example 3-2 Report Generated by the analyze_paths -path_type full Command

```
gca_shell> analyze_paths -to Tranx/pay_count__reg[1]/D -path_type full
Breaking loops for scenario: system
****************************************
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
****************************************
Summary of Paths:
Path Clocks
Startpoint                  Endpoint                Constraints Count  Launch/Capture
-------------------------------------------------------------------------------------
Tranx/pay_count__reg[1]/CPN Tranx/pay_count__reg[1]/D      6       (2 clocks)/(2
clocks)
enable                      Tranx/pay_count__reg[1]/D      4           sysclk/(2
clocks)
Tranx/pay_count__reg[0]/CPN Tranx/pay_count__reg[1]/D      4       (2 clocks)/(2
clocks)
init                        Tranx/pay_count__reg[1]/D      4           sysclk/(2
clocks)

Full report of all pins in the paths

Level Pin                                                    Attr      Constraints
--------------------------------------------------------------------------------
1     init                                                   Start
1     Tranx/pay_count__reg[0]/CPN                            Start
1     enable                                                 Start
1     Tranx/pay_count__reg[1]/CPN                            Start
2     Tranx/U146/A
2     Tranx/pay_count__reg[1]/Q
2     Tranx/pay_count__reg[0]/Q
2     Tranx/U925/A
3     Tranx/U925/Z
3     Tranx/U146/Z
3     Tranx/add_67/U1_1_1/A
3     Tranx/add_67/U1_1_1/B
4     Tranx/add_67/U1_1_1/S
2     Tranx/U1108/A
4     Tranx/U1108/B
4     Tranx/U384/A
4     Tranx/U384/B
5     Tranx/U1108/Z
5     Tranx/U384/Z
6     Tranx/U1107/A
6     Tranx/U383/A
7     Tranx/U1107/Z
7     Tranx/U383/Z
3     Tranx/U1132/A
8     Tranx/U1132/B
5     Tranx/U1132/C
8     Tranx/U1132/D
9     Tranx/U1132/Z
```

```
10    Tranx/U920/A
11    Tranx/U920/Z
12    Tranx/pay_count__reg[1]/D                            End
1
```

Example 3-3 Report Generated by the analyze_paths -path_type full Command

```
gca_shell> analyze_paths -path_type full -to Tranx/CRC_tranx/pay_out_reg/D
****************************************
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
****************************************
Summary of Paths:
Path Clocks
Startpoint                           Endpoint          Constraints Count  Launch/
Capture
---------------------------------------------------------------------------------------
Tranx/CRC_tranx/crc_out_reg/CPN Tranx/CRC_tranx/pay_out_reg/D  F1 2  (2 clocks)/(2
clocks)
Tranx/CRC_tranx/pay_in2_reg/CPN Tranx/CRC_tranx/pay_out_reg/D  M0 2  (2 clocks)/(2
clocks)
enable                          Tranx/CRC_tranx/pay_out_reg/D     2     sysclk/(2
clocks)


Exception Information:
  M0 multicycle_path          (run.ztcl, line 27)
  F1 false_path          (run.ztcl, line 28)

Full report of all pins in the paths

Level Pin                                                  Attr      Constraints
---------------------------------------------------------------------------------
1     enable                                               Start
1     Tranx/CRC_tranx/pay_in2_reg/CPN                      Start
1     Tranx/CRC_tranx/crc_out_reg/CPN                      Start
2     Tranx/CRC_tranx/crc_out_reg/Q                                  F1
2     Tranx/CRC_tranx/pay_in2_reg/Q                                  M0
3     Tranx/CRC_tranx/U20/A
3     Tranx/CRC_tranx/U20/C
2     Tranx/CRC_tranx/U20/D
4     Tranx/CRC_tranx/U20/Z
5     Tranx/CRC_tranx/U14/A
6     Tranx/CRC_tranx/U14/Z
7     Tranx/CRC_tranx/pay_out_reg/D                        End       M0,F1
1
```

Summary Output Report for the analyze_paths Command

Example 3-4 shows the `analyze_paths -path_type summary` report.

Example 3-4 Summary Output Report for analyze_paths Command

```
gca_shell> analyze_paths -to Tranx/CRC_tranx/pay_out_reg/D -path_type summary
****************************************
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
****************************************
Summary of Paths:
Path Clocks
Startpoint                           Endpoint                 Constraints Count
Launch/Capture
-------------------------------------------------------------------------------------
Tranx/CRC_tranx/crc_out_reg/CPN Tranx/CRC_tranx/pay_out_reg/D F1 2   (2 clocks)/
                                                                      (2 clocks)
Tranx/CRC_tranx/pay_in2_reg/CPN Tranx/CRC_tranx/pay_out_reg/D M0 2   (2 clocks)/
                                                                      (2 clocks)
enable                          Tranx/CRC_tranx/pay_out_reg/D    2    sysclk/
                                                                       (2 clocks)


Exception Information:
  M0 multicycle_path          (run.ztcl, line 27)
  F1 false_path          (run.ztcl, line 28)
1
```

Disabled Paths Output Report for the analyze_paths Command

When using the `analyze_paths` command with the `-traverse_disabled` option (see Example 3-5), the output report analysis traces through disabled paths of the given path set and shows

- Possible paths without data propagation blockage
- Reasons for datapaths being blocked or segmented

Example 3-5 Report for analyze_paths Command With the -traverse_disabled Option

```
gca_shell> analyze_paths -to Tranx/CRC_tranx/pay_out_reg/D -path_type full \
-traverse_disabled
****************************************
Report : analyze_paths
Design : b_top
Version: . . .
Date   : . . .
****************************************
Summary of Paths:
Path Clocks
Startpoint              Endpoint                    Constraints Count  Launch/Capture
-----------------------------------------------------------------------------------
enable                            Tranx/CRC_tranx/pay_out_reg/D       2
sysclk/(2 clocks)
enable                            Tranx/CRC_tranx/pay_out_reg/D   D0 2
sysclk/(2 clocks)
Tranx/CRC_tranx/crc_out_reg/CPN Tranx/CRC_tranx/pay_out_reg/D   F1 2 (2 clocks)/
                                                                       (2 clocks)
Tranx/CRC_tranx/pay_in2_reg/CPN Tranx/CRC_tranx/pay_out_reg/D   M0 2 (2 clocks)/
                                                                       (2 clocks)

Exception Information:
  M0 multicycle_path          (run.ztcl, line 27)
  F1 false_path          (run.ztcl, line 28)

Disabled Object Information:
  D0:
    set_disable_timing negative_unate Arc
      From: Tranx/CRC_tranx/U56/A
      To: Tranx/CRC_tranx/U56/Z

Full report of all pins in the paths

Level Pin                                                 Attr      Constraints
------------------------------------------------------------------------------
1     Tranx/CRC_tranx/pay_in2_reg/CPN                     Start
1     Tranx/CRC_tranx/crc_out_reg/CPN                     Start
1     enable                                              Start
2     Tranx/CRC_tranx/crc_out_reg/Q                                 F1
2     Tranx/CRC_tranx/pay_in2_reg/Q                                 M0
2     Tranx/CRC_tranx/U56/A
3     Tranx/CRC_tranx/U56/Z
3     Tranx/CRC_tranx/U20/A
4     Tranx/CRC_tranx/U20/B
3     Tranx/CRC_tranx/U20/C
2     Tranx/CRC_tranx/U20/D
5     Tranx/CRC_tranx/U20/Z
6     Tranx/CRC_tranx/U14/A
7     Tranx/CRC_tranx/U14/Z
8     Tranx/CRC_tranx/pay_out_reg/D                       End       M0,F1
1
```

Analyzing Unclocked Pins Using analyze_unclocked_pins

Use the `analyze_unclocked_pins` command to analyze a collection of pins for missing clock definitions or blocked clock propagation. By default, this command analyzes all pins in the design.

The `analyze_unclocked_pins` command by default reports

- Summary of analysis
- Possible points for missing clock definition
- Locations where clock propagation is blocked

Use the `-verbose` option to print out clock pins that are unclocked, case-disabled, and register-disabled. This option reports detailed lists of unclocked pins. From the output report, you can see register clock pins that do not have clock signals, and clock pins that have sequential checks disabled for various reasons. These are helpful for further analysis.

In addition to using the `analyze_unclocked_pins` command to help debug clock constraint problems, you can turn clock and case annotation on and off in the Schematic View using the Pin annotation drop-down menu. See "Analyzing a Violation" on page 2-7 for more information.

The analyze_unclocked_pins Verbose Output Report

Example 3-6 shows the output for the `analyze_unclocked_pins -verbose` command.

The report summarizes register clock pin status as being clocked, unclocked, case-disabled, or register-disabled. Case-disabled is caused by `case_analysis` propagation disabling timing check/arcs from these pins. Register-disabled is caused by various `set_disable_timing` constraints applied to all arcs from these pins.

In addition to the summary, the verbose report identifies possible startpoints for debugging missing clock definitions. It also identifies clocks that are blocked from register clock pins at specified locations. Check to see if the clock propagation blockage is intentional.

Example 3-6 Output Report for the analyze_unclocked_pins -verbose Command

```
gca_shell> analyze_unclocked_pins -verbose
****************************************
Report : analyze_unclocked_pins
        -verbose
Design : ChipLevel
Version: ...
Date   : ...
****************************************
Analyze Register Clock Pins that do not have clocks assigned
```

```
                            Summary
-----------------------------------------------------------------
Register Clock Pin Status                        Number of Pins
-----------------------------------------------------------------
Clocked                                                148
Unclocked                                               67
Case disabled clock pin                                  0
Register behavior disabled                               0
-----------------------------------------------------------------
Total register clock pins                              215

        Possible Startpoints for Missing Clock Definitions
                                       Number
Pin/Port Startpoints                   unclocked      Reason a Startpoint
--------------------------------------------------------------------------------
clk_adder_div2/Q                       50             constraint forced startpoint on
                                                       interior pin (default)
clk_8x8/Z                              16             constraint forced startpoint on
                                                       interior pin (default)
clk_adder/Z                            1              set_disable_timing (default)

    clk_adder/Z
    - fans out to 1 unclocked pins
       In fanin of clk_adder_div2/CP
    clk_adder_div2/Q
    - fans out to 50 unclocked pins
       In fanin of u1/ain_tmp_reg[15]/CP
     ...

Clocks are blocked from register clock pins at the following locations:
                                            Number
Clock      Blocking Pin                     unclocked      Reason
--------------------------------------------------------------------------------
mclk       clk_8x8/Z                        16             constraint forced
                                                            startpoint on interior pin
aclk       clk_adder/Z                      1              set_disable_timing


Clock mclk Blocking Locations (verbose)
Blocking Pin                 Reason
--------------------------------------------------------------------------------
clk_8x8/Z                    constraint forced startpoint on interior pin


Verbose Output for Blocked Clocks:

Blocking Pin                 Clock Pin
--------------------------------------------------------------------------------
clk_8x8/Z                    u4/res_reg[15]/CP
clk_8x8/Z                    u4/res_reg[12]/CP
...

Clock aclk Blocking Locations (verbose)
Blocking Pin                 Reason
--------------------------------------------------------------------------------
clk_adder/Z                  set_disable_timing
```

```
Verbose Output for Blocked Clocks:

Blocking Pin                 Clock Pin
--------------------------------------------------------------------------------
clk_adder/Z                  clk_adder_div2/CP
1
gca_shell>
.....
```

Analyzing Clock Networks Using analyze_clock_networks

In modern designs, clock networks grow more complex due to features such as generated clocks, clock-gating logic, and clock multiplexers. These structures can be difficult to analyze. In addition, the interaction of the clock network traversal with constraints that have an impact on clock networks can be very hard to understand.

To debug problems in complex clock networks, the PrimeTime GCA tool uses the `analyze_clock_networks` command to generate reports that describe the propagation of the clock network to and from specified points. A report can show the impact of constraints on

- The clock network
- The potential clock network (potential senses are detected with the `-traverse_disabled` option)
- A generated clock's source latency network

In addition to using the `analyze_clock_networks` command to help debug clock constraint problems, you can turn clock and case annotation on and off in the Schematic View using the Pin annotation drop-down menu. See "Analyzing a Violation" on page 2-7 for more information.

Output Reports for the analyze_clock_networks Command

Example 3-7 and Example 3-8 show the `analyze_clock_networks` command usage and output reports.

Example 3-7 Output Report With Potential Network for the analyze_clock_networks Command

```
gca_shell> analyze_clock_networks -from [get_clocks sysclk] -to Tranx/U0/ \
            o_lfsr1_reg[11]/CP -traverse_disabled -style full_expanded
****************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full_expanded
        -end_types  {register port clock_source }
        -traverse_disabled
```

```
Design : b_top
. . .
****************************************
Key for clock sense abbreviations:
   P = positive
   N = negative
   Potential senses detected with -traverse_disabled:
   [P] = potential positive

Full report for clock: sysclk

Branch 0:
       Branch
Level  Info     Sense Notes     Port/Pin
--------------------------------------------------------------------------------
    0           P     source    clk
    1           P               Tranx/U1775/A
    2           P               Tranx/U1775/Z
    3           P               Tranx/U1772/A
    4           N               Tranx/U1772/Z
    5           N               Tranx/U1771/A
    6           N               Tranx/U1771/Z
    7           N               Tranx/U1417/A
    8           P               Tranx/U1417/Z
    9           P               Tranx/U0/U1154/A
   10           P               Tranx/U0/U1154/Z
   11           P               Tranx/U0/U1139/A
   12           P               Tranx/U0/U1139/Z
   13           P               Tranx/U0/U1121/A
   14           [P]   DT#0      Tranx/U0/U1121/Z
   15           [P]   R         Tranx/U0/o_lfsr1_reg[11]/CP

Notes Column Information:

Clock Network End Type Abbreviations:
   R = register

Clock Blocking Constraints:
    DT#0 at pin: Tranx/U0/U1121/Z
      set_disable_timing
         run.ztcl, line 25
1
```

Example 3-8 Output Report With Generated Clock Path for the analyze_clock_networks Command

```
gca_shell> analyze_clock_networks -to gclk -traverse_disabled -style full_expanded

****************************************
Report : analyze_clock_networks
        -max_endpoints=1000
        -style full_expanded
        -end_types  {register port clock_source }
        -traverse_disabled
```

```
Design : b_top
Version: . . .
Date   : . . .
****************************************
Key for clock sense abbreviations:
   P = positive
   R = rise_triggered
   N,R = negative, rise_triggered

Source latency paths for generated clock: gclk
   from master clock: sys_clk - 2 partial path branches

Branch 0:
       Branch
Level  Info     Sense Notes     Port/Pin
--------------------------------------------------------------------------------
    0           P     source    clk
    1           P               Recx/U207/A
    2  S1       P               Recx/U207/Z
    3           P               Recx/CRC2/U69/A
    4           P               Recx/CRC2/U69/Z
    5           P     CG        Recx/CRC2/clk_and/A
    6  E1       N,R   CS        Recx/CRC2/clk_and/Z

Branch 1: from branch 0 reconverges to branch 0
       Branch
Level  Info     Sense Notes     Port/Pin
--------------------------------------------------------------------------------
    3           P     R         Recx/CRC2/clk_div/CP
    4           P               Recx/CRC2/clk_div/CP
    5           R               Recx/CRC2/clk_div/QN
    6           R     D         Recx/CRC2/clk_and/B

Notes Column Information:

Clock Network End Type Abbreviations:
   R = register
   CS = clock_source
   D = data
   CG = clock_gating
1
```

Analyzing Case and Constant Propagation Using report_case_details

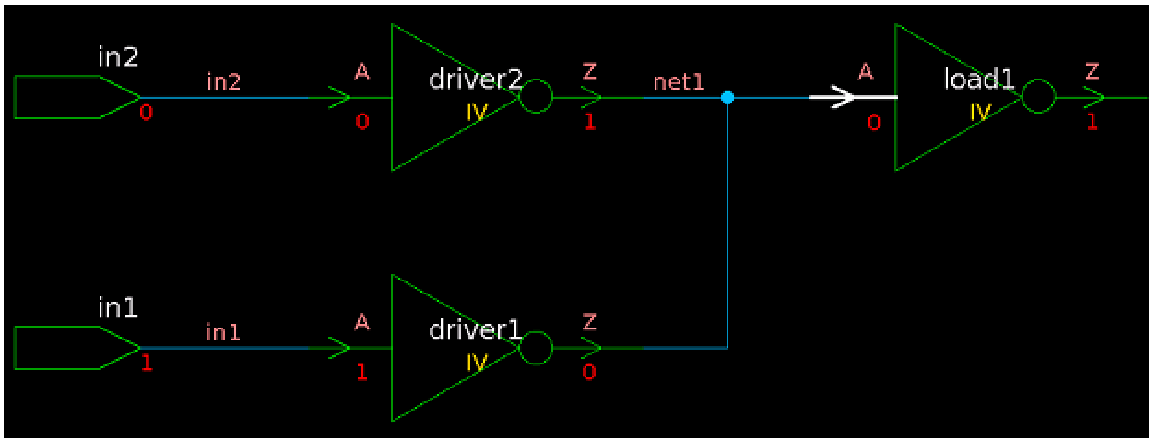
With complex design structures, analyzing case value and logic constant propagation can become very difficult. This analysis can become a very complicated task because case and constant propagation are controlled in a number of different ways, such as user-set values, constraints on the design, interaction between different case and constant values propagated to a gate, and by how the logic gate is modeled functionally.

To see how your case values and logic constant values propagate throughout the netlist, use the `report_case_details` command. This command reports the propagation of case values and logic constant values throughout your design. It can show how case values and logic constants propagate to or from specified pins and ports.

In addition to using the `report_case_details` command to help debug case propagation problems, you can turn clock and case annotation on and off in the Schematic View using the Pin annotation drop-down menu. See “Analyzing a Violation” on page 2-7 for more information.

If the GUI is open, the `report_case_details` command generates both a text report and a schematic showing a cone of logic with propagated values annotated on the cells, as shown in Figure 3-2.

Figure 3-2 Schematic Showing Case Propagation



The `-sources` option lists the ports and pins where case values or logic constants have been set and propagated to the list of ports and pins being examined. If you run the command with the `-sources` option and without a `-from` or `-to` option, a schematic is not generated because the cone of logic related to the propagated value is not reported. You can specify the maximum number of gates to be displayed in the schematic with the `-max_objects` option; if the logic cone is larger than this value, no schematic is generated.

If the GUI is not open or you use the `-text_only` option, the `report_case_details` command generates only a text report. An example of a text report is shown in Example 3-9.

Example 3-9 Text Report Generated by the report_case_details Command

```
gca_shell> report_case_details -to Tranx/my_fec_encoder/curr_state_reg[1]/D
****************************************
Report : case_details
Design : b_top
Version: . . .
Date   : . . .
****************************************
Properties         Value    Pin/Port
-------------------------------------------------------------------------------
from user case     0        Tranx/my_fec_encoder/curr_state_reg[1]/D
```

```
Case fanin report:
Verbose Source Trace for pin/port Tranx/my_fec_encoder/curr_state_reg[1]/D:
Path number: 1
Path Ref #   Value  Properties           Pin/Port
-------------------------------------------------------------------------------
             0                           Tranx/my_fec_encoder/curr_state_reg[1]/D
             0      F()=A ! B ! & C ! D ! & +
                                         Tranx/my_fec_encoder/U37/Z
2            1                           Tranx/my_fec_encoder/U37/B
3            1                           Tranx/my_fec_encoder/U37/D

Path number: 2
Path Ref #   Value  Properties           Pin/Port
-------------------------------------------------------------------------------
             1                           Tranx/my_fec_encoder/U37/B
             1      F()=A ! B ! C ! D ! + + +
                                         Tranx/my_fec_encoder/U36/Z
             0                           Tranx/my_fec_encoder/U36/D
             0      F()=A B C & &        Tranx/my_fec_encoder/U50/Z
             0                           Tranx/my_fec_encoder/U50/C
             0      F()=A !              Tranx/U1146/Z
             1                           Tranx/U1146/A
             1      F()=A                Tranx/U1122/Z
             1                           Tranx/U1122/A
             1      F()=A ! B ! C ! + +  Tranx/U1215/Z
             0                           Tranx/U1215/B
             0      F()=A !              Tranx/U1213/Z
             1                           Tranx/U1213/A
             1      F()=A ! B ! +        Tranx/U1212/Z
             0                           Tranx/U1212/B
             0      F()=A ! B ! &        Tranx/U1211/Z
             1                           Tranx/U1211/A
             1      F()=A                Tranx/U151/Z
             1                           Tranx/U151/A
             1      F()=A                Tranx/U1103/Z
             1                           Tranx/U1103/A
             1      user case            rst

Path number: 3
Path Ref #   Value  Properties           Pin/Port
-------------------------------------------------------------------------------
             1                           Tranx/my_fec_encoder/U37/D
             1      F()=A ! B ! +        Tranx/my_fec_encoder/U49/Z
             0      user case            Tranx/my_fec_encoder/U49/B
```

Analyzing Clock Interactions Using report_clock_crossing

The `report_clock_crossing` command reports the various interactions between clock domains in a text report. By default, it lists all clock combinations in which there is a topological path launched by one clock and captured by another one. Clock combinations that have been declared partially or completely false are noted. The report can be narrowed by providing a collection of `-from` and `-to` clocks. It can also be narrowed to only reporting the completely false paths or interactions with some valid paths.

Analyzing Constraint Coverage Using report_analysis_coverage

Many designs require complex and numerous constraints. The need for multimode and multicorner analysis increases the importance of constraint quality and completeness. The task of manually quantifying the completeness of a set of constraints is difficult and time consuming. To help automate this task, the PrimeTime GCA tool provides the `report_analysis_coverage` command, which enables you to gauge the completeness of a given set of constraints.

This section describes the PrimeTime GCA analysis coverage capabilities in the following subsections:

- Analysis Coverage Overview
- Using report_analysis_coverage in Single-Scenario Environments
- Narrowing Down Reports
- Using report_analysis_coverage in Multi-Scenario Environments

Analysis Coverage Overview

Use the `report_analysis_coverage` command to analyze constraint coverage. When you run the `report_analysis_coverage` command, the PrimeTime GCA tool looks at the endpoints and timing arcs in the design across all the design scenarios and reports a summary of all tested and untested points for the current constraints. After you review the summary, you can use the `report_analysis_coverage` command options to debug specific problems. These options enable you to obtain detailed information about the status of an endpoint. Detailed report information includes tested and untested endpoints, the reason why an endpoint is untested, what mode the endpoint is tested in, and so forth. These debugging features help speed-up constraint development.

The PrimeTime GCA tool supports both single-scenario and multi-scenario environments. A scenario is used to represent the constraints for an operating mode along with a particular process, voltage, and temperature (PVT) corner. Different PVT corners only affect delays and transition times in the design; they have no effect on whether the endpoints are constrained or not. It is important to ensure that the constraints of different scenarios cover the design for all expected modes of operation.

The valid check types for analysis coverage are:

- clock_gating_hold and clock_gating_setup
- hold and setup
- clock_separation

- max_skew
- min_pulse_width
- min_period
- recovery
- removal

The PrimeTime GCA tool generates a text report with the percentage and number of checks tested and untested.

Using report_analysis_coverage in Single-Scenario Environments

If there is only one scenario, the PrimeTime GCA tool automatically generates a report focusing on single-scenario analysis. This report identifies whether an endpoint or timing arc has been tested. If you define several scenarios in the current session, but you only want to generate a report for one of them, use the `-scenario` option. Example 3-10 shows a standard single-scenario report.

Example 3-10 Standard Single-Scenario Report

```
report_analysis_coverage
****************************************
Report : analysis_coverage
Design : zDesign
. . .
****************************************
Breaking loops for scenario: default
Updating Clock Gating Locations
  Inferring 294 clock-gating checks.

Type of Check              Total          Tested             Untested
--------------------------------------------------------------------
clock_gating_hold            294       294 (100%)             0 (  0%)
clock_gating_setup           294       294 (100%)             0 (  0%)
hold                       75481     45553 ( 60%)         29928 ( 40%)
min_period                   376       196 ( 52%)           180 ( 48%)
min_pulse_width            55778     39171 ( 70%)         16607 ( 30%)
out_hold                     359       333 ( 93%)            26 (  7%)
out_setup                    359       333 ( 93%)            26 (  7%)
recovery                   16801     16385 ( 98%)           416 (  2%)
removal                    16801     16385 ( 98%)           416 (  2%)
setup                      75479     45532 ( 60%)         29947 ( 40%)
--------------------------------------------------------------------
All Checks                242022    164476 ( 68%)         77546 ( 32%)
```

Example 3-10 shows a PrimeTime GCA summary report. After you identify some potential problems in the constraint coverage, the PrimeTime GCA tool can provide more detailed information for specific checks, as shown in the next section.

Narrowing Down Reports

In Example 3-10, some of the outputs have untested setup and hold checks. To find out more information about why these outputs are untested, run the following command:

```
gca_shell> report_analysis_coverage -status_details untested \
              -check_type [list out_setup out_hold]
```

Example 3-11 shows the PrimeTime GCA output, which provides a detailed list of all untested output setup and output hold checks along with a possible reason for this state.

Example 3-11 Detailed Output for the report_analysis_coverage Command

```
****************************************
Report : analysis_coverage
        -status_details {untested}
        -check_type {out_hold out_setup}
Design : zDesign
. . .
****************************************

Type of Check              Total           Tested             Untested
--------------------------------------------------------------------------------
out_hold                     359       333 ( 93%)            26 (   7%)
out_setup                    359       333 ( 93%)            26 (   7%)
--------------------------------------------------------------------------------
All Checks                   718       666 ( 93%)            52 (   7%)

Constrained Pin             Related Pin     Check Type          Untested Reason
--------------------------------------------------------------------------------
zOut1                                       out_hold            no_startpoint_clock
zOut2                                       out_setup           no_startpoint_clock
zOut3                                       out_hold            no_startpoint_clock
zOut4                                       out_setup           no_startpoint_clock
zOut5                                       out_hold            no_startpoint_clock
zOut6                                       out_setup           no_startpoint_clock
zOut7                                       out_hold            no_paths
zOut8                                       out_setup           no_paths
zOut9                                       out_hold            constant_disabled
zOut10                                      out_setup           constant_disabled
zOut11                                      out_hold            constant_disabled
. . .
```

After the PrimeTime GCA tool points out a potential untested reason, you can use the PrimeTime GCA debugging commands to find out more specific information about the source of the untested check. For example, if you want to know why “zOut9” is untested, run the following command:

```
gca_shell> report_case_details -to [get_ports zOut9]
```

Example 3-12 shows the output report.

Example 3-12 Output for the report_case_details Command

```
****************************************
Report : case_details
Design : zDesign
. . .
****************************************
Properties         Value    Pin/Port
------------------------------------------------------------------------------
constant net       0        zOut9

Case fanin report:
Verbose Source Trace for pin/port zOut9:
Path number: 1
Path Ref #    Value  Properties           Pin/Port
------------------------------------------------------------------------------
             0                            zOut9
             0      F()=A                 io_buffer_out_231/Y
             0                            io_buffer_out_231/A
             0      F()=A'                zDesign_top_i0/Y
             1                            zDesign_top_i0/A
             1      F()=A'                zDesign_top_i0/Y
             0                            zDesign_top_i0/A
             0      F()=A'                zDesign_top_i0/top_tieoff_i0/U100/Y
             1                            zDesign_top_i0/top_tieoff_i0/U100/A
             1      constant net          zDesign_top_i0/top_tieoff_i0/U42/HI
```

In Example 3-12, you can see that the output is constant because of the TIE cell “zDesign_top_i0/top_tieoff_i0/U42”.

Using report_analysis_coverage in Multi-Scenario Environments

The PrimeTime GCA tool analyzes several scenarios in one command. This is particularly important with analysis coverage reports. Not all the endpoints are tested in a given mode; however, you should try to cover as many endpoints or timing arcs as possible over all the design modes.

For example, if the design has scan chains, the scan-related pins in the flip-flops are not covered in functional mode. This results in a collection of untested checks in the design. However, after the PrimeTime GCA tool analyzes both the functional and test modes, all flip-flop related checks should be covered and reported as tested.

To summarize, because not all endpoints are covered in every mode, it is important to distinguish between those endpoints that are not tested in any mode and those endpoints that are tested in at least one of the modes.

The PrimeTime GCA tool takes this into account and generates a report slightly different from the one created in a single-scenario environment. Example 3-13 shows this variation:

Example 3-13 report_analysis_coverage for Multi-Scenario Environments

```
                                         Tested in          Untested in
Type of Check                Total      1+ scenario        all scenarios
----------------------------------------------------------------------
clock_gating_hold             9400       4165 ( 44%)         5235 ( 56%)
clock_gating_setup            9400       4165 ( 44%)         5235 ( 56%)
hold                       1328927     575149 ( 43%)       753778 ( 57%)
min_period                     352        351 (100%)            1 (  0%)
min_pulse_width             870706     491034 ( 56%)       379672 ( 44%)
out_hold                        36         36 (100%)            0 (  0%)
out_setup                       36         36 (100%)            0 (  0%)
recovery                    315842     295658 ( 94%)        20184 (  6%)
removal                     315838     295654 ( 94%)        20184 (  6%)
setup                      1328927     575149 ( 43%)       753778 ( 57%)
----------------------------------------------------------------------
All Checks                 4179464    2241397 ( 54%)      1938067 ( 46%)
```

The detailed reports in multi-scenario analysis show the modes where a particular check is tested. To obtain these details, run

```
gca_shell> report_analysis_coverage -check_type out_hold \
                   -style verbose -status_details tested
```

Example 3-14 shows the PrimeTime GCA output:

Example 3-14 report_analysis_coverage Detailed Output

```
                                         Tested in          Untested in
Type of Check                Total      1+ scenario        all scenarios
---------------------------------------------------------------------
out_hold                        36         36 (100%)            0 (  0%)
---------------------------------------------------------------------
All Checks                      36         36 (100%)            0 (  0%)
. . .
Constrained        Related                                     Untested
Pin                Pin          Check Type         Scenarios    Reasons
-----------------------------------------------------------------------
zOut1                           out_hold           default      x
                                                   FF_FUNC      x
                                                   FF_SCAN      t
zOut2                           out_hold           default      x
                                                   FF_FUNC      t
                                                   FF_SCAN      x
zOut3                           out_hold           default      x
                                                   FF_FUNC      t
                                                   FF_SCAN      x
. . .
```

In the “Untested Reasons” column, the “t” stands for tested. As a consequence, you can see that zOut1 is tested in scan mode, whereas zOut2 and zOut3 are tested in functional mode.

Analyzing Redundant and Dominant Constraints Using report_exceptions

The PrimeTime GCA tool helps you develop complete and efficient constraints. When developing a design with many constraints, it is common to encounter either incorrect, incomplete, or unnecessary constraints. Incorrect or incomplete constraints have a direct effect on the quality and performance of the design.

Unnecessary constraints are difficult to find and lead to increased memory usage and runtime. Therefore, it is important to have an automated way to detect them.

This section describes how to find redundant constraints and determine the dominant constraints in your design in the following subsections:

- Exception Reporting Overview
- Using report_exceptions
- Narrowing Down report_exceptions Reports
- Reporting Ignored Exceptions
- Reporting Dominant Exceptions
- Reporting Dominant Exceptions for Ignored Exceptions

Exception Reporting Overview

To refine your constraint sets and thereby improve your quality of results, use the `report_exceptions` command.

After you load the various constraints for each scenario into your design, use the `report_exceptions` command to analyze and report the status of your constraints. In addition to reporting various types of exceptions, the tool can analyze your design for redundant and dominant exceptions. This capability helps you minimize your constraints.

For example, run the `report_exceptions -ignored` command. The PrimeTime GCA tool analyzes your design and returns a list of exceptions that have no effect on the design and could be safely removed from the constraint sets. When you run the `report_exceptions -dominant` command, the tool returns a list of exceptions that do affect timing.

With the PrimeTime GCA tool, you can identify incorrect, incomplete, and unnecessary constraints as follows:

- Most incorrect constraints are flagged with one of the many built-in rules checked by the `analyze_design` command in the PrimeTime GCA tool.

- Incomplete constraints can be identified through either the built-in rules or the analysis coverage capability provided by the `report_analysis_coverage` command. See "Analysis Coverage Overview" on page 3-15 for more information.
- Unnecessary constraints are reported by the `report_exceptions` command.

To find exceptions that do not affect the timing of the circuit, but might have a negative impact on tool performance, use the `report_exceptions -ignored` command. You can also find these occurrences by turning on the EXC_0006 and EXC_0014 rules when you run the `analyze_design` command. These rules are disabled by default because they need extra runtime. See "Enabling and Disabling Rules" on page 2-33 for more information.

Using report_exceptions

The `report_exceptions` command works on the current scenario. If you have several scenarios loaded, you can switch from one scenario to another using the `current_scenario` command.

When used without any options, the `report_exceptions` command reports all timing exceptions as defined in your SDC (or Tcl) files. All of the exceptions and constraint specifications are reported in your constraint file. Example 3-15 shows an example of a default report.

Example 3-15 Default Report for the report_exceptions Command

```
****************************************
Report : exceptions
Design : zDesign
Scenario: SCAN_SCENARIO
. . .
****************************************
#  /remote/designs/zDesign/zDesign_scan.sdc, line 88
set_false_path -through [get_pins {jtag_wrap_inst/LVISION_JTAP_INST/
LVISION_JTAP_DR_CTRL/INT_DR_LATCH_reg_44_/q}]

#  /remote/designs/zDesign/zDesign_scan.sdc, line 89
set_false_path -through [get_pins {jtag_wrap_inst/LVISION_JTAP_INST/
LVISION_JTAP_DR_CTRL/INT_DR_LATCH_reg_69_/q}]

#  /remote/designs/zDesign/zDesign_scan.sdc, line 90
set_false_path -through [get_pins {jtag_wrap_inst/LVISION_JTAP_INST/
LVISION_JTAP_DR_CTRL/INT_DR_LATCH_reg_42_/q}]
. . .
```

Example 3-15 provides a list of all of the exceptions but, for this very reason, it can be difficult to find specific data within the report. The next section describes how to narrow down your reports to locate specific problems.

Narrowing Down report_exceptions Reports

The `report_exceptions` command offers all the standard filtering options, such as `-from/through/to` and `-rise_through/fall_through`. These can be used to reduce the number of exceptions reported. When a filter is used, only exceptions that have the same `-from/through/to` clause are reported.

To help understand how the `-from/through/to` filters are used, consider the constraints defined in Example 3-16.

Example 3-16 Exceptions Defined in Constraints File

```
set_multicycle_path 3 -from [get_clocks CLK1]
set_multicycle_path 3 -from ff1/CP
```

If ff1/CP is actually clocked by CLK1, when you run

```
gca_shell> report_exceptions -from [get_clocks CLK1]
```

the tool reports only the following timing exception:

```
set_multicycle_path 3 -from [get_clocks CLK1]
```

The following multicycle path (MCP) exception

```
set_multicycle_path 3 -from ff1/CP
```

is not reported. When you use the `-from/through/to` options with the `-ignored` or `-dominate` options, the set of reported exceptions is reduced based on whether the filtered set of exceptions are ignored for all timing paths or used for at least one timing path.

Reporting Ignored Exceptions

Ignored exceptions are timing exceptions that do not have any effect in the design. There can be many different reasons why an exception is ignored. When the PrimeTime GCA tool marks an exception as ignored, it also identifies the reason the constraint was ignored as one of the following conditions:

- The exception is fully covered by another exception in the SDC file.
- There is no path in the design related to this exception.
- No valid startpoint is specified in the exception.
- No valid endpoint is specified in the exception.

Example 3-17 shows an example report. You can also use the `-ignored` option in conjunction with other options, such as `-from`, to narrow the set of exceptions in the report.

Example 3-17 Output for the report_exceptions -ignored Command

```
****************************************
Report : exceptions
Design : zDesign
Scenario: FUNCTIONAL
. . .
****************************************
#########################################################################
###
## Redundant exceptions (totally overridden by other exceptions):

#  /remote/designs/zDesign/sdc/functional.sdc, line 440 \
set_false_path -from [get_clocks {Top_clk_1}] -through [get_pins
{zdesign_core_i0/
zPin1}] -to [get_clocks {Top_clk_2}]
#########################################################################
###
## Exceptions that do not cover any constrained paths:
#   /remote/designs/zDesign/sdc/functional.sdc, line 474
set_false_path -through [get_pins {zDesign_core_i0/i_resetb_pad_din}]
#########################################################################
###
## Exceptions with no valid -from objects:
#   /remote/designs/zDesign/sdc/functional.sdc, line 503
set_false_path -hold -from [get_ports {JTAG_CE}] -to [get_pins \
{zDesign_top/
jtag_control_inst/reg_127_/d}]
```

To reduce runtime and memory usage in future sessions, remove ignored exceptions from the SDC file.

Note:
The PrimeTime GCA tool does not modify constraints. Edit your constraints and rerun the tool for verification.

Reporting Dominant Exceptions

The PrimeTime GCA tool reports the dominant exceptions in the design. To report the dominant exceptions, run the following command:

```
gca_shell> report_exceptions -dominant
```

An exception is considered dominant if it affects the constraints on at least one path and its absence would result in a different set of constraints in the design. Example 3-18 shows output for the `report_exceptions -dominant` command.

Example 3-18 Output for the report_exceptions -dominant Command

```
****************************************
Report : exceptions
Design : zDesign
Scenario: FUNCTIONAL
. . .
****************************************
#####################################################################
## Exceptions that are dominant for at least one path:

#  /remote/designs/zDesign/sdc/functional.sdc, line 433
set_false_path -from [get_clocks {clk_1}] -to [get_clocks {clk_54}]

#  /remote/designs/zDesign/sdc/functional.sdc, line 434
set_false_path -from [get_clocks {clk_54r}] -to [get_clocks {clk_1}]
```

You can narrow down the set of reported exceptions by combining the `-dominant` option with other options.

Reporting Dominant Exceptions for Ignored Exceptions

The PrimeTime GCA tool reports the dominant exceptions for ignored exceptions in the design. To report these exceptions, run the following command:

gca_shell> **`report_exceptions -ignored -verbose`**

Example 3-19 shows output for the `report_exceptions -ignored -verbose` command.

Example 3-19 Output for the report_exceptions -ignored -verbose Command

```
****************************************
Report : exceptions
Design : test
Scenario: default
...
****************************************
#########################################################################
## Redundant exceptions (totally overridden by other exceptions):

# The following exception in test.tcl at line 62 is dominated by other
exceptions:
set_multicycle_path 2  -setup -from [get_clocks {CLK1_1}]
# The following exceptions dominate the above exception :
    #  test.tcl, line 62
    set_multicycle_path 3  -setup -from [get_pins {H1/H1/F1/CLK}]
    # test.tcl, line 64
    set_multicycle_path 2  -setup -from [get_pins {H2/H1/F1/CLK}]

# The following exception in test.tcl at line 65 is dominated by other
exceptions:
set_multicycle_path 3  -setup -from [get_clocks {CLK2_1}]
```

```
# The following exceptions dominate the above exception:
    #  test.tcl, line 62
    set_multicycle_path 3  -setup -from [get_pins {H1/H1/F1/CLK}]
    # test.tcl, line 64
    set_multicycle_path 2  -setup -from [get_pins {H2/H1/F1/CLK}]
```

4

Graphical User Interface

The PrimeTime GCA graphical user interface (GUI) provides an environment to view and debug constraint problems for PrimeTime and other Synopsys tools, such as Design Compiler and IC Compiler. The GUI environment is the main interface for working with the PrimeTime GCA tool.

The PrimeTime GCA GUI is described in the following sections:

- Using the GUI
- PrimeTime GCA Window
- User Message Browser
- Violation Browser
- Block-to-Top Violation Browser
- SDC-to-SDC Violation Browser
- Waiver Configuration Dialog Box
- Information Pane
- Console
- Online Help
- SDC View
- Hierarchy Browser and Schematic View

- Select By Name Toolbar
- Properties Dialog Box
- Schematic Display Options

Using the GUI

The GUI can help you visualize and understand the nature of the violations you might encounter when working with the PrimeTime GCA tool. You can use the visual analysis tools contained in the PrimeTime GCA tool after you read, link, constrain, and analyze the design.

An analysis session might consist of the following tasks:

1. Start the GUI.
2. Read, link, constrain, and analyze the design.
3. Perform in-depth analysis, examining specific aspects of the violation paths in detail.

The default behavior for starting a PrimeTime GCA session that automatically includes the GUI is to use `gca_shell`. Within the shell you can use the `gui_start` and `gui_stop` commands to start and stop the GUI.

If you want to run the PrimeTime GCA tool in shell-only mode, enter the `gca_shell -no_gui` commands.

You can display or hide elements of the GUI. For example, to display or hide a toolbar or panel, choose View > Palette/Toolbars and choose the appropriate command to display or to hide the corresponding area of the window. A check mark is shown next to the menu item if that element is currently being displayed in the window.

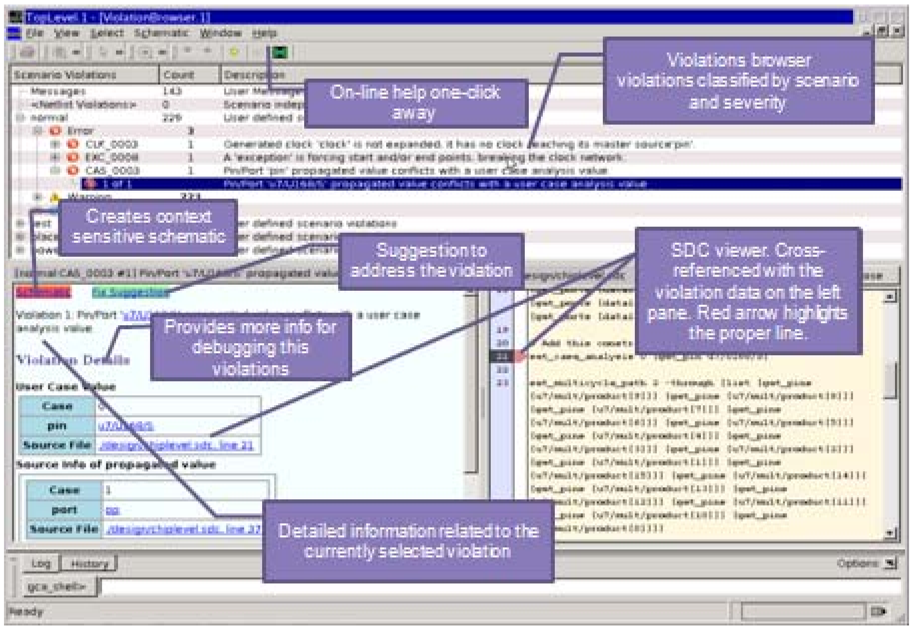
The lower area of the window contains the console. The `gca_shell` prompt is on the console. Use the Log and History tabs above the `gca_shell` prompt to display different types of information about the console.

To stop the GUI while still keeping the original `gca_shell` session going in the terminal window, use the `gui_stop` command or choose File > Close GUI. To exit the PrimeTime GCA tool, choose File > Exit.

PrimeTime GCA Window

Starting the GUI opens the PrimeTime GCA window. This is the main PrimeTime GCA window in which you can display many types of analysis windows. You can open, close, move, resize, dock, undock, maximize, and minimize various types of windows within the PrimeTime GCA window. Figure 4-1 shows the PrimeTime GCA window containing the menu bar, various toolbars, the violation browser, the SDC view, and the console. You can change the appearance by opening, resizing, and closing view windows within the PrimeTime GCA window. You can also open multiple instances of the PrimeTime GCA window to create different analysis views.

Figure 4-1 PrimeTime GCA Window


On-line help one-click away
Violations browser violations classified by scenario and severity
Creates context sensitive schematic
Suggestion to address the violation
SDC viewer. Cross-referenced with the violation data on the left pane. Red arrow highlights the proper line.
Provides more info for debugging this violations
Detailed information related to the currently selected violation

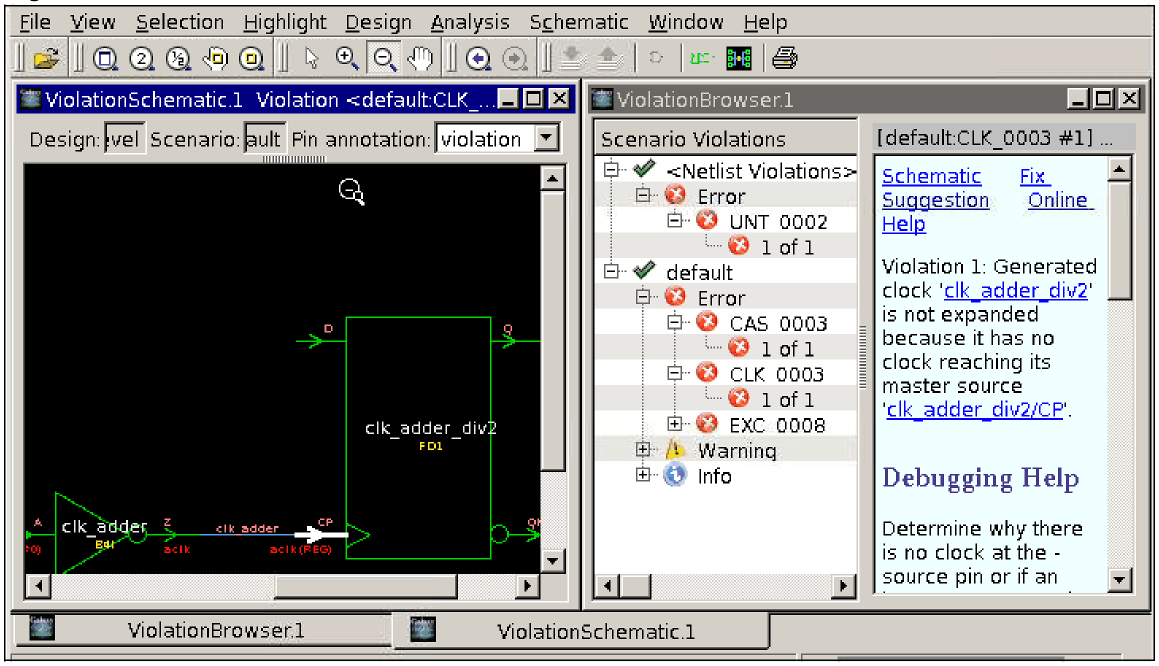
Manipulating Windows

Each window can be minimized or maximized. A window can be moved anywhere within the top-level window. Windows can overlap each other.

A window has a title bar with buttons to minimize, maximize, or close the window, and you can freely move and resize the window.

To lay windows out evenly like tiles, choose the menu command Window > Tile, as shown in Figure 4-2. To cascade the windows, choose Window > Cascade. You can arrange and resize the windows as shown in Figure 4-3.

Figure 4-2 Tiled Windows


File View Selection Highlight Design Analysis Schematic Window Help
ViolationSchematic.1 Violation <default:CLK_...
Design: Scenario: Pin annotation:
clk_adder_div2
clk_adder
ViolationBrowser.1
Scenario Violations
[default:CLK_0003 #1] ...
Schematic Fix Suggestion Online Help
ViolationBrowser.1 ViolationSchematic.1

Design: [vel] Scenario: [ault] Pin annotation: violation

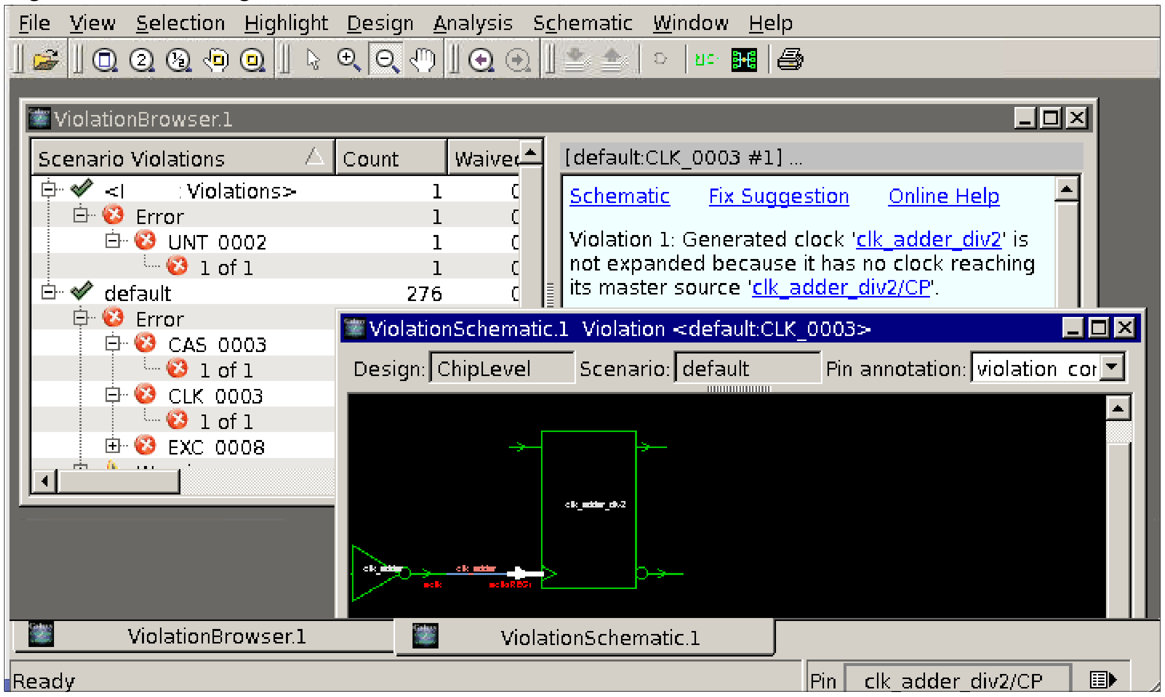
```
<Netlist Violations>
  Error
    UNT 0002
      1 of 1
default
  Error
    CAS 0003
      1 of 1
    CLK 0003
      1 of 1
    EXC 0008
  Warning
  Info
```

Violation 1: Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its master source 'clk_adder_div2/CP'.

Debugging Help

Determine why there is no clock at the source pin or if an

Figure 4-3 Arranged and Resized Windows


File View Selection Highlight Design Analysis Schematic Window Help
ViolationBrowser.1
[default:CLK_0003 #1] ...
Schematic Fix Suggestion Online Help
ViolationSchematic.1 Violation <default:CLK_0003>
clk_adder_div2
ViolationBrowser.1 ViolationSchematic.1
Ready
Pin clk_adder_div2/CP

```
Scenario Violations         Count  Waive
<l   Violations>                1  0
  Error                         1  0
    UNT 0002                    1  0
      1 of 1                    1  0
default                       276  0
  Error
    CAS 0003
      1 of 1
    CLK 0003
      1 of 1
    EXC 0008
```

Violation 1: Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its master source 'clk_adder_div2/CP'.

Design: ChipLevel Scenario: default Pin annotation: violation cor

You can move, resize, and minimize windows at any time. To move a window, point to the title bar and drag it to the position you want.

Window Types

Table 4-1 lists and briefly describes the types of windows that can be displayed within the PrimeTime GCA window. You can find additional information about each type of window in the remaining sections of this chapter.

Table 4-1 Types of Windows Within the PrimeTime GCA Window

Window type	**Description**
Violation Browser	Lists the violations in the current design, classified by scenario and severity.
B2T Mismatch Browser	Lists the violations between the constraints defined for the top level of the current design and the block level, classified by severity.
S2S Mismatch Browser	Lists the differences between two sets of constraints for the current design, classified by severity.
User Message Browser	Lists the user messages in the current design, classified by severity.
Information Pane	Lists detailed information related to the current selection.
Console Window	Lets you run `gca_shell` command and view the `gca_shell` log, command history, or error and warning messages.
SDC View	Displays the SDC constraint file and places a marker on the relevant line.
Hierarchy Browser and Schematic View	Lets you browse through the design hierarchy and select a cell, net, port, or pin in the design.
Path Schematic	Displays a circuit schematic of a hierarchical cell or a path in the design.
Properties Dialog Box	Displays a list of the object's properties.

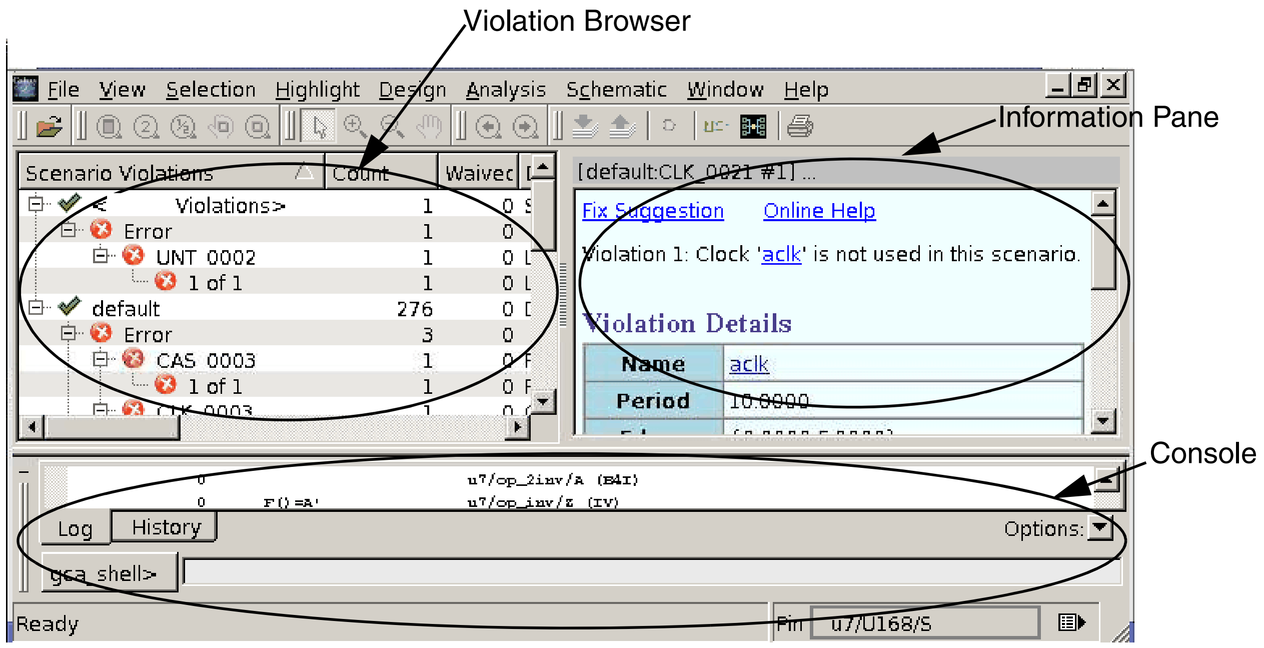
The PrimeTime GCA window starts out with the following windows:

- Violation browser
- Information pane

By default, the violation browser, information pane, and console windows are docked and displayed on the left side, as shown in Figure 4-4. You can undock or remove these default windows and open additional windows such as Schematics and Query. There is an "on-demand" hierarchy browser window that you can add to the standard appearance of the

GUI by modifying your default preferences. If the hierarchy browser window is not present, you can open one by choosing Window > New Hierarchy Browser.

Figure 4-4 PrimeTime GCA Window on Startup



Scenario Violations	Count	Waived	
< Violations>	1	0	S
Error	1	0	
UNT_0002	1	0	L
1 of 1	1	0	L
default	276	0	[
Error	3	0	
CAS_0003	1	0	F
1 of 1	1	0	F
CLK_0003	1	0	C

[default:CLK_0021 #1] ...

Fix Suggestion   Online Help

Violation 1: Clock 'aclk' is not used in this scenario.

Violation Details

Name	aclk
Period	10.0000

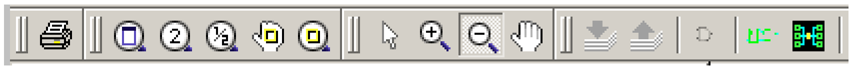
```
0                     u7/op_2inv/A (B4I)
0      F()=A'         u7/op_inv/Z (IV)
```

Ready   Pin u7/U168/S

Toolbars

The toolbar buttons provide quick access to often-used menu commands. Figure 4-5 shows an example of the toolbar buttons.

Figure 4-5 Toolbar Buttons



Buttons that cannot be used in the active view are dimmed. The current context changes when you make a new selection. For example, after you select a path, the toolbar options that operate on paths become enabled. When no paths are selected, those buttons functions are dimmed.

Menu Commands

Table 4-2 briefly describes each of the menu bar menu commands.

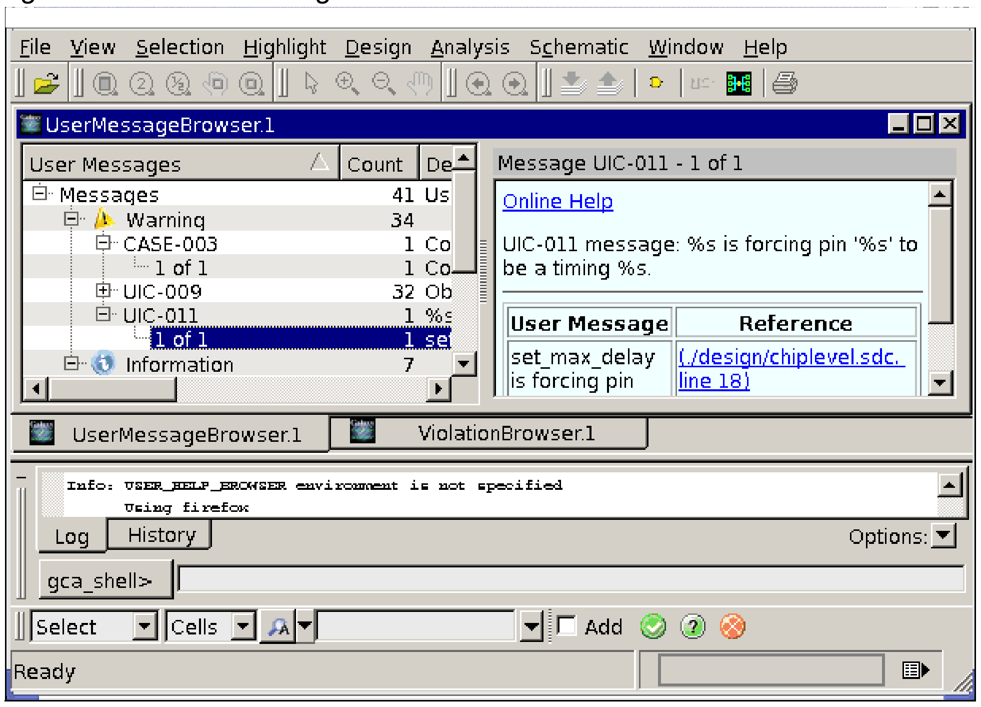
Table 4-2 Menu Commands

Menu commands	**Description**
File	Contains commands for executing scripts, printing schematics, and exiting the tool.
View	Contains commands for controlling the design views, mouse function, InfoTip display, toolbars, and other view settings.
Selection	Contains commands for selecting schematic objects and reporting the selected objects.
Highlight	Contains commands for coloring selected objects in schematic views.
Design	Contains commands for analyzing the design and opening the hierarchy browser and the waiver configuration dialog box.
Analysis	Contains commands for opening the violation browser, B2T mismatch browser, S2S mismatch browser, and user message browser.
Schematic	Contains commands for opening and controlling schematic views.
Window	Contains commands for controlling the display of windows within the PrimeTime GCA GUI.
Help	Contains commands for getting online Help.

User Message Browser

User messages are displayed in the user message browser shown in Figure 4-6. To open the user message browser, choose Analysis > UserMessage Browser. In the browser tree, the messages are sorted according to severity. When you select a message in the browser tree, the message details are displayed in the information pane. For the message selected in Figure 4-6, the information pane identifies the line number of the problem SDC constraint and a link to that line in the SDC file.

Figure 4-6 User Message Browser



User Messages	Count	De
Messages	41	Us
Warning	34	
CASE-003	1	Co
1 of 1	1	Co
UIC-009	32	Ob
UIC-011	1	%s
1 of 1	1	se
Information	7	

Message UIC-011 - 1 of 1

Online Help

UIC-011 message: %s is forcing pin '%s' to be a timing %s.

User Message	Reference
set_max_delay is forcing pin	(./design/chiplevel.sdc, line 18)

```
Info: USER_HELP_BROWSER environment is not specified
      Using firefox
```

Violation Browser

This section provides an overview and guidelines for navigating the violation browser, in the following sections:

- Violation Browser Overview
- Guidelines for Navigating in the Violation Browser

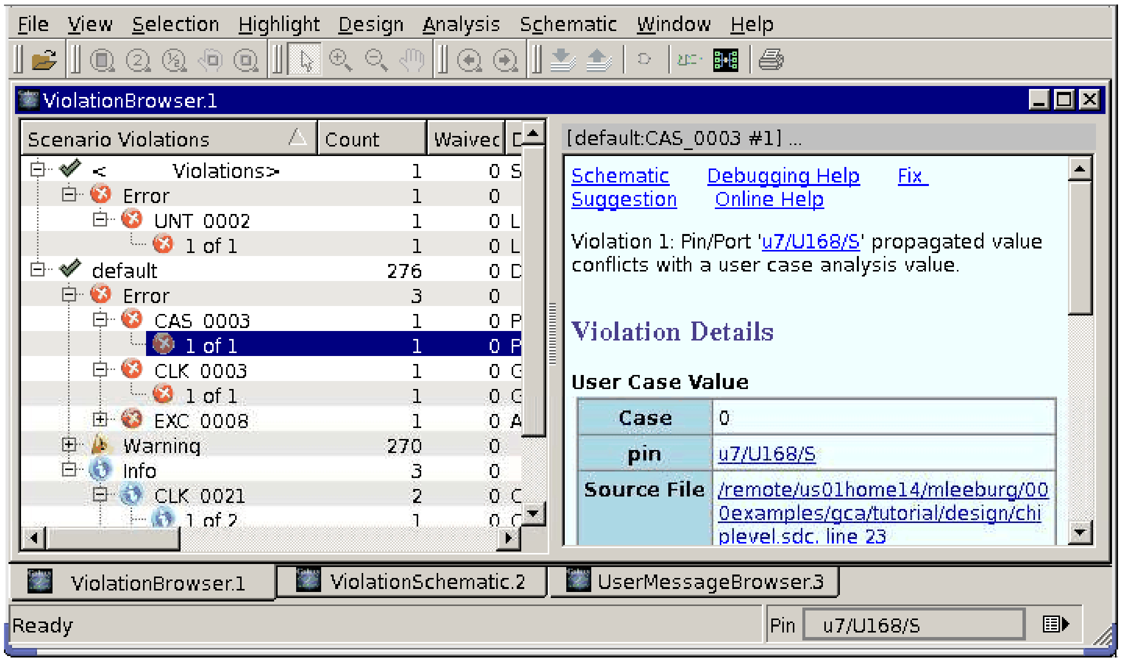
Violation Browser Overview

The violation browser is the main tool within the PrimeTime GCA tool for performing debugging tasks. It displays a hierarchical list of the design and constraint violations that have been detected by the tool along with suggestions on how to fix the problem. The violation browser supports netlist-related and scenario-related violations (one node per scenario). Violations are displayed in the following order: Error, Warning, and Information.

By default, the violation browser displays up to 1000 violation identifier nodes for each rule violated. Use the `display_violations_per_rule_limit` variable to specify a different limit.

Figure 4-7 shows a typical violation browser.

Figure 4-7 Violation Browser



Scenario Violations	Count	Waivec	D
< Violations>	1	0	S
Error	1	0	
UNT_0002	1	0	L
1 of 1	1	0	L
default	276	0	D
Error	3	0	
CAS_0003	1	0	P
1 of 1	1	0	P
CLK_0003	1	0	C
1 of 1	1	0	C
EXC_0008	1	0	A
Warning	270	0	
Info	3	0	
CLK_0021	2	0	C
1 of 2	1	0	C

[default:CAS_0003 #1] ...

Schematic Debugging Help Fix Suggestion Online Help

Violation 1: Pin/Port 'u7/U168/S' propagated value conflicts with a user case analysis value.

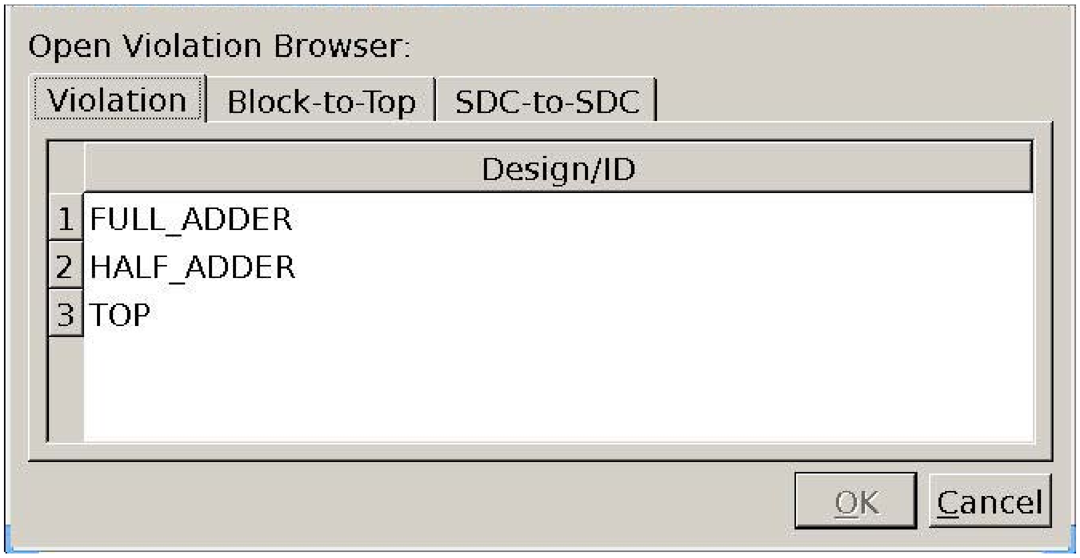
Violation Details

User Case Value

Case	0
pin	u7/U168/S
Source File	/remote/us01home14/mleeburg/000examples/gca/tutorial/design/chiplevel.sdc, line 23

During a session, the PrimeTime GCA tool stores the results of multiple analysis runs in multiple violation browsers. You use the analysis run chooser window, shown in Figure 4-8, to choose the specific run results to view. Access the analysis run chooser window from the Analysis menu on the menu bar.

Figure 4-8 Analysis Run Chooser



	Design/ID
1	FULL_ADDER
2	HALF_ADDER
3	TOP

This window displays the available results grouped by violation type. In addition to the standard violation browser, there is the B2T violation mismatch browser, and the S2S violation mismatch browser. For more information about these browsers, see Chapter 5, “Comparing Block- and Top-Level Constraints” and Chapter 6, “Comparing Two Sets of Design Constraints.”When the GUI is first opened, all of the severity nodes in the violation browser are expanded and all of the violation identifier nodes are collapsed. When expanded, a severity node displays all of the violations present in the design at that severity level.

You can specify the Web browser used to display the messages by setting an environment variable at the operating system prompt.

The browser executable needs to be in your environment variable setting.

```
setenv USER_HELP_BROWSER preferred_browser
```

Guidelines for Navigating in the Violation Browser

In the violation browser, clicking the +/- next to a violation identifier node expands the node and displays every individual violation of that type.

Whenever an individual violation is selected, the information pane shows the specifics of that particular violation. Whenever a violation instance is selected, the information pane shows the relevant data for the violation instance and a general message for the particular set of violations (for example, CLK_0004 message).

The violation browser groups violations of a rule if the violations contain similar object names. The browser displays these multiple violations as a single group, as shown in Figure 4-7. This feature helps you see the scope of all your design violations. When you select a group name, the information pane displays the Description field shown in the violation browser. The PrimeTime GCA tool allows you to suppress rule violations. You can either disable a rule completely or suppress a specific instance of a rule violation, also known as waiving a violation. To disable grouping or enable grouping for a specific number of levels, use the `grouping_violations_hierarchy_separator_limit` variable. Both waived and unwaived violations are displayed in a group. Rules without netlist objects in their parameters are not grouped because violation grouping is performed on netlist object names, such as pins, ports, and cells.

Block-to-Top Violation Browser

To compare a set of constraints from a block to the constraints from the top-level chip, use the `compare_block_to_top` command. You use the block-to-top violation browser to view the violations.

See Chapter 5, “Comparing Block- and Top-Level Constraints” for more information.

SDC-to-SDC Violation Browser

To determine the functional differences between the two sets of constraints using one common netlist, use the `compare_constraints` command. You use the SDC-to-SDC violation browser to view the violations.

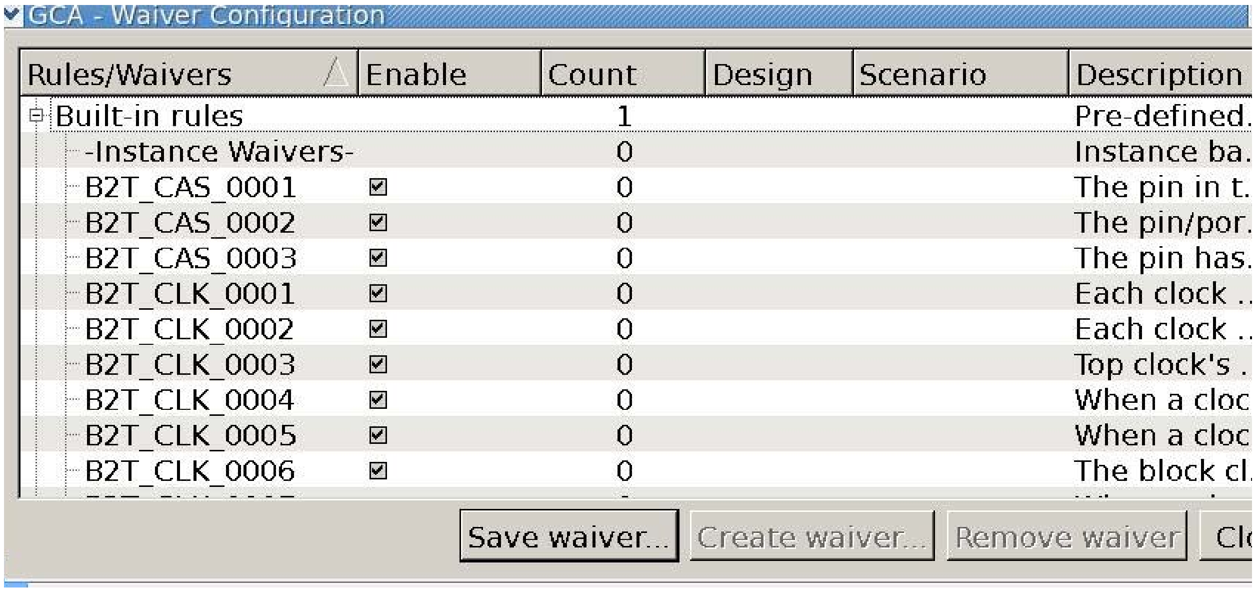
See Chapter 6, “Comparing Two Sets of Design Constraints” for more information.

Waiver Configuration Dialog Box

The waiver configuration dialog box allows you to waive either an entire rule or specific violations of a rule. Unlike the violation browser, you can use the waiver configuration dialog box both before and after running the `analyze_design` command. In addition, the waiver configuration dialog box gives you complete access to all the rules available in the PrimeTime GCA tool, whereas the violation browser shows only those rules with violations.

To open the waiver configuration dialog box, choose Design > Waiver Configuration. The PrimeTime GCA tool displays the waiver configuration dialog box, as shown in Figure 4-9.

Figure 4-9 Waiver Configuration Dialog Box



Rules/Waivers	Enable	Count	Design	Scenario	Description
Built-in rules		1			Pre-defined.
-Instance Waivers-		0			Instance ba.
B2T_CAS_0001	✓	0			The pin in t.
B2T_CAS_0002	✓	0			The pin/por.
B2T_CAS_0003	✓	0			The pin has.
B2T_CLK_0001	✓	0			Each clock ..
B2T_CLK_0002	✓	0			Each clock ..
B2T_CLK_0003	✓	0			Top clock's .
B2T_CLK_0004	✓	0			When a cloc
B2T_CLK_0005	✓	0			When a cloc
B2T_CLK_0006	✓	0			The block cl.

The columns in the waiver configuration dialog box are described in Table 4-3.

Table 4-3 Waiver Configuration Dialog Box Columns

Column	Description
Rules/ Waivers	Name of the rule. You can access all built-in and user-defined rules with the waiver configuration dialog. A plus sign next to the rule indicates that specific violations of the rule have been waived.
Enable	A check mark indicates that the rule is enabled. To disable the rule, uncheck the box. This method of enabling and disabling a rule applies to all scenarios.
Count	Shows the number of waivers for the rule.
Design	Specifies the design to which a waiver is applied. If no design is specified, the waiver applies to the current design.
Scenario	Lists the scenarios in which the waiver is active. If no scenario is listed, the waiver applies to all scenarios.
Description	Brief explanation of the built-in rule or user-defined comment for a user-defined rule. For detailed information about the built-in rules, see the rule reference documents in the online Help.

Information Pane

The information pane, shown in Figure 4-4 on page 4-7, is a central feature of the PrimeTime GCA GUI environment. Whereas the violation browser allows you to find general information about rule violations, the information pane displays the details of these violations.

Whenever a violation is selected in the violation browser, the information pane automatically displays the relevant information. In this way the information pane is always synchronized with whatever is selected in the violation browser window.

The information pane provides detailed information related to the current violation, including the full message of the violation and hyperlinks to related objects (whenever applicable).

The information pane also provides suggestions on how to debug the violation in more detail and provides the necessary setup details for creating some path schematics.

Console

The PrimeTime GCA tool provides a console, shown in Figure 4-4 on page 4-7, in which you can enter interactive commands; anything that can be done in a batch PrimeTime GCA session can also be done through the console.

Standard output and error output are displayed in the console log view.

To save the command history list into a file, click the History tab and click the Save Contents As button.

Online Help

An online Help system is provided as part of the PrimeTime GCA tool. It contains all documentation related to the PrimeTime GCA tool, including:

- User guides
- Man pages
- Rule documentation

You can access the online Help system from the Help menu in the GUI environment and from the command line when entering a `man` *`topic`* command.

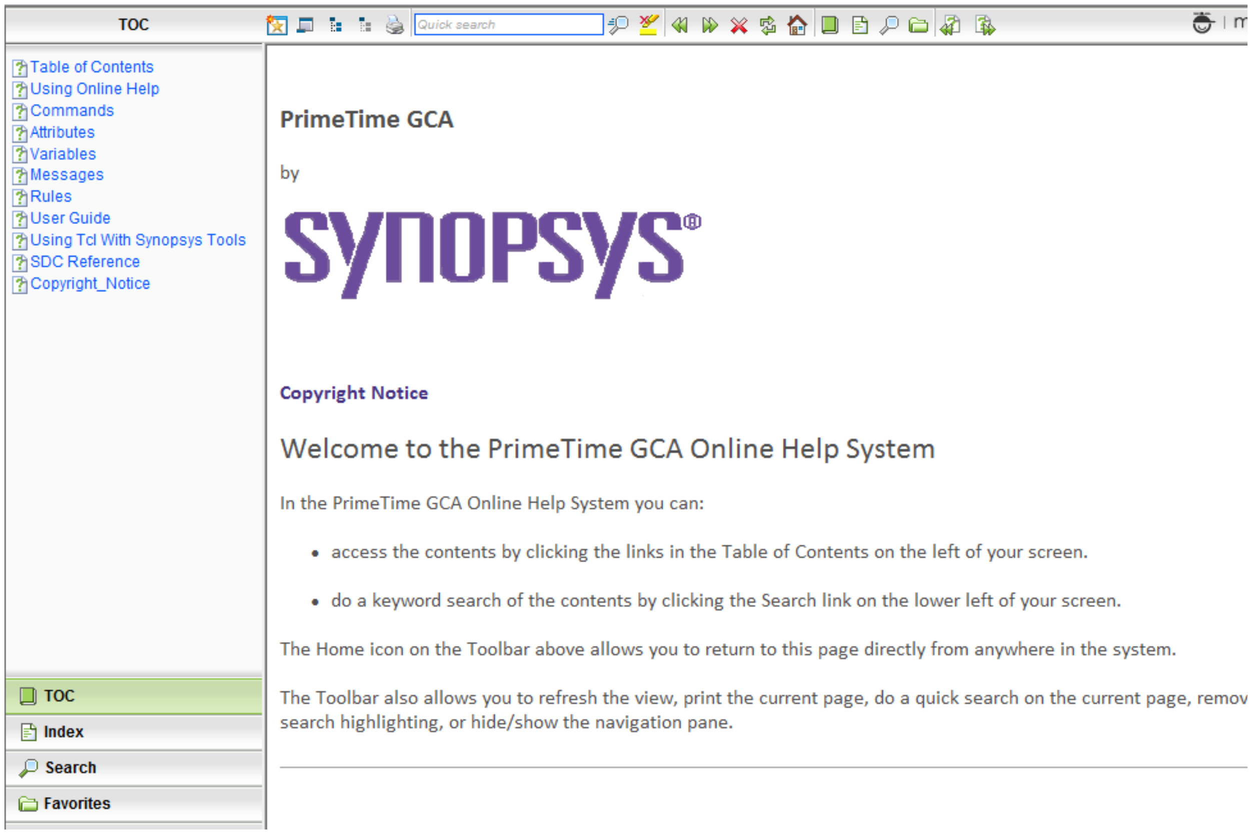
From the Help menu, choose online Help to open the PrimeTime GCA online Help system in a Web browser.

The PrimeTime GCA online Help system is also available through the information pane of any rule by clicking the Help hyperlink.

In the online Help system, which opens in a new Web browser window, the Rule Reference page for any rule contains an explanation of the rule, an example of a design containing a violation of that rule, and a suggestion on how to debug the issue.

Figure 4-10 shows the startup page of the PrimeTime GCA online Help system.

Figure 4-10 PrimeTime GCA Online Help System



PrimeTime GCA

by

Copyright Notice

Welcome to the PrimeTime GCA Online Help System

In the PrimeTime GCA Online Help System you can:

- access the contents by clicking the links in the Table of Contents on the left of your screen.
- do a keyword search of the contents by clicking the Search link on the lower left of your screen.

The Home icon on the Toolbar above allows you to return to this page directly from anywhere in the system.

The Toolbar also allows you to refresh the view, print the current page, do a quick search on the current page, remov
search highlighting, or hide/show the navigation pane.

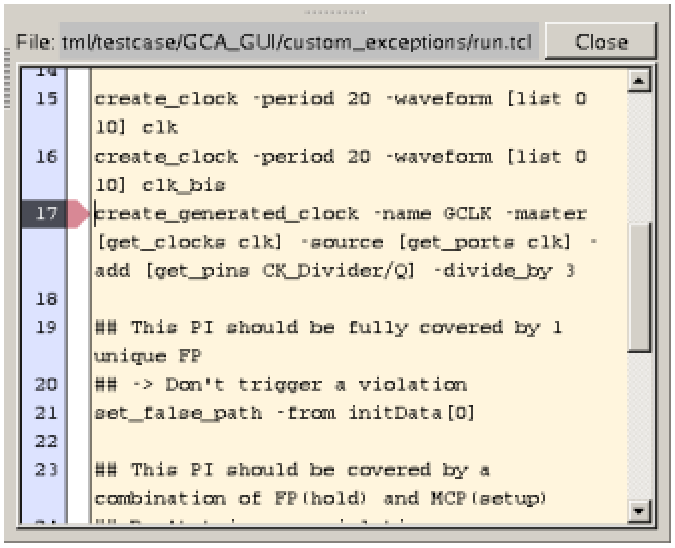
SDC View

Through the use of the SDC viewer, the PrimeTime GCA tool can identify constraints by their definition in a file (file name and line number). This type of information, when provided, is available through a hyperlink in the violation message in the violation browser and in the information pane.

By clicking the hyperlink of a constraint definition, the SDC viewer opens and automatically loads the relevant SDC constraint file and places a marker on the relevant line.

Figure 4-11 shows a typical SDC view.

Figure 4-11 SDC View



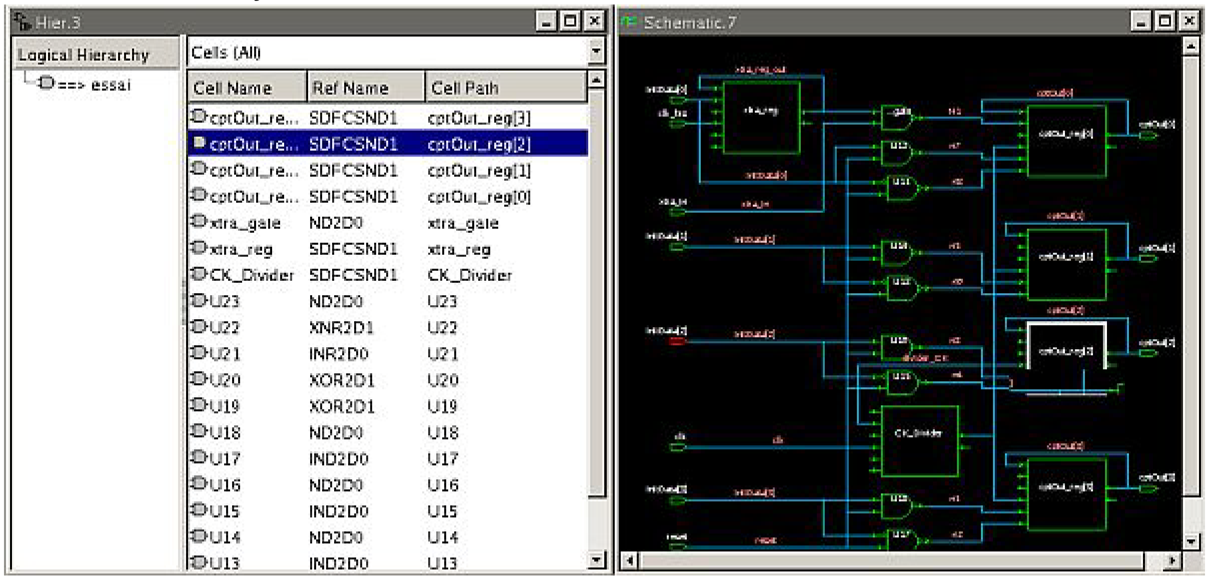
```
create_clock -period 20 -waveform [list 0
10] clk
create_clock -period 20 -waveform [list 0
10] clk_bis
create_generated_clock -name GCLK -master
[get_clocks clk] -source [get_ports clk] -
add [get_pins CK_Divider/Q] -divide_by 3

## This PI should be fully covered by 1
unique FP
## -> Don't trigger a violation
set_false_path -from initData[0]

## This PI should be covered by a
combination of FP(hold) and MCP(setup)
```

Hierarchy Browser and Schematic View

The hierarchy browser and schematic view let you examine the elements in the design hierarchy and display them in detailed schematic form.

Figure 4-12 shows a typical design in the hierarchy browser and schematic view.

Figure 4-12 Hierarchy Browser/Schematic View



Cell Name	Ref Name	Cell Path
cptOut_re...	SDFCSND1	cptOut_reg[3]
cptOut_re...	SDFCSND1	cptOut_reg[2]
cptOut_re...	SDFCSND1	cptOut_reg[1]
cptOut_re...	SDFCSND1	cptOut_reg[0]
xtra_gate	ND2D0	xtra_gate
xtra_reg	SDFCSND1	xtra_reg
CK_Divider	SDFCSND1	CK_Divider
U23	ND2D0	U23
U22	XNR2D1	U22
U21	INR2D0	U21
U20	XOR2D1	U20
U19	XOR2D1	U19
U18	ND2D0	U18
U17	IND2D0	U17
U16	ND2D0	U16
U15	IND2D0	U15
U14	ND2D0	U14
U13	IND2D0	U13

The look and feel of the PrimeTime GCA GUI are designed to be similar to those of other tools, such as the PrimeTime GUI or the Design Vision tool.

The primary function of the hierarchy browser is to display the logic hierarchy of the design. Schematics can be created for the whole selection by right-clicking any level of hierarchy or by clicking the toolbar buttons. Several schematics can be opened simultaneously, each in its own window. Each schematic view window opens so that you can see the logic.

The hierarchy browser also provides the following functionality:

- Whenever a level of hierarchy is selected, the cells belonging to that level appear on the cell list.
- The schematic view also allows you to search for a cell, net, or pin and to zoom in or zoom out to view a selection.

The schematic view can be annotated with the:

- Cell name
- Net name
- Library cell name

Selecting objects directly in the schematic also populates the relevant selection box on the status bar at the bottom of the PrimeTime GCA window.

The buttons available on the Schematics toolbar have the following functionality:

- The Create Instance Schematic button draws the level of logic currently selected. If nothing is selected, the current design is drawn.
- The Create Abstract Path Schematic of Selected Objects button draws only the selected objects and their immediate connections. If nothing is selected, the button is disabled.
- The Add Fanin/Fanout to Path Schematic button becomes active whenever a schematic window is opened. Otherwise the button is disabled.
- The zoom, pan, and select buttons on the View Tools and View Zoom/Pan toolbars enable you to navigate through the schematic.

The Schematic View also provides standard printing capabilities similar to other Synopsys GUI tools.

Path Schematic

The path schematic view in the GUI provides visualization of any part of the design rather than the full schematic view. Any path schematic you display is annotated with data, such as constant data and clock identifiers that are relevant to analyzing the issue.

The path schematic is displayed when you request it from the information pane (debugging commands).

There are a number of visibility features built into the path schematic view:

- You can expand or reduce the schematic.
- You can add or remove gates from the view to aid in the granularity of the view.
- You can select one or several objects to obtain more detailed information about them.
- You can display attributes in a separate attribute pane.

There is also a Create Path Schematic button on the toolbar you can use to create a schematic of whatever logic is currently selected.

When creating a path schematic, the tool displays a warning message beforehand if schematic generation is expected to take a long runtime. At this point, you have the option to cancel the current command. If significant runtime is anticipated when creating a path schematic, the tool displays a Work In Progress indicator to show that the tool is not failing.

Creating a Schematic

When a schematic view is applicable to a violation, the Schematic hyperlink appears in the information pane. You can automatically create a schematic for this violation by selecting the Schematic hyperlink. The schematic is annotated with information relevant to the violation.

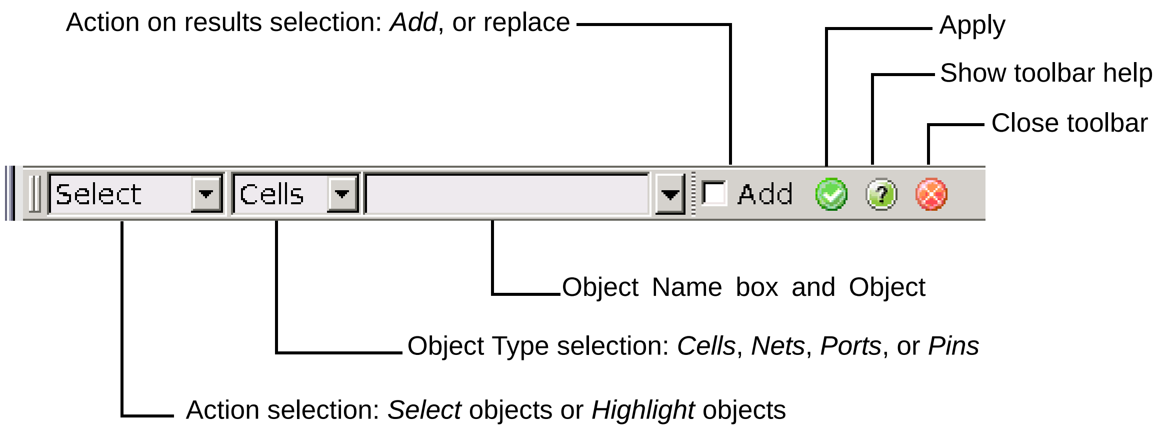
You can also create schematics at any level of logic by using the hierarchy browser or clicking the “Create Instance Schematic” button on the Schematics toolbar. The tool creates a schematic for all objects in the current selection list.

To open a hierarchy browser window, choose Design > Hierarchy Browser. Select the logic for the schematic and either right-click the selected cells and choose New Design Schematic View or choose Schematic > New Schematic View.

Select By Name Toolbar

Use the Select By Name toolbar, as shown in Figure 4-13, to quickly select or highlight objects by name in the current design.

Figure 4-13 Select By Name Toolbar


Action on results selection: *Add*, or replace
Apply
Show toolbar help
Close toolbar
Select
Cells
Add
Object Name box and Object
Object Type selection: *Cells*, *Nets*, *Ports*, or *Pins*
Action selection: *Select* objects or *Highlight* objects

The typical flow for the Select By Name toolbar involves the following steps:

1. Display the toolbar and set the keyboard focus: choose Selection > By Name Toolbar or press Ctrl+/.
2. Select the object type: press Page Up or Page Down to cycle through the object type list.
3. Set the action to be performed: press F1 to select objects or F2 to highlight objects.
4. Specify the search string: enter the search string, and press Tab to complete the name if necessary.
5. Choose whether to replace or add to the current selection: press Insert to toggle the Add option.
6. Select or highlight the object: press Enter.

When you press Enter, the tool searches the design for an object that matches the specified name or name pattern and automatically selects that object. The selected object is highlighted if it is visible in a schematic.

You can type part of a name and let the tool complete it by pressing Tab. The tool completes the name to its longest match. If multiple names match the text, a name completion list appears. You can select a name by pressing Enter or close the list by pressing Esc. Use the Up Arrow and Down Arrow keys to move up and down the list.

You can type multiple object names by separating them with blank spaces.

If an object name contains a space, enclose the name in curly braces ({}) to treat it as a single name. If an object name is invalid because nothing matches the specified name pattern, the name appears on a light red background.

The Select By Name toolbar in Table 4-4 provides the following interactive shortcut operations:

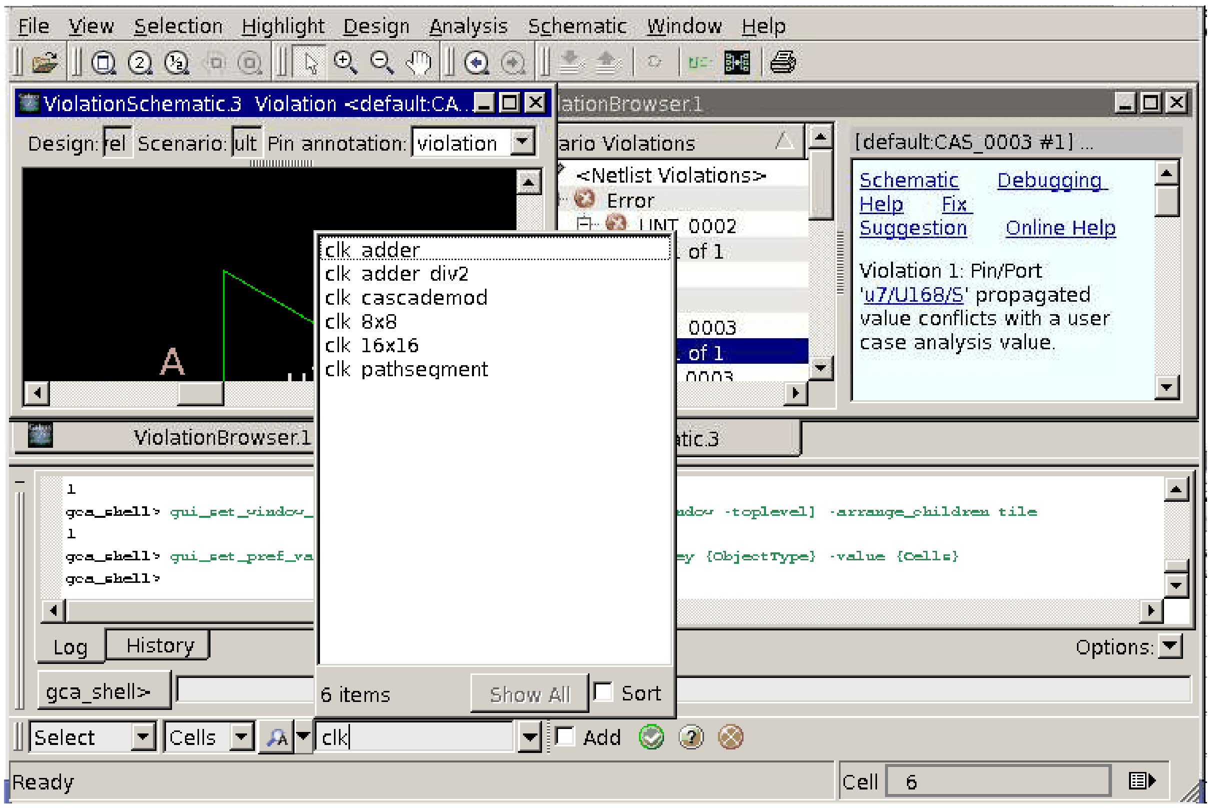
Table 4-4 Select By Name Toolbar Shortcuts

Key	**Action**
Enter	Select or highlight the objects
Tab	Complete current name to longest match
Down	Display completion menu
F1	Change to selection mode
F2	Change to highlight mode
Page Up	Move to the previous object type
Page Down	Move to the next object type
Insert	Switch between Add and Replace for selection mode
Escape	Hide the toolbar

Any characters you type while the object list is open automatically appear in the object name box. This allows you to continue typing the object name, which automatically updates the matching names in the name completion list.

Figure 4-14 shows an example of the Select By Name toolbar with the name completion list open.

Figure 4-14 Name Completion List



Violation 1: Pin/Port 'u7/U168/S' propagated value conflicts with a user case analysis value.

```
1
gca_shell> gui_set_window_          ndow -toplevel] -arrange_children tile
1
gca_shell> gui_set_pref_va          ey {ObjectType} -value {Cells}
gca_shell>
```

After a successful search, the tool automatically highlights the text in the object name box and you can start another search. You can also perform the search by clicking the Apply button.

The selection operation replaces the current selection by default. If you want to add objects to the current selection, select the Add option.

If you press the Tab key and the tool finds multiple objects that match the text you typed, the name completion list appears showing the first fifteen names.

If you enable name sorting, the option remains selected for future searches.

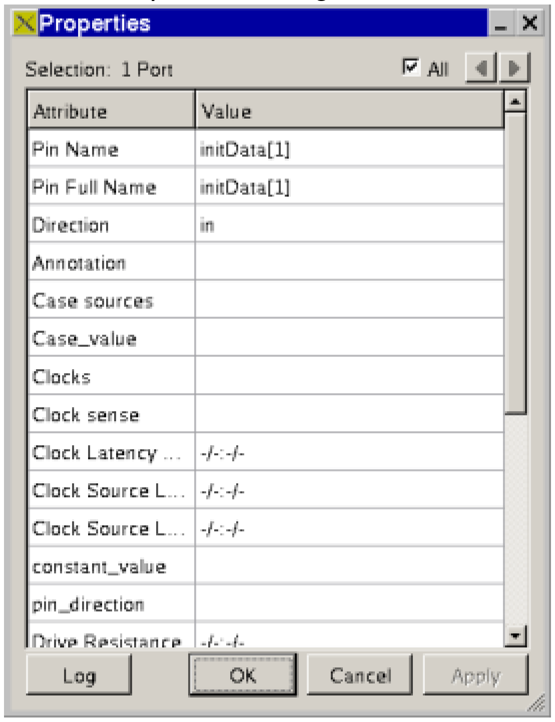
Properties Dialog Box

You can query the properties of any object in the design directly in the GUI by using the Properties dialog box. You can display an object's Properties dialog box by selecting the object in the properties dialog box. Objects that have properties are nets, pins/ports, and cells.

The Properties dialog box can also be accessed by right-clicking an object in a schematic and choosing Properties.

Figure 4-15 shows a typical attribute list in the Properties dialog box.

Figure 4-15 Properties Dialog Box



Selection: 1 Port ☑ All

Attribute	Value
Pin Name	initData[1]
Pin Full Name	initData[1]
Direction	in
Annotation	
Case sources	
Case_value	
Clocks	
Clock sense	
Clock Latency ...	-/-:-/-
Clock Source L...	-/-:-/-
Clock Source L...	-/-:-/-
constant_value	
pin_direction	
Drive Resistance	-/-:-/-

You can click the arrow buttons to scroll through the property lists for multiple selected objects. To show the combined characteristics of all selected objects in a single pane, select the All option.

Schematic Display Options

You can collapse and expand the display of buffer chains, hierarchical objects, and unconnected pins in schematic views. Collapsing these types of objects can greatly simplify the schematic display by hiding the unimportant details.

For example, in Figure 4-16, the chain of three buffers is selected. By choosing Collapse > Selected Buffers/Inverters on the pop-up menu, the chain is collapsed into a single “meta-buffer” that represents the functionality of the whole buffer chain, as shown in Figure 4-17.

Figure 4-16 Buffer Chain Selected

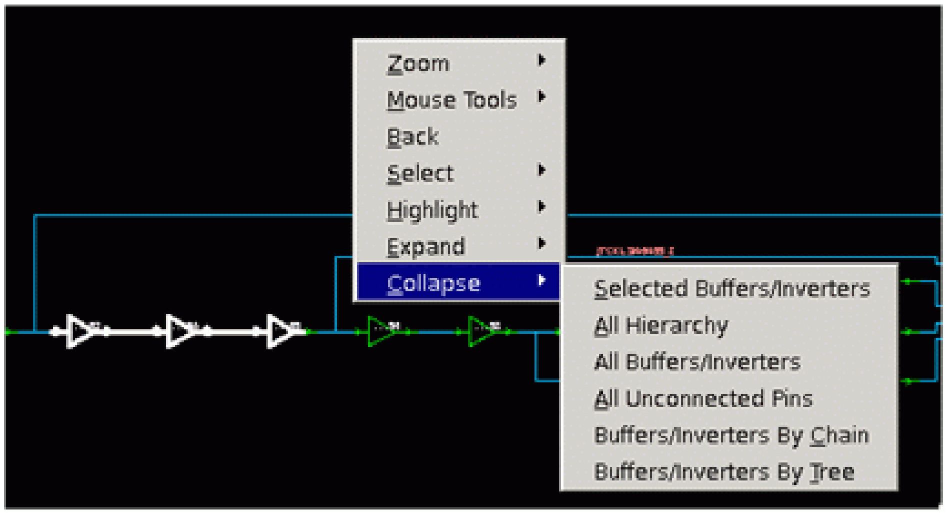
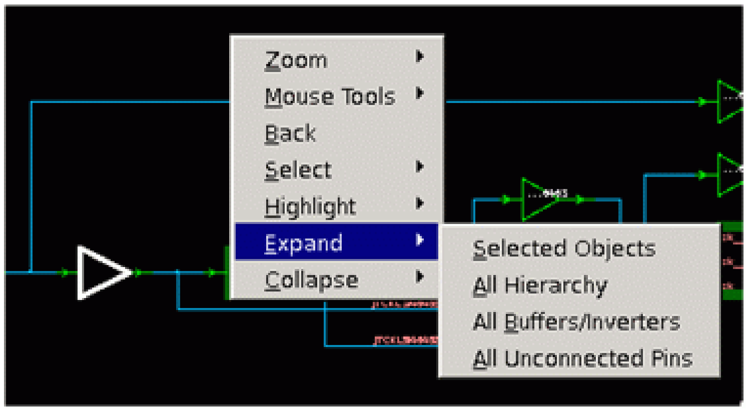


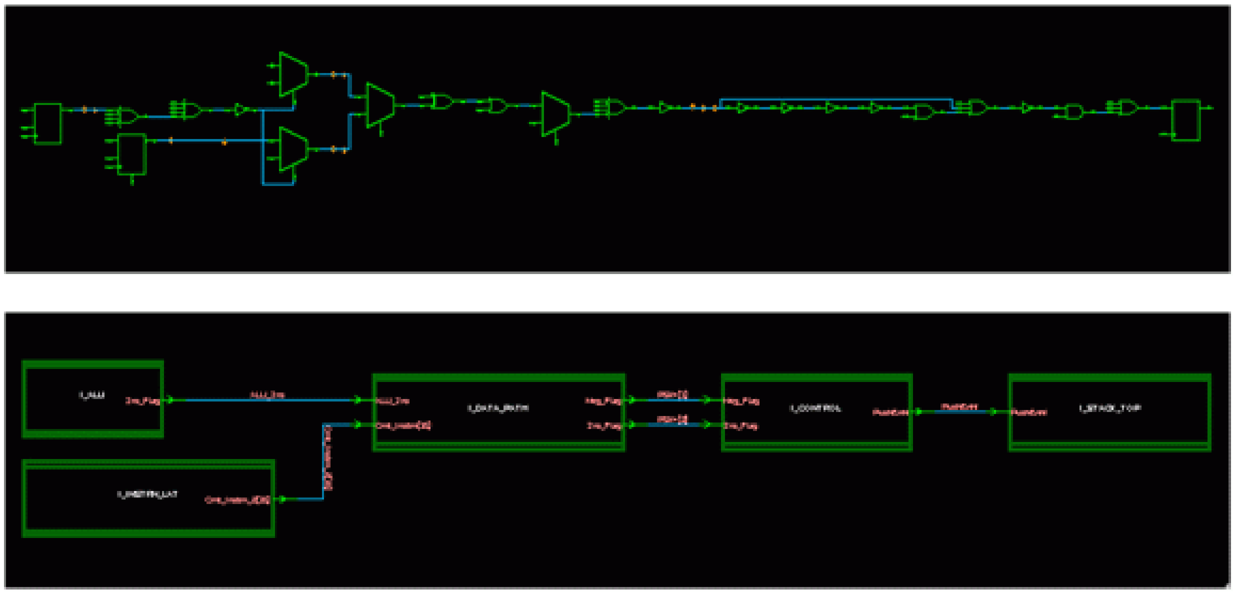
Figure 4-17 Buffer Chain Collapsed



The "meta-buffer" is drawn larger and with thicker lines to show that it is a compressed representation of the buffer chain, not an actual cell in the design. You can restore the view of the original buffer chain by choosing Expand > Selected Objects on the pop-up menu.

You can similarly collapse and expand the view of buffer trees, cells that belong to hierarchical blocks, and unconnected pins of large macro cells. The design functionality is often easier to understand with a collapsed view. Figure 4-18 shows an example of an expanded (flat) and a collapsed (hierarchical) view of a path schematic.

Figure 4-18 Expanded and Collapsed Views of a Design Hierarchy



The commands for collapsing and expanding objects are available on the pop-up menu and on the menu bar. On the menu bar, choose Schematic > Collapse or Schematic > Expand and choose the command. You can choose commands to collapse or expand the currently selected objects or all objects of a specified type, such as All Hierarchy, All Buffers/Inverters, Buffers/Inverters By Chain, or Buffers/Inverters By Tree.

5

Comparing Block- and Top-Level Constraints

Incorporating third-party IP blocks or reusing blocks from other designs is a common practice in chip development. Each of these predesigned blocks usually comes with a set of constraints for building the block. To guarantee that the block functions correctly, you must ensure that these block constraints are still valid after you incorporate the block in your bigger design.

To check the consistency between block- and top-level constraints, use the `compare_block_to_top` command to find inconsistencies between hierarchical constraints. To find potential issues within a single set of constraints, use the `analyze_design` command at the top or individual block level.

These features are described in the following sections:

- Creating Waivers
- Creating Text Reports

Overview of Hierarchical Constraint Comparison

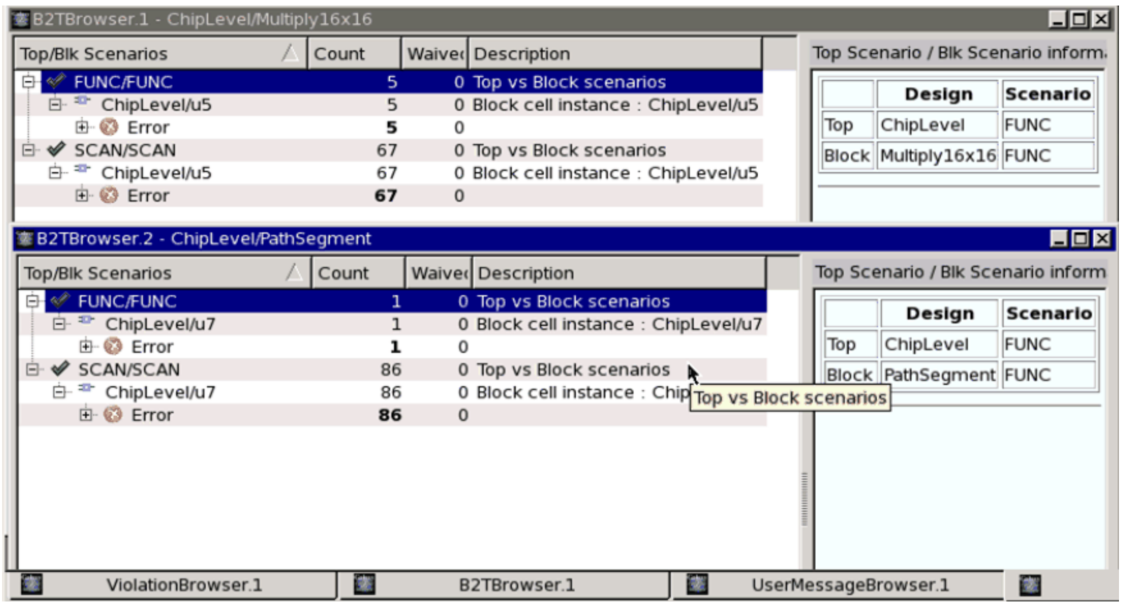
Design teams use various strategies to create block-level constraints from top-level constraints. Sometimes block-level constraints are propagated to the top level in a bottom-up design flow. During the design iteration process, new constraints can be manually inserted into both the top-level and block-level designs. To check whether the constraints match after such operations, use the `compare_block_to_top` command.

Each block comes with a set of constraints under which it was synthesized and placed. These constraints include clock definitions, timing exceptions, and boundary conditions.

When you use the `compare_block_to_top` command, the tool loads the constraints for both the block-level design and the top-level design and performs a comparison between the two sets of constraints. The constraint consistency-checking feature checks a unique netlist against the two sets of constraints. The tool generates a set of block-to-top rule violations that can be debugged just like the general design rules.

The tool preserves the block-to-top results for multiple blocks in a single session. Figure 5-1 shows violations for two blocks in two separate violation browser windows.

Figure 5-1 Violations for Two Blocks in Separate Violation Browser Windows


B2TBrowser.1 - ChipLevel/Multiply16x16

Top/Blk Scenarios	Count	Waived	Description
FUNC/FUNC	5	0	Top vs Block scenarios
ChipLevel/u5	5	0	Block cell instance : ChipLevel/u5
Error	5	0	
SCAN/SCAN	67	0	Top vs Block scenarios
ChipLevel/u5	67	0	Block cell instance : ChipLevel/u5
Error	67	0	

Top Scenario / Blk Scenario inform.

	Design	Scenario
Top	ChipLevel	FUNC
Block	Multiply16x16	FUNC

B2TBrowser.2 - ChipLevel/PathSegment

Top/Blk Scenarios	Count	Waived	Description
FUNC/FUNC	1	0	Top vs Block scenarios
ChipLevel/u7	1	0	Block cell instance : ChipLevel/u7
Error	1	0	
SCAN/SCAN	86	0	Top vs Block scenarios
ChipLevel/u7	86	0	Block cell instance : Chip
Error	86	0	

Top vs Block scenarios

Top Scenario / Blk Scenario inform

	Design	Scenario
Top	ChipLevel	FUNC
Block	PathSegment	FUNC

For more information about analyzing the results of multiple blocks in a single session, see “Reporting Rule Violations With the report_constraint_analysis Command” on page 2-25.

Methodology

The following sections describe the block-to-top flow:

- Keeping Multiple Linked Designs for Block-to-Top Constraint Checking
- Flow Script Example

Keeping Multiple Linked Designs for Block-to-Top Constraint Checking

When linking a design, the default behavior is to remove previously linked designs. However, the block-to-top flow is different from the standard flow because you need to keep two linked designs in memory:

- The block design with block-level constraints applied
- The top design with top-level constraints applied

To keep previously linked designs in memory, use the `link_design` command with the `-add` option. When there are multiple linked designs in memory at the same time, use the `current_design` command to switch between the designs.

Flow Script Example

The script in Example 5-1 shows the flow for the block-to-top constraint checking.

Example 5-1 Block-to-Top Constraint Checking Script

```
set_app_var search_path [list . ${TOP_DIR}/libs ${TOP_DIR}/design]
set_app_var link_path [list * libs.db]

read_verilog ${TOP_DIR}/design/ChipLevel_no2blocks.v
read_verilog ${TOP_DIR}/design/Multiply16x16.v
read_verilog ${TOP_DIR}/design/PathSegment.v

link_design ChipLevel
link_design -add Multiply16x16

current_design ChipLevel
source ${TOP_DIR}/dc_outdir/ChipLevel_propagate.sdc -echo
current_design Multiply16x16
source ${TOP_DIR}/design/Multiply16x16.sdc -echo

current_design ChipLevel
compare_block_to_top -block_design [get_designs Multiply16x16]
remove_linked_design [get_designs Multiply_16x16]
```

Checking Block and Top Constraints in the GUI

The tool provides comprehensive GUI windows for performing block-to-top comparisons and debugging block-to-top rule violations.

Built-In Block-to-Top Rules

Built-in block-to-top rules check top-level constraints against block-level constraints. When you run the `compare_block_to_top` command, the tool automatically loads the built-in block-to-top rules listed in Table 5-1.

Table 5-1 Built-In Block-to-Top Rules

Rule	Description
B2T_EXD_xxxx	Boundary conditions/external delays
B2T_CAS_xxxx	Case analysis Constraints/Exception analysis
B2T_EXC_xxxx	Timing exceptions Constraints/Exception analysis
B2T_CLK_xxxx	Clock properties
B2T_CLT_xxxx	Clock source latency and clock network latency
B2T_UNC_xxxx	Clock uncertainty
B2T_DIS_xxxx	Disabled objects and arcs
B2T_OPC_xxxx	Operating conditions

When appropriate, the rules that check these constraints have a tolerance rule property that allows some difference in the values being compared before a violation is issued. The tolerance is specified as a percentage of the smallest value compared.

By default, the tolerance is set at 1e-5 percent, which is equivalent to performing exact matches. To change the tolerance, use the `set_rule_property` command.

See the rule reference pages in the online Help for detailed rule descriptions.

Running Block-to-Top Analysis from the GUI

Typical steps to perform block-to-top analysis are:

1. Load the design.
2. Run the `analyze_design` command at the top level and at the block level and make sure that both the top-level and block-level constraints are clean.
3. Run the `compare_block_to_top` command specifying the block-level and top-level designs to be compared.
4. Open the Block-to-Top mismatch browser window: choose Analysis > B2T Mismatch Browser.
5. Debug the block-to-top rule violations.

   Note that it is not possible to waive block-to-top rule violations.

   The VBlock-to-Top mismatch browser window shows the design-level rule violations, if any.

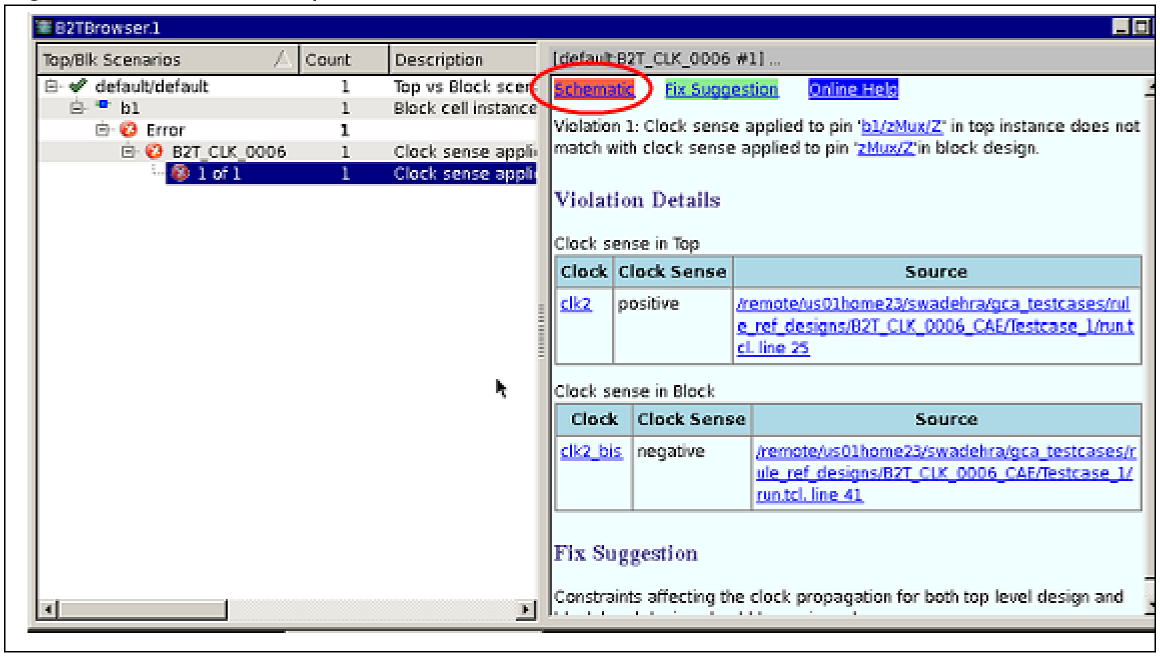
   Debug and correct the design to ensure that the block-to-top analysis results are correctly generated.

Displaying Schematics of Block and Top Rule Violations

You can display schematics of block and top-level constraint mismatches, as shown in Figure 5-1 and Figure 5-2. A schematic link is provided in the Block-to-Top (B2T) information pane, if the mismatch is schematic related.

Figure 5-1 shows a B2T_CLK_0006 violation. When you select this violation in the Block-to-Top mismatch browser, the information pane shows violation details that include a Schematic link (circled in red). When you click the Schematic link, the block and top-level schematics are displayed, as shown in Figure 5-2. The white-colored MUX outputs indicate the first place where the violation occurs.

Figure 5-2 Block-to-Top Information Pane With Schematic Link



B2TBrowser.1

Top/Blk Scenarios	Count	Description
default/default	1	Top vs Block scen
b1	1	Block cell instance
Error	1	
B2T_CLK_0006	1	Clock sense appli
1 of 1	1	Clock sense appli

[default:B2T_CLK_0006 #1] ...

Schematic Fix Suggestion Online Help

Violation 1: Clock sense applied to pin 'b1/zMux/Z' in top instance does not match with clock sense applied to pin 'zMux/Z' in block design.

Violation Details

Clock sense in Top

Clock	Clock Sense	Source
clk2	positive	/remote/us01home23/swadehra/gca_testcases/rule_ref_designs/B2T_CLK_0006_CAE/Testcase_1/run.tcl, line 25

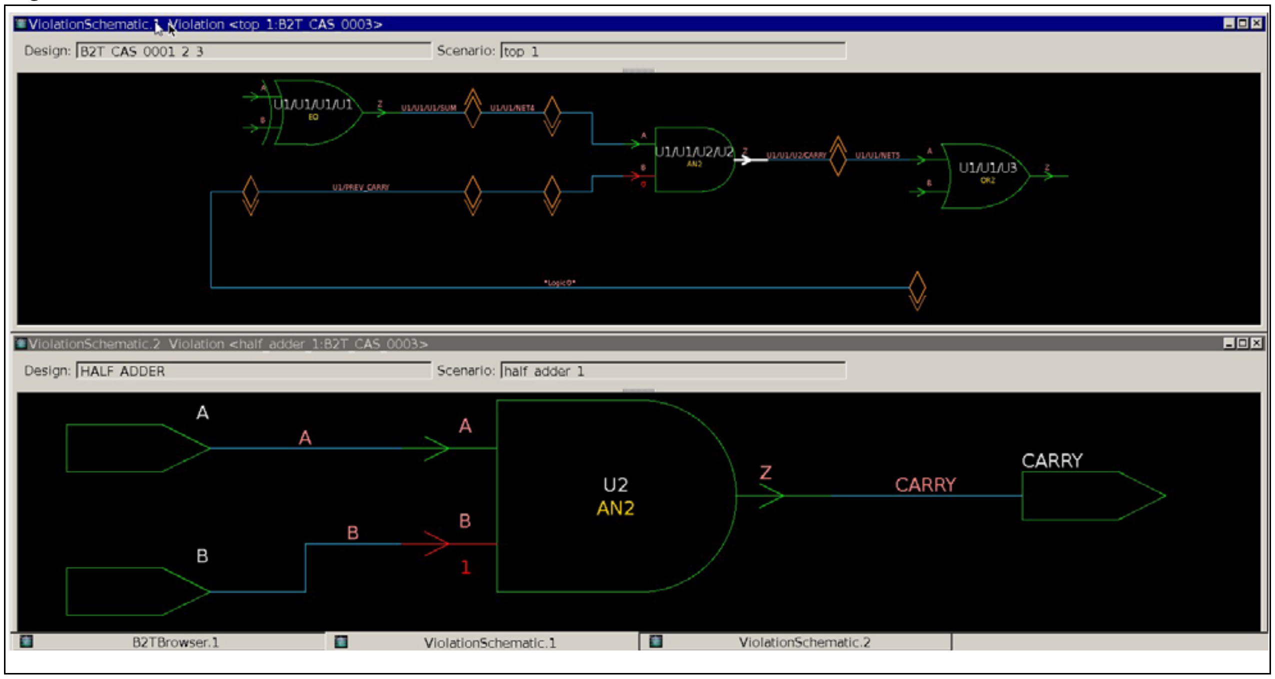
Clock sense in Block

Clock	Clock Sense	Source
clk2_bis	negative	/remote/us01home23/swadehra/gca_testcases/rule_ref_designs/B2T_CLK_0006_CAE/Testcase_1/run.tcl, line 41

Fix Suggestion

Constraints affecting the clock propagation for both top level design and

Figure 5-3 Dual Schematic



ViolationSchematic.1 Violation <top_1:B2T_CAS_0003>

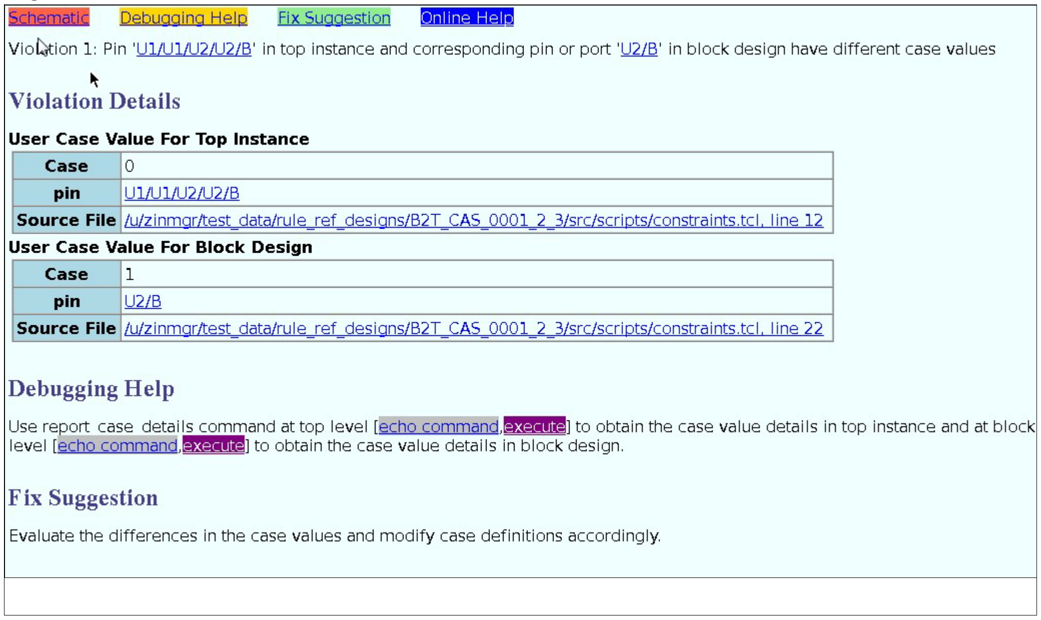
Design: B2T_CAS_0001_2_3 Scenario: top_1

ViolationSchematic.2 Violation <half_adder_1:B2T_CAS_0003>

Design: HALF_ADDER Scenario: half_adder_1

In addition to showing dual schematic views of the violation in Figure 5-3, the information pane in Figure 5-4 contains links to the SDC source files associated with both the block and top designs. When you click the source file link for either the block or top SDC file, the tool displays both top and block SDC source files side-by-side with the violating constraint highlighted in each file, as shown in Figure 5-5.

Figure 5-4 Information Pane When a Violation Is Selected



Violation 1: Pin 'U1/U1/U2/U2/B' in top instance and corresponding pin or port 'U2/B' in block design have different case values

Violation Details

User Case Value For Top Instance

Case	0
pin	U1/U1/U2/U2/B
Source File	/u/zinmgr/test_data/rule_ref_designs/B2T_CAS_0001_2_3/src/scripts/constraints.tcl, line 12

User Case Value For Block Design

Case	1
pin	U2/B
Source File	/u/zinmgr/test_data/rule_ref_designs/B2T_CAS_0001_2_3/src/scripts/constraints.tcl, line 22

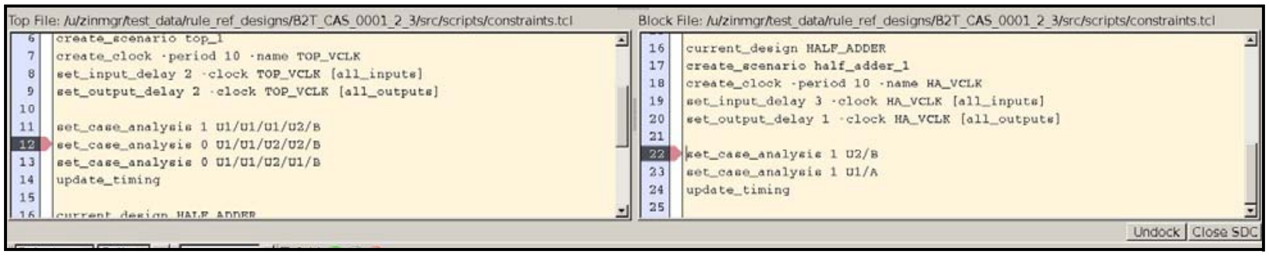
Debugging Help

Use report_case_details command at top level [echo command,execute] to obtain the case value details in top instance and at block level [echo command,execute] to obtain the case value details in block design.

Fix Suggestion

Evaluate the differences in the case values and modify case definitions accordingly.

Figure 5-5 Dual SDC Viewer



Top File: /u/zinmgr/test_data/rule_ref_designs/B2T_CAS_0001_2_3/src/scripts/constraints.tcl

```
 6  create_scenario top_1
 7  create_clock -period 10 -name TOP_VCLK
 8  set_input_delay 2 -clock TOP_VCLK [all_inputs]
 9  set_output_delay 2 -clock TOP_VCLK [all_outputs]
10
11  set_case_analysis 1 U1/U1/U1/U2/B
12  set_case_analysis 0 U1/U1/U2/U2/B
13  set_case_analysis 0 U1/U1/U2/U1/B
14  update_timing
15
16  current_design HALF_ADDER
```

Block File: /u/zinmgr/test_data/rule_ref_designs/B2T_CAS_0001_2_3/src/scripts/constraints.tcl

```
16  current_design HALF_ADDER
17  create_scenario half_adder_1
18  create_clock -period 10 -name HA_VCLK
19  set_input_delay 3 -clock HA_VCLK [all_inputs]
20  set_output_delay 1 -clock HA_VCLK [all_outputs]
21
22  set_case_analysis 1 U2/B
23  set_case_analysis 1 U1/A
24  update_timing
25
```

Constraint Comparison Example

The script in Example 5-2 uses the `compare_block_to_top` command to run block versus top consistency checks. The example loads the top-level netlist that includes the netlist for the block, links the top design, applies the top-level constraints, links the block-level design, applies block-level constraints, and sets the output directory. Finally, it runs block versus top consistency checks.

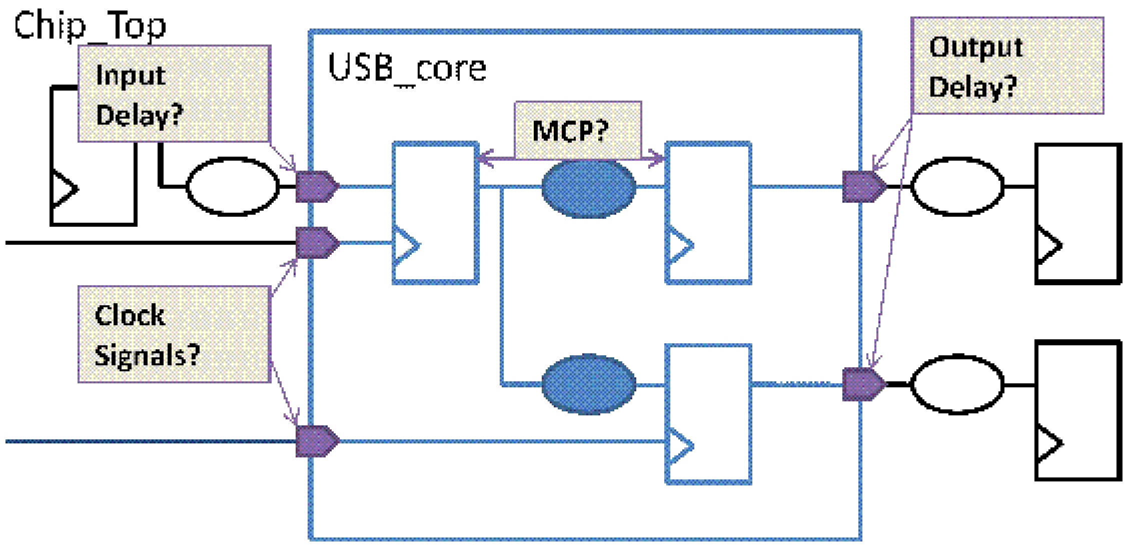
Example 5-2 Default Report for report_exceptions

```
gca_shell> read_verilog ./chip.v
gca_shell> link_design chip
gca_shell> source chip_constraints.tcl
gca_shell> link_design -add usb_core
gca_shell> source usb_core_constraints.tcl
gca_shell> set out_dir /user/abc/public_html/compare_chip
gca_shell> compare_block_to_top -block_design usb_core
```

Checking Block Versus Top Consistency

The `compare_block_to_top` command performs consistency checks between block-level design constraints and the corresponding top-level constraints. Figure 5-6 shows the block design USB_core inside the design Chip_Top and identifies some objects that can be compared.

Figure 5-6 Block Design USB_core Inside Design Chip_Top



The PrimeTime GCA tool looks for the following categories of constraints:

- Clock constraints
- Boundary condition constraints
- Exception type constraints
- Case value constraints
- Miscellaneous constraints

By default, all five categories are checked by the `compare_block_to_top` command. To restrict the categories, use the `-include` option.

The following sections describe the five categories checked by the PrimeTime GCA tool:

- Clock Checks
- Boundary Condition Checks
- Exception Checks
- Case Settings Checks
- Miscellaneous Settings Checks

Clock Checks

The clocks defined at the block level are compared to the clocks from the top-level reaching this block. The PrimeTime GCA tool matches the clocks for period, sense, waveform, and phase.

Tcl-based mapping is available as an alternative to the clock-check matching described previously.

Boundary Condition Checks

The input and output delays of the block are checked against the top level constraints. The actual input and output delay values are not subject to comparison. However, the following items are part of the comparison:

- Presence or absence of input or output delays
- Clock used to establish the input or output delays
- Triggering edge of the clock
- Edge-sensitive versus level-sensitive

Exception Checks

When running block-to-top consistency checks, the tool compares the following timing conditions:

- `set_min_delay` and `set_max_delay` exceptions
- `set_multicycle_path` exceptions
- `set_clock_groups` clock grouping
- `set_false_path` exceptions

Case Settings Checks

The tool compares case settings by looking at

- Case and constant values in the top design at the boundary of the block
- Case settings at the ports of the standalone block

If any case settings in the block design are not covered by the settings in the top design, then those are reported as missing block constraints in the report.

Miscellaneous Settings Checks

Miscellaneous checks compare the top- and block-level settings related to the `set_disable_timing` command, clock gating, clock sense, and operating conditions.

Checking Multiple Scenarios

You can compare multiple scenarios simultaneously. Use the `-block_scenarios` and `-top_scenarios` options to specify the pairs of scenarios to be compared. The first scenario passed in `-top_scenarios` is matched against the first scenario passed in `-block_scenarios`. Example 5-3 shows such a case.

Example 5-3 Comparing Multiple Scenarios

```
compare_block_to_top -block_design Multiply16x16 \
  -top_scenarios [list FUNC SCAN] \
  -block_scenarios [list BFUNC BTEST] \
```

In Example 5-3, the following scenarios are compared against each other:

- Top: FUNC; Block: BFUNC
- Top: SCAN; Block: BTEST

Checking Multiple Instances of One Block

When several instances of one block exist in the design, the PrimeTime GCA tool compares all of them to the top-level constraints at the same time. This implies, however, that all instances of the block were created under the same set of constraints. If this is not the case, run the `compare_block_to_top` command several times: one execution of `compare_block_to_top` for each combination of unique constraints per block.

To achieve this behavior, the constraint consistency-checking feature can also work on instances, as opposed to designs. Example 5-4 demonstrates this approach.

Example 5-4 Using the compare_block_to_top Command

```
compare_block_to_top -block_design Multiply16x16 \
  -top_scenarios [list FUNC SCAN] \
  -block_scenarios [list BFUNC BTEST] \
  -cells [get_cells [list TOP/Block1 TOP/Block2]]
```

Creating Waivers

You can create waivers for block-to-top rules through the GUI or the command line.

For more information about creating, editing, removing, and reporting waivers, see “Suppressing Violations” on page 2-11.

Creating Text Reports

In addition to GUI reports, you can create text reports using the `report_constraint_analysis` command.

For more information, see “Reporting Rule Violations With the report_constraint_analysis Command” on page 2-25.

6

Comparing Two Sets of Design Constraints

During design development, some modifications might be made to the original constraints file. These modifications could cause mismatches between the original constraints file and the modified constraints file. You can use the PrimeTime GCA tool to compare these two sets of design constraints using one common netlist, so you can determine the functional differences between the two sets of constraints.

Use the `compare_constraints` command to run the comparison. The following sections describe the `compare_constraints` command:

- Constraint Set Comparison Overview
- Methodology
- SDC-to-SDC Violation Browser
- Managing Rules and Violations When Comparing Constraint Sets
- Creating Waivers
- Text Reports

Constraint Set Comparison Overview

To compare constraint sets, define them as scenarios and then use the `compare_constraints` command to compare the scenarios. The scenarios must be created on the current design, which can be specified by the `current_design` command.

The `compare_constraints` command checks the following types of constraints:

- Clock definitions

  `create_clock`, `create_generated_clock`
- External delays

  `set_input_delay`, `set_output_delay`
- Timing exceptions

  `set_false_path`, `set_multicycle_path`, `set_min_delay`, `set_max_delay`, `set_clock_groups`
- Case analysis

  `set_case_analysis`
- Disable timing

  `set_disable_timing`
- Uncertainty

  `set_clock_uncertainty`
- Transitions

  `set_clock_transition`, `set_input_transition`
- Latency

  `set_clock_latency`
- Clock gating

  `set_clock_gating_check`
- Clock sense

  `set_sense`
- Loading

  `set_load`

When appropriate, the rules that check these constraints have a tolerance property that allows some difference in the values being compared before a violation is issued. The tolerance is specified as a percentage of the smallest value compared.

By default, the tolerance is set at 1e-5% which is equivalent to performing exact matches. To change the tolerance, use the `set_rule_property` command.

Methodology

Use the following methodology to compare constraint sets:

1. Use the `create_scenario` command to model your original constraint set and your modified constraint set.

   Your constraints can be in SDC files or in multiple Tcl scripts.

2. Use the `compare_constraints` command to compare the scenarios. For example,

   ```
   gca_shell> compare_constraints -constraints1 \
             my_original_constraint_scenarios  \
              -constraints2 my_modified_constraint_scenarios
   ```

   To match clock sources, include the `-match_clock_sources` option with the `compare_constraints` command.

3. Use the SDC-to-SDC browser to identify violations. Use the `report_constraint_analysis` command to obtain constraint comparison text reports.

4. Fix the violations in the following order:

   a. Clock constraint violations

   b. `set_disable_timing` violations

   c. `set_case_analysis` violations

   d. Exception violations

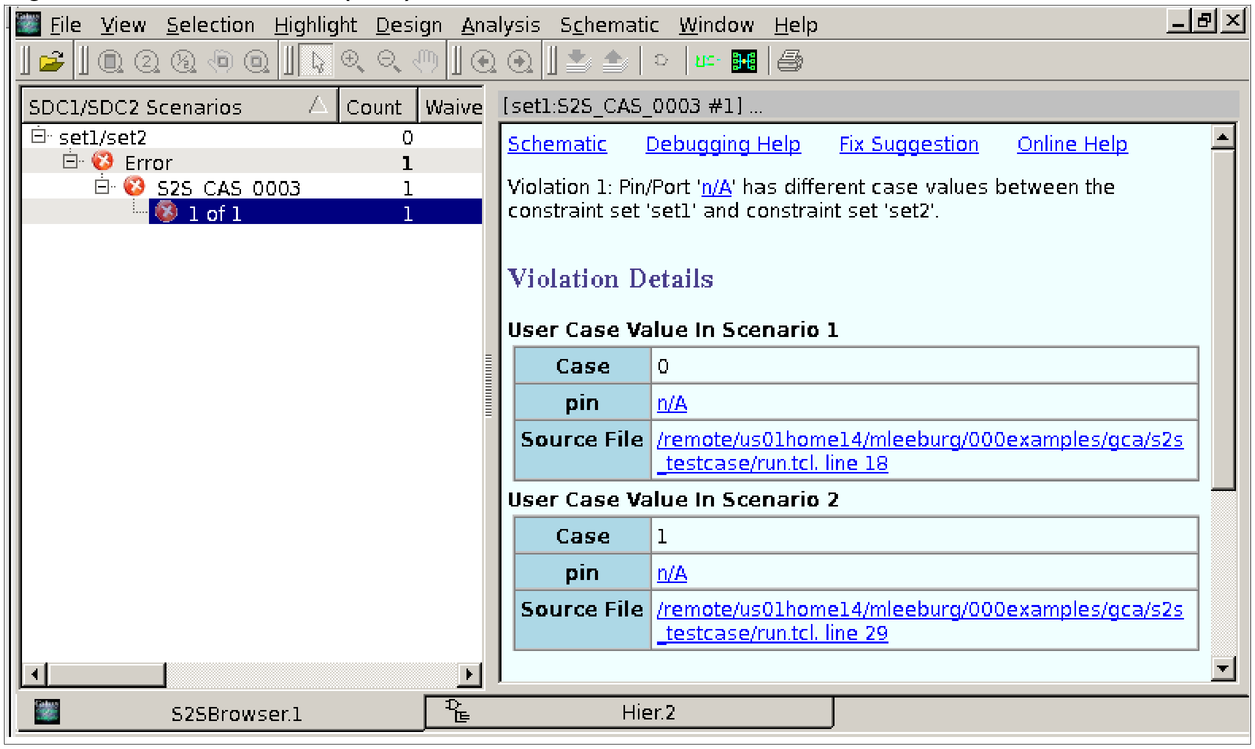
5. Repeat the `compare_constraints` command as needed.

SDC-to-SDC Violation Browser

After you compare your original constraints to your modified constraints, use the SDC-to-SDC (S2S) browser, shown in Figure 6-1, to identify and fix constraint mismatches.

The SDC-to-SDC violation browser is similar to the general violation browser used with the `analyze_design` command. All links in the information view pane function similarly to the links in the Block-to-Top (B2T) violation browser and the general violation browser. If schematics are appropriate for the violation, the information view pane contains a schematic link.

Figure 6-1   SDC-to-SDC (S2S) Violation Browser



Violation 1: Pin/Port 'n/A' has different case values between the constraint set 'set1' and constraint set 'set2'.

Violation Details

User Case Value In Scenario 1

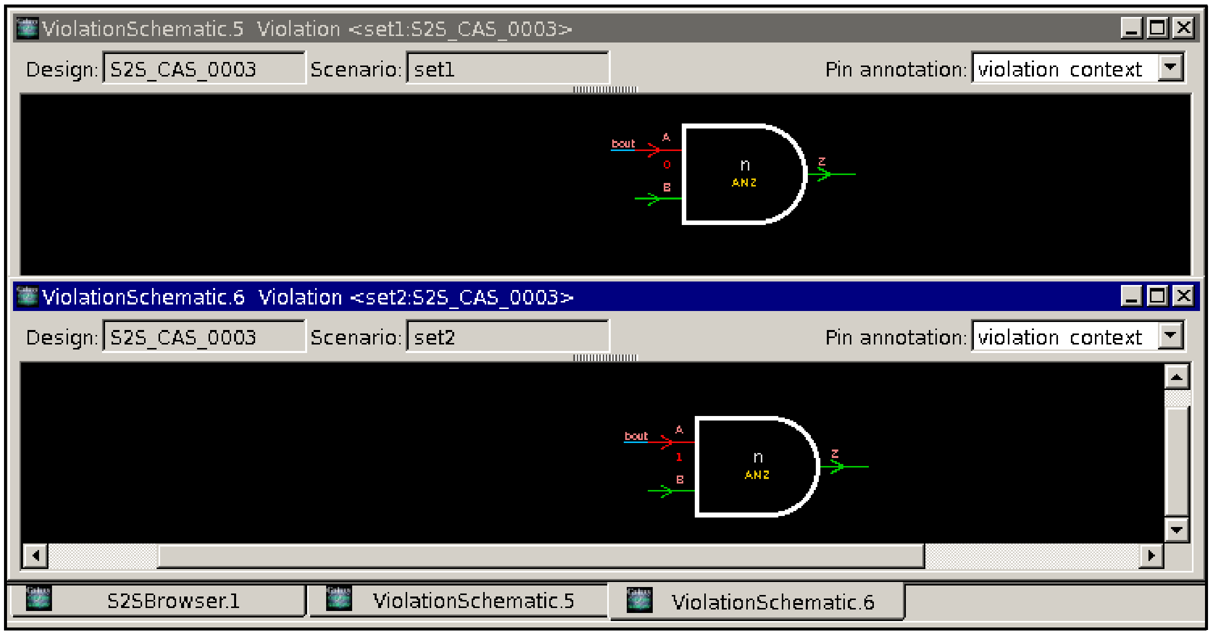
Case	0
pin	n/A
Source File	/remote/us01home14/mleeburg/000examples/gca/s2s_testcase/run.tcl, line 18

User Case Value In Scenario 2

Case	1
pin	n/A
Source File	/remote/us01home14/mleeburg/000examples/gca/s2s_testcase/run.tcl, line 29

When working with SDC-to-SDC constraint violations, the tool provides schematics for both sets of constraints and can display them side by side, as shown in Figure 6-2. Click the Schematic link to display the schematics. The schematics are annotated with data to help debug the constraint problem.

Figure 6-2 SDC-to-SDC (S2S) Schematic Display

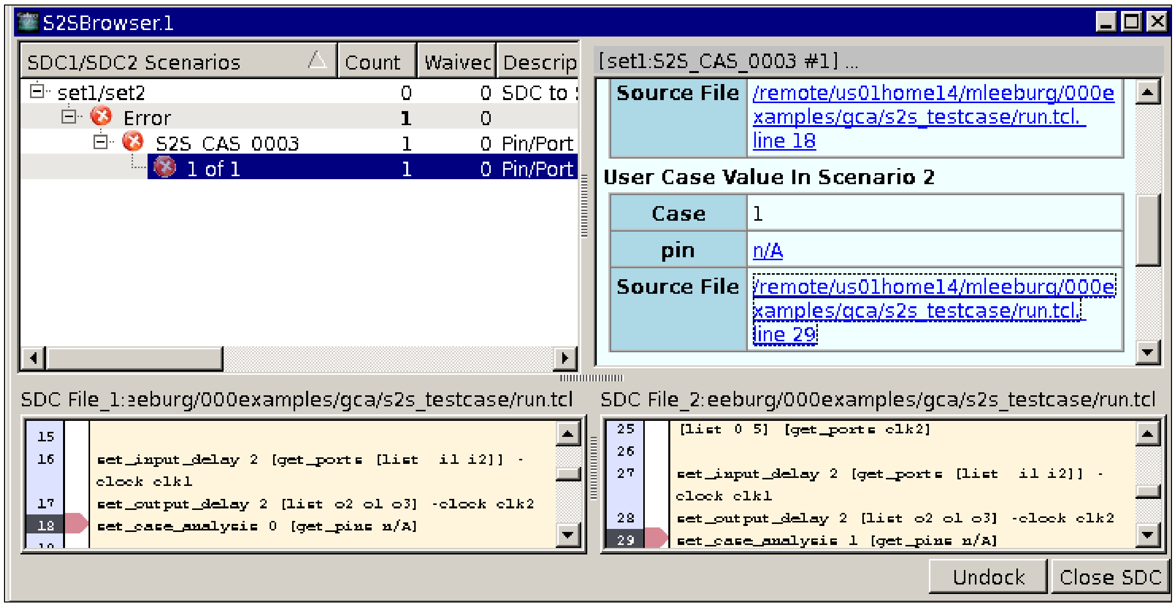

ViolationSchematic.5 Violation <set1:S2S_CAS_0003>
Design: S2S_CAS_0003
Scenario: set1
Pin annotation: violation context
bout
A
0
B
n
AN2
Z
ViolationSchematic.6 Violation <set2:S2S_CAS_0003>
Design: S2S_CAS_0003
Scenario: set2
Pin annotation: violation context
bout
A
1
B
n
AN2
Z
S2SBrowser.1
ViolationSchematic.5
ViolationSchematic.6

The tool can also display the constraints that caused the violation from each respective SDC file, as shown in Figure 6-3.

The SDC links are located in the Violations Details section of the information pane. Select the source file link for scenario 1 and scenario 2 to see the relevant constraints. You are not limited to displaying only schematics or SDC information; the tool can display both schematics and SDC file contents simultaneously.

To debug SDC-to-SDC violations, analyze the schematics and the constraint definitions in the SDC files. Follow the guidelines in the “Debugging Help” section in the information pane and use the fix suggestion described in the information pane. Additional resources, such as man pages and user guides are available in the online Help.

Figure 6-3 SDC-to-SDC (S2S) Scenarios


S2SBrowser.1

SDC1/SDC2 Scenarios	Count	Waived	Descrip
set1/set2	0	0	SDC to
Error	1	0	
S2S_CAS_0003	1	0	Pin/Port
1 of 1	1	0	Pin/Port

[set1:S2S_CAS_0003 #1] ...

Source File	/remote/us01home14/mleeburg/000examples/gca/s2s_testcase/run.tcl, line 18

User Case Value In Scenario 2

Case	1
pin	n/A
Source File	/remote/us01home14/mleeburg/000examples/gca/s2s_testcase/run.tcl, line 29

SDC File_1:eeburg/000examples/gca/s2s_testcase/run.tcl

```
set_input_delay 2 [get_ports [list  i1 i2]] -
clock clk1
set_output_delay 2 [list o2 o1 o3] -clock clk2
set_case_analysis 0 [get_pins n/A]
```

SDC File_2:eeburg/000examples/gca/s2s_testcase/run.tcl

```
[list 0 5] [get_ports clk2]

set_input_delay 2 [get_ports [list  i1 i2]] -
clock clk1
set_output_delay 2 [list o2 o1 o3] -clock clk2
set_case_analysis 1 [get_pins n/A]
```

Managing Rules and Violations When Comparing Constraint Sets

You can compare constraints that affect datapath propagation but you cannot compare constraints that affect clock network propagation. You cannot compare a subset of constraint types. However, constraint comparison violations can be disabled. When disabled, violations do not appear in the GUI or the text reports.

Creating Waivers

You create waivers for SDC-to-SDC rules through the GUI or command line. For more information about creating, editing, removing, and reporting waivers, see “Suppressing Violations” on page 2-11.

Text Reports

Use the `report_constraint_analysis` command to obtain constraint comparison text reports, as shown in Example 6-1. Use the SDC-to-SDC violation browser to view violations interactively. You can view where the mismatch occurs in the schematics and where the mismatch occurs in the SDC constraints file. See “Suppressing Violations” on page 2-11 for more information.

Example 6-1 Report Generated by the report_constraint_analysis Command for the compare_constraints Constraint

```
gca_shell> report_constraint_analysis
****************************************
Report : report_constraint_analysis
        -include {violations user_messages rule_info}
        -style {full}
Version: F-2011.06-Beta3
Date   : Mon Apr 25 21:34:34 2011
****************************************

Constraint Comparison Violations
                          Count Waived Description
------------------------------------------------------------------------------
set1/set2                    1    0
    error                    1    0
      S2S_CAS_0003           1    0    Pin/Port 'pin_port' has different case values
between the constraint set 'scenario1' and constraint set 'scenario2'.
        1 of 1                    0    Pin/Port 'n/A' has different case values
between the constraint set ‘set1' and constraint set 'set2'.
------------------------------------------------------------------------------
Total Error Messages         1    0
Total Warning Messages       0    0
Total Info Messages          0    0


User Messages

MessageID            Count   Description
------------------------------------------------------------------------------
Information          5
  ADES-016           1       waiver '%s' removed.
    1 of 1           1       waiver 'waiver_0' removed.
  CMP-006            3       Reading %s configuration file: %s
    1 of 3           1       Reading PrimeTime configuration file: /remote/srm489/
image/GCA/nightly/zinmgin_dev/110423.1/Testing/auxx/gca/tcl/primetime.pcx
    2 of 3           1       Reading IC Compiler configuration file: /remote/srm489/
image/GCA/nightly/zin_zin_dev/110423.1/Testing/auxx/gca/tcl/icc.pcx
    3 of 3           1       Reading Design Compiler configuration file: /remote/
srm489/image/GCA/nightlynmgr_zin_dev/110423.1/Testing/auxx/gca/tcl/syn.pcx
  LNK-040            1       %sunits loaded from library '%s'
    1 of 1           1       units loaded from library 'lsi_10k'
Rule Information

Rule          Severity  Status    Message
...
```

A

Tutorial

The PrimeTime GCA tool can help you identify problems with your constraints by checking them against a predefined set of rules. By using the debugging capabilities, you can investigate and resolve constraint problems.

This tutorial assumes that you are familiar with key components of the PrimeTime GCA tool, which are described previously in this user guide.

This appendix shows you how to set up and run the PrimeTime GCA tool on the tutorial design and investigate the rule violations in the design.

This appendix contains the following sections:

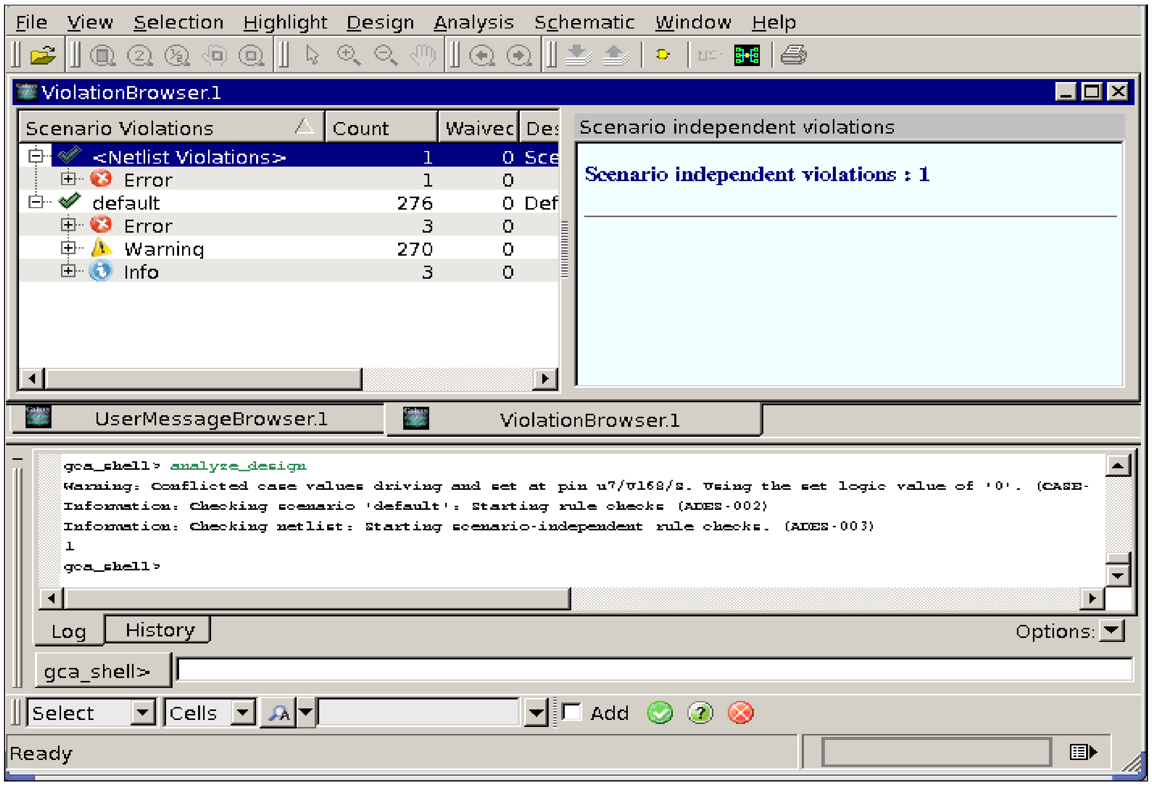
Overview of the PrimeTime GCA Tool

The violation browser, shown in Figure A-1, is a key feature of the PrimeTime GCA tool. The violation browser displays a summary of the rule violations found by the `analyze_design` command. Each scenario has summary of Error, Warning, and Info rule violations. View the specific rule violations in one of the categories by double-clicking it or by clicking the "+" icon for that row. Double-clicking on a specific rule expands the view in the violation Browser to list all violations of that rule for that scenario.

The violation browser contains an area referred to as the information pane. This section displays information specific to the item selected in the violation browser. If a rule is selected in the violation browser, then more information is displayed about that rule in the information pane. If a violation for a rule is selected, the information pane displays specific information about that violation along with links to help you investigate the violation.

The console contains a command line for entering commands interactively during a session. It also displays the log view, which provides a transcript of the session. You can use the scroll bar to the right of the window to view previous commands and their output.

Figure A-1 PrimeTime GCA GUI Window



Scenario Violations	Count	Waived	Des
<Netlist Violations>	1	0	Sce
Error	1	0	
default	276	0	Def
Error	3	0	
Warning	270	0	
Info	3	0	

Scenario independent violations

Scenario independent violations : 1

```
gca_shell> analyze_design
Warning: Conflicted case values driving and set at pin u7/U168/S. Using the set logic value of '0'. (CASE-
Information: Checking scenario 'default': Starting rule checks (ADES-002)
Information: Checking netlist: Starting scenario-independent rule checks. (ADES-003)
1
gca_shell>
```

Use tabs to switch between different views that are available in some of the GUI components. The console displays the log view by default, but if you click the History tab, you can see a list of the commands you used in the session. If you click a Schematic link in the information pane, a schematic view replaces the violation browser. To revert to the previous view, click the violation browser tab. You can use all of these items in the GUI as you begin debugging constraint problems in the tutorial design. If you need to exit the tutorial, see "Ending and Restarting the Tutorial Session" on page A-26.

When you run the `analyze_design` command, the design loaded in the tool is checked against a set of predefined rules. These rules check clock definitions, case analysis propagation, timing exceptions, boundary conditions, and other general checks. Table A-1 provides an overview of the different types of rule definitions.

Table A-1 Rule Types

Design Analysis Intent	**Rule**	**Rule Analysis**
Boundary Conditions	CAP_xxxx rules	Capacitance values
	DRV_xxxx rules	Drive constraints
	EXD_xxxx rules	External delays
Constraints/Exceptions Analysis	CAS_xxxx rules	Case analysis
	EXC_xxxx rules	Timing exceptions
Clocks	CGR_xxxx rules	Clock groups
	CLK_xxxx rules	Clock properties
	CNL_xxxx rules	Network latencies
	CSL_xxxx rules	Clock source latencies
	CTR_xxxx rules	Clock transition
	PRF_xxxx rules	Clock count
	UNC_xxxx rules	Clock uncertainties
General	DES_xxxx rules	Design constraints
	LOOP_xxxx rules	Timing loops
	NTL_xxxx rules	Net properties
	UNT_xxxx rules	Library units

Table A-1 Rule Types (Continued)

Design Analysis Intent	Rule	Rule Analysis
Block-To-Top	B2T_xxx_xxxx rules	Top-level constraints vs. block-level constraints

Each rule defines a specific check that is performed on the design and its constraints. To see a short description of each case analysis rule, enter the following command:

```
gca_shell> report_rule CAS_*
```

The command returns a report like the one in Example A-1.

Example A-1 report_rule Output for CAS_* Rules

```
****************************************
Report : rule
...
****************************************
Rule           Severity  Status    Message
-------------------------------------------------------------------------
CAS_0001       error     enabled   Net 'net' has conflicting user case
                                   analysis values on load pins.
CAS_0002       error     enabled   Pin/Port 'pin' is driven by conflicting
                                   values. A value of zero is used.
CAS_0003       error     enabled   Pin/Port 'pin' propagated value
                                   conflicts with a user case analysis
                                   value.
CAS_0004       error     enabled   Bidirectional pin 'pin' has conflicting
                                   values on the load and driver side.
```

For more details, see the rules in the online Help. To launch the online Help, choose Help > Online Help from the menu in the GUI.

About the Tutorial Design

The tutorial uses a design called “ChipLevel” that implements several binary arithmetic functions such as addition and multiplication.

The ChipLevel design is hierarchical and consists of the following major blocks:

- Adder16
- CascadeMod
- Comparator
- Multiply8x8
- Multiply16x16

- MuxMod
- PathSegment

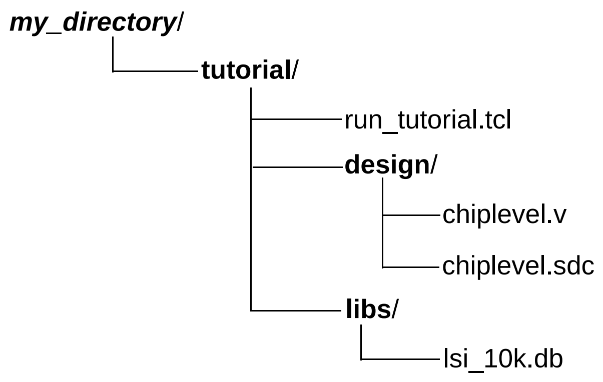
This pre-layout design is in the early development phase, with the clock network and constraints still under construction. You can use the tool to identify constraint problems early in the design cycle.

The tool must be installed before you run the tutorial. The tutorial directories and files are located under the `doc/gca/tutorial` directory of the PrimeTime installation. Copy this directory to a working directory by entering the following UNIX command:

```
% cp -r $SYNOPSYS/doc/gca/tutorial .
```

This command creates a new directory called “tutorial” under the current directory. If you copied the tutorial design files from the PrimeTime installation, you have the directory structure shown in Figure A-2 in your working directory.

Figure A-2 Tutorial Directory Structure



Run the tutorial from the tutorial directory using the run_tutorial.tcl script. The subdirectories are:

- design.

  This directory contains the Verilog netlist (chiplevel.v) and design constraints (chiplevel.sdc).

- libs.

  This directory contains the lsi_10k.db technology library.

Starting the PrimeTime GCA Tool

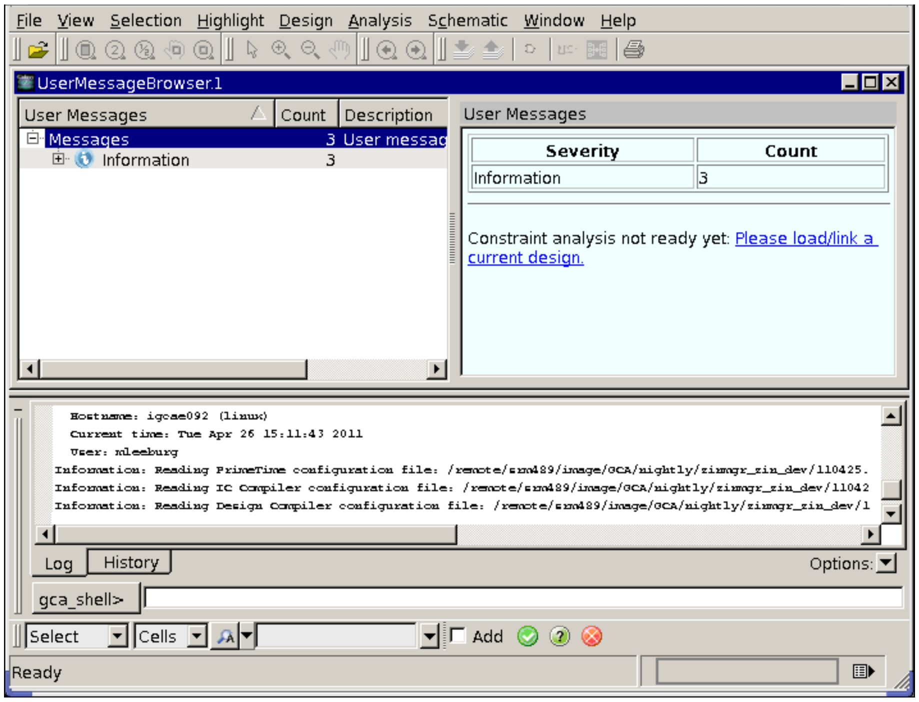
Start the PrimeTime GCA tool from the tutorial directory. To start the tool:

1. Open a UNIX shell.
2. Change to the ***user_directory*/tutorial** directory.
3. At the UNIX prompt, enter:

```
% gca_shell
```

The tool opens displaying the GUI as shown in Figure A-3. Files can be sourced and commands can be entered at the `gca_shell` prompt. All commands and their output are saved to the gca_shell_command.log file.

Figure A-3 PrimeTime GCA GUI



Severity	Count
Information	3

Constraint analysis not ready yet: Please load/link a current design.

```
  Hostname: igcae092 (linux)
  Current time: Tue Apr 26 15:11:43 2011
  User: mleeburg
Information: Reading PrimeTime configuration file: /remote/srm489/image/GCA/nightly/zimgr_zin_dev/110425.
Information: Reading IC Compiler configuration file: /remote/srm489/image/GCA/nightly/zimgr_zin_dev/11042
Information: Reading Design Compiler configuration file: /remote/srm489/image/GCA/nightly/zimgr_zin_dev/1
```

If the PrimeTime GCA tool fails to start, check the following:

- Was the PrimeTime GCA tool correctly installed?
- Is the PrimeTime installation path included in your path definition?

- Is the Synopsys license server running?
- Is your Synopsys license key file available and current, and does it include a PrimeTime and PrimeTime GCA license? The PrimeTime GCA tool requires that you have a PrimeTime license available, although it is not checked out.

If you need assistance, ask your system administrator or consult the installation and licensing documentation.

Analyzing the ChipLevel Design

The PrimeTime GCA tool requires the design libraries, the gate-level Verilog netlists of the design, and the files containing the constraints for the design. You must read in the design, link it to the technology libraries, and load the constraints before the tool can analyze the constraints.

Follow the steps below to analyze the constraints in the ChipLevel design.

1. Set the `search_path` variable:

   ```
   gca_shell> set_app_var search_path [list . ./libs ./design ] \
   . ./libs ./design
   ```

   The `search_path` variable specifies locations to search for design and library files.

2. Set the `link_path` variable:

   ```
   gca_shell> set_app_var link_path [list * lsi_10k.db] * lsi_10k.db
   ```

   Instead of the Tcl `set` command, the `set_app_var` command is recommended for assigning values to application shell variables because it verifies that the variable is an application variable and that it is a legal value for that variable in the application.

   The `link_path` variable specifies a list of libraries, design files, and library files to use for resolving references during linking. The asterisk (*) indicates that the tool should use the designs and libraries that have already been read into memory for design linking.

3. Read in the Verilog netlist:

   ```
   gca_shell> read_verilog chiplevel.v
   Loading verilog file '/remote/techp5/grayc/gca/tutorial/design \
   chiplevel.v'
   1
   ```

   The `read_verilog` command reads in the gate-level Verilog netlist for the design.

4. Set the current design to the top-level design:

   ```
   gca_shell> current_design ChipLevel
   {"ChipLevel"}
   ```

   The `current_design` command specifies the working design for the session.

5. Link the design:

   ```
   gca_shell> link_design
   Loading db file '/remote/techp5/grayc/gca/tutorial/libs/lsi_10k.db' \
   Information: units loaded from library 'lsi_10k' (LNK-040) \
   Removing previously linked designs ... \
   Design 'ChipLevel' was successfully linked.
   1
   ```

   The `link_design` command resolves the references in the Verilog netlist to the library components and builds an internal representation of the design for analysis.

6. Read the constraints file:

   ```
   gca_shell> source ./design/chiplevel.sdc
   Information: Setting sdc_version outside of an SDC file has no
   effect (SDC-1)
   Warning: set_max_delay is forcing pin 'clk_8x8/Z' to be a timing
   startpoint. (UIC-011) \
   Warning: Object 'u5/res_reg[0]/Q' is not a valid endpoint. The
   set_multicycle_path command will not match this object. (UIC-009)
   Warning: Object 'u5/res_reg[1]/Q' is not a valid endpoint. The
   set_multicycle_path command will not match this object. (UIC-009)
   ...
   Warning: Object 'u5/res_reg[31]/Q' is not a valid endpoint. The
   set_multicycle_path command will not match this object. (UIC-009)
   1
   ```

   The `source` command reads in the constraint file for the design. The `read_sdc` command could also be used because the file only contains SDC commands.

7. Analyze the design:

   ```
   gca_shell> analyze_design
   Warning: Conflicted case values driving and set at pin u7/U168/S.
   Using the set logicvalue of '0'. (CASE-003)
   Information: Checking scenario 'default': Starting rule
   checks (ADES-002)
   Information: Checking netlist: Starting scenario-independent rule
   checks. (ADES-003)
   1
   gca_shell>
   ```

After you run the `analyze_design` command, the tool displays the results of the rule checking in the violation browser, as shown in Figure A-4.

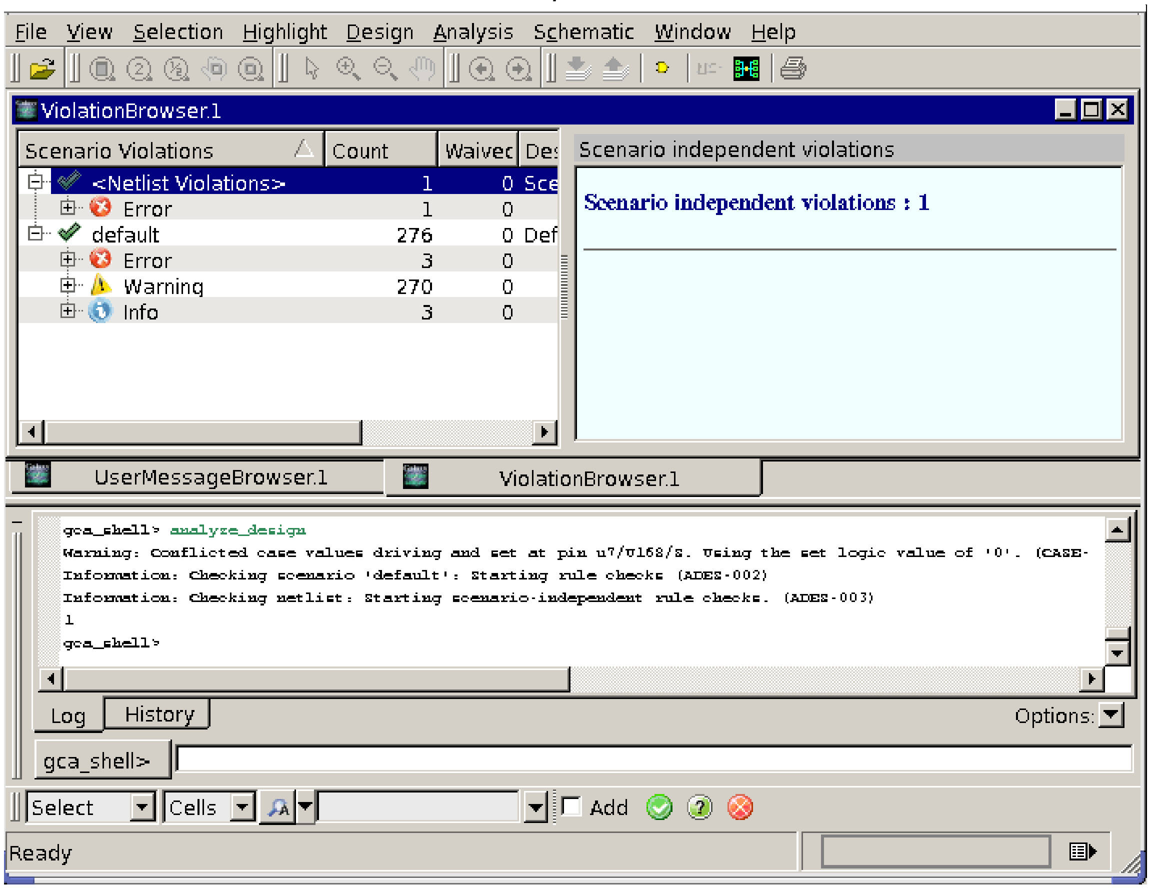
The `analyze_design` command is used to run analysis on your design after the necessary inputs have been loaded in the tool. The design and its constraints are checked against a predefined set of rules, and the results of the analysis are summarized in the violation browser in the GUI.

Note:
The commands used in the steps above are provided in a script file located in the tutorial directory. The file name is run_tutorial.tcl. The content of the run_tutorial.tcl script is shown in Example A-2.

Example A-2 ChipLevel Script to Read in Design, Link, and Load Constraints

```
######################################################################
#       link design
######################################################################
set_app_var search_path [list . ./libs ./design ]
set_app_var link_path [list * lsi_10k.db]
read_verilog chiplevel.v
current_design ChipLevel
link_design
######################################################################
#       read SDC
######################################################################
source ./design/chiplevel.sdc
######################################################################
#       run design analysis
######################################################################
analyze_design
```

Figure A-4 PrimeTime GCA Window on Startup



Scenario Violations	Count	Waivec	Des
<Netlist Violations>	1	0	Sce
Error	1	0	
default	276	0	Def
Error	3	0	
Warning	270	0	
Info	3	0	

Scenario independent violations : 1

```
gca_shell> analyze_design
Warning: Conflicted case values driving and set at pin u7/U168/S. Using the set logic value of '0'. (CASE-
Information: Checking scenario 'default': Starting rule checks (ADES-002)
Information: Checking netlist: Starting scenario-independent rule checks. (ADES-003)
1
gca_shell>
```

gca_shell>

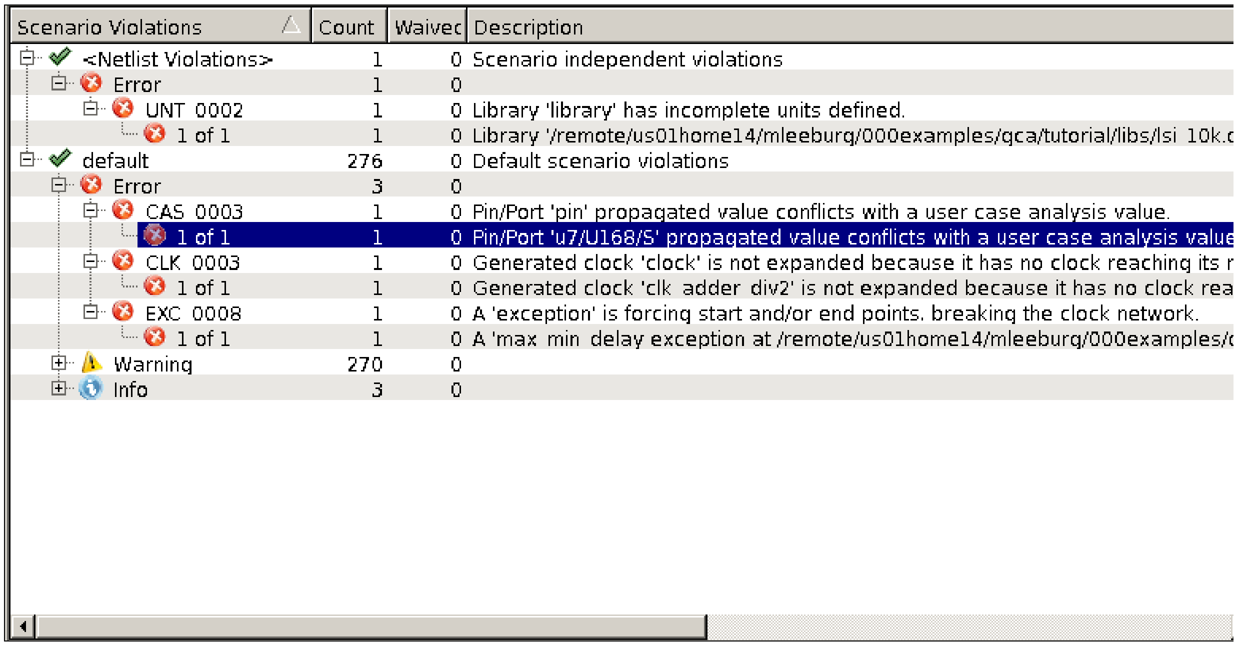
The PrimeTime GCA tool checks constraints for multiple scenarios of the same design. For example, a design can have both a functional mode and test mode, and it can have different boundary conditions for worst-case and best-case analysis. For more information, see "Additional Analysis Features."

Debugging Constraint Problems

The constraint problems found by the `analyze_design` command are displayed in the violation browser, as shown in Figure A-5. The violations are organized by scenario in the browser and are grouped by their severity: error, warning, or information. Error and Warning violations can identify constraint inconsistencies and impact design analysis, runtime, and memory usage. Error violations are the most severe violations. Information violations notify you of situations that might be the result of a mistake in the constraints.

Use the GUI to investigate and debug the constraint problems reported by the tool.

Figure A-5 Violation Browser for the ChipLevel Design



Scenario Violations	Count	Waivec	Description
<Netlist Violations>	1	0	Scenario independent violations
Error	1	0	
UNT 0002	1	0	Library 'library' has incomplete units defined.
1 of 1	1	0	Library '/remote/us01home14/mleeburg/000examples/gca/tutorial/libs/lsi 10k.c
default	276	0	Default scenario violations
Error	3	0	
CAS 0003	1	0	Pin/Port 'pin' propagated value conflicts with a user case analysis value.
1 of 1	1	0	Pin/Port 'u7/U168/S' propagated value conflicts with a user case analysis value
CLK 0003	1	0	Generated clock 'clock' is not expanded because it has no clock reaching its r
1 of 1	1	0	Generated clock 'clk adder div2' is not expanded because it has no clock rea
EXC 0008	1	0	A 'exception' is forcing start and/or end points, breaking the clock network.
1 of 1	1	0	A 'max min delay exception at /remote/us01home14/mleeburg/000examples/c
Warning	270	0	
Info	3	0	

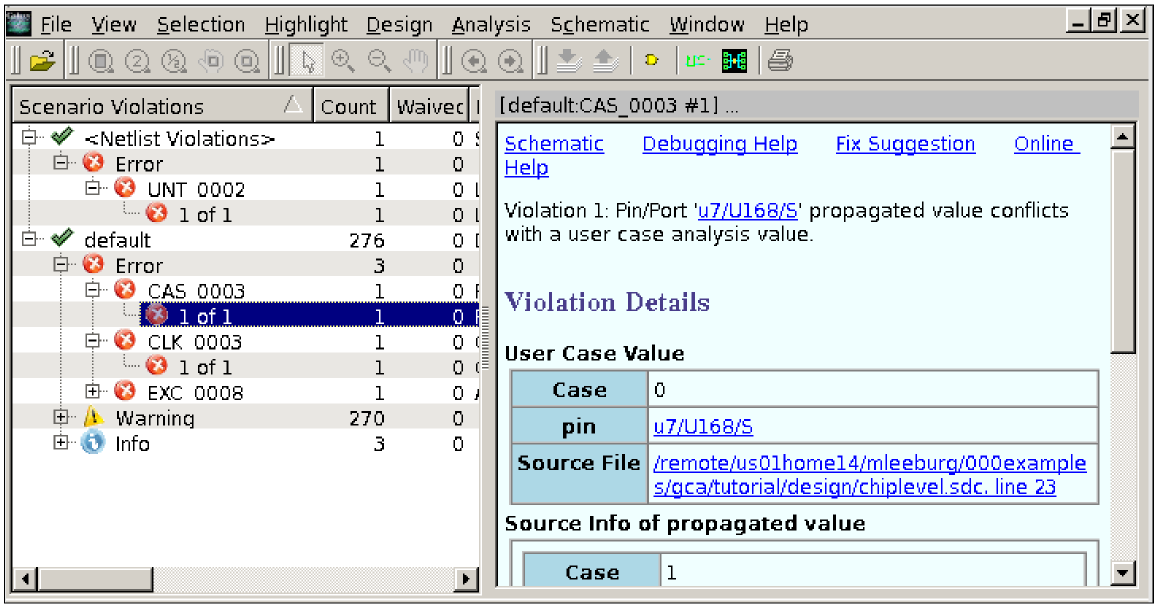
You can display the violations in each category and view the number of violations for each rule by double-clicking on the row of interest in the violation browser or by clicking on the '+' icon at the beginning of each row.

Investigating the CAS_0003 Violation

Begin investigating the CAS_0003 violation by double-clicking the row containing this rule in the violation browser and then clicking on the row listing the violation for this rule, as shown in Figure A-6.

The information pane to the right displays details about the CAS_0003 violation for the ChipLevel design.

Figure A-6 Information Pane Displaying CAS_0003 Rule Violation Details


File View Selection Highlight Design Analysis Schematic Window Help

Scenario Violations	Count	Waived
<Netlist Violations>	1	0
Error	1	0
UNT_0002	1	0
1 of 1	1	0
default	276	0
Error	3	0
CAS_0003	1	0
1 of 1	1	0
CLK_0003	1	0
1 of 1	1	0
EXC_0008	1	0
Warning	270	0
Info	3	0

[default:CAS_0003 #1] ...

Schematic Debugging Help Fix Suggestion Online Help

Violation 1: Pin/Port 'u7/U168/S' propagated value conflicts with a user case analysis value.

Violation Details

User Case Value

Case	0
pin	u7/U168/S
Source File	/remote/us01home14/mleeburg/000examples/gca/tutorial/design/chiplevel.sdc, line 23

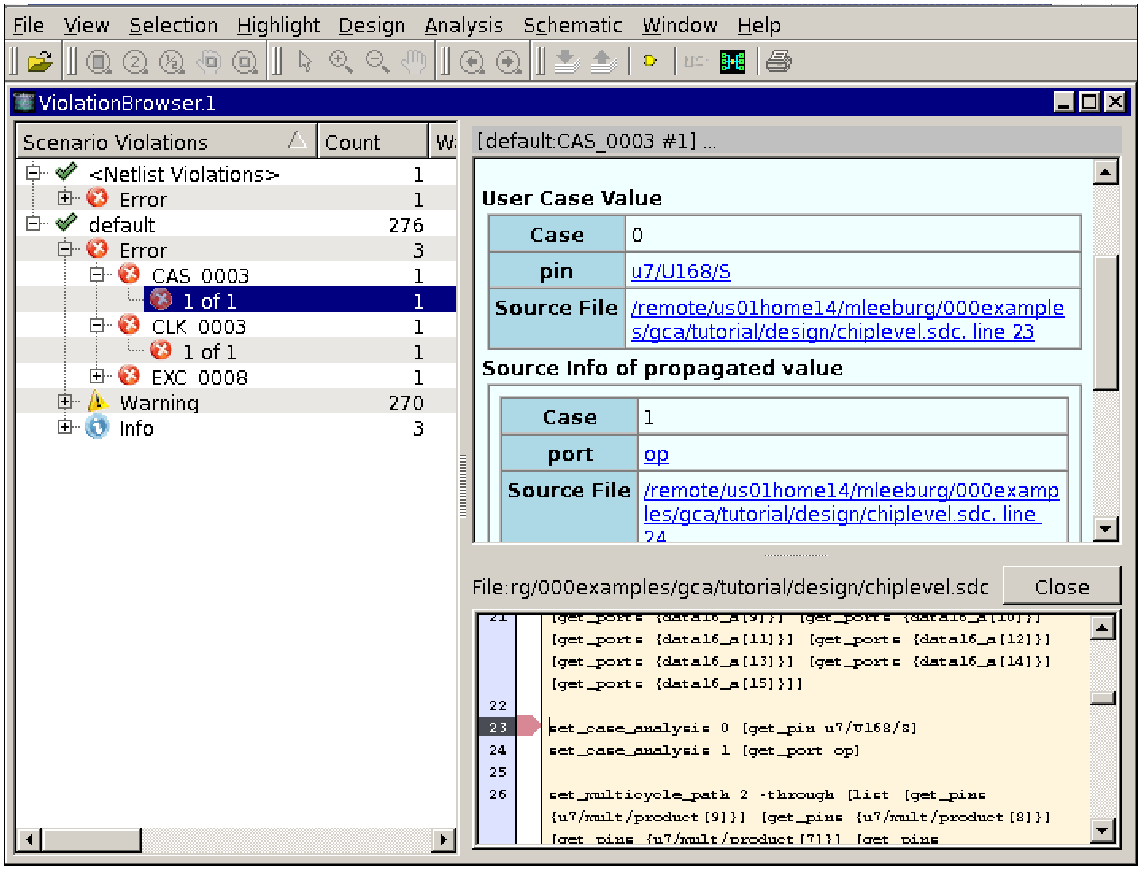
Source Info of propagated value

Case	1

The Violation Message (Violation 1) in the information pane states that a propagated logic constant or case analysis value conflicts with a user-specified case analysis value on a pin or port. The message includes the pin name, u7/U168/S, with the conflicting values.

The User Case Value section shows that a user-specified case analysis value of 0 has been set on this pin. The source of the conflicting case analysis value is the port named “op” as shown in the Source Info of propagated value in the Violation Details. To view the constraints related to the case analysis values, click the link in the Source File box. When you click the link, the constraint browser opens in the information pane, and a marker points to the constraint in the SDC file as shown in Figure A-7.

Figure A-7 Constraint Browser for the User Case Value in the Violation Details


File View Selection Highlight Design Analysis Schematic Window Help
ViolationBrowser.1
Close

Scenario Violations	Count
<Netlist Violations>	1
Error	1
default	276
Error	3
CAS_0003	1
1 of 1	1
CLK_0003	1
1 of 1	1
EXC_0008	1
Warning	270
Info	3

[default:CAS_0003 #1] ...

User Case Value

Case	0
pin	u7/U168/S
Source File	/remote/us01home14/mleeburg/000examples/gca/tutorial/design/chiplevel.sdc, line 23

Source Info of propagated value

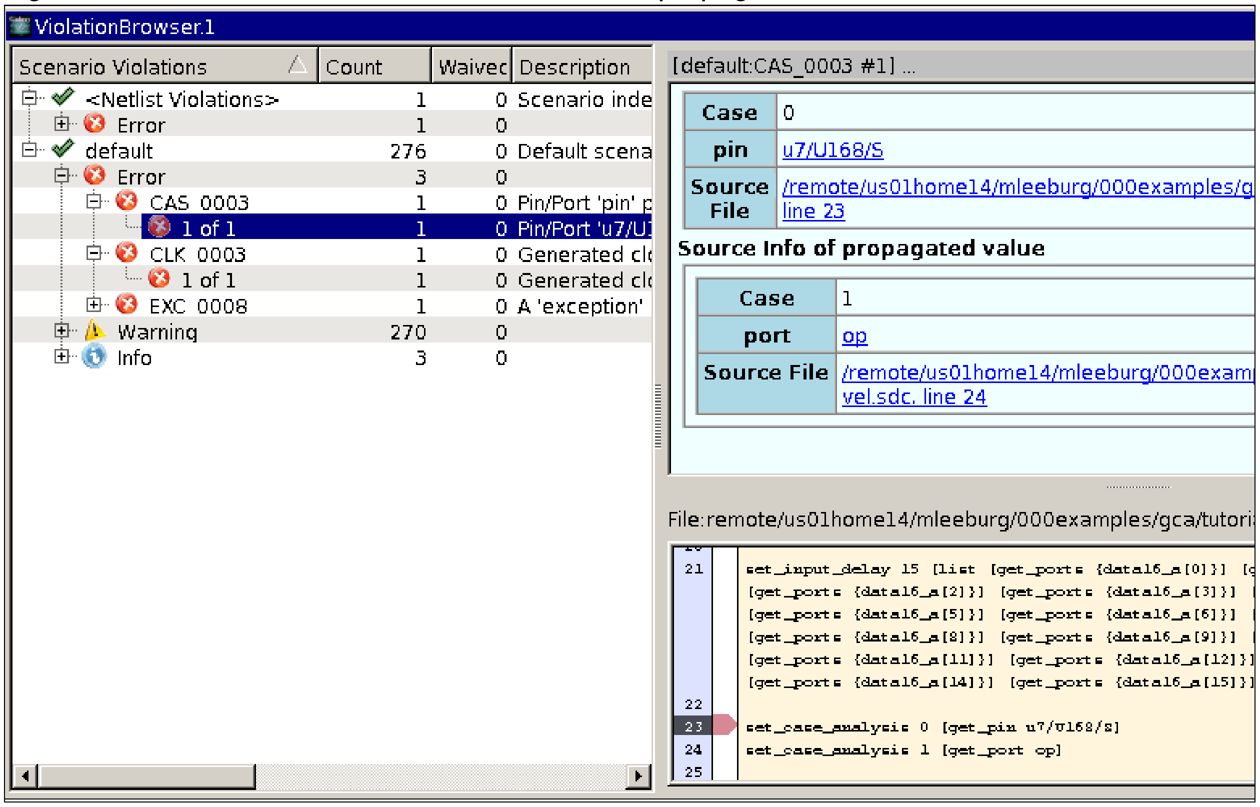
Case	1
port	op
Source File	/remote/us01home14/mleeburg/000examples/gca/tutorial/design/chiplevel.sdc, line 24

File:rg/000examples/gca/tutorial/design/chiplevel.sdc

```
[get_ports {data16_a[11]}] [get_ports {data16_a[12]}]
[get_ports {data16_a[13]}] [get_ports {data16_a[14]}]
[get_ports {data16_a[15]}]]

set_case_analysis 0 [get_pin u7/U168/S]
set_case_analysis 1 [get_port op]

set_multicycle_path 2 -through [list [get_pins
{u7/mult/product[9]}] [get_pins {u7/mult/product[8]}]
```

Similarly, by clicking on the Source File link under Source Info of propagated value, the constraint browser moves to the line in the SDC file related to the port named “op” as shown in Figure A-8.

Figure A-8 Constraint Browser for Source Info of propagated value in Violation Details



ViolationBrowser.1

Scenario Violations	Count	Waivec	Description
<Netlist Violations>	1	0	Scenario inde
Error	1	0	
default	276	0	Default scena
Error	3	0	
CAS_0003	1	0	Pin/Port 'pin' p
1 of 1	1	0	Pin/Port 'u7/U1
CLK_0003	1	0	Generated clo
1 of 1	1	0	Generated clo
EXC_0008	1	0	A 'exception'
Warning	270	0	
Info	3	0	

[default:CAS_0003 #1] ...

Case	0
pin	u7/U168/S
Source File	/remote/us01home14/mleeburg/000examples/g line 23

Source Info of propagated value

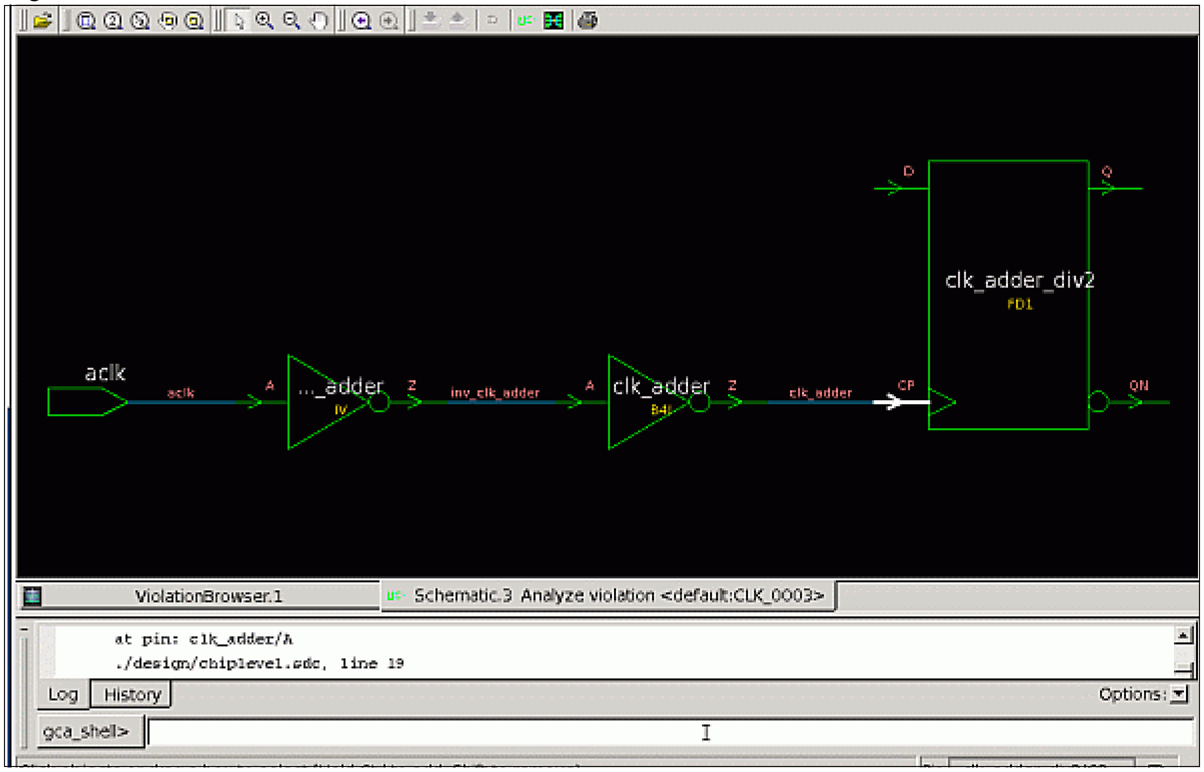
Case	1
port	op
Source File	/remote/us01home14/mleeburg/000exam vel.sdc, line 24

File:remote/us01home14/mleeburg/000examples/gca/tutori

```
21   set_input_delay 15 [list [get_ports {data16_a[0]}] [g
     [get_ports {data16_a[2]}] [get_ports {data16_a[3]}] [
     [get_ports {data16_a[5]}] [get_ports {data16_a[6]}] [
     [get_ports {data16_a[8]}] [get_ports {data16_a[9]}] [
     [get_ports {data16_a[11]}] [get_ports {data16_a[12]}]
     [get_ports {data16_a[14]}] [get_ports {data16_a[15]}]
22
23   set_case_analysis 0 [get_pin u7/U168/S]
24   set_case_analysis 1 [get_port op]
25
```

To understand how the case analysis value on the port named "op" propagates to the violating pin, scroll up to the top of the information pane and click the Schematic link. The link opens a separate window, shown in Figure A-9 that indicates the location of the pin with the conflicting case analysis value.

Figure A-9 Schematic Related to CAS_0003 Rule Violation

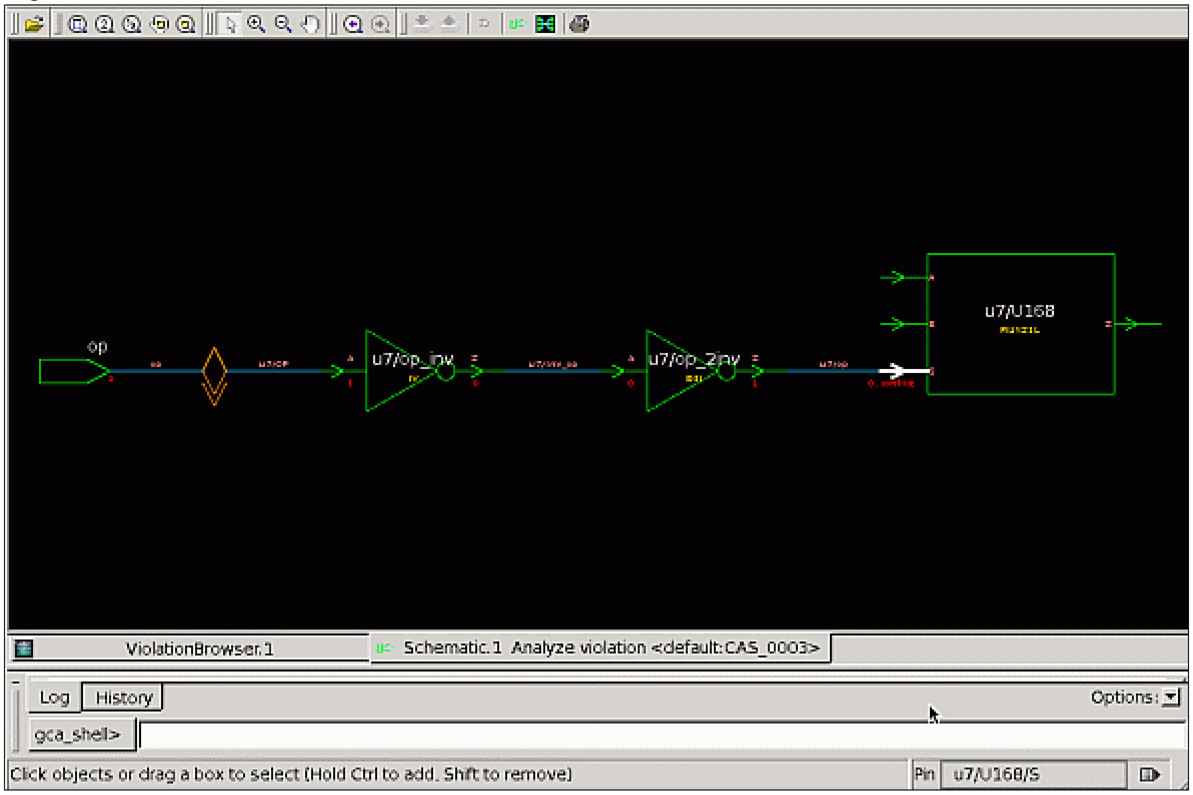


```
at pin: clk_adder/A
./design/chiplevel.sdc, line 19
```

The schematic shows that a case value of 1 is present at the u7/op_2inv/Z pin, and a case value of 0 is present at the u7/U168/S pin. This pin has been highlighted in the schematic and marked as the source of the case analysis conflict.

The schematic shows the case analysis value of 1 on the op port, which propagates through the u7/op_inv inverter to a case value of 0 on its output, and then through the u7/op_2inv inverter, resulting in a case value of 1 on its output. The PrimeTime GCA tool highlights this situation as a case analysis conflict.

Figure A-10 Schematic View of CAS_0003 Rule Violation

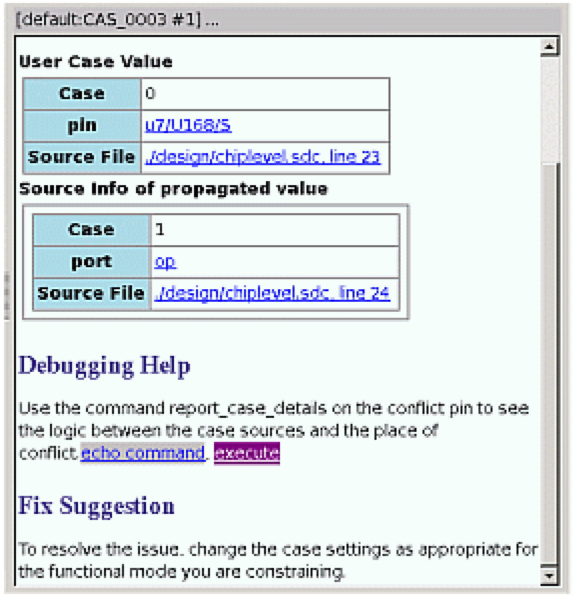


Click objects or drag a box to select (Hold Ctrl to add. Shift to remove)

You can also view the same information in report form. First, click the ViolationBrowser.1 tab to go back to the violation browser and information pane. Then click the Debugging Help link in the information pane.

The Debugging Help section related to the violation is displayed in the information pane as shown in Figure A-11.

Figure A-11 Viewing the Debugging Help for the CAS_0003 Rule Violation



[default:CAS_0003 #1] ...

User Case Value

Case	0
pin	u7/U168/S
Source File	./design/chiplevel.sdc, line 23

Source Info of propagated value

Case	1
port	op
Source File	./design/chiplevel.sdc, line 24

Debugging Help

Use the command report_case_details on the conflict pin to see the logic between the case sources and the place of conflict. echo command, execute

Fix Suggestion

To resolve the issue, change the case settings as appropriate for the functional mode you are constraining.

When you click the Execute link under Debugging Help, the tool runs the `report_case_details` command related to the violation. A report similar to the one shown in Example A-3 is displayed in the console log view.

Example A-3 Case Propagation Details Report

```
gca_shell> report_case_details -to u7/U168/S
****************************************
Report : case_details
         -to
         -max_objects = 1000
Design : ChipLevel
...
****************************************
Properties         Value    Pin/Port
-------------------------------------------------------------------------
user case          0        u7/U168/S (MUX21L)
Case fanin report:
Verbose Source Trace for pin/port u7/U168/S:
Path number: 1
Path Ref #   Value  Properties           Pin/Port
-------------------------------------------------------------------------
             0      user case            u7/U168/S (MUX21L)
             1      F()=A'               u7/op_2inv/Z (B4I)
             0                           u7/op_2inv/A (B4I)
             0      F()=A'               u7/op_inv/Z (IV)
             1                           u7/op_inv/A (IV)
             1      user case            op (in)
```

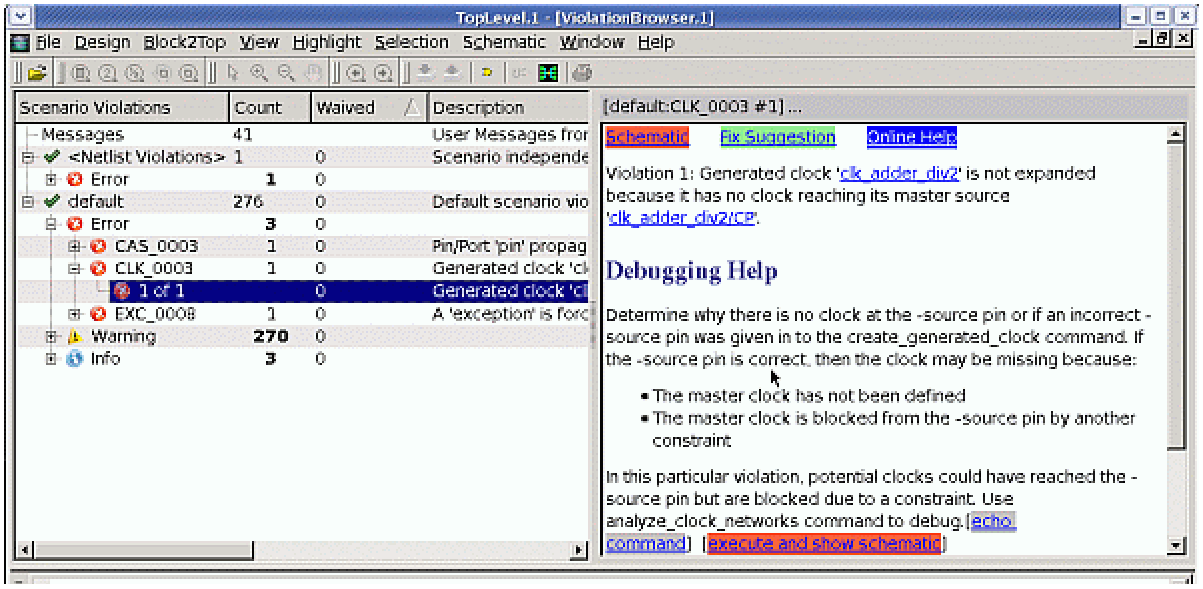
The beginning of the report shows the user-set case analysis value set directly on the u7/U168/S pin. The “Case fanin report” section shows the conflicting value propagated to the same pin, along with the path of the propagated values through multiple gates, originating from a case analysis value set on the "op" port.

The decision on how to resolve the CAS_0003 rule violation is design-dependent and based on knowledge of the design and mode being analyzed. In some cases, removing the case analysis on the u7/U168/S pin might be the solution, and in other cases, removing the case analysis on the port might be the solution. The conflict between propagated and user-specified case values should be resolved to avoid incomplete timing analysis or analysis of false paths for the operational mode of the design.

Investigating the CLK_0003 Violation

To investigate the CLK_0003 rule violation, go to the violation browser and click the violation for the CLK_0003 rule. The information pane is updated with information about the CLK_0003 rule violation as shown in Figure A-12.

Figure A-12 Violation Browser and Information Pane Displaying CLK_0003 Rule Violation Details


TopLevel.1 - [ViolationBrowser.1]
File Design Block2Top View Highlight Selection Schematic Window Help

Scenario Violations	Count	Waived	Description
Messages	41		User Messages fror
<Netlist Violations>	1	0	Scenario independe
Error	1	0	
default	276	0	Default scenario vio
Error	3	0	
CAS_0003	1	0	Pin/Port 'pin' propag
CLK_0003	1	0	Generated clock 'cl
1 of 1		0	Generated clock 'cl
EXC_0008	1	0	A 'exception' is forc
Warning	270	0	
Info	3	0	

[default:CLK_0003 #1] ...

Schematic Fix Suggestion Online Help

Violation 1: Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its master source 'clk_adder_div2/CP'.

Debugging Help

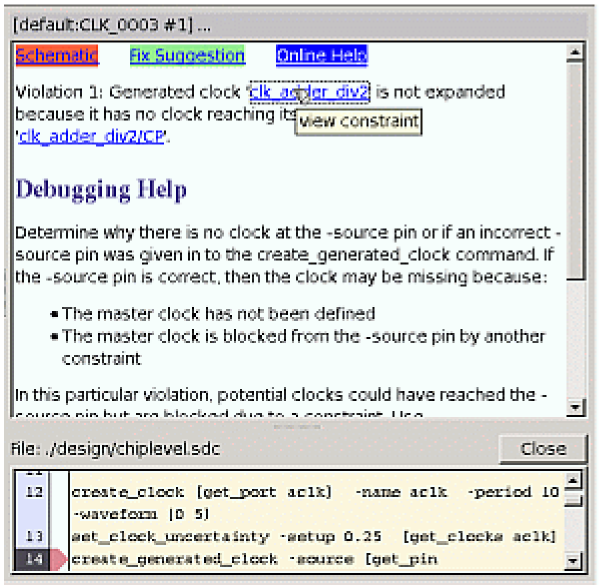
Determine why there is no clock at the -source pin or if an incorrect -source pin was given in to the create_generated_clock command. If the -source pin is correct, then the clock may be missing because:

- The master clock has not been defined
- The master clock is blocked from the -source pin by another constraint

In this particular violation, potential clocks could have reached the -source pin but are blocked due to a constraint. Use analyze_clock_networks command to debug.[echo command] [execute and show schematic]

The CLK_0003 rule highlights a problem with generated clock propagation due to the lack of a master source as shown in the Violation 1 message. The pin or port specified as the master source of the generated clock, clk_adder_div2/CP, does not have a clock reaching it. You can view the definition of the generated clock in the constraint file by clicking on the clk_adder_div2 link (the name of the generated clock) in the Violation 1 message. The constraint browser opens and displays the generated clock definition as shown in Figure A-13.

Figure A-13 Definition of clk_adder_div2 Generated Clock in Constraint File



[default:CLK_0003 #1] ...

Schematic   Fix Suggestion   Online Help

Violation 1: Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its 'clk_adder_div2/CP'.

View constraint

Debugging Help

Determine why there is no clock at the -source pin or if an incorrect -source pin was given in to the create_generated_clock command. If the -source pin is correct, then the clock may be missing because:

- The master clock has not been defined
- The master clock is blocked from the -source pin by another constraint

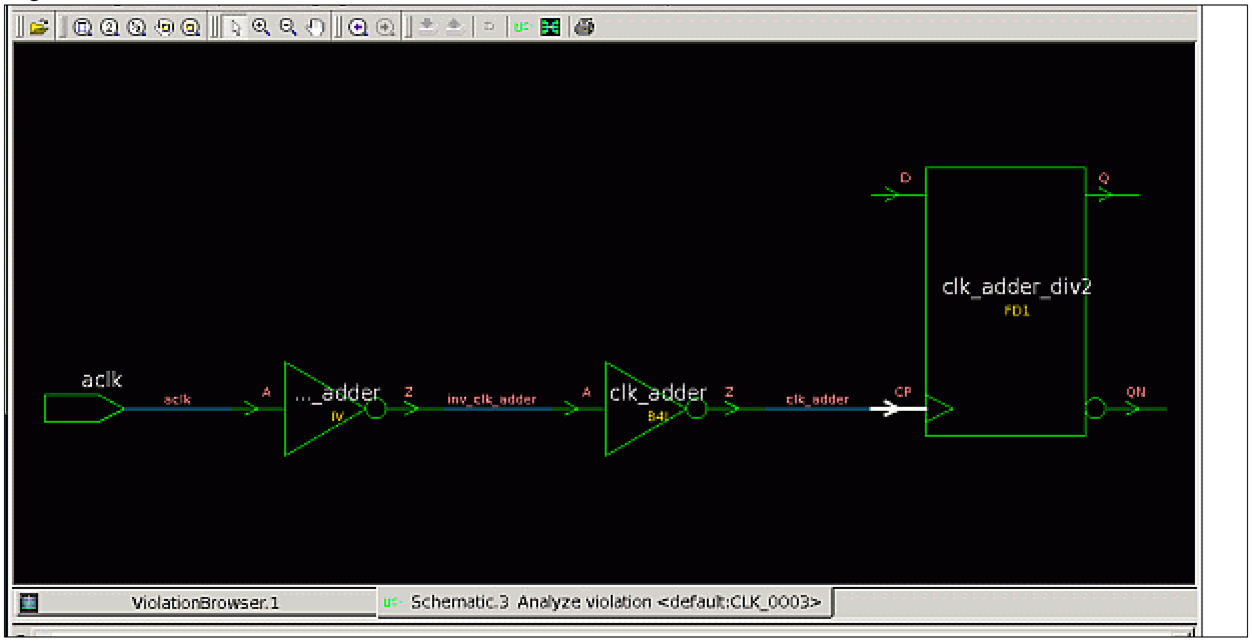
In this particular violation, potential clocks could have reached the -

File: ./design/chiplevel.sdc   Close

```
create_clock [get_port aclk]  -name aclk  -period 10
-waveform {0 5}
set_clock_uncertainty -setup 0.25  [get_clocks aclk]
create_generated_clock -source [get_pin
```

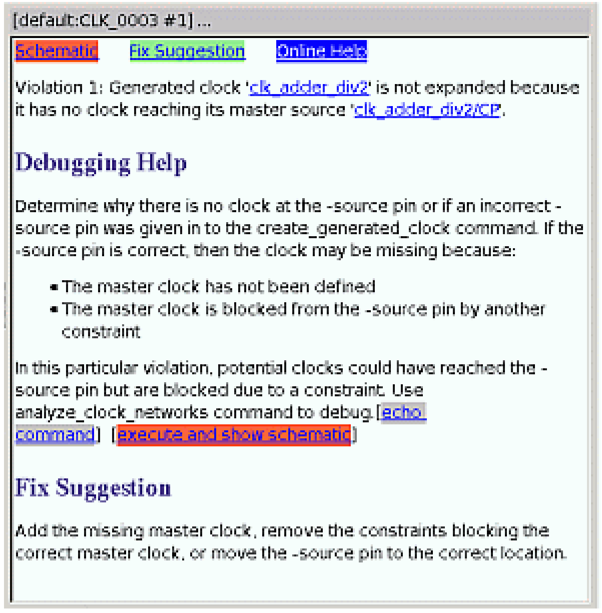
The constraint browser confirms that the pin clk_adder_div2/CP has been defined as the master clock source for the generated clock. To view the related schematic, click the Close button of the constraint browser and then click the Schematic link in the information pane. The schematic opens in a separate window as shown in Figure A-14.

Figure A-14 Schematic for CLK_0003 Rule Violation



The schematic shows that there is a path from the aclk port to the master source for the generated clock, clk_adder_div2/CP. To see what might be causing the problem, go back to the violation browser and look at the Debugging Help section following the violation message, as shown in Figure A-15.

Figure A-15 Debugging Help for the CLK_0003 Rule Violation



[default:CLK_0003 #1] ...

Schematic Fix Suggestion Online Help

Violation 1: Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its master source 'clk_adder_div2/CP'.

Debugging Help

Determine why there is no clock at the -source pin or if an incorrect -source pin was given in to the create_generated_clock command. If the -source pin is correct, then the clock may be missing because:

- The master clock has not been defined
- The master clock is blocked from the -source pin by another constraint

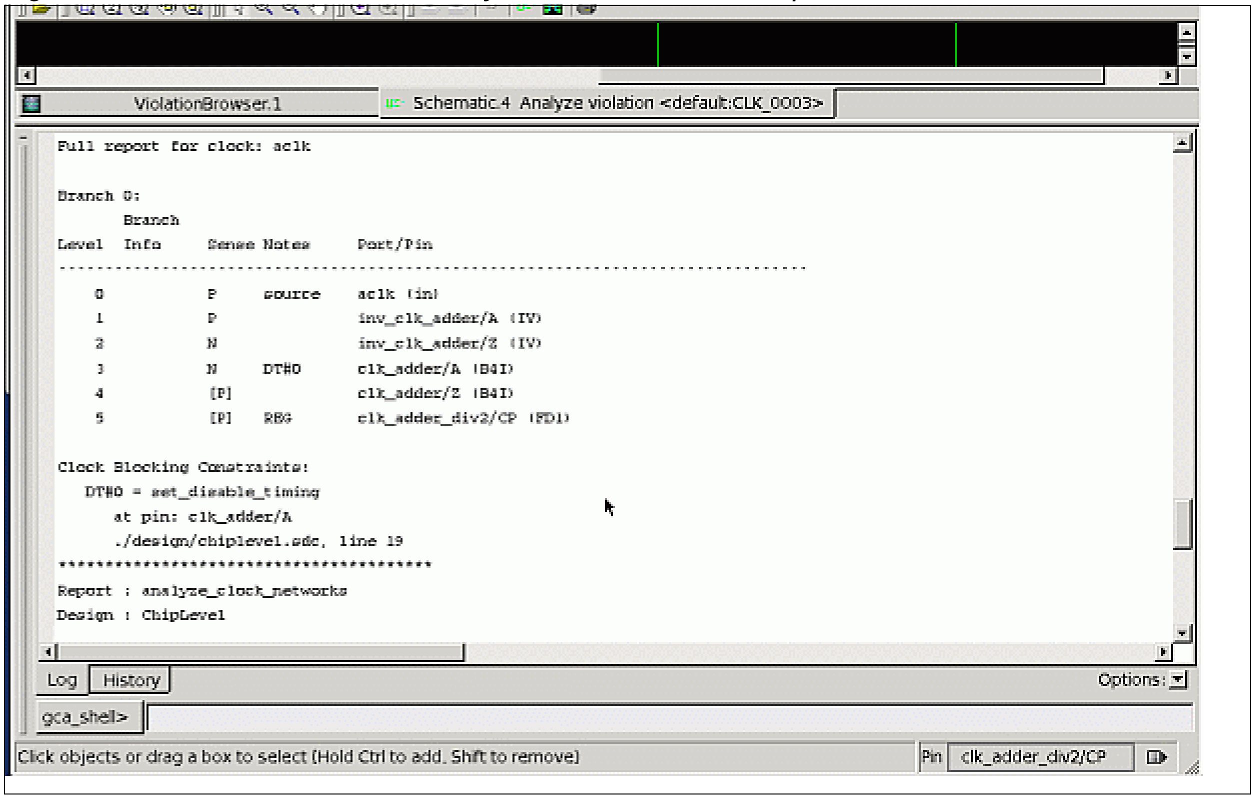
In this particular violation, potential clocks could have reached the -source pin but are blocked due to a constraint. Use analyze_clock_networks command to debug.[echo command] [execute and show schematic]

Fix Suggestion

Add the missing master clock, remove the constraints blocking the correct master clock, or move the -source pin to the correct location.

The Debugging Help states that there might be potential clocks that have reached the clk_adder_div2/CP pin. To run the `analyze_clock_networks` command to help debug this violation, click the “execute and show schematic” link under Debugging Help. The schematic related to the violation is displayed again, and the report for the `analyze_clock_networks` command is displayed in the GUI console as shown in Figure A-16.

Figure A-16 Partial View of the analyze_clock_networks Report in GUI Console



```
Full report for clock: aclk

Branch 0:
       Branch
Level  Info     Sense Notes     Port/Pin
--------------------------------------------------------------------------------

    0           P     source    aclk (in)
    1           P               inv_clk_adder/A (IV)
    2           N               inv_clk_adder/Z (IV)
    3           N     DT#0      clk_adder/A (B4I)
    4           [P]             clk_adder/Z (B4I)
    5           [P]   REG       clk_adder_div2/CP (FD1)

Clock Blocking Constraints:
   DT#0 = set_disable_timing
      at pin: clk_adder/A
      ./design/chiplevel.sdc, line 19
****************************************
Report : analyze_clock_networks
Design : ChipLevel
```

```
Click objects or drag a box to select (Hold Ctrl to add, Shift to remove)
```

View the report by scrolling up and down in the GUI console window, or use the echo command link under Debugging Help and the redirect command to send the entire report to a text file. For this violation, the following is an example of the `redirect` command:

```
gca_shell> redirect -file clk_0003_report.txt \
  { analyze_clock_networks -from [get_clocks {aclk}] \
  -through [get_pins {clk_adder_div2/CP}] -max_endpoints 1 -style full \
  -end_types  {register port clock_source } -traverse_disabled -nosplit }
```

The `-traverse_disabled` option of the `analyze_clock_networks` command ensures that the portions of the clock network that are disabled due to a constraint are reported as potential clock networks and that the related constraints that are causing the clock to be blocked are included. The full report is shown in Example A-4.

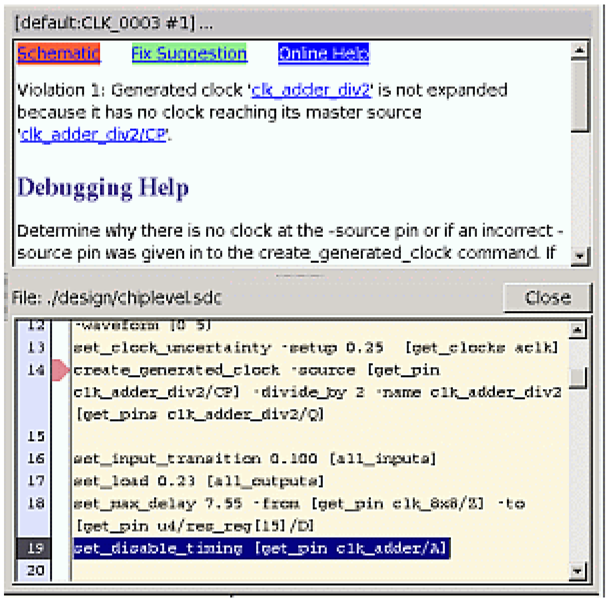
Example A-4 Report for analyze_clock_networks for CLK_0003 Violation

```
****************************************
Report : analyze_clock_networks
Design : ChipLevel
Scenario: default
...
****************************************
Clock Sense Abbreviations:
   P - positive
   N - negative
Potential senses detected with -traverse_disabled:
   [P] - potential positive
Clock Network End Type Abbreviations:
   REG - register
Full report for clock: aclk
Branch 0:
       Branch
Level  Info     Sense Notes     Port/Pin
--------------------------------------------------------------------------
    0           P     source    aclk (in)
    1           P               inv_clk_adder/A (IV)
    2           N               inv_clk_adder/Z (IV)
    3           N     DT#0      clk_adder/A (B4I)
    4           [P]             clk_adder/Z (B4I)
    5           [P]   REG       clk_adder_div2/CP (FD1)
Clock Blocking Constraints:
   DT#0 = set_disable_timing
      at pin: clk_adder/A
      ./design/chiplevel.sdc, line 19
```

The report traces the clock network that begins at port aclk and identifies the point in the clock path where the clock stops propagating and the related constraint. A detailed examination of the report shows that the clock aclk is a positive-sense clock as shown by the P in the Sense column of the report. It then becomes a negative-sense clock N at the output of the inv_clk_adder inverter in the clock network. The output of the next inverter, clk_adder, inverts the clock again to a positive-sense clock, but the report shows that the clock propagation has stopped.

A potential clock has been detected on the output of the clk_adder inverter with a positive sense, as shown by the [P] in the Sense column of the report. The Notes column shows a reference, DT#0, to the Clock Blocking Constraints section of the report. This shows that a `set_disable_timing` constraint has been specified on the pin clk_adder/A which is blocking the clock propagation. Similarly, the generated clock master source, clk_adder_div2/CP, has a potential positive clock [P] because it is downstream of the clk_adder/A pin. The Clock Blocking Constraints section also indicates that the relevant constraint is on line 19 of the constraint file. Because this is the same constraint file that contains the generated clock definition, you can open the Constraint Browser again by clicking on the clk_adder_div2 link and scrolling to line 19, as shown in Figure A-17.

Figure A-17 Viewing Clock Blocking Constraint Identified in analyze_clock_networks Report



Violation 1: Generated clock 'clk_adder_div2' is not expanded because it has no clock reaching its master source 'clk_adder_div2/CP'.

Debugging Help

Determine why there is no clock at the -source pin or if an incorrect -source pin was given in to the create_generated_clock command. If

```
-waveform {0 5}
set_clock_uncertainty -setup 0.25  [get_clocks aclk]
create_generated_clock -source [get_pin
clk_adder_div2/CP] -divide_by 2 -name clk_adder_div2
[get_pins clk_adder_div2/Q]

set_input_transition 0.100 [all_inputs]
set_load 0.23 [all_outputs]
set_max_delay 7.55 -from [get_pin clk_8x8/Z] -to
[get_pin u4/res_reg[15]/D]
set_disable_timing [get_pin clk_adder/A]
```

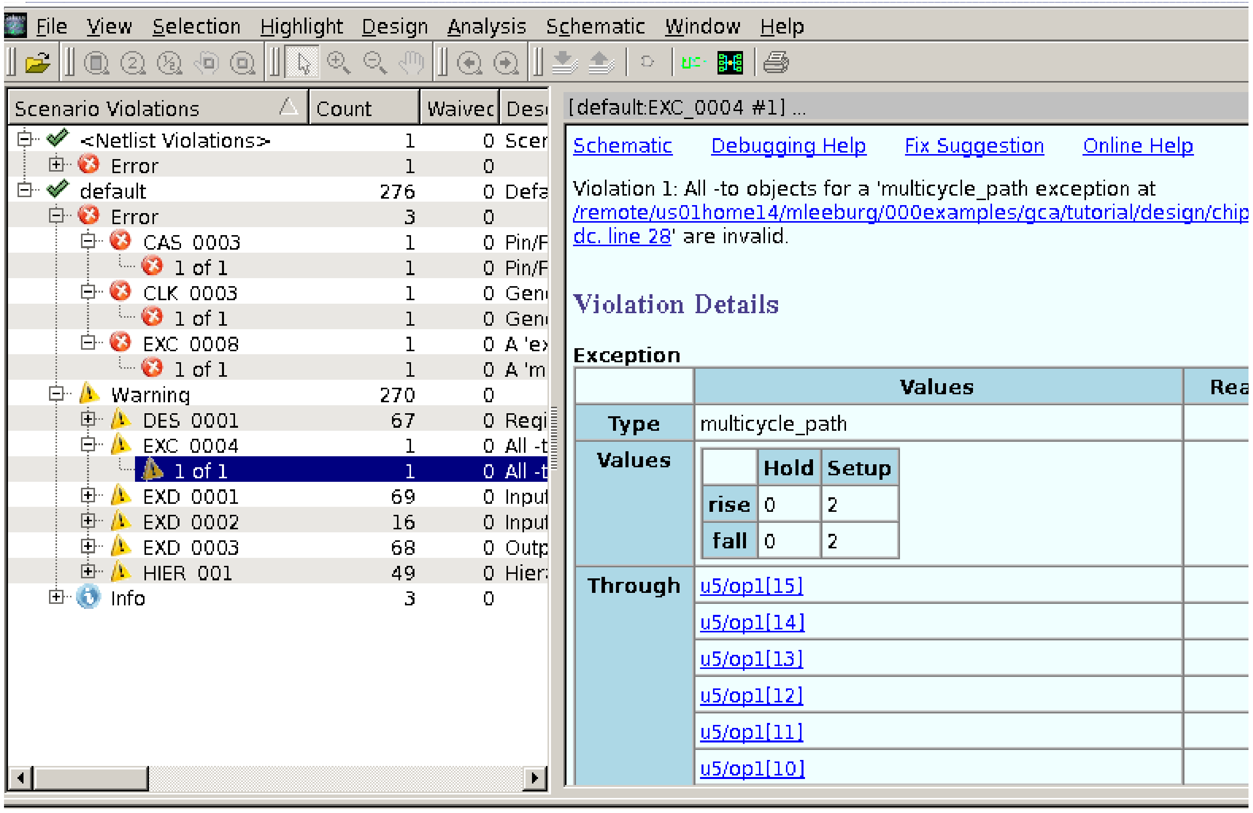
The set_disable_timing constraint disables all timing paths through the pin clk_adder/A and it is the root cause for the clock not reaching the master source clk_adder_div2/CP of the generated clock.

Based on your knowledge of the design, there can be a valid reason for disabling the clock network. If so, the generated clock definition should be reexamined. If left without a clock at the master source, the generated clock does not propagate.

Investigating the EXC_0004 Violation

To examine the EXC_0004 rule violation, expand the Warning category in the violation browser and select the EXC_0004 rule violation. The information pane displays the violation details, as shown in Figure A-18.

Figure A-18 Information Pane for EXC_0004 Rule Violation



Scenario Violations	Count	Waivec	Desi
<Netlist Violations>	1	0	Scer
Error	1	0	
default	276	0	Defa
Error	3	0	
CAS_0003	1	0	Pin/F
1 of 1	1	0	Pin/F
CLK_0003	1	0	Gen
1 of 1	1	0	Gen
EXC_0008	1	0	A 'ex
1 of 1	1	0	A 'm
Warning	270	0	
DES_0001	67	0	Reqi
EXC_0004	1	0	All -t
1 of 1	1	0	All -t
EXD_0001	69	0	Input
EXD_0002	16	0	Input
EXD_0003	68	0	Outp
HIER_001	49	0	Hier:
Info	3	0	

[default:EXC_0004 #1] ...

Schematic Debugging Help Fix Suggestion Online Help

Violation 1: All -to objects for a 'multicycle_path exception at /remote/us01home14/mleeburg/000examples/gca/tutorial/design/chip dc, line 28' are invalid.

Violation Details

Exception

	Values	Rea
Type	multicycle_path	
Values	Hold / Setup: rise 0 / 2; fall 0 / 2	
Through	u5/op1[15]	
	u5/op1[14]	
	u5/op1[13]	
	u5/op1[12]	
	u5/op1[11]	
	u5/op1[10]	

	Hold	Setup
rise	0	2
fall	0	2

The EXC_0004 rule flags exceptions with invalid path endpoints in the `-to` option of the constraint. The Violation Details show that the Type of exception is a multicycle path for this violation. If you browse further down the Violation Details, you can see that many Through and To points have been listed as part of this violation. The Reason listed for the To points is “nonendpoint”.

To help understand the Reason provided, scroll back to the top of the information pane, and click the Debugging Help link.

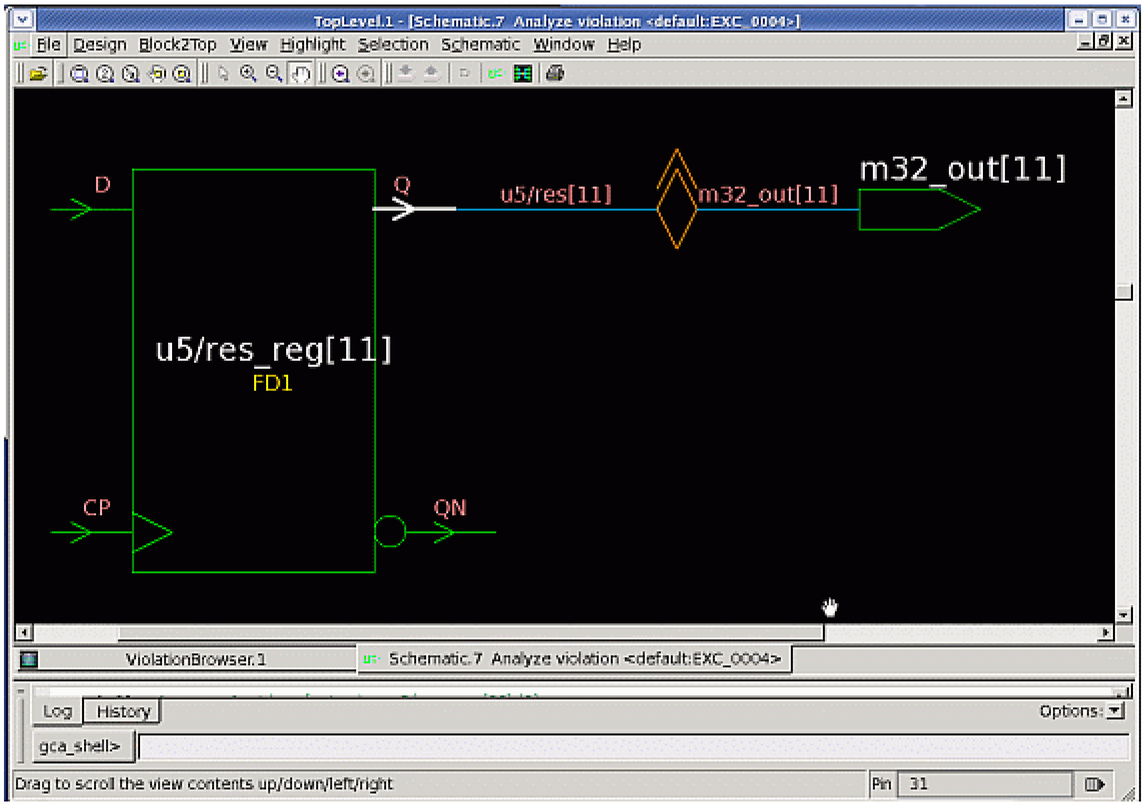
The Debugging Help suggests using the `report_disable_timing -all_arcs` command if the Reason code is case disabled, check disabled, sequential disabled, or node disabled. Because the Debugging Help does not have a command suggestion for the nonendpoint Reason, click the online Help link at the top of the information pane for more help. The rule reference for the EXC_0004 rule is displayed in your Web browser.

The Description section of the rule reference explains the different Reason codes for the EXC_0004 rule. The nonendpoint reason is given when the `-to` pin is an internal pin instead of a timing path endpoint. The Description section also states that the exception is ignored during timing analysis unless a valid endpoint is provided for the constraint. The explanation in the rule reference for nonendpoint indicates that the pins specified for the `-to` option of the `set_multicycle_path` command are internal pins and are not timing path endpoints.

The `-to` points listed in the Violation Details are all pins with the name Q. View the schematic related to the violation by clicking the Schematic link at the top of the information pane. The schematic shows all the affected `-to` pins. Use the Zoom In button on the Toolbar to view individual pins.

The Q pin of the register is highlighted in white, indicating it is one of the `-to` points related to the violation. Double-click the pin to show more of the schematic, as shown in Figure A-19.

Figure A-19 Expanded Schematic View of Register Q Pin


TopLevel.1 - [Schematic.7 Analyze violation <default:EXC_0004>]
File Design Block2Top View Highlight Selection Schematic Window Help
D
Q
u5/res[11]
m32_out[11]
m32_out[11]
u5/res_reg[11]
FD1
CP
QN
ViolationBrowser.1
Schematic.7 Analyze violation <default:EXC_0004>
Log History
Options:
gca_shell>
Drag to scroll the view contents up/down/left/right
Pin 31

The schematic shows that the Q pin of the register is connected to an output port. The Q pin of the register might have been incorrectly specified instead of the D pin in the constraint. Changing the constraint to the D pin of the register might resolve the violation, but you should verify that this is the intent of the constraint based on your knowledge of the design.

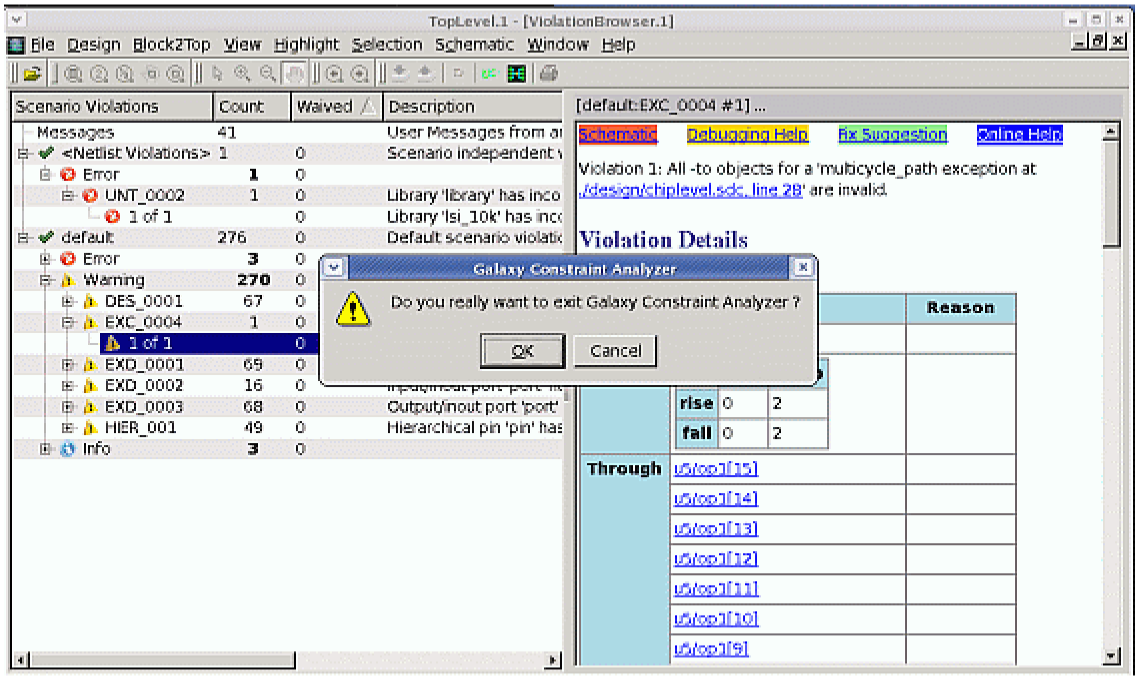
Ending and Restarting the Tutorial Session

You have completed investigation of several constraint problems found by the PrimeTime GCA tool. To quit the shell, enter `exit` at the `gca_shell>` prompt in the GUI:

```
gca_shell> exit
```

A message asks if you want to exit the PrimeTime GCA tool, as shown in Figure A-20.

Figure A-20 GUI Message for Exiting the PrimeTime GCA Tool



TopLevel.1 - [ViolationBrowser.1]

Scenario Violations	Count	Waived	Description
Messages	41		User Messages from a
<Netlist Violations>	1	0	Scenario independent
Error	1	0	
UNT_0002	1	0	Library 'library' has inco
1 of 1		0	Library 'lsi_10k' has inc
default	276	0	Default scenario violati
Error	3	0	
Warning	270	0	
DES_0001	67	0	
EXC_0004	1	0	
1 of 1		0	
EXD_0001	69	0	
EXD_0002	16	0	
EXD_0003	68	0	Output/inout port 'port'
HIER_001	49	0	Hierarchical pin 'pin' has
Info	3	0	

[default:EXC_0004 #1] ...

Violation 1: All -to objects for a 'multicycle_path exception at ./design/chiplevel.sdc, line 28' are invalid.

Violation Details

Galaxy Constraint Analyzer

Do you really want to exit Galaxy Constraint Analyzer ?

Click OK to exit the PrimeTime GCA tool. The tool closes and reports its memory and CPU time usage:

```
gca_shell> exit
Maximum memory usage for this session: 24.38 MB
CPU usage for this session: 2 seconds
```

If you have ended the tutorial session and want to begin analysis of violations again, you rerun the ChipLevel design. To specify your run script at the command line, use the `-f` option:.

```
% gca_shell -f run_tutorial.tcl
```

In this case, the PrimeTime GCA tool loads the libraries, design, and constraints and performs rule checking on the design. The GUI then displays the results of the `analyze_design` command in the violation browser.

If you want to rerun the analysis to investigate one rule, specify the `-rules` option for the `analyze_design` command. This limits the scope of the checking to the rules specified. The command below checks the ChipLevel design for violations of the CAS_0003 rule.

```
analyze_design -rules { CAS_0003 }
```