SpyGlass® Explorer

User Guide

Version N-2017.12-SP2, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Third-Party Links

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

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

Boost Process

Project homepage: *http://www.highscore.de/boost/process0.5/index.html*

Project license:

Boris Schaeling

Report an Error

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

Contents

Introduction

The SpyGlass platform provides designers with insight about their design, early in the process at RTL, using many advanced algorithms and analysis techniques. It functions like an interactive guidance system for design engineers, enabling them to solve various design issues in the early stages of the design, so as to facilitate successful implementation for complex SoCs.

The SpyGlass Explorer User Guide describes various features, menus, and windows of SpyGlass Explorer.

In This Guide

The SpyGlass Explorer User Guide has the following sections:

Section	Description
Introduction	Provides an overview of SpyGlass and SpyGlass Explorer
Getting Started	Describes the process of invoking SpyGlass Explorer
Working with SpyGlass Explorer User Interface	Explains the menus and options available in SpyGlass Explorer
Windows and Panes in SpyGlass Explorer	Explains the windows and panes in SpyGlass Explorer
Understanding SpyGlass Concepts	Describes concepts and flow of SpyGlass
Understanding Design Stages in SpyGlass	Describes the design stages in SpyGlass
Working with Input Design and Libraries	Describes the process of reading the design
Working with Methodologies	Describes the methodologies and goals in SpyGlass
Working with SpyGlass Design Constraints	Introduces SpyGlass constraints and explains the process of using SpyGlass Constraints
Working with SpyGlass Messages	Explains the messages generated by SpyGlass and waiving messages
The Configuration File in SpyGlass	Describes how to specify various configuration settings in the SpyGlass Configuration File.
Design Read Options in SpyGlass Explorer	Describes the various design read options in SpyGlass Explorer
Product-Specific Interfaces in SpyGlass Explorer	Explains various product-specific interfaces available using SpyGlass Explorer
Batch Mode in SpyGlass	Explains how to use SpyGlass in the batch mode
Reports Generated in SpyGlass	Describes various reports generated in SpyGlass

Section	Description
Special Features in SpyGlass	Describes various special features in SpyGlass Explorer
Appendix	Describes using HDL directives, re-using simulation scripts, and using the order file

Intended Audience

This Guide is intended for all the beginner and advanced users of SpyGlass and SpyGlass Explorer.

SpyGlass Overview

SpyGlass® provides the following features:

- Full language support for Verilog (IEEE 1364, Verilog 2001, and SystemVerilog) and VHDL, that is VHDL (IEEE Std 1987) and VHDL (IEEE Std 1993).
- A rich suite of in-built rules, including:
  - File checks, such as file names, design units per file and headers
  - Naming checks on signals, ports, parameters, constants, clocks and other constructs
  - Style and related checks
  - Coding for synthesis and related checks
  - Design practice and related checks
  - Area, timing, and synchronization checks
  - Clock and reset checks
  - SpyGlass DFT solution, SpyGlass Power Verify solution, SpyGlass Constraints solution, SpyGlass ERC solution, and similar checks (cost options)
  - Support for several industry standard HDL analysis and assessment programs, including OpenMORE^{TM} and STARC^{TM}
- A variety of report format options so you can set up your own reports
- Built-in engines, including RTL synthesis and flattening, to enable detailed implementation tests including clocking, reset and synchronization of asynchronous signals
- A Graphical User Interface (GUI) called SpyGlass Explorer
- A batch execution program for integration in corporate design flows

Additional features enable you to customize SpyGlass to meet your company’s unique requirements. Refer to the *SpyGlass Policy Customization Guide* for more details. Customization features include:

- A perl interface that provides extensive programmability, and customization of error messages, error severity, and other parameters
- User-created rules built as dynamically linked C libraries

- User-programmable reporting that enables you to generate message reports as screen displays, hard copies, files, e-mail, Web pages, and other formats

SpyGlass top-level is a full perl interpreter that allows extensive integration, enabling you to take advantage of the open-source libraries for Web programming, GUI management, and many other capabilities.

SpyGlass Explorer Overview

SpyGlass Explorer (SE) is the new graphical user-interface for SpyGlass. It provides enhanced UI features and uses Tcl Shell Interface of SpyGlass, `sg_shell`, at its core. The Tcl Shell Interface provides you with the following capabilities:

- Execute all Tcl commands or Tcl scripts directly in the GUI
- Interactively control the various stages of the design analysis session
- Perform easy design read, goal setup, goal execution, and customized reporting to focus on the intended violations

Following are the advantages of SpyGlass Explorer over Atrenta Console:

- **Integrated single process**: SpyGlass Explorer uses a single thread for GUI and batch processes, and therefore, takes care of the performance and synchronization issues experienced in Atrenta Console.
- **Shared data structures**: Since the GUI and batch processes are integrated in SpyGlass Explorer, the data structures and algorithms are shared. This saves disk input/output, therefore, improving timing and performance.
- **Faster access to data**: At the end of the SpyGlass run, GUI can reuse the data created in memory by the batch engine. This provides faster access to the data.
- **Fully integrated Tcl engine**: With a fully-integrated Tcl engine, SpyGlass Explorer can now leverage the Tcl command infrastructure. For more information, see *sg_shell*.
- **Full support for ADC/AWL:** SpyGlass Explorer provides full-capability to create ADC/AWL commands.
- **Fully compliant with the existing project/sg_shell flow:** To transition from Atrenta Console to SpyGlass Explorer, you do not need to setup the project file again.
- **Configurable standard view**: You can customize the appearance of your workspace in SpyGlass Explorer using dockable widgets.
- **Improved Message Display:** The Violations pane in SpyGlass Explorer provides additional information, such as, message details, test file name, and violating line number. Additionally, you can highlight the dynamic parts of the message.

- **Operational Similarity**: SpyGlass Explorer is functionally similar to Atrenta Console. This ensures easy transition from Atrenta Console to SpyGlass Explorer.

Features of SpyGlass Explorer

SpyGlass Explorer ()is graphical user interface of SpyGlass that enables you to read your design, run various rules to check for design issues, analyze the results, and debug the design issues.

Some of the features of SpyGlass Explorer are as follows:

- *Dockable Widgets*
- *Support for sg_shell*
- *Enhanced Design Setup View*
- *Dedicated Help Viewer*
- *Enhanced Design Setup View*
- *Enhanced Schematic Window*
- *Multi-line Highlight*
- *Improved Waiver Editor*
- *Object-Based Waivers*

Dockable Widgets

Enables you to customize the workspace by allowing you to drag-and-drop any widget anywhere on the screen. You can dock or undock a widget using one of the following options:

- Click and hold a widget title and drag to dock or undock
- Double-click on the widget title to dock or undock

To show/hide Views in SpyGlass Explorer, use one of the following methods:

- Select/deselect the views from the View->Windows menu
- Right-click on the menu bar and select/deselect the views

Also enables you to view multiple module, instance, or file trees.

Support for sg_shell

SpyGlass Explorer has an integrated sg_shell window, which enables you to perform UI actions using Tcl commands. For example, you can add a design file in SpyGlass Explorer using the following Tcl command:

```
set_option read_file -type
```

The actions performed in the GUI are also translated to the corresponding Tcl command. Also, these actions are saved in the *gui_command.log* file. For more information, see *sg_shell*.

Dedicated Help Viewer

SpyGlass Explorer provides integrated Help Viewer, which displays help for the options clicked in SpyGlass Explorer. Additionally, you can view SpyGlass Help documents using Help Viewer, without the need of an external browser.

Enhanced Design Setup View

The Design Setup view in SpyGlass Explorer provides a unified file view. where all input files, such as, Design Files, Constraints, Tech Libs, and HDL libs are listed. For more information, see *Design Setup*.

Enhanced Schematic Window

The Incremental Schematic window in SpyGlass Explorer includes the Minimap pane, which provides a concise view of the whole schematic. This facilitates in debugging very large designs.

In addition, the Incremental Schematic window includes the Properties pane. Hovering over any object or clicking any object in the schematic updates the values in the Properties pane. For more information, see *The Incremental Schematic Window*.

Multi-line Highlight

SpyGlass Explorer provides the capability of using multiple markers for violating cases in your RTL file. This helps in tracking and debugging the design.

Improved Waiver Editor

The **Waiver Editor** window in SpyGlass Explorer displays the key elements of the waived messages, such as, IP block, design unit, rule, and message, in separate columns.

For more information, see *The Waiver Editor Window*.

Object-Based Waivers

SpyGlass Explorer provides the capability of filtering and waiving messages based on the design objects.

For more information, see *Waiving Messages through GUI*.

Related Documents

The SpyGlass Explorer User Guide explains various tasks related to the SpyGlass flow. For more information, refer to the following SpyGlass documents:

- *SpyGlass Tcl Shell Interface User Guide*
- *SpyGlass GuideWare 2.0 User Guide*
- *SpyGlass Built-in Rules Reference Guide*

Getting Started

Before running SpyGlass, ensure that you have specified the mandatory environment variables. For details, see *Setting Environment Variables*.

This section explains the following topics:

- *Invoking SpyGlass Explorer*
- *Exiting SpyGlass Explorer*
- *The SpyGlass Explorer GUI*
- *Files/Directories Created in SpyGlass Explorer*
- *Running SpyGlass in Batch Mode*

Setting Environment Variables

You can set the following environment variables to customize SpyGlass:

Environment Variable	**Purpose**	**Default Value**	**Optional (O)/ Mandatory (M)**
SPYGLASS_HOME	Indicates SpyGlass home directory	`<your-installation_dir>/SPYGLASS_HOME`	O
SNPSLMD_LICENSE_FILE	Indicates SpyGlass license server	`<port-number>@<host name>`	M

Setting Up License Queuing

License queuing enables SpyGlass to check for the availability of SpyGlass licenses, in a multi-user environment.

When license queuing is not enabled, the requested SpyGlass process is aborted, in case of unavailability of a license.

To use the license queuing feature, perform the following steps:

1. Specify the -licqueue command on the command-line.

   or

   Set the value of the environment variable, SPYGLASS_ENABLE_LICENSE_QUEUE, to 1.

2. Set the `LICENSE_QUEUING_INTERVALS_IN_SECS` key in the setup file

   Following is the syntax of setting this key:

   ```
   LICENSE_QUEUING_INTERVALS_IN_SECS=<num1> <num2>
   ```

   Where:

   - *<num1>* refers to the maximum time period a process should wait for the license. The default value for this argument is 24000.
   - *<num2>* refers to the time interval after which a process should check if the license is free. The default value for this argument is 10.

   The process aborts in case license is mandatory for run and is not checked out successfully by the time out period. License is made available to first SpyGlass run that queues for a given license on the server and the run is aborted due to time out.

   Consider the following example:

   ```
   LICENSE_QUEUING_INTERVALS_IN_SECS = 100 10
   ```

   The above setting means that a process can wait for maximum 100 seconds to get the license. During this period, the process checks for the availability of license after every 10 seconds.

Invoking SpyGlass Explorer

You can invoke SpyGlass Explorer by:

- *Starting a New Session*
- *Loading a Previous Session*
- *Using the sg_shell Mode*

Starting a New Session

To start a new SpyGlass session, specify any of the following commands on the command-line:

- `%>spyglass`
- `%>spyglass -gui`

Once you specify any of the above commands, SpyGlass Explorer opens with a default project, as shown in *Figure 1*.

NOTE: *To launch Atrenta Console, specify **-gui=console** in the command-line.*

The details of the current GUI session are saved in a Project File, so that you can restart this session later with all the saved data. For more details on a project file, see *Project File*.

Loading a Previous Session

To load a previous session, load a saved project file using one of the following methods:

- Specify the name of a project file by using the `-project` command while invoking SpyGlass.

  ```
  %>spyglass -project <your_project_file>.prj
  ```

- Invoke SpyGlass Explorer, and then select a project by using the *File -> Open Project* menu option.

When you load a project, SpyGlass loads the session from the stage at which you earlier closed that session in the specified project.

Using the sg_shell Mode

Invoke SpyGlass Explorer from sg_shell by specifying the following command at the sg_shell prompt:

```
gui_start
```

NOTE: *To invoke Atrenta Console, specify* `gui_start -console` *at the* `sg_shell` *prompt.*

Specifying the above command displays the SpyGlass Explorer window, as shown in *Figure 1*. The SpyGlass Explorer window displays the Shell pane, wherein, you can work in the GUI using the Tcl commands. Additionally, when you work on the GUI, equivalent Tcl commands are also displayed in the Shell pane.

To exit sg_shell, specify the exit command at the `sg_shell` prompt.

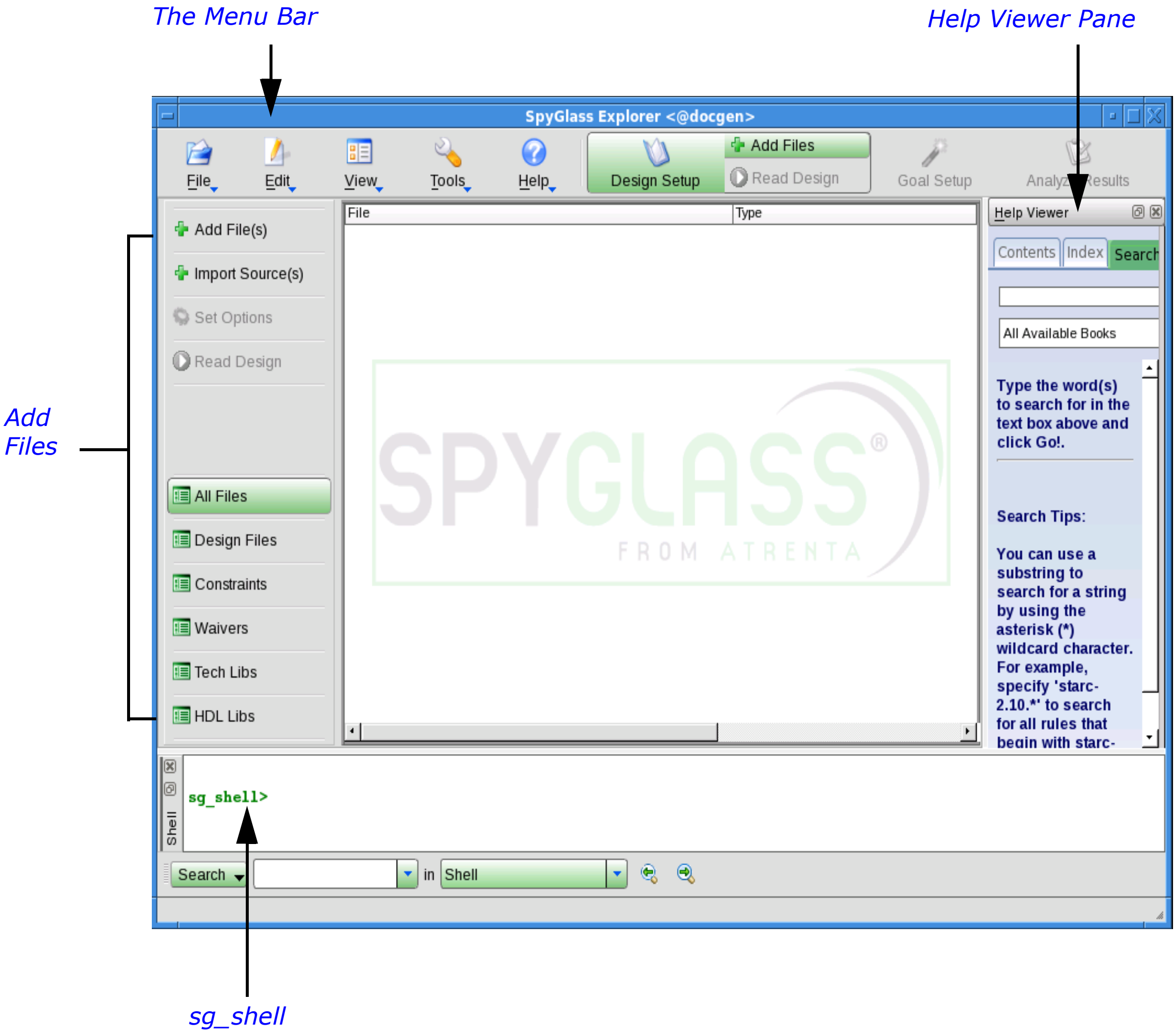
For more information on the sg_shell mode in SpyGlass, refer to *SpyGlass Tcl Shell Interface User Guide*.

Exiting SpyGlass Explorer

Before exiting an existing session of SpyGlass Explorer, save the project using the **File->Save** option. A project is saved in a project file (.prj).

The SpyGlass Explorer GUI

When you invoke SpyGlass Explorer, the SpyGlass Explorer window is displayed as shown in *Figure 1*:



Type the word(s) to search for in the text box above and click Go!.

Search Tips:

You can use a substring to search for a string by using the asterisk (*) wildcard character. For example, specify 'starc-2.10.*' to search for all rules that begin with starc-

FIGURE 1. SpyGlass Explorer Window

Files/Directories Created in SpyGlass Explorer

SpyGlass generates different files to log runtime information, such as reports and log files.

Some files/directories, such as *spyglass.log*, *spyglass_reports*, *spyglass_spysch*, and *spyglass.vdb* are generated every time you run SpyGlass.

Some files, however, are generated only when you use some special features. These files include *spyglass.db*, *WORK*, and *spyglass_cmdline_debug.log*.

These files are generated in the following directory:

```
<projectwdir>/<projectname>/<top_name>/<goal_name>/
<scenario_name>/
```

The spyglass.vdb file is generated in the `<projectwdir>/ projectname>/<top_name>/` directory.

Project File

A project file is a Tcl format file that enables easy reading and editing of this file outside the SpyGlass Explorer.

NOTE: *In SpyGlass, Tcl format is supported only in a project file. However, this format is not supported in SpyGlass GUI and the .spyglass.setup file. Therefore, for GUI and the .spyglass.setup file, you should specify commands in batch console format and not Tcl format.*

A project file (.prj) contains the following data about a particular SpyGlass session:

- Input HDL files and language settings
- Run options
- State of project (design read, goal setup, goal run, or results analysis)
- Constraint files and parameter settings for goals
- Status of goal setup and analysis

For details on creating a project file, see *Creating a Project File*.

When you load a project file in SpyGlass Explorer, the stage at which you last closed the session gets loaded with all the saved data. For more details on a project file, see *Structure of a Project File*.

By default, a project file is saved in the current working directory. You can specify a different directory by using the *File > Save Project As* menu option.

NOTE: *You can also create a project file using a Tcl (Tool Command Language) scripting interface. This enables you to configure a SpyGlass batch session without using the SpyGlass Explorer.*

This section explains the following topics:

- *Creating a Project File*
- *Structure of a Project File*
- *Example of a Tcl-based Project File*
- *Project Working Directory*
- *Project Current Working Directory*

Creating a Project File

A project file contains files, option settings, and parameter settings. The easiest way to create this file is to invoke the SpyGlass GUI, and save the settings into a project file.

SpyGlass saves the status information in a separate file. This file is not meant for user editing and is modified exclusively by the SpyGlass GUI.

A project file can be created and edited outside the SpyGlass GUI and can be loaded by using a normal text editor.

SpyGlass processes all the Tcl commands. However, while re-saving, it only retains the project-specific commands that are printed in a predetermined order. The built-in Tcl commands and structures are not stored for later saving.

A command can be used more than once. However, in case of conflicting values, only the last command issued is retained.

Structure of a Project File

A project file contains a header that displays the file type, version, copyright information, and date. In addition, the project file is organized into different sections, each having a group of commands.

A Tcl-based project file is divided into the following sections:

- *Data Import Section*
- *Common Options Section*
- *Goal Setup Section*

NOTE: *For using environment variables in the project file, standard Tcl syntax is supported. For example, use* `$env(DIR)` *to access a environment variable named* `DIR`*.*

Data Import Section

The Data Import Section includes commands that are used to add source files, HDL files, HDL libraries, and technology libraries. The following is the structure of the data import section:

```
##Data Import Section

read_file -type <type> <file-name>

set_option lib <logical-lib-name> <directory-path>

set_option libhdlfiles <logical-lib-name> { file-list }
```

The `read_file` command is used to add the source files. The `read_file` command takes the following arguments:

<type>

Specifies the type of source file. The type can be any of the following:

Value	Description
sourcelist	Specifies the source files in .spp or .f format
verilog	Specifies the Verilog (.v) files
vhdl	Specifies the VHDL (.vhdl/.vhd) files
def	Specifies the design exchange format (.def) files
sglib	Specifies the SpyGlass library (.sglib) files
gateslib	Specifies the gates library (.gateslib) files
lef	Specifies the library exchange format (.lef) files
plib	Specifies the power library (.plib) files
sgdc	Specifies the SpyGlass Design Constraints (.sgdc) files
waiver	Specifies the waiver files

<file-name>

Name of the source file.

The `set_option lib` command is used to specify the HDL libraries.

NOTE: *Refer to the Adding Files in GUI section for details on how to specify the HDL libraries using the SpyGlass GUI.*

The `set_option lib` command accepts the following values:

<logical-lib-name>

Specifies the logical name of the library.

<directory-path>

Specifies the path to the directory where the library is located.

The `set_option libhdlfiles` command is used to specify a mapping between the logical library and its HDL files. This option needs to be specified in the order of dependency of the libraries being compiled.

The `set_option libhdlfiles` command accepts the following values:

<logical-lib-name>

Specifies the logical name of the library.

<file-list>

Specifies the list of HDL files.

Common Options Section

The *Common Option Section* is used to set additional important options for design analysis that are not associated with any goal.

By default, the Tcl-based project file contains entries of only the modified design read options.

NOTE: *Arguments using $ or other meta characters that are meant to be passed directly to SpyGlass should be enclosed in curly brackets to prevent evaluation by the Tcl interpreter.*

Specifying a Report

You can specify reports and their formats by using various options of the `set_option` command, as discussed below:

- The following command specifies the name of the report to be generated:

```
set_option report <name>
```

NOTE: *This option is applicable for all goals that you specify in the Goal Setup Section. So if any goal specified in the Goal Setup Section does not generate the report specified by the set_option report <name> command, SpyGlass reports a fatal violation. In such cases, use the set_goal_option report <name> command in the Goal Setup Section to specify the name of goal-specific reports.*

- The following command specifies the name and location of the report file:

```
set_option reportfile <file-name>
```

- The following command specifies the maximum number of messages for sorted reports (simple, moresimple, and waiver reports):

```
set_option report_max_size <value>
```

- The following command specifies the report style:

```
set_option report_style <style-name>
```

Here, the *`<style-name>`* argument can accept any of the following values:

Value	**Description**
flat	Displays the report in an ungrouped format.
grouped	Groups the content of the report (for example, by goals).
display_msgid	Enables the display of the message index column in the reports.
hide_msgid	Hides the message index column in the reports
display_rulegroup	Allows grouping of rule messages in the reports by rule group.
display_sdcgroup	Groups messages of the sdc_data based rules in the SpyGlass Constraints solution based on the sdc_data specified in the SGDC file.

Value	Description
hide_rulegroup	Disallows grouping of rule messages by rule group in the report.
display_taggroup	Groups messages of the Ac_sync_group rules of SpyGlass CDC solution based on instance names or user-specified names.

Sorting Messages in Reports

SpyGlass generates the following reports sorted for better usability:

count	moresimple	simple	summary	waiver

By default, SpyGlass sorts messages in these reports by the following criteria (provided the criterion is applicable to the report):

- Severity Class (decreasing from FATAL, Error, Warning, and Info)
- Rule Name (alphabetical)
- Source HDL Filename (alphabetical)
- Line number (ascending)

You can modify the above sorting order by using the `sortrule` option of the `set_option` command, as shown below:

```
set_option sortrule <value>
```

Where, *`<value>`* can be specified in the following format:

`<language>`+*`<rule-name>`*+*`<sort-order>`*

Please note that there are no spaces between any of the values in the above format.

Details of different values of this format are given in the following table:

Value	Description
<language>	Refers to the rule language, which can be `Verilog`, `VHDL`, or `Mixed`. The rule for which message sorting order is being defined must be registered for the specified language. If the rule is specified for both languages, you can optionally specify only one of the languages if you want to specify the message sorting order for only that language.
<rule-name>	Refers to the rule name
<sort-order>	Refers to the user-defined sort order. This value is specified in the following format: *`<arg-number><arg-type><arg-sort-order>`*
Details of the <sort-order> value	
<arg-number>	Refers to the argument number To get the argument number, refer the rule message goal in the product ruledeck file. For example, the LPFSM16 rule of the SpyGlass Power Verify solution has the following message goal: `Attribute '%1' found on enumerated type '%2' used for encoding FSM states` Therefore, the first argument is the attribute name and is specified as 1. The second argument is the state variable name and is specified as 2. Refer to the corresponding rules reference document for an explanation of the rule message arguments.
<arg-type>	Refers to the argument type Argument types can be string (specified as s), numerals (specified as n), and enumerated types (specified as e).
<arg-sort-order>	Refers to the argument value sorting order as ascending (specified as a) or descending (specified as d)

Consider the following example:

```
set_option sortrule Verilog+R1+2sa+1nd+3e/val1/val2/val3
```

The above specification indicates that it defines the message sorting order of the `R1` rule in the Verilog mode. Further, the message sorting order is as follows:

1. First, sort the messages by `2sa`, that is, sort by the value of the second argument (2) which is a string argument (s) in ascending order (a).
2. For messages with the same second argument value, sort by `1nd`, that is, sort by the first argument (1) which is a numeral argument (n) in descending order (d).
3. For messages with the same first argument value, sort by `3e/val1/val2/val3`, that is, sort by the third argument (3) which is an enumerated type argument (e) based argument values `val1`, `val2`, and `val3` in that order.

In addition to the argument-based sorting orders described above, you can specify message sorting order by file (specified as f) and by line number (specified as l), both in either ascending order (specified as a) or descending order (specified as d). Thus, fd means to sort the messages by file name in descending order. And la means to sort by line number in ascending order.

Consider the `sortrule` specification in the above example with addition values as follows:

```
set_option sortrule Verilog+R1+2sa+1nd+3e/val1/val2/
val3+fd+la
```

This specification means that any sorting after the argument-based sorting will be done first by file names in descending order and then by line numbers in ascending order.

By default, the argument values are sorted in a case-sensitive manner. Specify `i` (for ignore case) to indicate that the argument values are to be sorted in a case-insensitive manner. Consider the following example:

```
set_option sortrule Verilog+R1+2sai+1nd+3e/val1/val2/
val3+fdi+la
```

The above specification indicates that argument-based sorting indicated by `2sa` and file-based sorting indicated by `fd` should be performed in a case-insensitive manner.

Goal Setup Section

The *Goal Setup* Section is used to add the setup information for goals (including rules and parameters), SGDC files, and reports.

Before you use any of the related commands, you must declare the goal scope. For details, refer to the *Specifying the Goal Scope* topic. If you specify a command without specifying the scope of a goal, a warning message is displayed in the session log that all such commands will be ignored.

The scope of each goal is confined within the scope of a current methodology. For details, see *Specifying a Current Methodology*.

Specifying the Goal Scope

To specify the goal scope, use the *current_goal* command as shown below:

```
current_goal <goal_path_and_name> [-top <module_name> | -
alltop ]
```

Where:

`<goal_path_and_name>`

(Mandatory) The relative path of the location where the goal is located.

`-top <module_name>`

(Optional) Name of the top-module. The `<module-name>` option defines the goal settings for the given top module only.

`-alltop`

(Optional) Use this option to define goal settings for cases in which no top-level design unit is specified, and goals are run for all top-level design units found in a design.

NOTE: *If you do not specify any of the `-top` or `-alltop` option in batch, goal settings are defined for a top module specified by the `set_option top` command in the project file. However, if the `set_option top` command is also missing in the project file, SpyGlass considers the behavior of `-alltop` option, that is, goal settings are defined for all the top modules.*
While saving a project file, the `current_goal` command is always written with either `-top <top>` or `-alltop` in the project file so that the settings remain a part of the current top, even if the user changes a top-level design unit for the project file.

For more information on the Tcl-based usage of the *current_goal* command, refer to the *current_goal* section of the *SpyGlass Tcl Shell Interface User Guide*.

Selecting a Goal Setting From a Project File

For a goal, a project file may contain multiple settings corresponding to different top-level modules. When you specify a goal to be executed in batch with the `current_goal` command, only one of the goal settings specified in the project file is selected for that goal depending upon a top-level module specified in a project file.

The following points discuss different settings that are selected when the following command is specified in batch mode:

```
spyglass -batch project Project-1.prj -goal G1
```

- The following goal setting is selected as a top-level module, `T1`, is specified by using the `set_option top` command and this module is the same as the top-level module specified with the `-top` command for the `G1` goal in the below highlighted setting:

  ```
  //Project-1.prj

  set_option top T1

  current_goal G1 -top T1
  set_parameter fa_modulelist {M1 M2}

  current_goal G1 -top T2
  set_parameter fa_modulelist {M3 M4}

  current_goal G1 -alltop
  set_parameter fa_modulelist {M1 M2 M3 M4}
  ```

- The following goal setting is selected for the `G1` goal in the following case:

  ```
  //Project-1.prj
  ```

```
current_goal G1 -top T1
set_parameter fa_modulelist {M1 M2}

current_goal G1 -top T2
set_parameter fa_modulelist {M3 M4}

current_goal G1 -alltop
set_parameter fa_modulelist {M1 M2 M3 M4}
```

- Consider the following case in which both the `set_option top` and `-alltop` commands are specified:

  ```
  //Project-1.prj

  set_option top T1
  current_goal G1 -alltop
  set_parameter fa_modulelist {M1 M2 M3 M4}
  ```

  In this case, there is no goal for which `T1` is specified as a top-level module. Therefore, the `G1` goal will be executed with default settings, and `-alltop` will not be considered in this case.

Specifying Goal-Specific Options

You can specify goal specific options that are applicable to the scope of the specified goal. To specify goal specific options, use the `set_goal_option` command. The syntax of this command is as follows:

```
set_goal_option <option> [<value>]
```

Where:

<option>

(Mandatory) Specifies the name of the option, such as `report`, `sdc2sgdcfile`, `sdc2sgdc`, etc.

<value>

Specifies the value of the option.

Specifying a Parameter

The command to setup a parameter is as follows:

```
set_parameter <param-name> <value>
```

Where:

<param-name>

Name of the parameter

<value>

Value of the parameter

NOTE: *The names of the parameter and their values are as they exist currently. Sanity check is not performed on parameter arguments.*

Enabling/Disabling an SGDC File for a Goal

You can enable or disable an SGDC file that has been added for a goal in the *Data Import Section* using the `read_file` command as follows:

- Enabling an SGDC file:

  To enable an SGDC file for a goal, use the `read_file` command as follows:

  ```
  read_file -type sgdc <file-name>
  ```

  Where:

  `<file-name>` is the name of the SGDC file.

NOTE: *If you are not specifying the* `read_file` *command in a particular goal scope, the specified SGDC file is considered as a global SGDC file and is enabled for all the goals (unless explicitly disabled for a particular goal).*

- Disabling an SGDC file:

  To disable the global SGDC file for a goal, use the `remove_file` command as follows:

  ```
  remove_file -type sgdc <file-name>
  ```

Defining Custom Goals

To define a custom goal, use the *define_goal* command, as shown below:

```
define_goal <goal_name> [-policy <product_list>]
{<goal_settings>}
```

Where:

<goal_name>

(Mandatory) Specifies the name of the custom goal.

-policy <product_list>

(Optional) Specifies a list of products that should be a part of the custom goal.

<goal_settings>

Specifies goal-specific settings or options, such as rules, parameters, overload-rule, reports, and so on. The settings must be specified within brackets.

The following is an example of using this command:

```
define_goal CUSTOM_GOAL_1 -policy { lint } {set_parameter abc
def}
```

For more information on the Tcl-based usage of the *define_goal* command, refer to the *define_goal* section of the *SpyGlass Tcl Shell Interface User Guide*.

Example of a Tcl-based Project File

The following is an example of a Tcl-based project file:

```
#!SPYGLASS_PROJECT_FILE
#!VERSION 3.0
#  ---------------------------------------------------
#  Copyright Atrenta, Inc 2009
#  Last Updated By: SpyGlass 4.3.0
#  Last Updated On Tue Aug 4 19:11:48 2009
```

```
#
#  -----------------------------------------------------

##Data Import Section

read_file -type sgdc netlist/constraints_netlist.sgdc
read_file -type verilog netlist/test_netlist.v
read_file -type gateslib lsi_10k.lib

##Common Options Section

set_option language_mode mixed
set_option projectwdir JUNK1
set_option projectcwd /atrenta/testcases
/DDR/case100
set_option enable_gateslib_autocompile yes

##Goal Setup Section

current_methodology /atrenta/spyint/integration/4.3.0/
RELEASE/SpyGlass-4.3.0/SPYGLASS_HOME/GuideWare/New_RTL
current_goal DDR_Flow/SDC_Equivalence_Dual_Design
set_goal_option reference_design_sgdc { rtl
/constraints_rtl.sgdc }
set_goal_option reference_design_projectfile Project-1.prj
set_parameter equiv_sdc_design_equivalence_file
equiv_file.txt

current_methodology /case100/New_RTL

current_goal DDR_Flow/SDC_Equivalence_Dual_Design

set_goal_option reference_design_projectfile /atrenta/
testcases/DDR/case100/Project-1.prj

set_goal_option reference_design_sgdc { /atrenta
/testcases/DDR/case100/rtl/constraints_rtl.sgdc }

set_parameter equiv_sdc_design_equivalence_file /
atrenta/testcases/DDR/case100/equiv_file.txt
```

Project Working Directory

A project working directory is the output directory of a project file.

Use the following command in a project file to specify a project working directory:

```
set_option projectwdir <dir-name>
```

NOTE: *You must have write permission in the directory specified when using this option.*

If the project file name is test.prj, SpyGlass creates a sub-directory, test, in the directory specified by this option. All run results are then stored inside this directory.

By default, the project working directory is the same directory in which the project file itself is stored.

Sub-Directories in a Project Working Directory

The project directory contains the following sub-directories:

- Design_Read

  This directory stores the results of SpyGlass run. It contains files, such as .vdb file, .log file, .out file, and SpyGlass reports.

- Run _Summary

  This directory contains information about the project, the goal run, and message information, such as message severity.

- *<module-name>*

  This directory contains all the results for a module, which is specified as a top-level block. There can be multiple module directories as a project file supports runs with different top-level blocks. The directory structure beneath the module name contains the same hierarchy structure of the methodology used for analysis. If no module has been specified as the top-level block, the name of the methodology will be used instead.

- WORK

  This directory contains the precompiled VHDL design units.

Project Current Working Directory

A project current working directory is the directory where a project was initially created.

This directory serves as an input directory from which various files, such as design files are picked. Any relative path inside a project file is assumed to be relative to this directory.

You may or may not have write permission for this directory.

Use the following command in a project file to specify this directory:

```
set_option projectcwd <directory-name>
```

While loading a project, if the current working directory is not same as the directory specified by this option, SpyGlass reports a warning and allows you to internally switch to the working directory specified by this option.

NOTE: *If you are working on sg_shell, errors are reported for such cases and the tool prompts you to change the directory to <projectcwd> and reopen the project.*

You can change the project current working directory if you want to move or copy a project to a new location and it is intended that the new project should pick up the design files relative to the new location.

File Generated in GUI

When you run SpyGlass Explorer, spyglass.out is generated.

This file is the screen-out file of SpyGlass Explorer in which runtime information (output) and rule-checking information is saved.

By default, the screen-out file name is spyglass.out and is saved in the current working directory along with spyglass.log. During runtime, however, if spyglass.log name or location is changed, the screen-out file is also changed accordingly.

Files/Directories Generated by Default

By default, SpyGlass creates the following files/directories:

- spyglass.log

  This file contains SpyGlass run details. These details include general information about SpyGlass run, such as SpyGlass version and the arguments passed for a particular run. This file is saved in the current working directory.

- spyglass_reports

  This directory is created in the current working directory. Various standard reports, such as simple.rpt, moresimple.rpt, inline.rpt, count.rpt, and sign-off.rpt are saved in this directory. This directory also contains product-specific reports.

- spyglass_spysch

  This directory contains internally generated files that are used for internal SpyGlass processing or to support GUI features. This directory is saved in the current working directory.

- spyglass.vdb

  This file stores all the violations generated during SpyGlass run.

Files Generated to Support Special Features

When you run SpyGlass with some special features, the following additional files/directories are generated to save data related to those features:

- spyglass.db

  This file is generated when design save-restore feature is used. It contains the synthesized view of the design during the first analysis run.

- WORK

  This directory is generated when you specify precompiled libraries during SpyGlass run. This directory contains the precompiled libraries.

  You can change the name of this directory by using the following command:

  ```
  set_option work <value>
  ```

- spyglass_cmdline_debug.log

  This file is generated in the current working directory when you specify the `set_option enable_cmdline_debug yes` command in the project file.

  Details of this file are summarized below:

❒ It contains a command-line trace log that enables you to understand how SpyGlass arrives at the final set of command-line options using the initial option set provided by you.

❒ It enables you to trace some GUI operations, such as parameter changes, goal selection, etc.

❒ It contains batch-mode command-line processing details. This file logs tracing of internal processing of command-line options; expansion of command files and goals; internal aliasing of certain command-line options; default options set by SpyGlass; configuration keys settings; and wildcard expansions of profile files, design files, paths specified by the `set_option I` {space-separated list of directory name } command, SGDC file, and waivers.

❒ When used with spyglass.log file, this file helps to understand the processing of various options.

SpyGlass generates a new file for each new run. However, if SpyGlass is run incrementally from the same invocation, the tracing data of subsequent runs is appended to the already existing file.

The trace done in GUI and batch mode is logged in different sections of the file, marked by `START` and `END` blocks.

Running SpyGlass in Batch Mode

To use SpyGlass in batch mode, specify the `-batch` command-line option as shown below:

```
spyglass -batch [-project <project-file>]
```

For more information on running SpyGlass in batch mode, see *Batch Mode in SpyGlass*.

To invoke sg_shell in batch mode, specify one of the following commands:

- sg_shell
- spyglass -shell

Working with SpyGlass Explorer User Interface

This section explains the following GUI options available in SpyGlass Explorer:

- *The Menu Bar*
- *The Methodology Configuration System Menu Bar*
- *sg_shell*
- *Design Setup*
- *Goal Setup*
- *Analyze Results*

The Menu Bar

The menu bar for SpyGlass Explorer (*Figure 2*) contains the following menus:

File Menu	*Edit Menu*	*View Menu*	*Tools Menu*	*Help Menu*

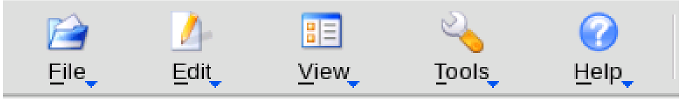


FIGURE 2. SpyGlass Explorer Menu Bar

File Menu

When you click the *File* menu, the following options appear:

File > New Project	*File > Open Project*
File > Edit Project	*File > Reload Project*
File > Close Project	*File > Save Project*
File > Save Project As	*File > Import Source(s)*
File > Exit	

File > New Project

This menu option is used to create a project (.prj) file.

When you invoke SpyGlass Explorer, a new Untitled -1.prj project file is opened automatically. You need to click *File > Save Project* to save the project.

You can also open a new project by pressing the <*Ctrl*>+<*N*> key combination on your keyboard.

File > Open Project

This menu option is used to open an already saved project (.prj) file.

When you select this menu option, the Open File window appears. You can navigate through your directory structure to locate the required goal file. Open the file by either double-clicking the file or selecting the file and then clicking *Open*.

You can also open an already saved project by pressing the *<Ctrl>+<O>* key combination on your keyboard.

NOTE: *When you open a saved project file, the stage that you were working the last time when you saved the project opens automatically.*

File > Edit Project

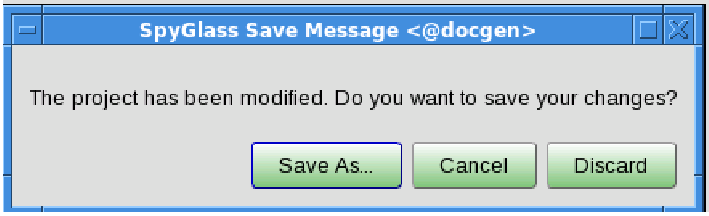
This menu option is used to edit the project. It opens the project in a text editor.

You can also edit an opened project by pressing the *<Ctrl>+<J>* key combination on your keyboard.

File > Reload Project

This menu option is used to reload the current project if that project has been modified through an external editor.

When you select this menu option, the following *SpyGlass Save Message* dialog box is displayed:



SpyGlass Save Message <@docgen>

The project has been modified. Do you want to save your changes?

FIGURE 3. Unsaved Project Warning Message

You can perform one of the following tasks from the *SpyGlass Save Message* dialog:

- Click *Discard* to reload the project and discard all the unsaved changes.

- Click *Save As* to open the *Save Project As* dialog, which allows you to save the current project in a different project file before reloading the current project.
- Click *Cancel* to abort the reload operation.

File > Close Project

This menu option is used to close an already open project.

File > Save Project

This menu option is used to save a currently open project.

When you click this menu option, the project is saved in the same location from where it was opened.

You can also save a project by pressing the *<Ctrl>*+*<S>* key combination on your keyboard.

File > Save Project As

This menu option is used to save the project with a different name and to a different location.

When you select this menu option, the *Save Project As* dialog appears as shown in the following figure:

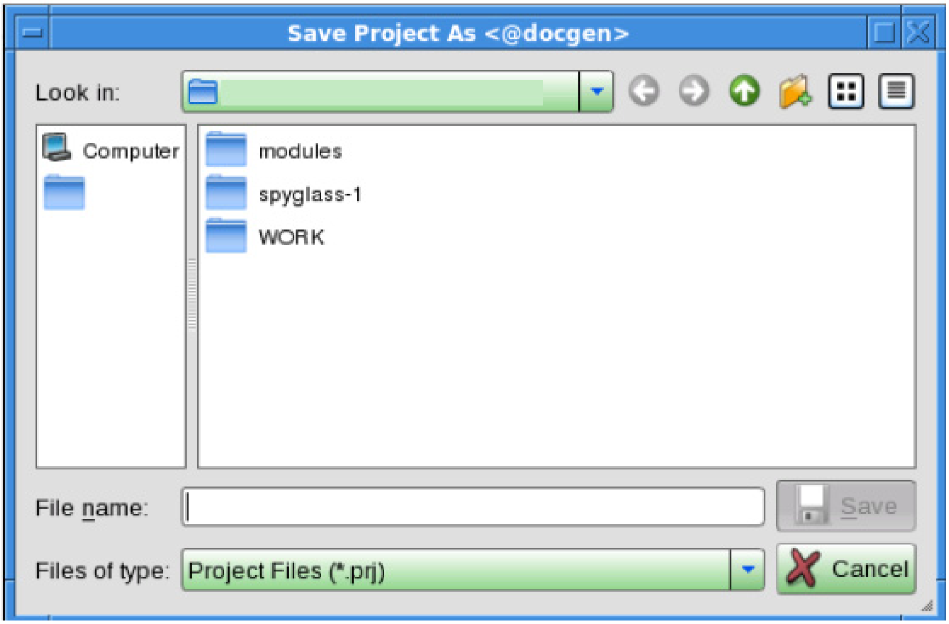


FIGURE 4. Save Project As

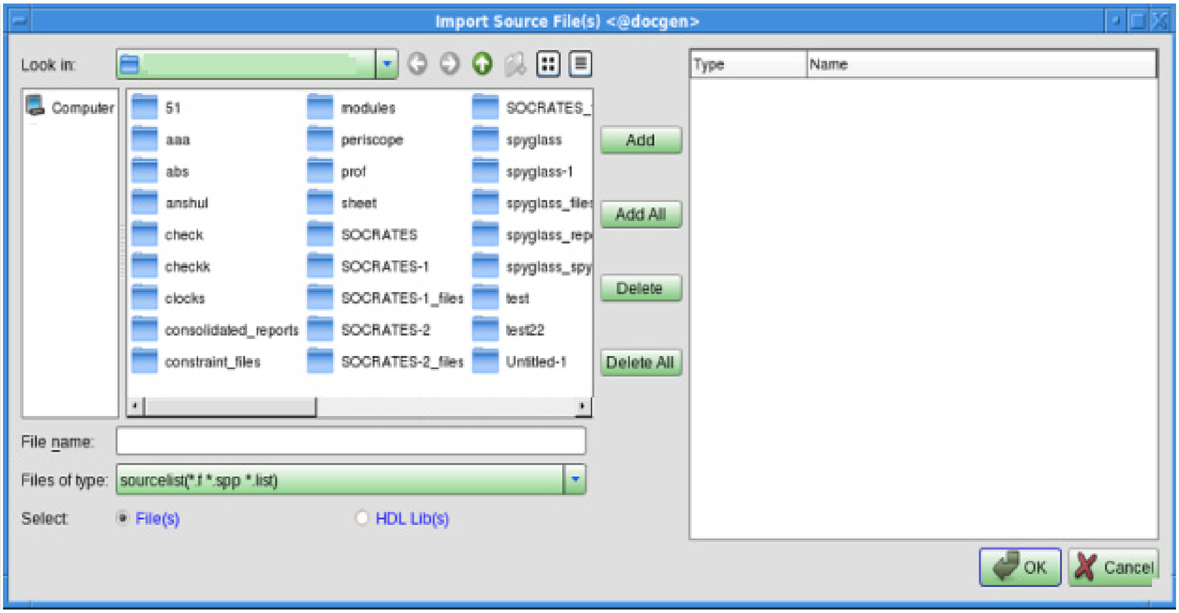
Specify the project name in the *File Name* text field and navigate to the directory where you want to save the project file. Next, click *Save* to save the project file in the selected directory.

NOTE: *If a project by the name Project -1.prj already exists, SpyGlass names the project as Project-2.prj and so on.*

File > Import Source(s)

This menu option is used to import an existing SpyGlass profile (.spp) file that contains the setup information from an earlier SpyGlass run, such as the current working directory, VHDL library mapping information, and so on.

When you select this menu option, the *Import Source File(s)* window appears, as shown in the following figure:



Import Source File(s) <@docgen>

Files of type: sourcelist(*.f *.spp *.list)

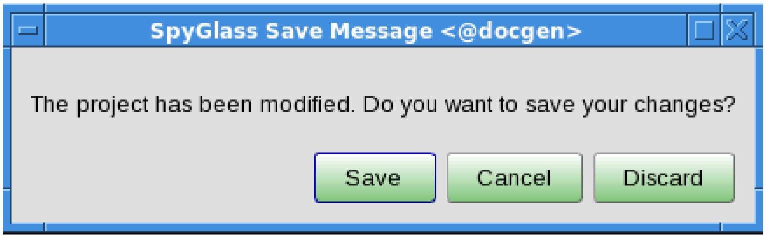
Select: ● File(s) ○ HDL Lib(s)

FIGURE 5. Import Source File(s)

You can navigate through your directory structure to locate the .spp file that contains the information you want to analyze.

File > Exit

The menu option is used to exit SpyGlass Explorer. When you select this option, SpyGlass Explorer prompts you to save the current project.



SpyGlass Save Message <@docgen>

The project has been modified. Do you want to save your changes?

FIGURE 6. SpyGlass Exit Warning

Click *Save* to save the changes and exit SpyGlass. Click *Discard* to exit SpyGlass without saving the changes. Click *Cancel* to return to the SpyGlass Explorer user interface.

You can also exit SpyGlass by pressing the <*Ctrl*>+<*Q*> key combination on your keyboard.

Edit Menu

When you click the *Edit* menu, the following options appear:

Edit > Clear All Selections	*Edit > Find*	*Edit > Preferences*

Edit > Clear All Selections

This menu clears all the selections in the Results View. This option is only visible when the Results View is visible.

Edit > Find

This menu option displays the Search bar, which allows you to search for a term in the Shell pane.

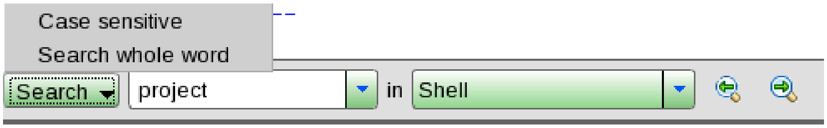


FIGURE 7. Search Utility

Click on the Search button to chose one of the following search options:

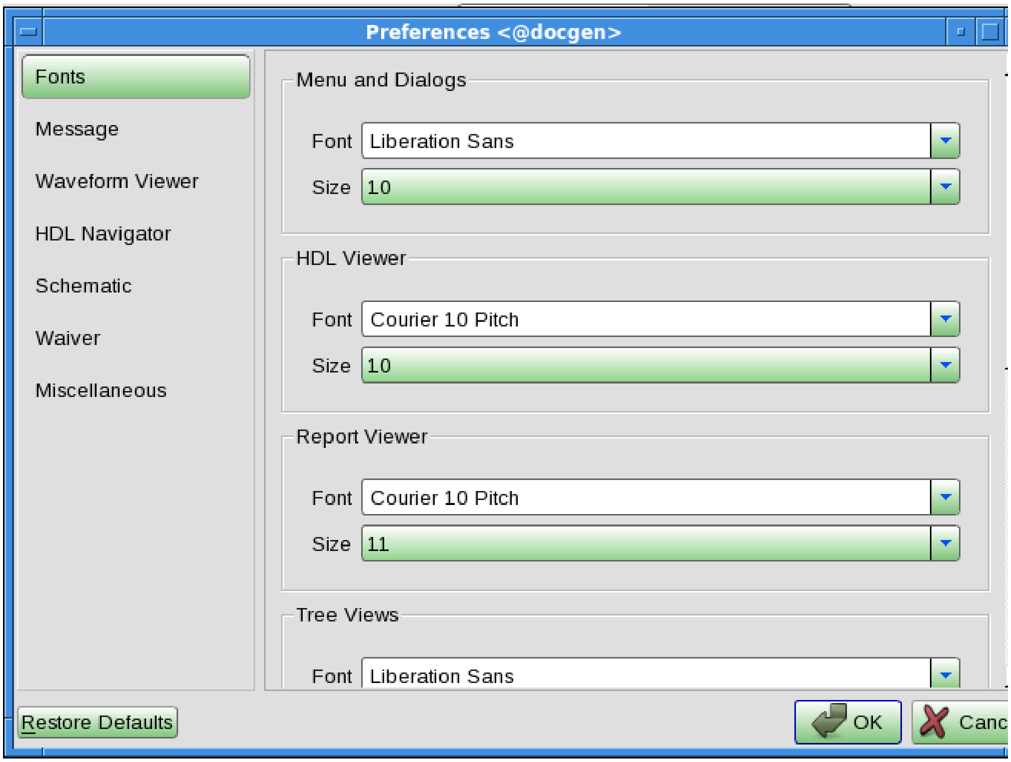
- Case-sensitive
- Search whole word

The Left (⇦)and Right (⇨) icons allow you to navigate through the search in the Shell pane.

You can also display the Search bar by pressing the <*Ctrl*>+<*F*> key combination on your keyboard.

Edit > Preferences

This menu displays the Preferences dialog box (*Figure 8*). For details on the Preferences dialog box, see *Tools > Preferences*.


Preferences <@docgen>

Fonts

Message

Waveform Viewer

HDL Navigator

Schematic

Waiver

Miscellaneous

Menu and Dialogs

Font Liberation Sans

Size 10

HDL Viewer

Font Courier 10 Pitch

Size 10

Report Viewer

Font Courier 10 Pitch

Size 11

Tree Views

Font Liberation Sans

FIGURE 8. Preferences Dialog Box

You can also view the Preferences dialog box by pressing the <*Ctrl*>+<*R*> key combination on your keyboard.

View Menu

When you click the *View* menu, the following options appear:

View > Design Setup	*View > Goal Setup*
View > Analyze Results	*View > Skins*
View > Styles	*View > Results View*
View > Windows	

View > Design Setup

This menu displays the *Design Setup* page.

View > Goal Setup

This menu displays the *Goal Setup* page.

View > Analyze Results

This menu displays the *Analyze Results* page.

View > Skins

This menu displays the following skins available to modify the color schemes for the SpyGlass Explorer GUI:

- Atrenta
- Console
- Black

View > Styles

This menu displays the following styles available to change the appearance of the SpyGlass Explorer GUI:

- Cleanlooks
- Motif
- CDE
- Plastique

View > Results View

This option displays the following views available to change the window placement in the results view:

- New
- Classic

View > Windows

This option allows you to show/hide the following panes in SpyGlass Explorer:

- Debug Bar
- Search Bar
- Shell
- Violations
- Waiver Tree
- HDL Navigator
- Files
- Constraints
- Modules
- Instances
- Help Viewer

Tools Menu

When you click the *Tools* menu, the following options appear:

Tools > Modular Schematic	*Tools > Incremental Schematic*
Tools > Waiver Editor	*Tools > Text Viewer*
Tools > View Logfiles	*Tools > Aggregated Project Results*
Tools > Datasheet Report	*Tools > Dashboard Report*

Tools >Methodology Configuration Window	*Tools > Push Button Flow Config File Editor*
Tools > Preferences	

Tools > Modular Schematic

This menu option is used to display the Modular Schematic window. The Modular Schematic window displays a hierarchical schematic layout of the modules analyzed by SpyGlass.

For details of the Modular Schematic window, see The Modular Schematic Window.

Tools > Incremental Schematic

This menu options is used to display the Incremental Schematic window. The Incremental Schematic window displays the selected portions of (flattened) design schematic across hierarchical boundaries.

For details of the Incremental Schematic window, see The Incremental Schematic Window.

Tools > Waiver Editor

This menu option is used to waive messages displayed in the *Results* pane.

When you select this menu option, the *SpyGlass Waiver Editor window* appears. Alternatively, hit the <*F*> key on the keyboard to display this window.

For details on the Waiver Editor window, see The Waiver Editor Window.

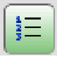
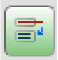
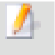
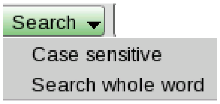
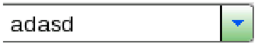
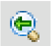
Tools > Text Viewer

This menu option is used to view text files in the SpyGlass Text file viewer.

When you select this menu option, the Open File dialog box is displayed, which allows you to select the text file to be viewed. The SpyGlass Text file viewer appears displaying the selected text file. If the SpyGlass Text file viewer is already open displaying a text file selected earlier, the new text file replaces the earlier text file.

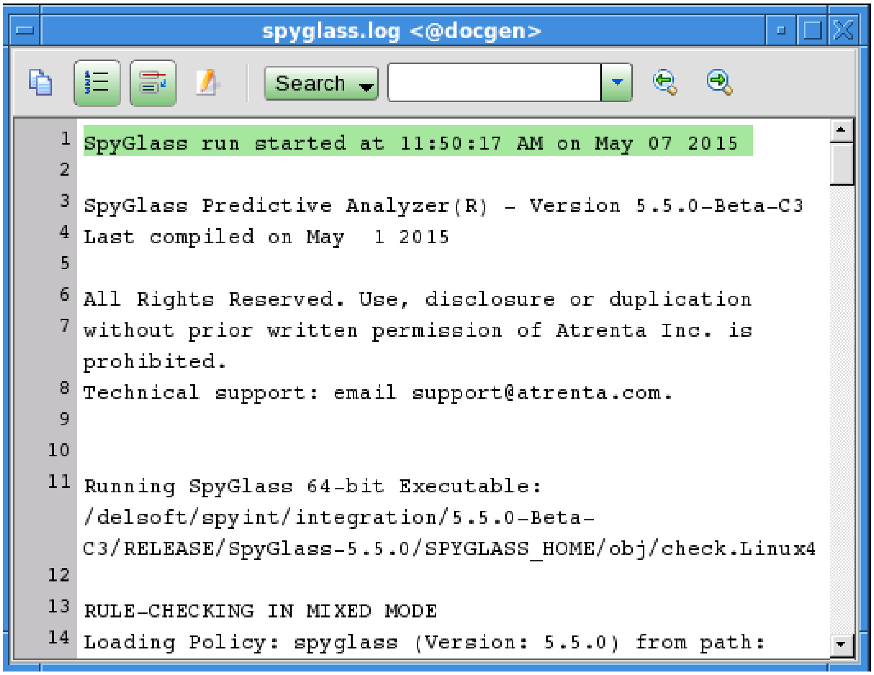
The following table lists the options available in the Text file viewer window:

TABLE 1 Text file viewer options

Option	Description
	Enables you to toggle the line numbering in the text viewer
	Enables you to toggle the text wrapping
	Opens the Text Editor window
	Allows you to select one of the available search options
 adasd	Allows you to specify the search text
	Allows you to search the previous instance of the searched term
	Allows you to search the next instance of the searched term

Tools > View Logfiles

This menu option is used to view to the log files created during the SpyGlass run. When you select this option, a sub-menu displaying the various log files generated during the run is displayed. Select a log file to view it in the *LogFile Viewer*, as shown in the following figure:



```
SpyGlass run started at 11:50:17 AM on May 07 2015

SpyGlass Predictive Analyzer(R) - Version 5.5.0-Beta-C3
Last compiled on May  1 2015

All Rights Reserved. Use, disclosure or duplication
without prior written permission of Atrenta Inc. is
prohibited.
Technical support: email support@atrenta.com.



Running SpyGlass 64-bit Executable:
/delsoft/spyint/integration/5.5.0-Beta-
C3/RELEASE/SpyGlass-5.5.0/SPYGLASS_HOME/obj/check.Linux4

RULE-CHECKING IN MIXED MODE
Loading Policy: spyglass (Version: 5.5.0) from path:
```

FIGURE 9. LogFile Viewer

Tools > Aggregated Project Results

This menu option is used to view aggregate report that contains the combined results from multiple single-user projects.

For details on this report, see *Generating Aggregate Reports*.

Tools > Datasheet Report

This menu option is used to view the DataSheet report. For details on this report, see *Generating DataSheet Report*.

Tools > Dashboard Report

This menu option is used to view the Dashboard report. For details on this report, see *Generating Dashboard Report*.

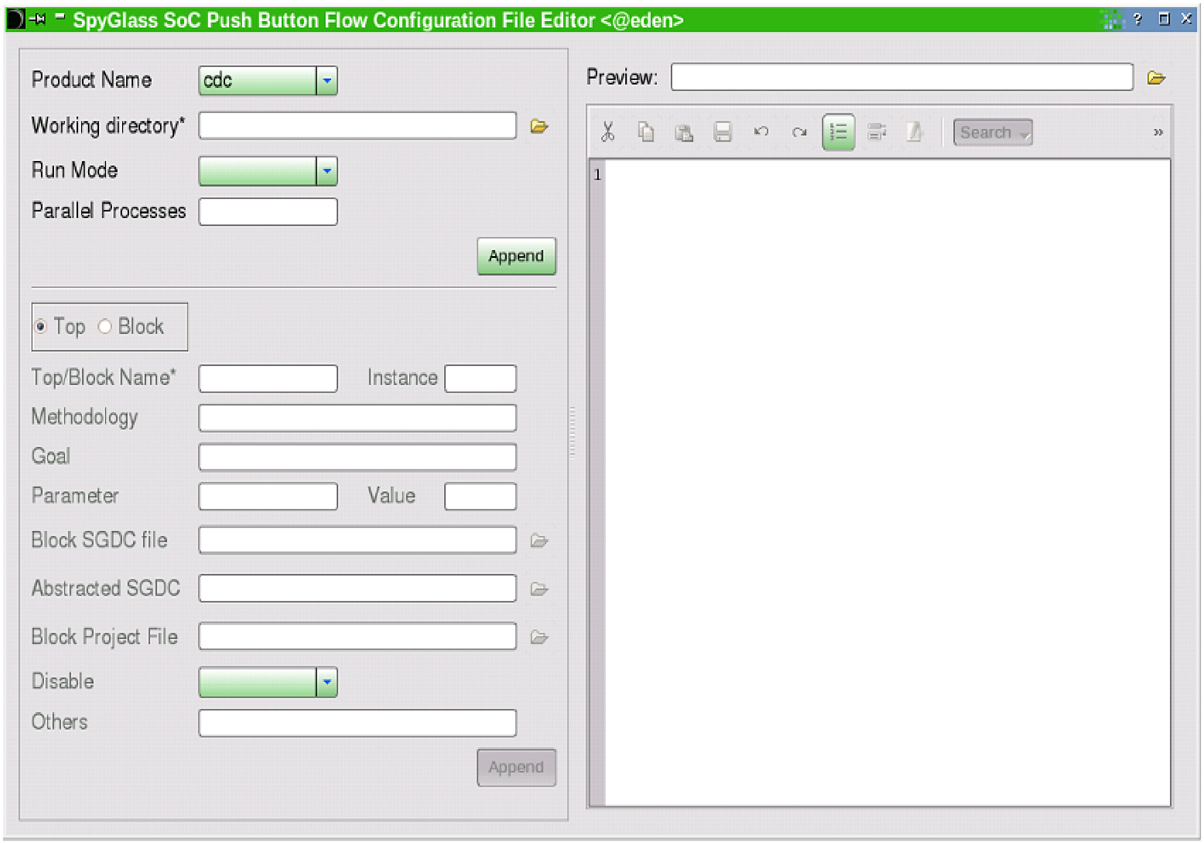
Tools >Methodology Configuration Window

This menu option is used to open the Methodology Configuration System window. For details, see *Working with Methodologies*.

Tools > Push Button Flow Config File Editor

This menu option is used to generate configuration file to run the auto_soc Tcl command.

The following figure shows a sample SoC Push Button Flow Configuration File Editor:



Product Name: cdc

● Top ○ Block

FIGURE 10. SoC Push Button Flow Configuration File Editor

In the above figure, the Push Button Flow Configuration File Editor window is divided into two panes:

- **Form pane**: Fill up the relevant form fields to create a Tcl file. After filling the first four fields of the form, click the Append button to add this information to the Tcl file and enable the remaining form. Select either Top or Block option, if you want to add information for top modules or blocks, respectively. Click Append to add the specified information to the Tcl file. You can specify information for multiple top and block modules.
- **Preview pane**: Provides you a preview of the Tcl file. You can also edit the Tcl file in the preview area. Also, you can change the Tcl file or

create a new one using the browse button provided above the preview area.

Tools > Preferences

This menu option is used to set various preferences in SpyGlass Explorer:

When you select this option, the *Preferences* dialog appears. You can view the following pages in this dialog:

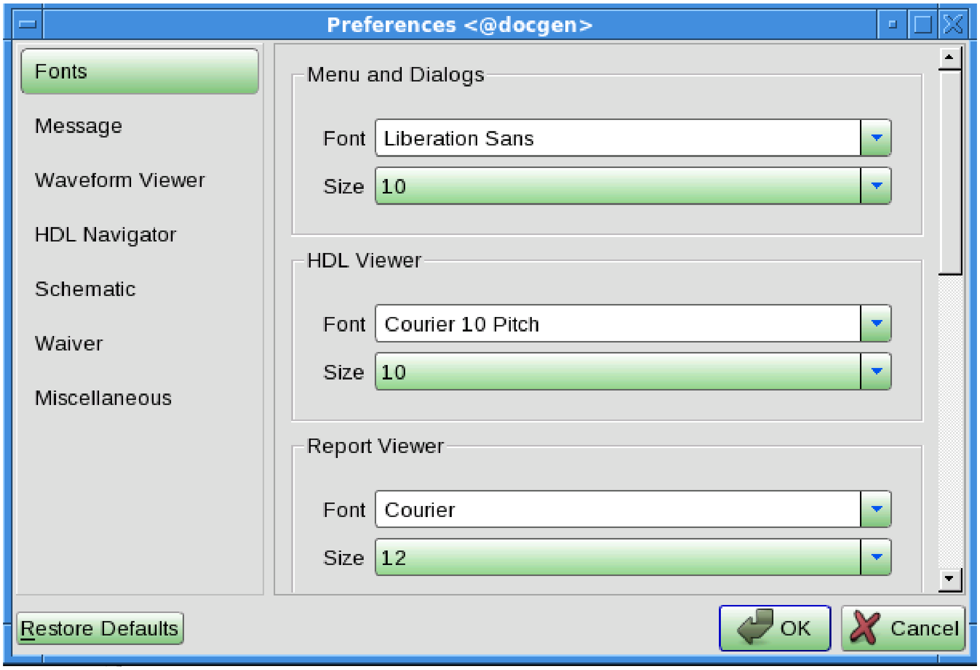
Font Page	*Message Page*
Waveform Viewer Page	*HDL Navigator Page*
Schematic Page	*Waiver Page*
Miscellaneous Page	

Font Page

Use the Font page to set a font style and size of text appearing in the following GUI options:

- Menu and Dialogs
- HDL Viewer
- Report Viewer
- Tree Views
- Table Views
- Help Window
- Shell Window

The following figure shows the Font page:



Menu and Dialogs

Font: Liberation Sans

Size: 10

HDL Viewer

Font: Courier 10 Pitch

Size: 10

Report Viewer

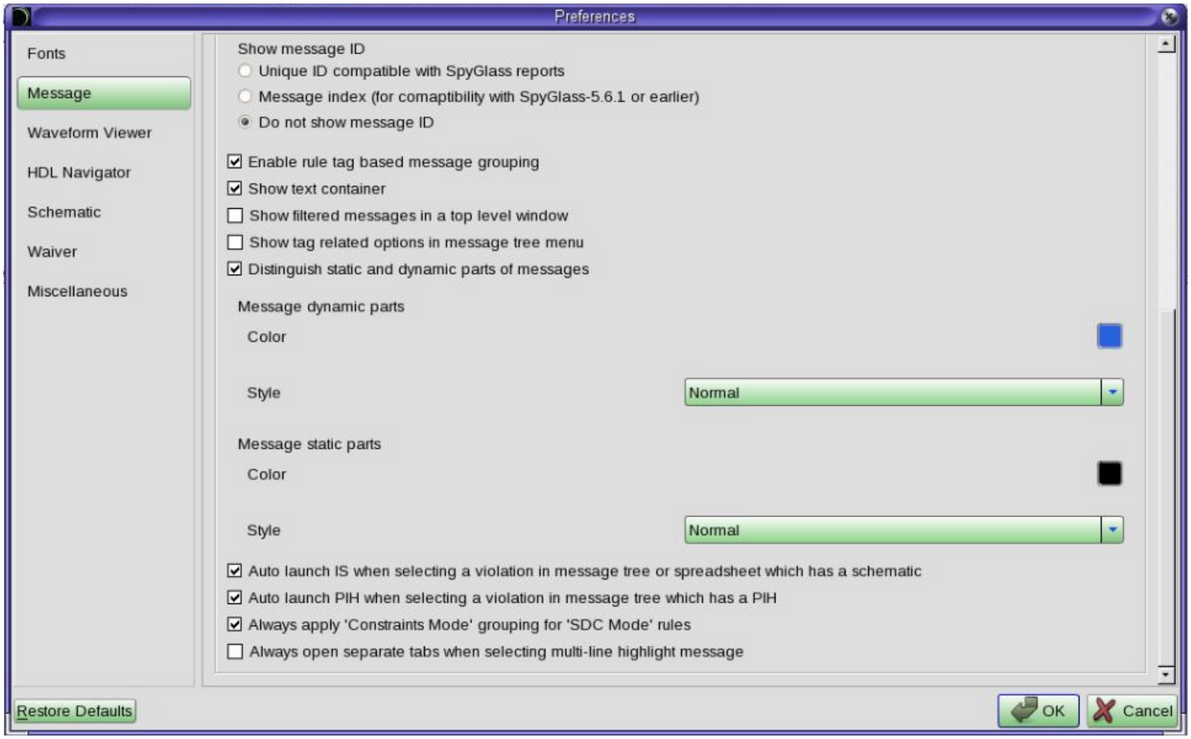
Font: Courier

Size: 12

FIGURE 11. The SpyGlass Preferences Window - Font Page

Message Page

Use the Message page to set the color for each severity class and configure other advanced settings, such as, color and style for message parts, and so on.



Show message ID

○ Unique ID compatible with SpyGlass reports
○ Message index (for comaptibility with SpyGlass-5.6.1 or earlier)
● Do not show message ID

☑ Enable rule tag based message grouping
☑ Show text container
□ Show filtered messages in a top level window
□ Show tag related options in message tree menu
☑ Distinguish static and dynamic parts of messages

Message dynamic parts

Color 🟦

Style Normal

Message static parts

Color ■

Style Normal

☑ Auto launch IS when selecting a violation in message tree or spreadsheet which has a schematic
☑ Auto launch PIH when selecting a violation in message tree which has a PIH
☑ Always apply 'Constraints Mode' grouping for 'SDC Mode' rules
□ Always open separate tabs when selecting multi-line highlight message

FIGURE 12. The SpyGlass Preferences Window - Message Page

You can set the following options:

Severity Class

You can set the color for each severity class so that a message of the given severity class is highlighted in the selected color. To set the color, click the displayed color and the Choose color dialog appears:

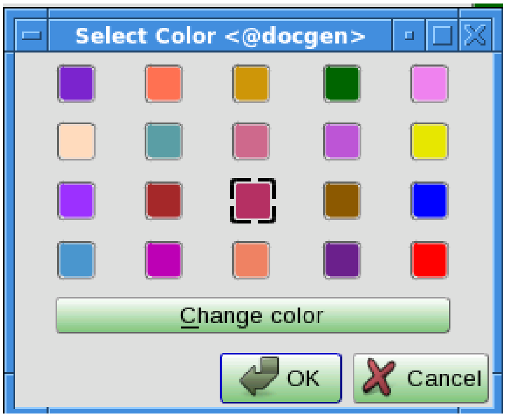


FIGURE 13. The SpyGlass Preferences Window - Choose Color

Choose a highlighting color from the predefined color palette or click *Change Color* to select more colors for the palette. Click *OK* to apply the selected color.

The sample text is also displayed for each severity class.

Set the color scheme to your liking and click *OK* to apply the color scheme to the HDL Viewer.

Wrap messages in the violation/waiver tree

Set this option to wrap the text in the violation or waiver tree.

Vertical Spacing between messages, when wrapped (pixels)

Specify a positive integer value to set spacing between messages when text wrap is applied.

Show tool-tip

Set this option to enable or display the information that is displayed as a tool-tip in the Msg Tree window, Msg Summary window and so on.

Show VDB path in message tree

Set this option to display the path of the violation database in the message tree.

`Display SDC Parser messages in dumped order`

Set this option to ensure that the SDC parser messages are displayed in their dumped order in the message tree. If this option is set, then message tree would internally chose the grouping `Constraints Mode-I` instead of `Constraint Mode` where messages would be displayed by following grouping criteria:

```
SDCMode -> BuiltIn -> Messages
```

`Show message ID`

Set this option to enable or disable unique hexadecimal message IDs in the relevant sections of the *Violations Pane* (by default, this is enabled). These indexes (when enabled) allow you to cross-probe between the *Message Window* and other SpyGlass Explorer windows.

`Show text container`

Set this option to automatically display the text window on selecting the violation, if the violation has associated text data.

`Show filtered messages in a top level window`

Set this option to show the message collection in a top-level window. By default, it shows it in the tab next to other violations.

`Show tag related options in message tree menu`

Set this option to turn on tagging of messages.

`Distinguish static and dynamic parts of messages`

Set this option to highlight the variable arguments of a violation in a different color for better readability.

`Auto launch IS when selecting a violation in message tree or spreadsheet which has a schematic`

Set this option to automatically display the Incremental Schematic window on selecting the violation, if the violation has associated data in Incremental schematic.

`Auto launch PIH when selecting a violation in message tree which has`

a PIH

Set this option to automatically display the PIH window on selecting the violation.

Always apply 'Constraints Mode' grouping for 'SDC Mode' rules

Set this option to organize the violations according to the constraint mode, when SpyGlass Constraints product is run.

Always open separate tabs when selecting multi-line highlight message

Set this option to open multiple tabs when you need to highlight multiple lines in a single file, numerically equating to number of distinctive highlighting lines.

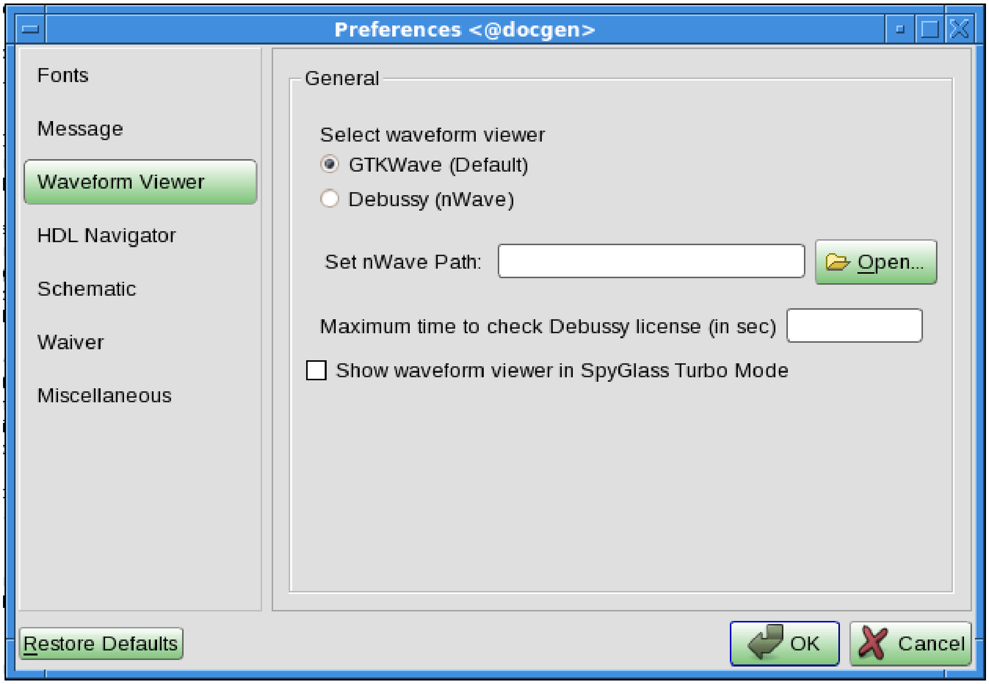
If more than one file needs to be cross-probed, they are also opened while retaining the above property.

Files on each tab should be scrolled to display a different highlighted line.

If you de-select this option, while multiple tabs of same file are opened, the excess tabs will be automatically closed when repeating the same task.

Waveform Viewer Page

Use the Waveform Viewer Page to set the waveform viewer to view the waveform associated with a message.



General

Select waveform viewer

● GTKWave (Default)

○ Debussy (nWave)

Set nWave Path:

Maximum time to check Debussy license (in sec)

□ Show waveform viewer in SpyGlass Turbo Mode

FIGURE 14. The SpyGlass Preferences Window - Waveform Viewer Page

You can set the following waveform options:

Select Waveform Viewer

You can select either *GTKWave (default)* viewer or the *Debussy (nWave)* waveform viewer.

Set nWave Path

If you select Debussy nWave waveform viewer, you need to set the path of nWave in the corresponding field. Click the *Open* button to browse to the directory location of nWave executable and select the required filename.

Maximum time to check Debussy license (in secs)

You can also specify the maximum time to check the *Debussy* license.

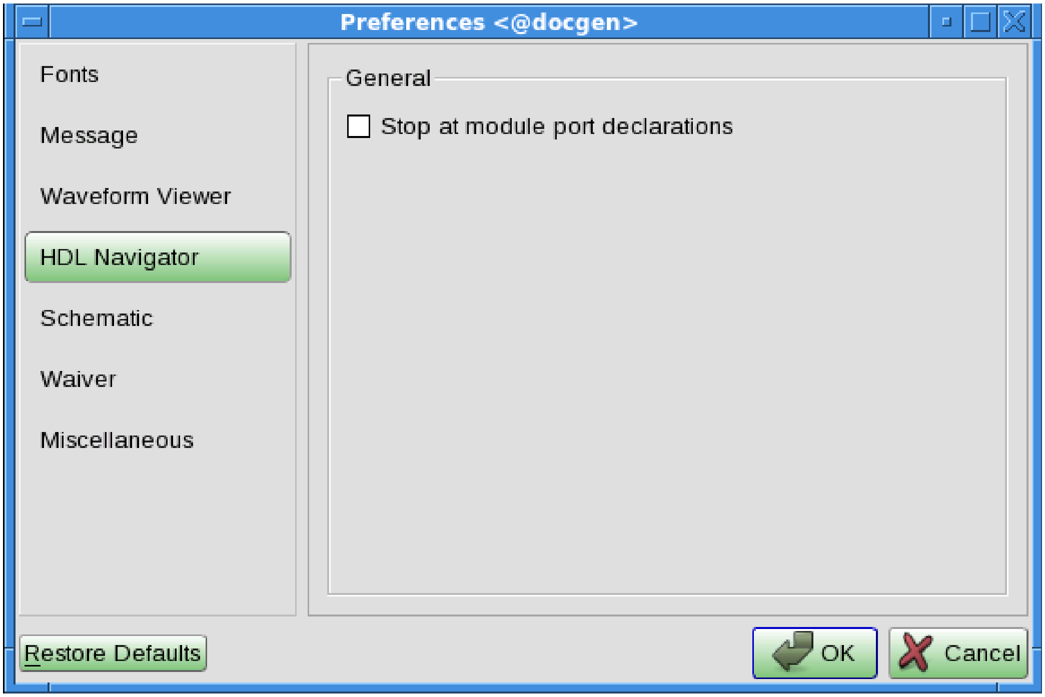
NOTE: *You need to have a license to run the Debussy nWave waveform viewer. Atrenta does not provide any license to run the Debussy waveform viewer.*

`Show waveform viewer in SpyGlass Turbo Mode`

Set this option to display waveform viewer when SpyGlass is run in Turbo mode.

HDL Navigator Page

Use the HDL Navigator page to set the *Stop at module port declarations* option. This option allows you to restrict the Loads and Drivers results such that while traversing the input/output cone, if module boundary is found before any other instance, the declaration is returned as the load/driver. When this option is not selected, even if module boundary is found, the traversing continues until an instance or a primary port (outside the module) is found.

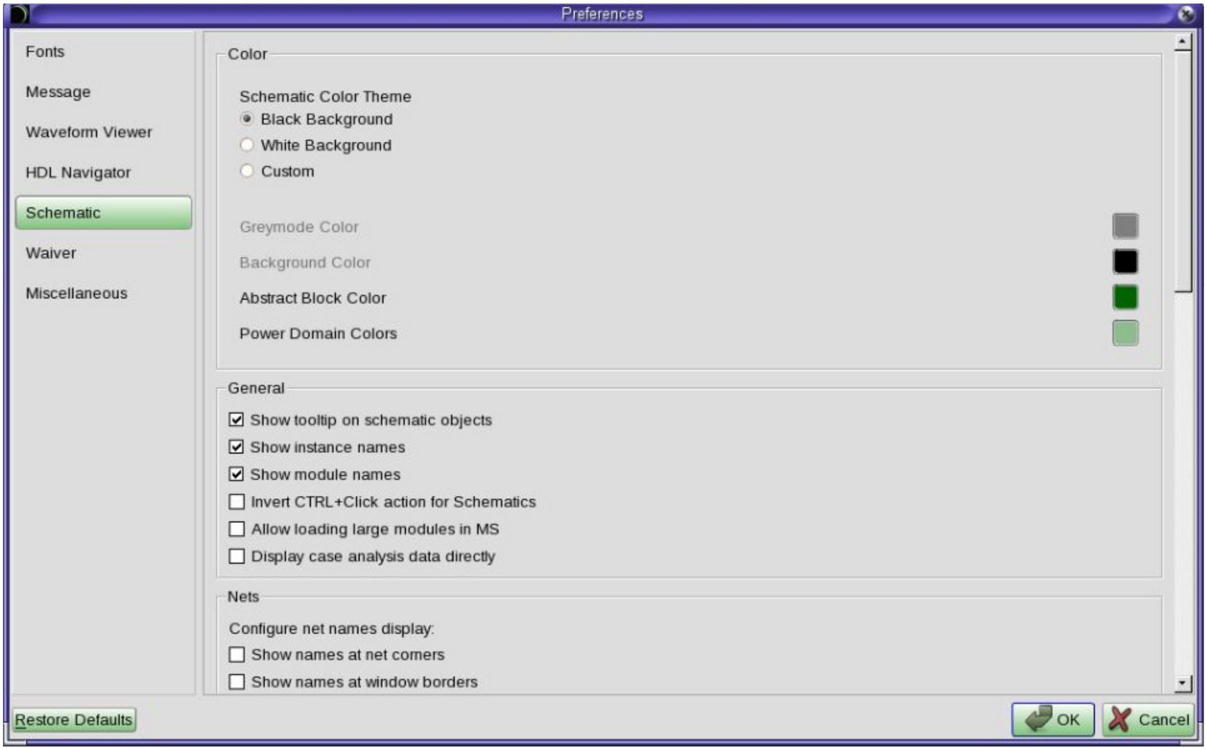


□ Stop at module port declarations

FIGURE 15. The SpyGlass Preferences Window - HDL Navigator Page

Schematic Page

Use the Schematic page to set the appearance of the schematic windows.



Preferences

Fonts
Message
Waveform Viewer
HDL Navigator
Schematic
Waiver
Miscellaneous

Color

Schematic Color Theme
- ● Black Background
- ○ White Background
- ○ Custom

Greymode Color

Background Color

Abstract Block Color

Power Domain Colors

General
- ☑ Show tooltip on schematic objects
- ☑ Show instance names
- ☑ Show module names
- □ Invert CTRL+Click action for Schematics
- □ Allow loading large modules in MS
- □ Display case analysis data directly

Nets

Configure net names display:
- □ Show names at net corners
- □ Show names at window borders

FIGURE 16. The SpyGlass Preferences Window - Schematic Page

You can set the following schematic preferences:

Schematic Color Theme

Choose the highlighting color of the selected object in the schematic windows by clicking the displayed color and by selecting a different color from the color palette.

Greymode Color

Choose the color of the schematic display (except the message path) when a message is cross-probed to the schematic windows (from the Message tree view in Msg Tree page of the Message window) by clicking the displayed color and selecting a different color from the color palette.

Background Color

Choose the background color of the schematic windows by clicking the displayed color and selecting a different color from the color palette.

Abstract Block Color

Allows you to set the color to highlight abstracted blocks in the SoC Abstraction flow.

Power Domain Colors

You can change the default colors used for Power and Voltage domains displayed in the Schematic by clicking on a color on the right. This opens the color palette, which allows you to change individual colors similar to the highlighted color palette.

SpyGlass now provides a new set of palette with lighter colors added to reduce color interference and better visibility. If you want to use the old palette and overwrite the new color set, added the following setting in the .spyglass.setup file:

```
SDE_CONFIG_OPTIONS=initial_preference:spy_AllPDColors=#5f7d9
eb8a082 #cdd268728978 #ba9f553fd3b5 #ffffffffffff
#9ba53021ffff #a5a12a3d2a3d #b0b030306060 #8b855a5e0000
#00000000ffff #4f5c947acdd2 #bdf30000b0a3 #eeee82826262
#686822228b8b #8faabcab8f93 #645c9592edd2 #bdb1b76f6b5d
#f0e4f8edffff #7f7cffffd4d1 #1e3590e8ffff #ffffd8920000
```

You can also specify the colors of your choice using the above command. However, the list specified should have 20 color items.

Show Tool-tip On Schematic Objects

Set this option to view a tool-tip whenever the cursor is placed over an object in the schematic windows. The tool-tip displays the full name of the object (net, pin, port, pin-bus, or instance). By default, this option is set and the tool-tip is displayed whenever the cursor is placed over an object in the schematic windows.

Show Instance Names

Set this option to turn on/off the display of instance names in the schematic windows.

NOTE: *This option is useful because in a netlist design, the debugging of a violation message using the schematics becomes difficult if the instance names are too long. In such a scenario, part of the instance name of the first gate is overwritten by the instance name of the next gate and thus the information becomes unreadable.*

Show Module Names

Set this option to turn on/off the display of module names in the schematic windows.

Invert CTRL+Click action for Schematics

Set this option to invert current Ctrl+click behavior. Then, schematic objects are added to the selection, when clicked. You cannot de-select an object by re-clicking on it.

However, when you press the Ctrl key and click on an object, other objects are de-selected.

The preference value is saved throughout sessions.

Allow loading large modules in MS

By default, if a module being viewed in modular schematic is large and expected to take significant time, it is not loaded in the modular schematic. The rationale is that the modular schematic would be too large and complex to be of any real value.

Set this option to view the modular schematic for such large modules.

Display case analysis data directly

Set this option to display data for informational rules, such as *Info_testmode*, *Info_Case_Analysis*, and *Show_Case_Analysis*.

Such informational rules generate a lot of data in the schematic, which results in large loading time and high memory usage. Therefore, by default, this option is turned off and the data for such informational rules is not loaded in the schematic.

If this option is turned off and you try to load the violation of such rules, SpyGlass displays the Huge Schematic Data warning message.

`Show names at net corners`

Displays net names at net corners.

`Show names at window borders`

Displays net names at a window boundary if a net does not fit in the current display.

`When double-clicked on a PinBus in IS`

You can select one of the following options:

- Expand all the bits
- Let the user to select the bits to be expanded
- Continue the probe, and ask the user only when required

`Expand to all loads/drivers on double clicking`

Controls the behavior when double-clicking on nets/pins in the Incremental Schematic.

When you set this option click on an output pin, then all the gates in the immediate fan-out are loaded. Similarly, when clicking on input pin, all the gates in immediate fan-in are loaded.

If this option is not selected, then only one gate is loaded at a time.

`Show symbols on unconnected pins of IS`

For violations which use schematic symbols, there is some performance penalty in showing the symbols on pins whose connected net is not loaded into the Incremental Schematic.

Set this option to display all symbols irrespective of the net connection being loaded or not.

`Show symbol on constrained objects`

For any object where a constraint is set, ⊕ symbol is shown in Incremental Schematic and the Schematic Legend window.

By default, this option is enabled.

Show full cone

Set this option to display all possible paths in the schematic windows while extracting the fan-in/fan-out cone. By default, not all paths are displayed.

Show net bundles in IS

Set this option to enable the display of net bundles in the Incremental Schematic (IS). The net bundle takes the attributes of the constituent nets. For example, if a net contained in the net bundle is partial (dashed) or highlighted (colored), the net bundle would also appear the same.

Enable single click probing

Set this option to enable single-click probe in the schematic.

Maximum nets to probe

Set this option to probe the specified number of net bits in a net bundle. The default value for this option is 64, that is, SpyGlass will probe only 64 or lesser number of bits in a net bundle.

Skip text annotation for instances with pins more than

Set this option to disable showing the text attribute on instances with pins more than set value. The default value for this option is 1024.

Violation Hierarchy Depth

Some messages show schematic highlight in a hierarchical manner. The initial view shows only objects up to a specific level of hierarchy and then you can interactively dive into deeper hierarchies.

Set this option to control the depth of hierarchy shown in the initial view and when a specific hierarchy is loaded.

Hierarchical view enabled for instance count greater than

Set this option to switch to the hierarchical view when the number of instances reported reach the specified value. The default value of this option is 100.

NOTE: *This option is enabled only for the SpyGlass DFT product. Please refer to the DFT Rules Reference Guide for more information on the rules that support hierarchical view.*

Schematic debug data selection method

Schematic debug data widget automatically updates based on the option you select, namely, Mouse hover, Single click, or No update.

Highlight Width

Choose the highlighting width of the selected object in the schematic windows from 1(thinnest) to 9 (thickest). The default width is 3.

Auto Expand

Use this option to auto-expand all groups and their sub-groups in the current module. The other modules remain unaffected.

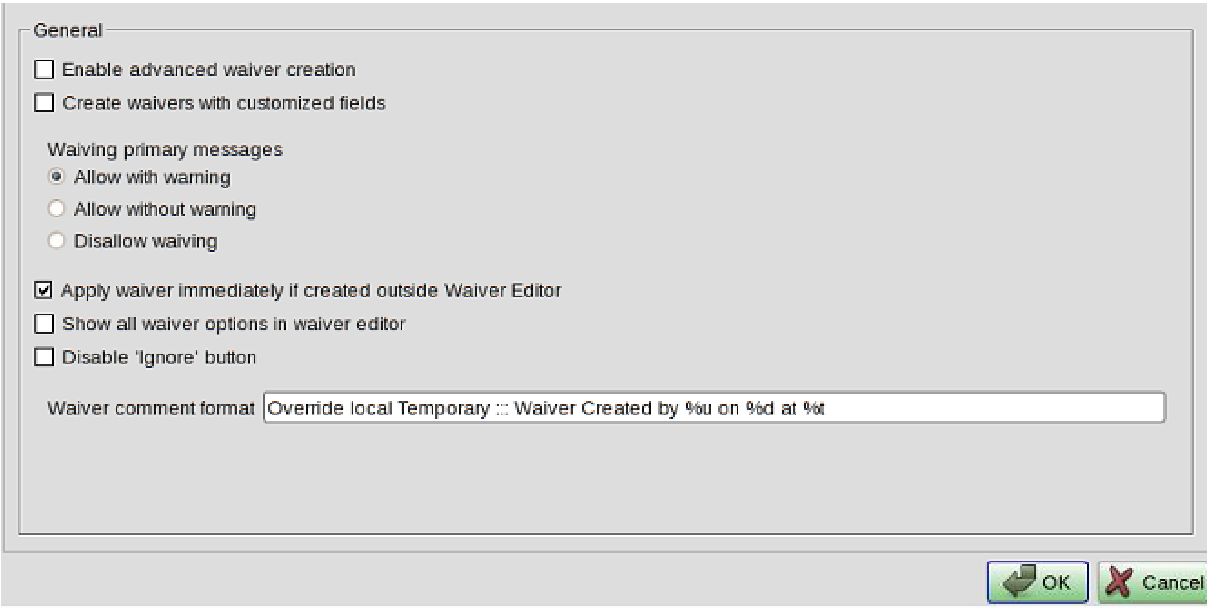
Allow sequential logic in IS clouds, if allowed by rule

Use this option to show the sequential logic grouped under clouds in the Incremental Schematic, if the rule allows. When this option is not set, the sequential logic will not be displayed as grouped under clouds. By default, this option is enabled.

Waiver Page

Use the *Waiver* page to set waiver options for violation messages.

The following figure shows the *Waiver* page:



General

□ Enable advanced waiver creation

□ Create waivers with customized fields

Waiving primary messages

● Allow with warning

○ Allow without warning

○ Disallow waiving

☑ Apply waiver immediately if created outside Waiver Editor

□ Show all waiver options in waiver editor

□ Disable 'Ignore' button

Waiver comment format: Override local Temporary ::: Waiver Created by %u on %d at %t

FIGURE 17. The SpyGlass Preferences Window - Waiver Page

You can set the following waiver options:

Enable advanced waiver creation

Set this option to apply waivers based on different criteria, such as waive by file, module, rule, severity, or selected message.

Create waivers with customized fields

Considers custom fields in a waiver, while waiving selected messages.

By default, this option is disabled.

Waiving primary messages

Allows you to define the preferred behavior for waiving primary messages. You can select one of the following options:

- **Allow with warning**: This option is selected by default. If you have selected a primary message in the message list, SpyGlass Explorer displays a warning/confirmation box.

- **Allow without warning**: Select this option to waive the primary messages without displaying any warning/confirmation box.
- **Disallow waiving**: Select this option to disable waiving of primary messages. In this case, SpyGlass Explorer does not display any warning/confirmation box.

NOTE: *Currently, this option will work only in the turbo mode.*

`Apply waiver immediately if created outside Waiver Editor`

By default, this option is enabled. In this case, waivers created outside the Waiver Editor window, are automatically applied.

When this option is disabled, you need to click Apply or Ok button in the Waiver Editor window to apply the waiver.

`Show all waiver options in waiver editor`

Shows the advanced options in waiver editor.

`Disable 'Ignore' button`

Set this option to disable the **Ignore** button.

`Waiver Comment Format`

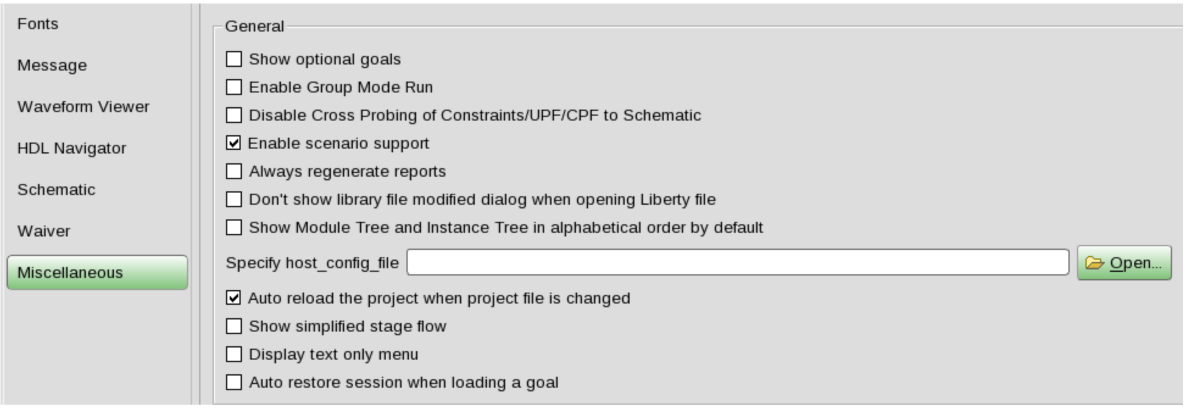
Enables you to specify the format of the auto-generated comment, while creating waivers using the GUI. The following is the format for the auto-generated comment:

```
Created by %u on %d at %t
```

Here, %u denotes user name, whereas, %d and %t denote current date and time, respectively.

Miscellaneous Page

Use the *Miscellaneous* page to set miscellaneous preferences.



General

□ Show optional goals
□ Enable Group Mode Run
□ Disable Cross Probing of Constraints/UPF/CPF to Schematic
☑ Enable scenario support
□ Always regenerate reports
□ Don't show library file modified dialog when opening Liberty file
□ Show Module Tree and Instance Tree in alphabetical order by default

Specify host_config_file

☑ Auto reload the project when project file is changed
□ Show simplified stage flow
□ Display text only menu
□ Auto restore session when loading a goal

FIGURE 18. The Miscellaneous Page

You can set the following miscellaneous preferences:

Show optional goals

Set this option to display optional goals under the *Select Goal* tab.

Enable scenario support

Use this option to enable a user to work with scenarios, such as creating, modifying, and deleting scenarios.

For details on scenarios, see *Working With Scenarios*.

Always regenerate reports

Set this option to re-generate the report even if the report already exists at the default location. By default, SpyGlass Explorer prompts for the report location.

Don't show Library file modified dialog when opening Liberty file

Use this option to suppress the pop-up while opening liberty file from schematic, in case the file has been modified.

Show Module tree and Instance tree in alphabetical order by default

Displays the module tree and instance tree items in the alphabetical order.

By default, this preference is disabled. Enabling this preference displays the items in the instance and module trees in the alphabetical order in the subsequent spyglass sessions.

Specify host_config_file

Specify a host configuration file in this field.

If a host configuration file is already specified by the `HOST_CONFIG_FILE` key in .spyglass.setup, the details of that file appear under this field so that you can edit the details as per your requirement.

This file is used for parallel execution of goals on different machines.

Auto reload the project when the project file is changed

If this preference is set, the GUI automatically reloads the project, if it is edited.

Auto restore session when loading a goal

Use this option to auto restore the saved session.

The current session gets saved automatically in a Tcl file when you switche from a goal or when you exit the GUI. All the current GUI items, such as, the violation message being debugged, active Incremental Schematic, Modular Schematic, PIH window, and spreadsheet window are saved in this file.

On enabling this option, the auto saved session is restored during the goal run.

PDF Reader

This option sets the path to the PDF file viewer that is required for viewing the printable SpyGlass documentation files.

This field is controlled as follows:

- The value of the `SG_PDF_VIEWER` environment variable, if set, has the highest priority. Every time, you start SpyGlass, this field shows the value of the `SG_PDF_VIEWER` environment variable, if set and uses that PDF viewer.
- You can change the PDF viewer for the current session by typing the complete path and file name of the PDF viewer executable or by clicking *Browse* and then search and select the executable file. If the

SG_PDF_VIEWER environment variable has not been set, this setting will persist from session to session.

- If the SG_PDF_VIEWER environment variable has not been set and you do not set the *Specify pdf file reader* preference, SpyGlass will try to invoke the commonly-used PDF viewer of the particular platform.

HTML Browser

This field sets the path to the HTML File Browser for viewing the SpyGlass HTML-based Help files.

This field is controlled as follows:

- The value of the SG_HTML_BROWSER environment variable, if set, has the highest priority. Every time, you start SpyGlass, this field shows the value of the SG_HTML_BROWSER environment variable, if set and uses that HTML File Browser.
- You can change the HTML File Browser for the current session by typing the complete path and file name of the HTML File Browser executable or by clicking *Browse* and then search and select the executable file. If the SG_HTML_BROWSER environment variable has not been set, this setting will persist from session to session.
- If the SG_HTML_BROWSER environment variable has not been set and you also do not set this preference, SpyGlass will try to invoke the commonly-used HTML File Browser of the particular platform.

Text Editor

This field sets the path to the text editor for viewing and editing text files.

Type the complete path and file name of the executable text editor or browse to the executable file.

If you do not set this preference, SpyGlass invokes the text editor specified in the EDITOR environment variable or the commonly-used text editor of the particular platform.

Text Editor line flag

This field sets the line flag (line number format) specific to an editor, which is specified in the Text Editor field.

Line flag is a way of specifying the line number from which an editor should open.

Specify the format by using `%s`, which gets replaced with the line number specified by the user while opening that editor.

The following table shows examples of few editors and their respective line flags:

Editor	Line Flag	Example
gvim	+%s	To display the test.v file from line 400 in `gvim`, the following command is used: `gvim +400 test.v`
nedit	-line %s	To display the test.v file from line 400 in `nedit`, the following command is used: `nedit -line 400 test.v`

Use external editor for displaying reports/logs

Select this option to open reports and log files in the text editor specified by the *Text Editor* field.

Number of spreadsheet viewer

Specify a positive integer value to control the spreadsheet viewer configuration.

Auto-launch a Spreadsheet viewer when selecting a violation in message tree

Select this option to automatically display the associated spreadsheet when you double-click a message.

Show help in Spreadsheet viewer

Enables/disables Spreadsheet viewer window globally

Open auto SoC flow results in a new session

If this preference is set, the results of SoC flow are opened in a new GUI session, instead of opening in the same session.

Help Menu

When you click the *Help* menu, the following options appear:

Help > SpyGlass Help	*Help > SpyGlass Manuals*
Help > Search SpyGlass Manuals	*Help > Shortcut Keys*
Help > Icons Quick Reference	*Help > SpyGlass Console User Guide*
Help > SpyGlass Console Reference Guide	*Help > BuiltIn Rules Reference*
Help > Methodology Guides	*Help > SpyGlass Release Summary*
Help > SpyGlass Release Notes	*Help > SpyGlass KPNS*
Help > Online Support	*Help > About SpyGlass*

Help > SpyGlass Help

This menu option invokes the spyhelpviewer utility to display SpyGlass documentation in an HTML format.

By default, the spyhelpviewer utility searches for the netscape executable in your machine's path for displaying the PDF files. Use the *Tools > Preferences* > *Miscellaneous Page* > *HTML Browser* option or the new SG_HTML_BROWSER environment variable to set your HTML Browser.

Help > SpyGlass Manuals

This menu option invokes the spydocviewer utility that displays SpyGlass printable (PDF) documentation in a tree format for easy access. In addition, the Atrenta Standard Rule-Primitive documentation (in text format) is also accessible.

By default, the spydocviewer utility searches for the acroread or xpdf executable in your machine's path for displaying the PDF files. Use the *Tools > Preferences* > *Miscellaneous Page* > *PDF Reader* option or the new SG_PDF_VIEWER environment variable to set your PDF viewer.

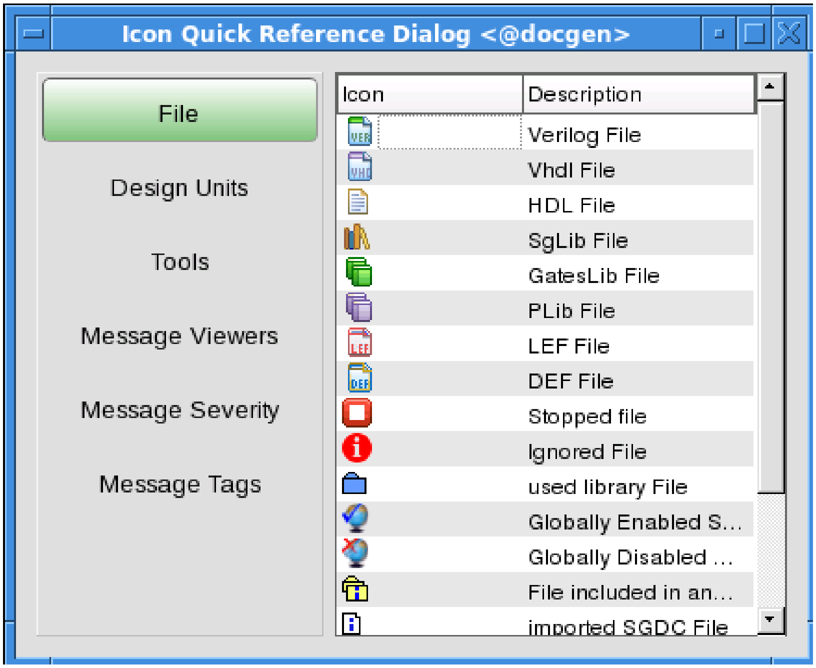
You can also use the F1 key on your keyboard to invoke the on-line manuals.

Help > Search SpyGlass Manuals

This menu option displays the *Search* dialog using which you can search for specific information in various SpyGlass documents.

Help > Icons Quick Reference

This menu option invokes the *Icons Quick Reference* window that displays various icons used in SpyGlass.

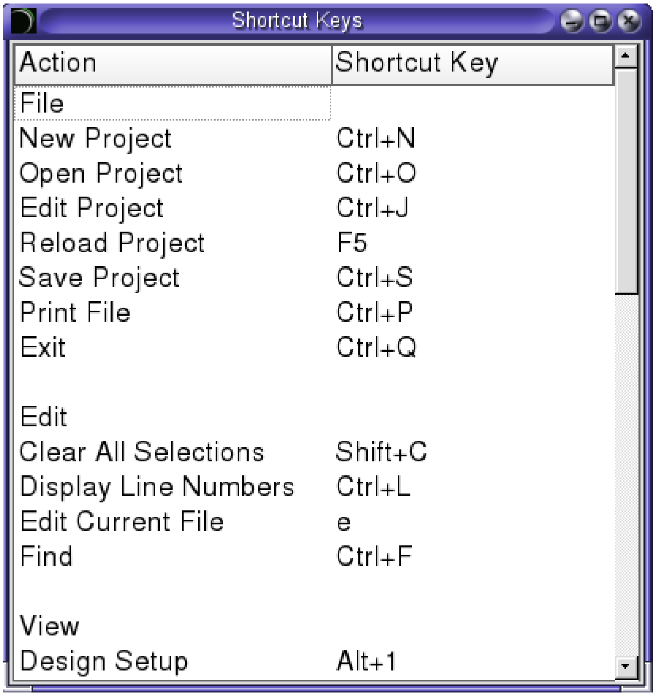


Icon	Description
	Verilog File
	Vhdl File
	HDL File
	SgLib File
	GatesLib File
	PLib File
	LEF File
	DEF File
	Stopped file
	Ignored File
	used library File
	Globally Enabled S...
	Globally Disabled ...
	File included in an...
	imported SGDC File

FIGURE 19. Icon Quick Reference

Help > Shortcut Keys

This menu option invokes the Shortcut Keys window that contains the names of the shortcut keys combination used to open different SpyGlass windows.



Action	Shortcut Key
File	
New Project	Ctrl+N
Open Project	Ctrl+O
Edit Project	Ctrl+J
Reload Project	F5
Save Project	Ctrl+S
Print File	Ctrl+P
Exit	Ctrl+Q
Edit	
Clear All Selections	Shift+C
Display Line Numbers	Ctrl+L
Edit Current File	e
Find	Ctrl+F
View	
Design Setup	Alt+1

FIGURE 20. Shortcut Keys

You can also use the *<Ctrl>+F1* key combination on your keyboard to invoke the Shortcut Keys window.

Help > SpyGlass Console User Guide

This menu option displays the *SpyGlass Console User Guide* document.

Help > SpyGlass Console Reference Guide

This menu option displays the *SpyGlass Console Reference Guide* document.

Help > BuiltIn Rules Reference

This menu option displays the *BuiltIn Rules Reference Guide* document.

Help > Methodology Guides

This menu option displays a sub-menu that lists all methodology guides. Select the required option from the sub-menu to display the corresponding methodology guide.

Help > SpyGlass Release Summary

This menu option displays the *SpyGlass Release Summary* document that contains a summary of enhancements made in the current SpyGlass version.

Help > SpyGlass Release Notes

This menu option displays the *SpyGlass Release Notes* document that contains details about all enhancements made in the current SpyGlass version.

Help > SpyGlass KPNS

This menu option displays the *SpyGlass Known Problems and Solutions* document that contains details about known problems in SpyGlass and their corresponding solution.

Help > Online Support

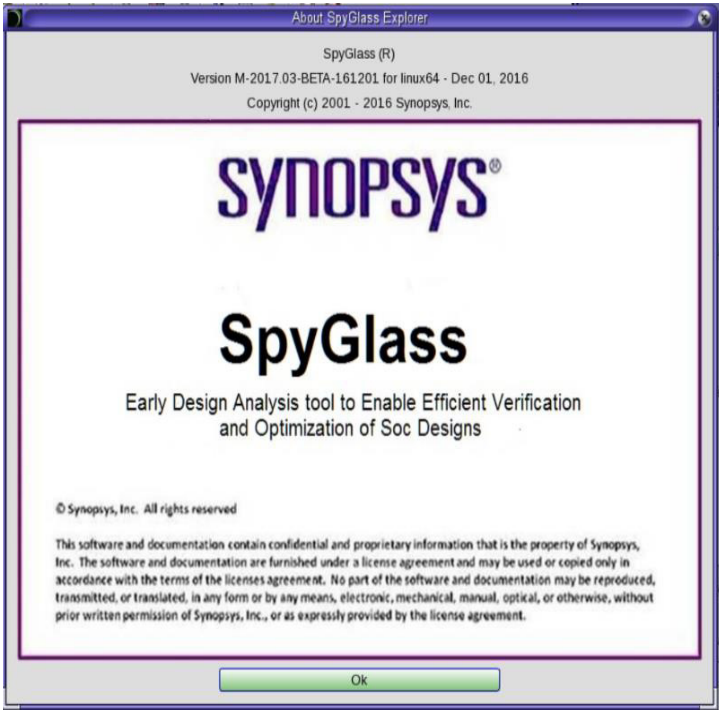
This menu option provides online support for various details, such as updates, training, and FAQ.

When you select this menu, a sub-menu appears that lists various options to provide different types of online support.

Help > About SpyGlass

This menu option is used to display SpyGlass version number and Synopsys copyright and disclaimer.

When you select this menu option, the About SpyGlass window is displayed.



About SpyGlass Explorer

SpyGlass (R)

Version M-2017.03-BETA-161201 for linux64 - Dec 01, 2016

Copyright (c) 2001 - 2016 Synopsys, Inc.

SYNOPSYS®

SpyGlass

Early Design Analysis tool to Enable Efficient Verification and Optimization of Soc Designs

© Synopsys, Inc. All rights reserved

This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the licenses agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

FIGURE 21. About SpyGlass

Click the *Ok* button to close this window.

The Methodology Configuration System Menu Bar

The Methodology Configuration System menu bar provides you access to the functions that have been categorized into the following menus:

- *File Menu*
- *Edit Menu*
- *Tools menu*
- *Help Menu*

File Menu

When you click the *File* menu, the following options appear:

File > New Methodology	*File > Open Methodology*
File > Save Methodology	*File > Save Methodology As*
File > Reload Methodology	*File > Methodology Properties*
File > Close	

File > New Methodology

This menu option is used to create a new methodology.

To know details on creating a new methodology, see *Creating and Modifying a Methodology*.

File > Open Methodology

This menu option is used to open an existing methodology.

When you select this menu option, SpyGlass Explorer first prompts you to save the currently loaded methodology by displaying the *Save Methodology* dialog. Once you perform the required actions in this dialog, SpyGlass Explorer displays the *Open Methodology* dialog, as shown in the following figure:

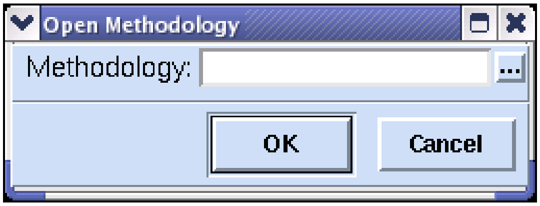


FIGURE 22. Open Methodology

In the above dialog, you can specify the path of the directory where the methodology resides in the *Methodology* text field. Alternatively, click the (...) button, and browse to the directory where the methodology is present.

NOTE: *You can also use the <Ctrl> + <O> key combination on the keyboard to display the Open Methodology dialog.*

File > Save Methodology

This menu option is used to save the currently loaded methodology.

When you select this menu option, the *Save Methodology* dialog appears, as shown in the following figure:

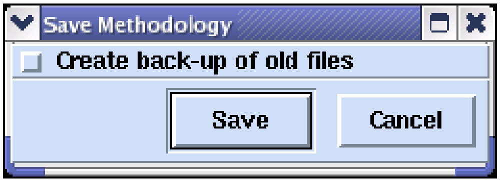


FIGURE 23. Save Methodology

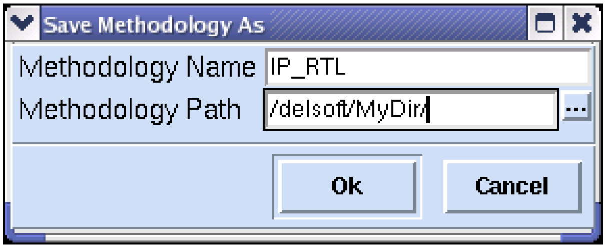
In the above dialog, click the *Save* button to save the methodology. If you want to create a backup of the methodology, select the *Create back-up of old files* option, and then click the *Save* button.

NOTE: *You can also use the <Ctrl> + <S> key combination on the keyboard to open the Save Methodology dialog.*

File > Save Methodology As

This menu option is used to save the current methodology to a different location.

When you select this menu option, the *Save Methodology As* dialog appears, as shown in the following figure:



Methodology Name IP_RTL

Methodology Path /delsoft/MyDir/

FIGURE 24. Save Methodology As

In the above dialog, you can specify the methodology name and path in appropriate fields. Once you specify the required details, click the *OK* button to save the changes.

File > Reload Methodology

This menu option is used to reload a methodology.

File > Methodology Properties

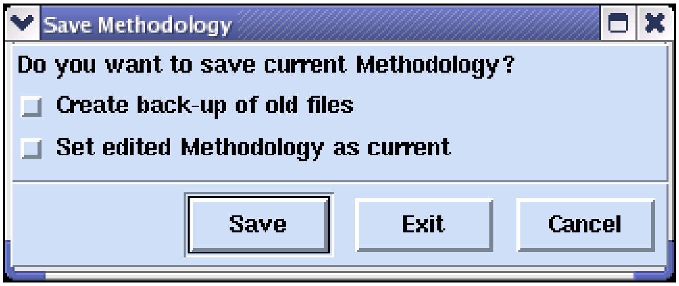
This menu option is used to view/change methodology properties, such as name, path, and help descriptions.

For details on modifying the methodology properties, see *Creating and Modifying a Methodology*.

File > Close

This menu option is used to close the *Methodology Configuration System* window.

When you select this menu option, SpyGlass Explorer prompts you to save the methodology by displaying the *Save Methodology* dialog, as shown in the following figure:



Save Methodology

Do you want to save current Methodology?

□ Create back-up of old files

□ Set edited Methodology as current

FIGURE 25. Save Methodology

If you want to load the edited methodology in the *Goal Selection* window, select the *Set edited Methodology as current* option. In addition, if you want to create a backup of the old methodology files, select the *Create back-up of old files* option. Once you select the required options, click the *Save* button. However, if you want to exit MCS without saving the changes in the currently loaded methodology, click the *Exit* button.

NOTE: *You can also use the <Ctrl> + <Q> key combination on the keyboard to open the Close confirmation dialog.*

Edit Menu

When you click the *Edit* menu, the following menu options appear:

Edit > Add Sub-Methodology	*Edit > Add New Goal*
Edit > Import Goal(s)	*Edit > Copy Sub-Methodology*
Edit > Paste Sub-Methodology	*Edit > Copy Goal*
Edit > Paste Goal	

Edit > Add Sub-Methodology

This menu option is used to add a sub-methodology to an existing methodology.

For more details, see *Creating a Sub-Methodology*.

NOTE: *You can also use the <Ctrl> + <M> key combination on the keyboard to open the Add Sub-Methodology dialog.*

Edit > Add New Goal

This menu option is used to add a goal to the selected methodology. For details on adding a new goal, see *Creating Goals*.

Edit > Import Goal(s)

This menu option is used to import goal/goals into the selected the methodology.

For details on importing goals, see *Importing Goals*.

Edit > Copy Sub-Methodology

This menu option is used to copy a sub-methodology to create multiple instances of a sub-methodology.

NOTE: *This menu option is visible in the Edit menu when you have selected a sub-methodology in the Goals section.*

When you copy a sub-methodology, it is displayed in italics in the *Goals* section of the MCS window.

When you copy a sub-methodology, the goals displayed under the sub-methodology tree are also copied.

You can also copy a sub-methodology by performing any of the following:

- Right-click a sub-methodology, and select the *Copy Sub-Methodology* shortcut menu option.
- Use the *<Ctrl> + C* key combination on your keyboard to copy a sub-methodology.

Edit > Paste Sub-Methodology

Use this menu option is used to create multiple instances of the copied sub-methodology.

NOTE: *This menu option is visible in the Edit menu only if you have copied a sub-methodology from the Goals section.*

You can also paste a sub-methodology by performing any of the following:

- Right-click on a sub-methodology, and select the *Paste Sub-Methodology* option from the shortcut menu.
- Use the <*Ctrl*> +V key combination on your keyboard to paste the sub-methodology in a different location.

Edit > Copy Goal

This menu option is used to copy goals from one sub-methodology to another thus enabling you to create multiple instances of the same goal.

When you copy a goal, it is displayed in italics in the *Goals* section of the MCS window.

You can also copy a goal by performing any of the following:

- Right-click on a goal, and select the *Copy Goal* option from the shortcut menu.
- Use the <*Ctrl*> +C key combination on your keyboard to paste a goal in a different location.

Edit > Paste Goal

This menu option is used to create another instance of the same goal across different sub-methodologies

You can also paste a goal by performing any of the following:

- Right-click a goal, and select the *Paste Goal* shortcut menu option.
- Use the <*Ctrl*> + V key combination on your keyboard to paste a goal under a different sub-methodology.

Tools menu

When you click the *Tools* menu, the following options appear:

Tools > Compare	*Tools > Preferences*

Tools > Compare

When you select this menu option, the following options appear in the sub-menu:

- *Goal(s) with Goal(s)*
- *Methodologies*

Goal(s) with Goal(s)

Select this menu option to compare two methodologies or goal files.

The comparison result shows the comparison between the rules and parameters present in the two methodologies or goal files. See *Figure 29*.

One methodology or goal is called the reference data and the other methodology or goal is called the target data. The target data is compared with the reference data.

When you select this option, the *Methodology Comparison* window appears, as shown in the following figure:

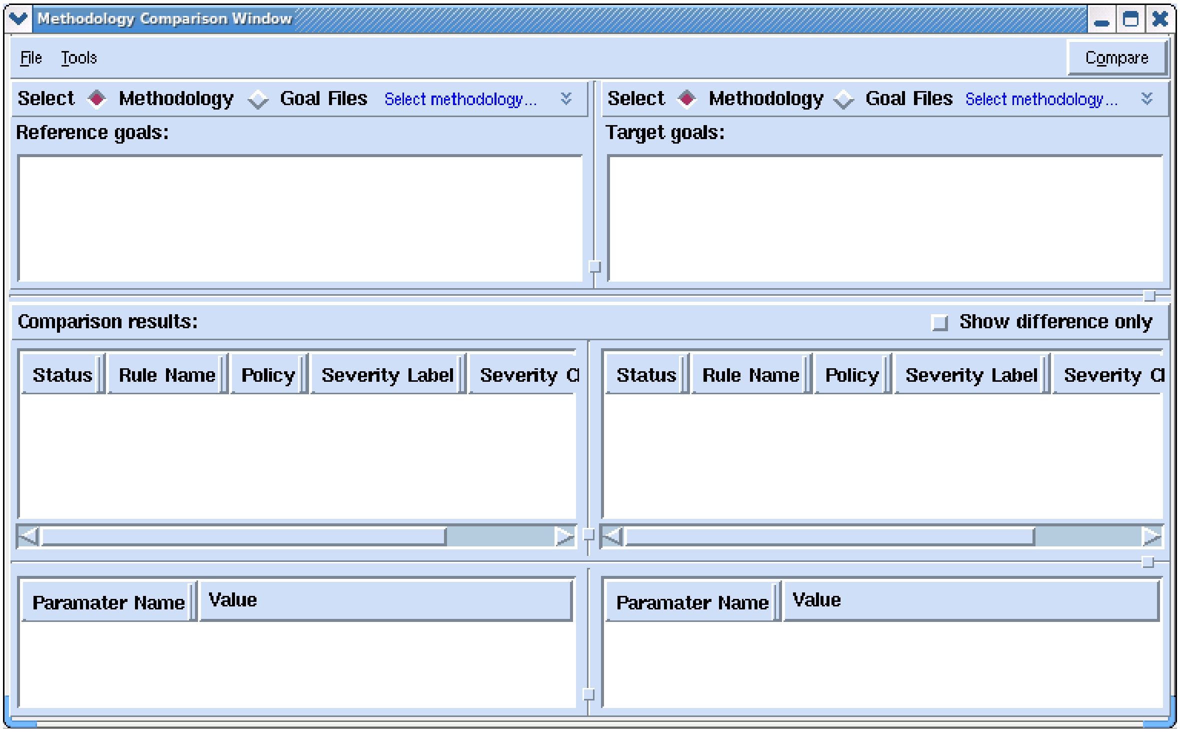


FIGURE 26. Methodology Comparison Window

To compare two methodologies, perform the following steps:

1. Select the *Methodology* option in the *Reference goals* section.
2. Click the *Select Methodology* link.

The *Select Methodology* dialog appears. The following figure shows the *Select Methodology* dialog:

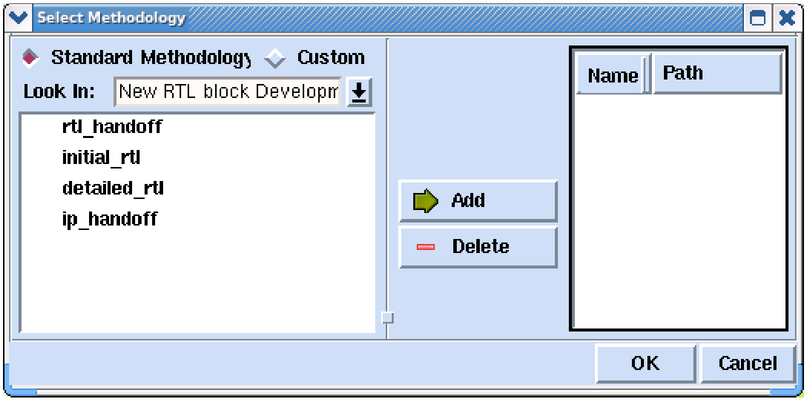


FIGURE 27. Select Methodology

3. In the above dialog, select the *Standard Methodology* or the *Custom* option depending upon the type of methodology you want to set as the reference data.
4. Select a methodology in the left-most pane and click the *Add* button.

   The name and path of the selected methodology appears in the right-most pane in this dialog.
5. Click the *OK* button to close the above dialog.

   After performing the above steps, SpyGlass Explorer loads the details of the selected methodology in the *Reference goals* section, as shown in the following figure:

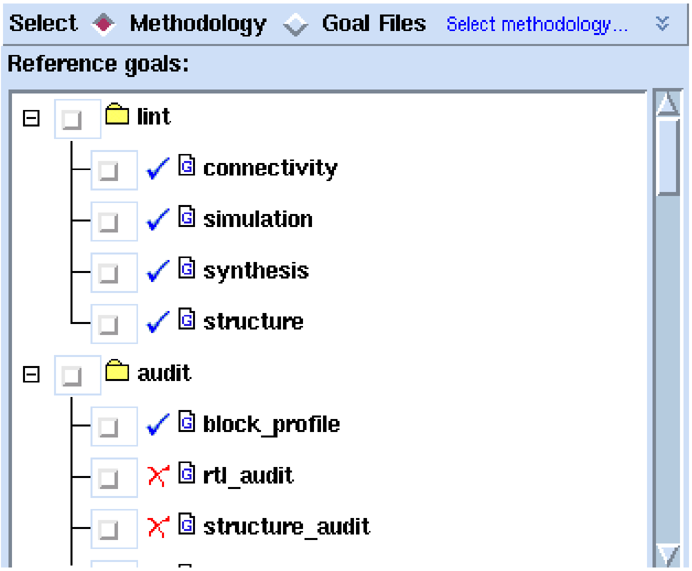


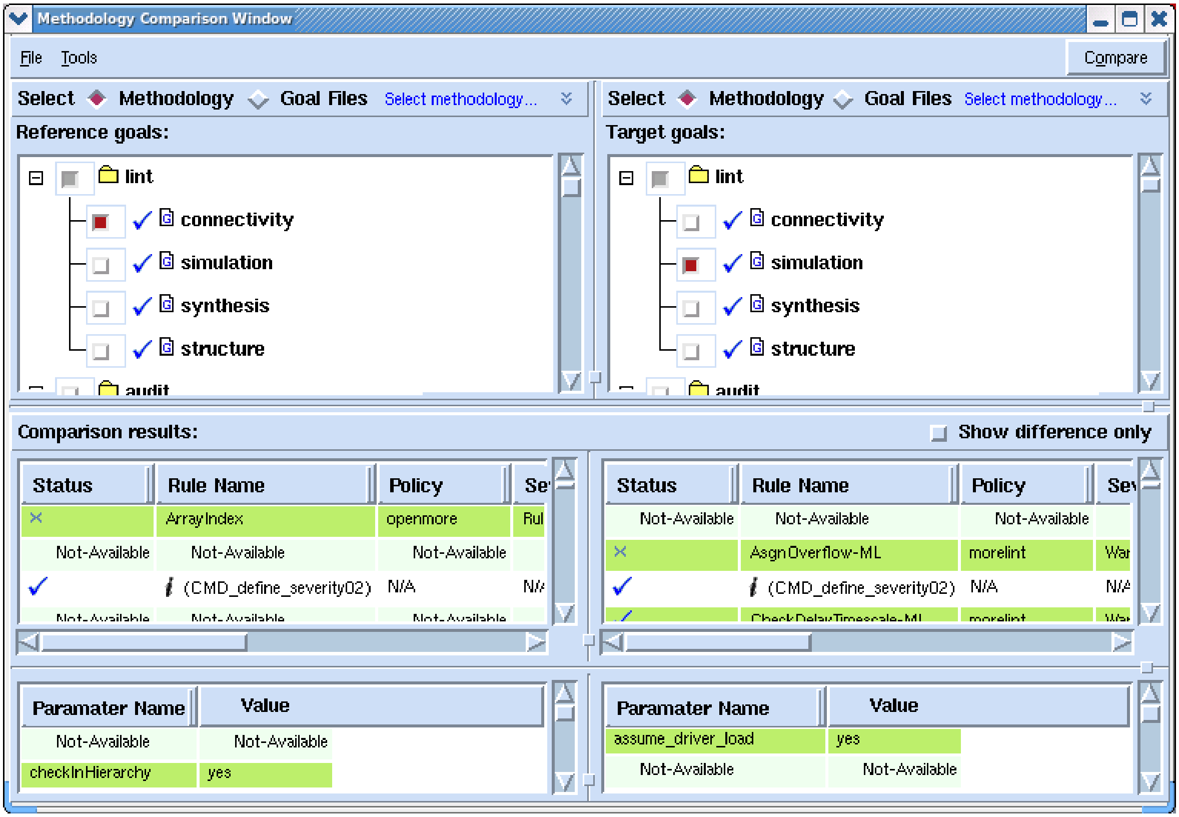
FIGURE 28. Select Reference Goals

6. Select the *Methodology* option in the *Target goals* section.
7. Repeat steps *3* to *5* to load the details of the target methodology.
8. Select the goals to be compared from the *Reference goals* section and the *Target goals* section.

NOTE: *You can specify a map file to trace back the reference of the new goal (present in GuideWare 2.0) to the original goal (present in GuideWare 1.0).*

9. Click the *Compare* button.

After performing the above steps, the comparison results appear in the *Comparison Results* section, as shown in the following figure:


Methodology Comparison Window
File
Tools
Compare
Select ◆ Methodology ◇ Goal Files
Select methodology...
Select ◆ Methodology ◇ Goal Files
Select methodology...
Show difference only

Reference goals:

- lint
  - ✓ connectivity
  - ✓ simulation
  - ✓ synthesis
  - ✓ structure
- audit

Target goals:

- lint
  - ✓ connectivity
  - ✓ simulation
  - ✓ synthesis
  - ✓ structure
- audit

Comparison results: □ Show difference only

Status	Rule Name	Policy	Se
✕	ArrayIndex	openmore	Rul
Not-Available	Not-Available	Not-Available	
✓	(CMD_define_severity02)	N/A	N/A
Not-Available	Not-Available	Not-Available	

Status	Rule Name	Policy	Se
Not-Available	Not-Available	Not-Available	
✕	AsgnOverflow-ML	morelint	War
✓	(CMD_define_severity02)	N/A	N/A
✓	CheckDelayTimescale-ML	morelint	War

Paramater Name	Value
Not-Available	Not-Available
checkInHierarchy	yes

Paramater Name	Value
assume_driver_load	yes
Not-Available	Not-Available

FIGURE 29. Methodology Comparison Results

Similarly, you can compare two goals by using the above window.

Methodologies

Select this option to compare two methodologies and to merge the contents of one goal into another.

When you select this menu option, the *Methodology Comparison* dialog appears, as shown in the following figure:

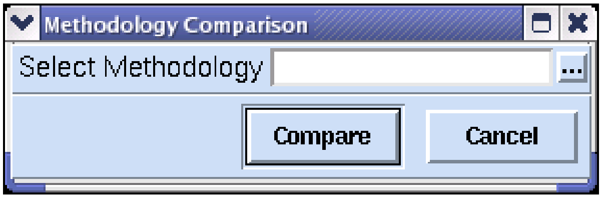


FIGURE 30. Select Methodology for Comparison

For comparison, the currently selected methodology is considered as a reference. In this dialog, specify the location of the methodology that you want to compare in the *Select Methodology* field, and click the *Compare* button.

The differences in the methodology will be shown in a hierarchical format. The reference methodology is displayed in the left column of the Methodology Comparison window and the methodology being compared is displayed in the right column of the Methodology Comparison window.

The differences are displayed with levels like goal, product, rules, parameter, and parameter value as shown below.

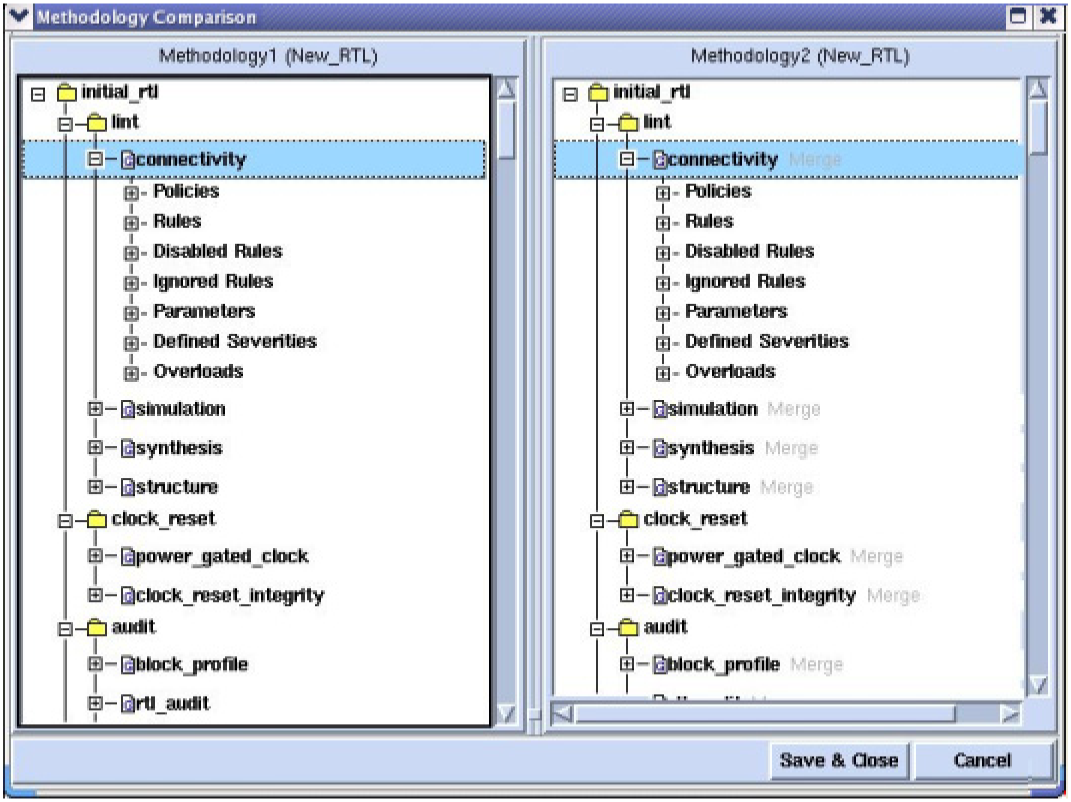


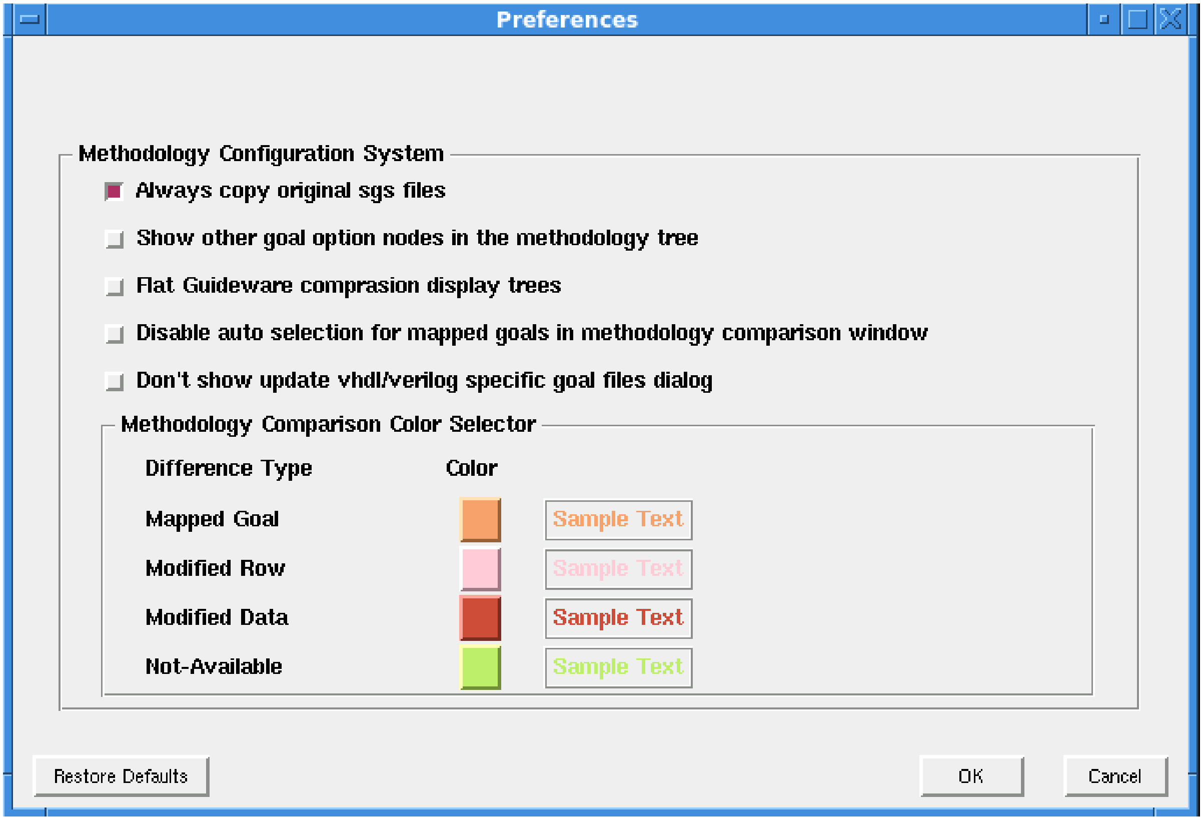
FIGURE 31. Methodology Comparison Result Levels

You can also merge settings of a methodology being compared on basis of goals into the base/reference methodology.

To do this, click the *Merge* link on the methodology that is being compared. Then the changes made in the methodology are merged into the reference methodology.

Tools > Preferences

This menu option displays the *Preferences* dialog, as shown in the following figure:



Methodology Configuration System

- ☒ Always copy original sgs files
- □ Show other goal option nodes in the methodology tree
- □ Flat Guideware comprasion display trees
- □ Disable auto selection for mapped goals in methodology comparison window
- □ Don't show update vhdl/verilog specific goal files dialog

Methodology Comparison Color Selector

Difference Type	Color	
Mapped Goal	🟧	Sample Text
Modified Row	🟪	Sample Text
Modified Data	🟥	Sample Text
Not-Available	🟩	Sample Text

FIGURE 32. Preferences - Font

In the above dialog, you can provide the required settings for *goals and methodology comparison colors.*

Help Menu

When you click the *Help* menu, the following options appear:

Help > Methodology Configuration System Help	*Help > On-line Manuals*
Help > On-line Help	*Help > Icons Quick Reference*
Help > About SpyGlass	

Help > Methodology Configuration System Help

This menu option is used to display the online help for Methodology Configuration System.

Help > On-line Manuals

This menu option is used to display the *Documentation* window that contains links to various on-line manuals in SpyGlass. To open a particular manual, double-click on that manual name.

Help > On-line Help

This menu option is used to display the online help of SpyGlass documentation. When you select this menu option, SpyGlass Explorer displays the browser window in which you can traverse through the online help on different topics.

Help > Icons Quick Reference

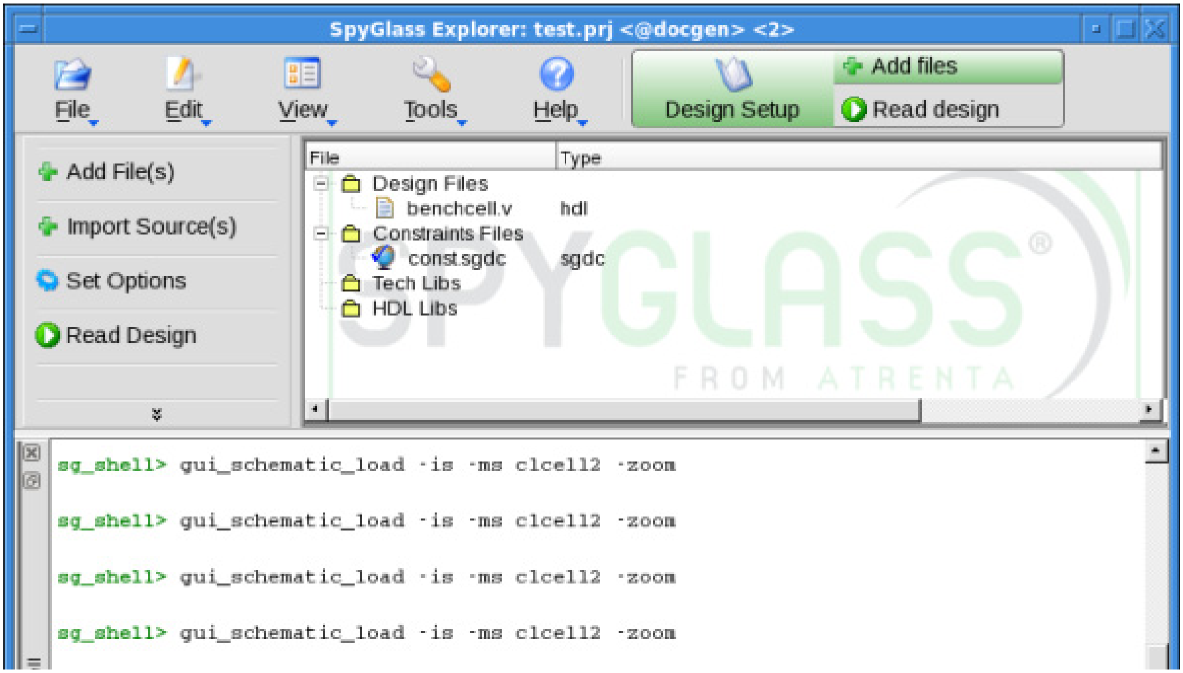
This menu option invokes the *Icons Quick Reference* window that contains a brief description of various icons used in the *Methodology Configuration System* window.

Help > About SpyGlass

This menu option is used to invoke the *About SpyGlass* window that displays various details such as SpyGlass version number and Atrenta Contact information.

sg_shell

Enables you to add files using Tcl commands and has an embedded Tcl Shell Interface. The UI actions are translated to the respective Tcl commands, which are saved in the *gui_command.log* file. Command completion and emacs like editing features are supported in the sg_shell. You can use these files for using the setup in a later run.

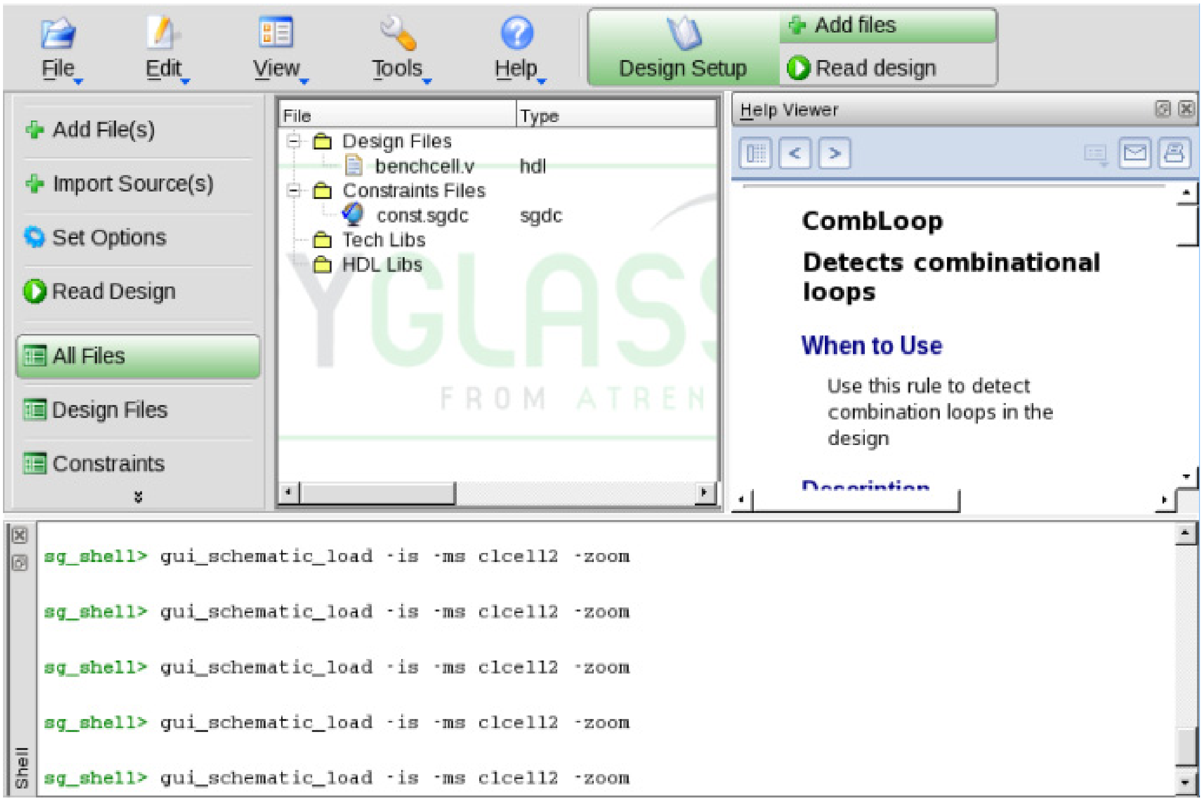


```
sg_shell> gui_schematic_load -is -ms clcell2 -zoom
sg_shell> gui_schematic_load -is -ms clcell2 -zoom
sg_shell> gui_schematic_load -is -ms clcell2 -zoom
sg_shell> gui_schematic_load -is -ms clcell2 -zoom
```

FIGURE 33. SpyGlass Explorer - sg_shell

Design Setup

The Design Setup view for SpyGlass Explorer shows a unified file view, where all types of files, such as, Design Files, Constraints, Tech Libs, and HDL libs are listed. To view the details of each of these file types, you can click the respective option in the left pane of the Design Setup window. *Figure 34* shows the **Design Setup** window.



File	Type
Design Files	
benchcell.v	hdl
Constraints Files	
const.sgdc	sgdc
Tech Libs	
HDL Libs	

CombLoop

Detects combinational loops

When to Use

Use this rule to detect combination loops in the design

Description

```
sg_shell> gui_schematic_load -is -ms clcell2 -zoom

sg_shell> gui_schematic_load -is -ms clcell2 -zoom

sg_shell> gui_schematic_load -is -ms clcell2 -zoom

sg_shell> gui_schematic_load -is -ms clcell2 -zoom

sg_shell> gui_schematic_load -is -ms clcell2 -zoom
```

FIGURE 34. SpyGlass Explorer - Design Input

Clicking the Design Setup tab enables the following options:

- *Add Files*
- *Read Design*

Add Files

The Add Files pane lists the options described in Table:

Option	Description
Add Files	Displays the Add Files dialog box. Allows you to add design files. You can add following types of design files: ● Design Files ● Constraints ● Waivers ● Tech Libs ● HDL Files
Import Source(s)	used to import an existing SpyGlass profile (.spp) file that contains the setup information from an earlier SpyGlass run, such as the current working directory, VHDL library mapping information, and so on.
Set Option(s)	Enables you to specify various settings, command and advanced, during the design setup stage. For details see Setting the Design Read Option(s).
Read Design	Enables you to perform first-level HDL analysis.
All Files	Displays all the selected design files. You can also add the design files by right-clicking in the File View Pane and selecting the type of design file to be loaded from the following options: ● HDL File(s) ● Constraint File(s) ● Waiver File(s) ● Design Files ● ADC File(s) ● AWL File(s) ● Tech Lib File(s) ● Constraints ● HDL Library File(s)
Design Files	Allows you to view the HDL files. You can also add the HDL files by right-clicking in the File View pane and selecting the Add HDL File(s) option.

Option	Description
Constraints	Allows you to view the Constraint(s)/ADC files. You can also add the HDL files by right-clicking in the File View pane and selecting one of the following options: ● Add Constraints File(s) ● Add ADC File(s)
Waivers	Allows you to view the Waiver files. You can also add the waiver files by right-clicking in the File View pane and selecting one of the following options: ● Add Waiver File(s) ● Add AWL File(s) ● Create New Waiver File ● Save Waiver Files
Tech Libs	Allows you to view the tech lib files. You can also add the tech lib files by right-clicking in the File View pane and selecting one of the following options: ● Add Waiver File(s) ● Add AWL File(s) ● Create New Waiver File ● Save Waiver Files
HDL Libs	Allows you to add HDL files. You can also add the HDL files by right-clicking in the File View pane and selecting the Add HDL Library File(s) option.

Read Design

Selecting this option enables you to perform first-level HDL analysis. When you select the Read Design tab, following options are displayed:

Option	Description
Run Design Read	Initiates the design read process.
Synthesize netlist	Enables the synthesis mode in design read.

Goal Setup

The Goal Setup window allows you to Select or Setup goals.

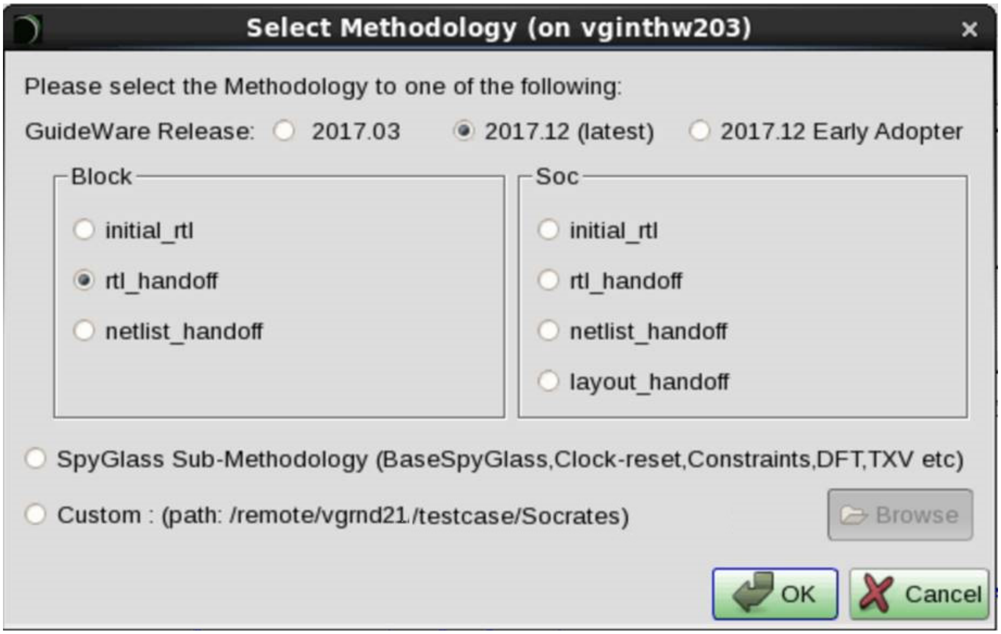
Select

Displays the various goals available for a product for the selected methodology. It also displays the run status of the goals and pre-requisite goal, if any.

You can select any of the listed goals to run.

You can also change the selected methodology by clicking the Select Methodology button.

The Select Methodology window is displayed as shown below:



Select Methodology (on vginthw203)

Please select the Methodology to one of the following:

GuideWare Release: ○ 2017.03 ● 2017.12 (latest) ○ 2017.12 Early Adopter

Block
- ○ initial_rtl
- ● rtl_handoff
- ○ netlist_handoff

Soc
- ○ initial_rtl
- ○ rtl_handoff
- ○ netlist_handoff
- ○ layout_handoff

○ SpyGlass Sub-Methodology (BaseSpyGlass,Clock-reset,Constraints,DFT,TXV etc)

○ Custom : (path: /remote/vgrnd21./testcase/Socrates)

FIGURE 35. Select Methodology

Select one of the methodologies available under the GuideWare Release section. Additionally, you can select SpyGlass Sub-Methodology or can specify a Custom methodology.

The following right-click menu options are available for the Select pane:

Option	Description
Create Scenario	Enables you to create scenarios for a goal. For details, see Working with Scenarios.
Edit Rules and Parameters	Allows you to modify the parameters and/or rules for the selected scenario.
Setup Goal	Allows you to modify the parameters and specify the required constraints for a goal.
Configure Columns	Enables you to select/deselect the columns to be displayed.

Setup

Goal setup includes setting up the setting up the recommended parameters and specifying the required constraints for a goal.

When you select the Setup tab, following options are displayed:

Option	Description
Add Constraint/ Waiver File(s)	Allows you to add SGDC/ADC files to the selected goal.
Show Common Parameters Only	Displays only the common parameters for the selected goal. When de-selected, displays advanced parameters also.
Restore Defaults	Restore all parameters to the default values.
Incremental Mode	Enables the incremental mode, which allows you to compare results between two runs.
Reports	Allows you to select reports to be generated while running the goals.
Restore Settings	Restores the default settings for the current goal.

Option	Description
Import Settings	Allows you to import setup information from some other goal.
Select Goal(s)	Displays the Goal Selection pane.
Analyze Goal(s)	Displays the Analyze Results pane.

Also, following options are displayed when you right-click on a file in the File view pane of the Goal Setup pane:

Option	Description
Enable File/ Disable File	Enables or disables the selected SGDC/ADC files globally.
Edit File	Allows you to edit the selected ADC/SGDC file.
Copy File Path	Allows you to copy the path of the selected file
Add/Constraints/ Waiver File(s)	Allows you to add SGDC/ADC files to the selected goal.

Analyze Results

The Analyze Results window displays the results of a goal run.

When you select the Analyze Results tab, following options are displayed:

Option	Description
Run Goal	Runs the selected goal.
Incremental Mode	Enables the incremental mode, which allows you to compare results between two runs. Click the 📂 icon to specify the reference VDB file for incremental run.

Windows and Panes in SpyGlass Explorer

This section explains following windows and panes in SpyGlass Explorer:

Modules Page

The *Modules* page displays the modules and instances in each source and library file that you are analyzing with the current project.

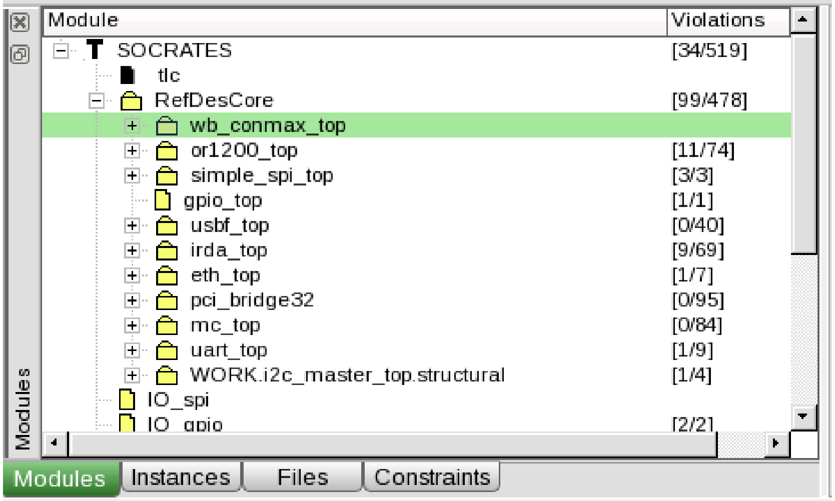
This page remains blank until SpyGlass analysis has been completed or a valid project file is loaded.

This section explains the following topics:

- *Overview of the Modules Page*
- *Right-Click Menu Options of Modules Page*

Overview of the Modules Page

The following figure displays the *Modules* page:



Module	Violations
SOCRATES	[34/519]
tlc	
RefDesCore	[99/478]
wb_conmax_top	
or1200_top	[11/74]
simple_spi_top	[3/3]
gpio_top	[1/1]
usbf_top	[0/40]
irda_top	[9/69]
eth_top	[1/7]
pci_bridge32	[0/95]
mc_top	[0/84]
uart_top	[1/9]
WORK.i2c_master_top.structural	[1/4]
IO_spi	
IO_gpio	[2/2]

FIGURE 36. Module View

In the above view, a module name appears in the following format:

<module-name> [x/y]

Where:

- `x` refers to the number of violations reported for the module *`<module-name>`*.
- `y` refers to `x` + (number of violations reported within the hierarchy of the module *`<module-name>`*)

Right-clicking over the module/instance displays a right-click menu as discussed in the *Right-Click Menu Options of Modules Page* section.

When you double-click on a module, the source code of that module appears in the source section with the first line of the module description highlighted (initial commented lines are skipped). Similarly, the corresponding gate is also highlighted in *The Modular Schematic Window*.

Details of the above page are discussed in the following points:

- The design hierarchy in this page displays the multiple layers and dependencies of the modules that you have analyzed using SpyGlass through a hierarchical tree.
- Top-level modules are displayed as either root-level modules or leaf-level modules, depicted as yellow folders in the hierarchical tree.
- Root-level modules are modules that contain child modules (instances of other modules). A child module can contain leaf-level modules, black box modules, or children of its own.
- Leaf-level modules are modules that contain no children. Leaf-level modules can be top-level modules, or children of other modules (depicted as white file icons).
- Un-synthesizable modules are the modules that contain coded behavior and descriptions, but cannot be synthesized. In the design hierarchy, root-level un-synthesizable modules are depicted as green folders and leaf-level un-synthesizable modules are depicted as green files.
- Black box modules, depicted as black squares with a white “X” in the hierarchical tree, are modules that contain no coded behavior or description. These modules are usually children of other modules.
- Encrypted design modules are indicated by the 🔒 icon.

Right-Click Menu Options of Modules Page

When you right-click on a module in the *Modules* page, the following options appear in the shortcut menu:

Waive Messages By Module

Select this menu option to waive rules by the selected module.

When you select this option, the *The Waiver Editor Window* appears in which you can specify the required details.

Waive Messages By IP

Select this menu option to waive rules by IP.

When you select this option, the *The Waiver Editor Window* appears in which you can specify the required details.

Copy

Select this menu option to copy instance module name or module file path.

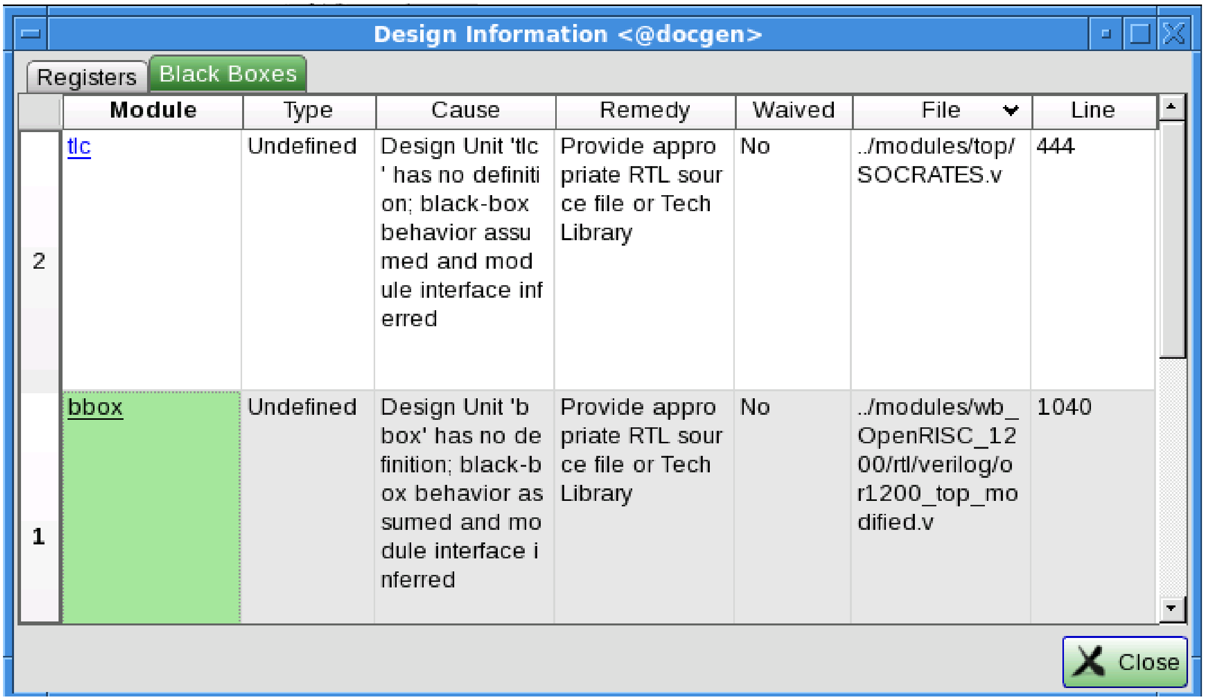
When you select this option, a sub-menu appears containing two options, Module name and Module file path.

Show BlackBox Info

Select this menu option to view the information about all the black boxes in the currently loaded module.

A black box can be identified with the ■ icon.

When you select this option, the *BlackBox Viewer* window appears, as shown in the following figure:



	Module	Type	Cause	Remedy	Waived	File	Line
2	tlc	Undefined	Design Unit 'tlc' has no definition; black-box behavior assumed and module interface inferred	Provide appropriate RTL source file or Tech Library	No	../modules/top/SOCRATES.v	444
1	bbox	Undefined	Design Unit 'bbox' has no definition; black-box behavior assumed and module interface inferred	Provide appropriate RTL source file or Tech Library	No	../modules/wb_OpenRISC_1200/rtl/verilog/or1200_top_modified.v	1040

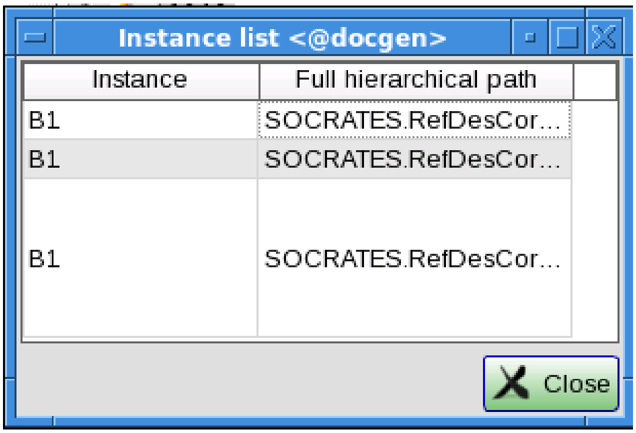
FIGURE 37. Black Box Info

For details on the above window, see *Black Box Viewer Window*.

Show Instance List

Select this menu option to view a list of instances in the design.

When you select this option, the *Instances* window appears, as shown in the following figure:



Instance	Full hierarchical path
B1	SOCRATES.RefDesCor...
B1	SOCRATES.RefDesCor...
B1	SOCRATES.RefDesCor...

FIGURE 38. Module View - Instance List

The above window lists instances along with complete hierarchical path of each instance.

Encrypted instances are indicated by the 🔒 icon.

When you right-click on an instance in the above window, following options appear in the right-click menu:

- *Properties*

  Select this option to display the *Instance List: Properties* window that shows instance properties, such as instance name, master module name, and complete hierarchical path of the instance.

- *Copy Instance Path*

  Select this option to copy instance path. You can then paste the path at a desired location.

- *Save Instance List*

  Select this option to save the instance list at the desired location.

- *Configure Page Size*

  Select this option to specify the maximum number of instances to be displayed at a time in the *Instances* window.

Set Stop Module

Select this menu option to set the selected module as stop module.

The specified module is the prefixed with the 🟥 icon to indicate it as the stop module.

Remove Stop Module

Select this menu option to not consider the selected module as a stopped module.

Set Ignore Module

Select this menu option to ignore the selected VHDL design unit or Verilog module for SpyGlass analysis.

Remove Ignore Module

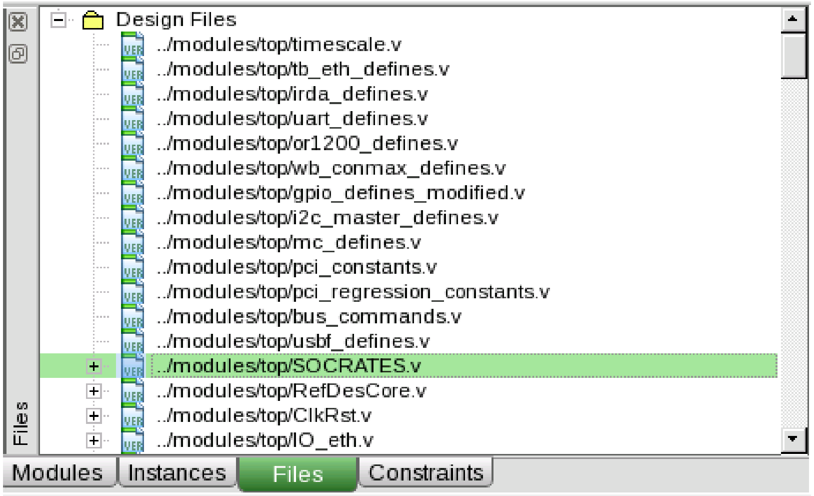
Select this menu option to consider the selected VHDL design unit or Verilog module for SpyGlass analysis.

File View Page

The File View page displays the list of source and library files under appropriate groups.

When you double-click on a file name in the *File View* page, SpyGlass displays the corresponding file in the *Source* section of the *HDL Viewer* with the cursor at the first line of the design unit description. Commented lines at the beginning of the file are skipped.

Figure 39 displays the File View page:



Design Files
- ../modules/top/timescale.v
- ../modules/top/tb_eth_defines.v
- ../modules/top/irda_defines.v
- ../modules/top/uart_defines.v
- ../modules/top/or1200_defines.v
- ../modules/top/wb_conmax_defines.v
- ../modules/top/gpio_defines_modified.v
- ../modules/top/i2c_master_defines.v
- ../modules/top/mc_defines.v
- ../modules/top/pci_constants.v
- ../modules/top/pci_regression_constants.v
- ../modules/top/bus_commands.v
- ../modules/top/usbf_defines.v
- ../modules/top/SOCRATES.v
- ../modules/top/RefDesCore.v
- ../modules/top/ClkRst.v
- ../modules/top/IO_eth.v

FIGURE 39. File View

This section explains the following topics:

- *Precompiled Files in the File View Page*
- *File Names in the File View Page*
- *Right-Click Menu Options in File View Page*

Precompiled Files in the File View Page

The File View page lists all the precompiled files under the Precompiled Files node. Under this node, the precompiled files are displayed in a hierarchical structure based on their logical library names.

Different icons are used to identify different types of precompiled files, as explained in the following table:

Icon	Indicates
📁	Precompiled files that are used in the current SpyGlass run
📁	Precompiled files that are not used in the current SpyGlass run but are a part of the libhdlfile specification
🔒	Precompiled files that are encrypted

To add/remove a precompiled file, right-click on that file and select the Add/Remove Precompile File option from the shortcut menu. The Precompile File Mapping dialog appears in which you can delete the required mapping.

File Names in the File View Page

SpyGlass displays the source and library file names based on their position relative to the current working directory at the time the file was added.

If you have added files located below the current working directory in the directory tree (call this Scenario1), SpyGlass lists the files in the *File View* page by their position in the directory path relative to the position of the project file. For example, if the current working directory's full path name is /SPY/verilog/version1/ and the file to be analyzed has the full path name /SPY/verilog/version1/sourcefiles/chip.v, the *File View* page entry will be sourcefiles/chip.v.

On the other hand, if you have added files located above the current working directory in the directory tree (call this Scenario2), the files are listed in the *File View* page by their position in the full directory path position. For example, if the current working directory's full path name is /SPY/verilog/version1/sourcefiles/analysis1/ and the file to be analyzed has the full path name /SPY/verilog/version1/sourcefiles/chip.v, the File view page entry will be as follows:

/SPY/verilog/version1/sourcefiles/chip.v

This difference in two scenarios will become relevant in the following predicaments:

In Scenario1, suppose that the working directory (/SPY/verilog/version1/) does not have a project file. Now consider that you invoke SpyGlass GUI from the working directory. Then, you add the /SPY/verilog/version1/sourcefiles/chip.v file as a source file (once you set up the rest of the options (such as design read and goal selection), and run SpyGlass analysis.

Suppose, that you do not run analysis at this point in time. Instead, you save the setup that you have made in a profile file (such as testcase1.spp) for analysis at a future time (such as later that night).

In Scenario2, suppose that the working directory (/SPY/verilog/version1/sourcefiles/analysis1/) does not have a project file. Now consider that you launch SpyGlass GUI from the working directory. Then, you add the /SPY/verilog/version1/sourcefiles/chip.v file as a source file, once you set up the rest of the options (such as design read and goal selection), you are ready to run the SpyGlass analysis.

Suppose, again, that you DO NOT run the analysis at this point in time. Instead, you save the setup that you have made in a profile file (such as testcase2.spp) for analysis at a future time (such as later that night).

Now, the predicament occurs when the current working directory's full path name has changed and if someone were to rename the /SPY/verilog/version1 directory to /SPY/verilog/version2. If everything else were left the same, testcase1.spp from Scenario1 would still be valid while testcase2.spp from Scenario2 would be unable to find the chip.v file.

The reason for this result is the difference in how the source files are recorded. With the relative relationship that the profile file and the source files maintain in Scenario1, changing the path does not affect the relationship of the files. However, with Scenario2, there is no relative relationship, and a change in the full directory path will cause the profile file to look for the source files in a directory that no longer contains them (or even exists).

Right-Click Menu Options in File View Page

The right-click options on the File View page are divided into the following categories:

- *Common Right-Click Options*

- *Right-Click Option for a Pre-compiled File*

Common Right-Click Options

When you right-click on a file name in the *File View* page, the following options appear in the shortcut menu:

Add File

Select this menu option to add a file in the *File View* page.

When you select this menu option, the *Add File(s)* dialog appears in which you can select the required file to be added.

Copy

Select this menu option to copy the path of the selected file.

Edit File

Select this menu option to edit the selected file appearing in the *File View* page.

When you select this menu option, an editor window appears in which you can make the required updates.

You can specify the type of editor window to be displayed in the *Text Editor* field of the *Miscellaneous Page* in the *Preferences* dialog. If you have not specified any value in this field, SpyGlass invokes the text editor specified by the EDITOR environment variable.

If you edit and save a source file listed in the *File View* page, the file name is displayed in red color to indicate that the file has been modified since last analysis run.

Waive Messages By File

Select this menu option to waive rule messages corresponding to the selected file.

When you select this option, the *The Waiver Editor Window* appears in which you can specify the required details.

Set Stop File

Select this menu option to set the selected file as a stop file.

The stop file is indicated with the stop file icon (STOP) prefixed to the design file name.

Set Ignore File

Select this menu option to ignore the selected file so that all design units specified in that file are ignored during SpyGlass analysis.

The ignored file is indicated with the ℹ icon prefixed to the design file name.

Right-Click Option for a Pre-compiled File

When you right-click on a precompiled file in the *File View* page, the Add/ Remove Precompile File option is displayed.

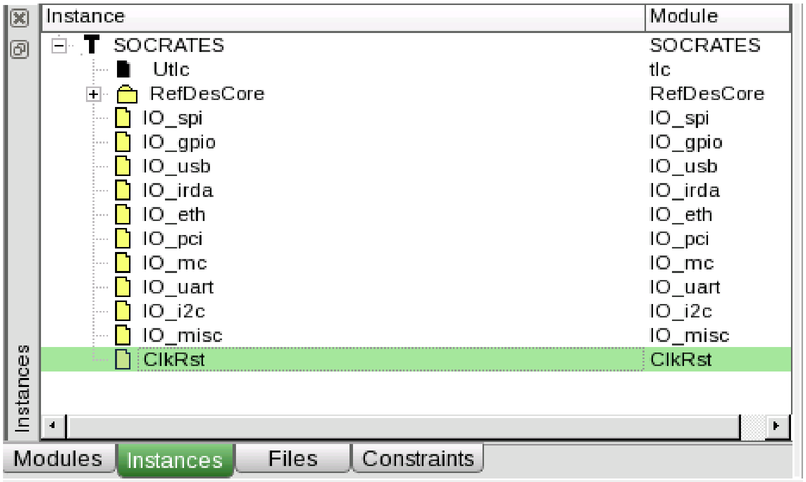
Select this menu option to add/remove a precompiled from a logical library.

When you select this menu option, the *Precompile File Mapping* dialog appears in which you can make the required modifications.

Instance View Page

The *Instance View* page is the instance hierarchy browser that displays design unit instances in each source and library file that you are analyzing with the current project file. This page remains blank until SpyGlass analysis has been completed or a valid project file is loaded.

Figure 40 illustrates the *Instance View* page:



Instance	Module
SOCRATES	SOCRATES
Utlc	tlc
RefDesCore	RefDesCore
IO_spi	IO_spi
IO_gpio	IO_gpio
IO_usb	IO_usb
IO_irda	IO_irda
IO_eth	IO_eth
IO_pci	IO_pci
IO_mc	IO_mc
IO_uart	IO_uart
IO_i2c	IO_i2c
IO_misc	IO_misc
ClkRst	ClkRst

FIGURE 40. Instance View

This section describes the following topics:

- *Overview of the Instance View Page*
- *Right-Click Options of the Instance View Page*

Overview of the Instance View Page

When you double-click on an instance name, the corresponding source file containing that instance appears in the source code section of the HDL Viewer. In addition, SpyGlass highlights the first line of the instance description (initial commented lines are skipped). Similarly, the corresponding gate is also highlighted in *The Modular Schematic Window*.

The details of the above page are discussed in the following points:

- The page displays the instance hierarchy containing multiple levels and dependencies of design unit instances that you have analyzed using SpyGlass.
- The first column displays instance names of a design unit.
- The second column displays module/design unit corresponding to the instances listed in the first column.
- Top-level instances are displayed as either root-level instances or leaf-level instances, depicted as yellow folders in the hierarchical tree.
- Root-level instances are instances that contain child instances, that is, instances of other design units. A child instance can contain leaf-level instances, instances of black box modules, or children of its own.
- Leaf-level instances are instances that contain no children. Leaf-level instances can be top-level instances, or children of other design unit instances (depicted as white file icons).
- Un-synthesizable design unit instances are the instances that contain coded behavior and descriptions, but cannot be synthesized. In the hierarchical tree, root-level un-synthesizable instances are depicted as green folders and leaf-level un-synthesizable instances are depicted as green files.
- Instances of black box modules, depicted with black file icons in the hierarchical tree, are design unit instances that contain no coded behavior or description.

Right-Click Options of the Instance View Page

When you right-click on an instance in the *Instance View* page, the following options appear in the shortcut menu:

Waive Messages by Module

Select this option to waive messages for the selected module.

Waive Messages by IP

Select this option to waive messages for the selected IP.

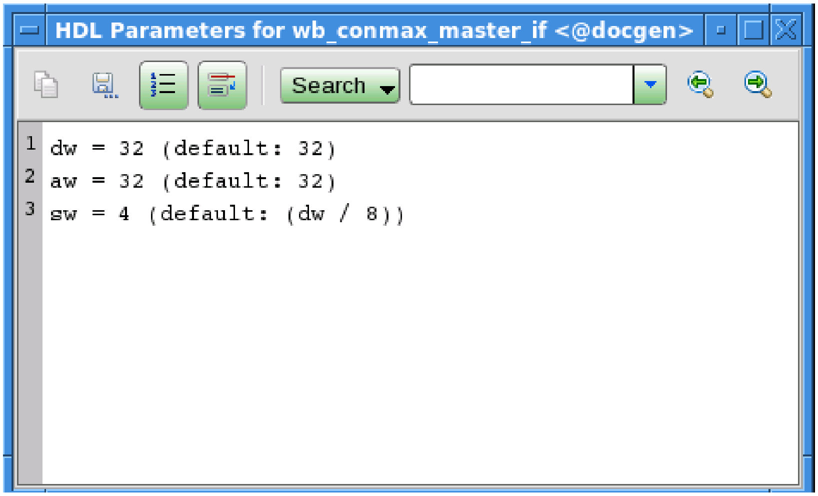
Copy

Select this menu option to copy instance name, complete instance path, module name, or module file path.

When you select this option, a sub-menu appears containing four options, Module name, Module file path, Instance path, Instance name.

Show HDL Parameters

Shows the list of HDL parameters for an instance. This option is available only for those modules or instances which have at least one HDL parameter defined. Following figure illustrates the sample HDL Parameters window for the module, `dct`:



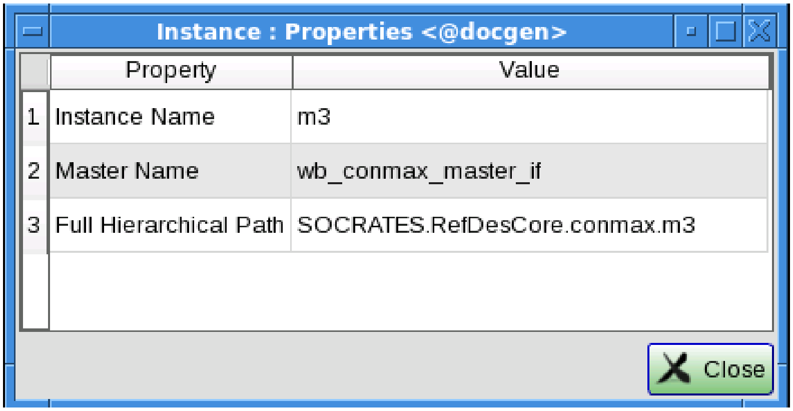
```
dw = 32 (default: 32)
aw = 32 (default: 32)
sw = 4 (default: (dw / 8))
```

FIGURE 41. Show HDL Parameters

Properties

Select this menu option to view the properties of listed instances.

When you select this option, the *Instance: Properties* dialog appears, as shown in the following figure:



Instance : Properties <@docgen>

	Property	Value
1	Instance Name	m3
2	Master Name	wb_conmax_master_if
3	Full Hierarchical Path	SOCRATES.RefDesCore.conmax.m3

FIGURE 42. Instance View - Properties

The above dialog displays the instance name, master module name, and the complete hierarchical path of the instance.

You can individually copy contents of this window by using the right-click shortcut menu options or <*Ctrl*>+<*C*> key combination.

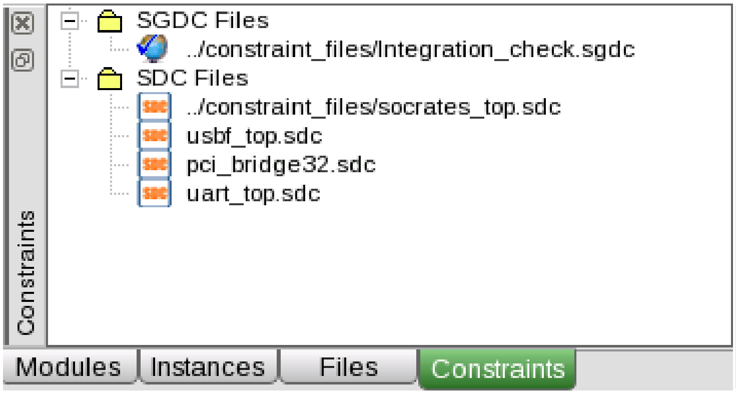
Configure Columns

Select this option to select the columns to be displayed in the Instance View page.

Constraints View Page

This page displays the list of constraint files (.sgdc), CPF files (.cpf), and UPF files (.upf) specified in the current SpyGlass run. These files are listed under appropriate categories, that is, *SGDC Files*, *CPF Files*, and *UPF Files* categories.

The following figure shows the *Constraints View Page* listing SGDC files:



SGDC Files
../constraint_files/Integration_check.sgdc
SDC Files
../constraint_files/socrates_top.sdc
usbf_top.sdc
pci_bridge32.sdc
uart_top.sdc

FIGURE 43. Constraints View

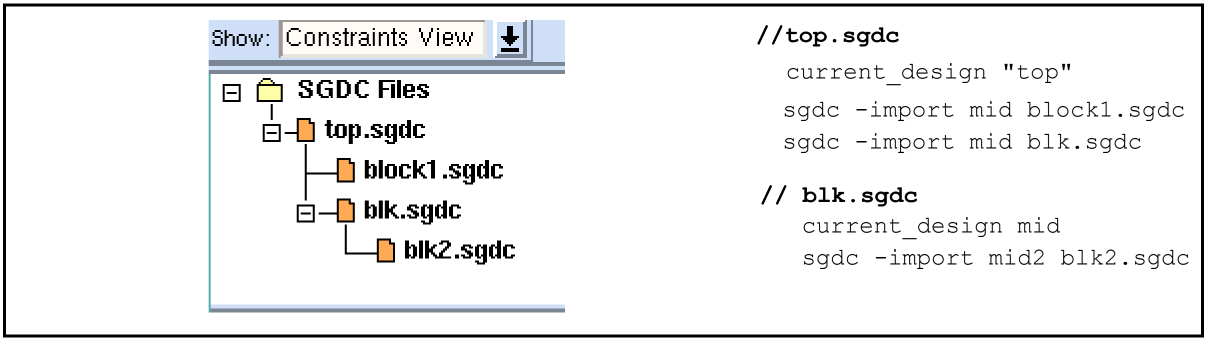
In the above view, when you double-click on a constraints file name in the *Constraints View* page, the corresponding file appears in the *Source* section with the cursor at the first line of the constraints file.

This section explains the following topics:

- *Viewing the Imported SGDC Files*
- *Viewing the Abstract Block-Level Files*
- *Right-Click Menu Options of the Constraints View Page*

Viewing the Imported SGDC Files

If an SGDC file contains the `-import` command specifications to import other SGDC files, the *Constraints View Page* shows the SGDC files hierarchically in the manner they are imported. This is explained in the following figure:



Show: Constraints View

```
//top.sgdc
  current_design "top"
  sgdc -import mid block1.sgdc
  sgdc -import mid blk.sgdc

// blk.sgdc
   current_design mid
   sgdc -import mid2 blk2.sgdc
```

FIGURE 44. View Imported Constraints

Viewing the Abstract Block-Level Files

If you specify an abstract block level SGDC file by using the `abstract_file` constraint, the *Constraints View Page* shows that file under the imported SGDC file.

For example, consider the following example:

```
# top.sgdc
current_design top
sgdc -import mid block/spyglass_reports/abstract_view/
block1_cdc_abstract.sgdc

# block1_cdc_abstract.sgdc
abstract_file -version 0.0 -scope cdc -block_file  block/
spyglass_reports/abstract_view/cdc/clock.sgdc
current_design mid
set_case_analysis -name am -value 1
```

For the above example, the following *Constraints View Page* appears:

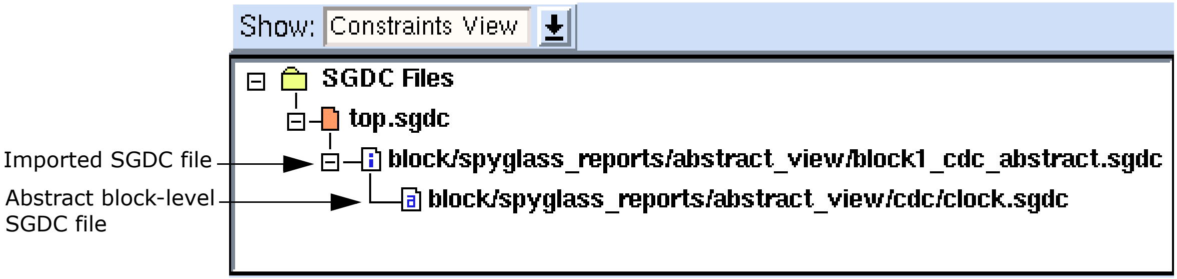

Show: Constraints View
SGDC Files
top.sgdc
block/spyglass_reports/abstract_view/block1_cdc_abstract.sgdc
block/spyglass_reports/abstract_view/cdc/clock.sgdc
Imported SGDC file
Abstract block-level SGDC file

FIGURE 45. Sample Constraints View Page

Right-Click Menu Options of the Constraints View Page

When you right-click on a module in the *Constraints View* page, the following options appear in the shortcut menu:

Add Files

Select this menu option to add a constraint file.

Copy

Select this menu option to copy the constraint file.

Edit File

Select this menu option to edit the selected file.

When you select this option, an editor window appears in which you can perform the required updates.

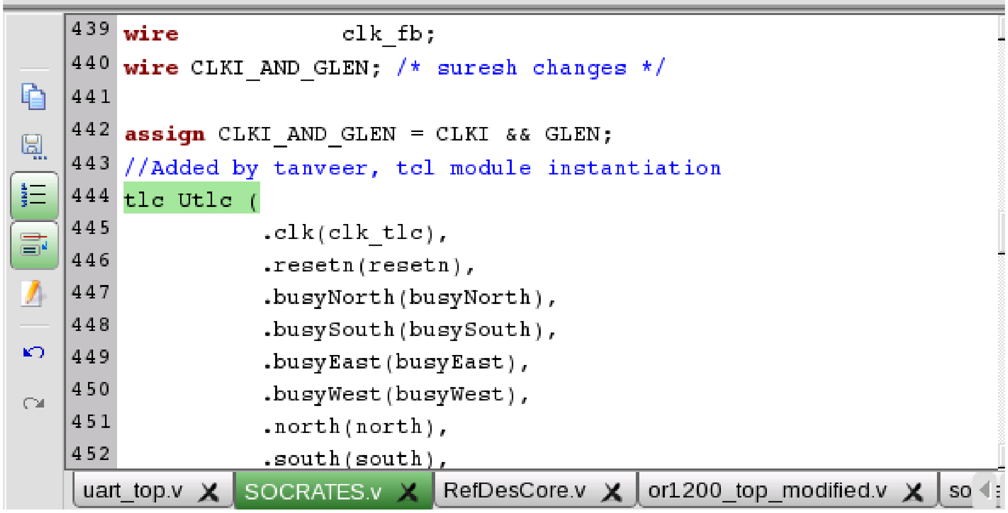
Waive Messages by File

Select this menu option to waive violation messages for the selected constraints file.

HDL Viewer Pane

The *HDL Viewer* pane displays the selected file, such as source file or SGDC file under separate tabs.

The following figure shows the *HDL Viewer* pane:



```
wire            clk_fb;
wire CLKI_AND_GLEN; /* suresh changes */

assign CLKI_AND_GLEN = CLKI && GLEN;
//Added by tanveer, tcl module instantiation
tlc Utlc (
            .clk(clk_tlc),
            .resetn(resetn),
            .busyNorth(busyNorth),
            .busySouth(busySouth),
            .busyEast(busyEast),
            .busyWest(busyWest),
            .north(north),
            .south(south),
```

FIGURE 46. HDL Viewer

When you select a particular pathway or gate in the schematic, the corresponding line is highlighted in the source code in the *HDL Viewer* pane.

You can display or hide *HDL Viewer* by clicking the *Show HDL Viewer* or *Hide HDL Viewer* option on the ribbon bar.

This section explains the following topics:

- *Color-Coding Scheme in HDL Viewer*
- *Searching in HDL Viewer*
- *Inactive Code Display*
- *Interaction with Other Windows*
- *Right-Click Options in the HDL Viewer Pane*
- *Navigation Bar*

Color-Coding Scheme in HDL Viewer

A specific color-coding scheme is used for the source code displayed in the *HDL Viewer* pane. For example, keywords are displayed in the maroon color.

To disable the color scheme in *HDL Viewer*, de-select the *Show Color Coded syntax in RTL Window* option in the *HDL Navigator Pane* page of the Preferences dialog:

NOTE: *The color-coding is visible only after the design is run. The coding of keywords differentiates between the Verilog and VHDL files and the modes (v2k, no_v2k, and so on) defined for VHDL files.*

Searching in HDL Viewer

To search for any text in the source code, select the *Edit > Find* menu option. When you select this option, the *Search* dialog appears in which you can specify the required search criteria.

Inactive Code Display

An RTL of a design unit can have the following inactive sections:

- ifdef...endif directives in a Verilog module

  These directives make specific sections of the module inactive if the corresponding condition is evaluated to false.
- translate_off/translate_on directives in a design unit

  These directives turn off sections of design on which SpyGlass analysis happens either partially or not at all.
- Comments in the source input files

Such inactive sections are highlighted in the green color in the *HDL Viewer* pane.

NOTE: *The inactive code display feature is not available for the precompiled VHDL library files.*

Interaction with Other Windows

When you double-click on an object in a source design file, that object is highlighted in various windows.

The following table specifies different types of objects that is highlighted in different windows:

Object Type	Window Actions	Highlight Details
Nets	Modular Schematic window is redrawn to show the design unit containing the probed net.	Net is highlighted.
	Net and all connected components are added to the Incremental Schematic window.	Net is highlighted across the hierarchical boundaries.
	Net is added to the Legend window list.	-
Ports	Modular Schematic window is redrawn to show the module containing the probed port.	The Port and its connected net are highlighted.
	Port and all components connected to the port’s connected net are added to the Incremental Schematic window.	Port and its connected net are highlighted across the hierarchical boundaries.
	Port is added to the Legend window list.	-
Instances	Modular Schematic window is redrawn to show the parent module containing the probed instance.	Instance is highlighted.
	Instance icon is added to the Incremental Schematic window.	Instance is highlighted
	Instance is added to the Legend window list.	-

Cross-probing of Nets To and From Schematic

For a current module, a net and all its end connections up to ports or leaf-level instances are highlighted in *The Modular Schematic Window*.

To view the schematic pathways or gates associated with a given object, open the *Modular Schematic* window and double-click the object in the *HDL Viewer*. Similarly, double-click an object in the *Modular Schematic* window and the corresponding file is opened in the *HDL Viewer*, if not already open, and the name is highlighted.

NOTE: *You can also select more than one objects for probing or remove an object from a set of probed objects in the schematic windows by clicking the relevant object while holding down the <Ctrl> key on the keyboard.*

SGDC to Schematic Cross-Probing

To implement SGDC to schematic cross-probing, double-click on a SGDC file appearing in the *Constraints View Page*. This displays the SGDC source file in the *HDL Viewer*.

Objects in this source file appear as hyperlinks. So when you select an object in the *HDL Viewer* and click the *Modular Schematic* or *Incremental Schematic* icons on the toolbar, the corresponding object is highlighted in the schematic window.

Right-Click Options in the HDL Viewer Pane

When you right-click on a signal in the *HDL Viewer* pane, the following options appear in the shortcut menu:

- *Common Options*
- *Options Visible at the Analyze Results Stage*

Common Options

Following are the common set of options that appear when you right-click on the *HDL Viewer* pane:

Waive Messages on this Line

Use this option to waive the violation messages, if available, for the selected code.

Clicking on this option opens the *The Waiver Editor Window*.

Open Editor

Use this option to edit a file.

When you select this option, a text-editing program appear as defined by the *Text Editor* setting in the *Miscellaneous Page* of the *Tools > Preferences* menu option.

If you have not specified a text editor in this setting, the text editor pointed to by the EDITOR environment variable is invoked. Otherwise, SpyGlass attempts to invoke the commonly-used text editor of your operating platform.

If you edit and save a source file listed in the *File View* page, the file name is displayed in the red color to indicate that the file has been modified since last analysis run.

Preferences

Use this option to open the *Tools > Preferences* window.

Options Visible at the Analyze Results Stage

The following additional options are visible only at the *Analyze Results* stage:

Instance

Use this option to view the information about an instance, such as, the scope of the instance and the location where the instance is declared.

Copy Hierarchical Path

Use this option to copy the complete hierarchical path of the module.

Properties

Use this option to view the signal properties. When you click the *Properties...* option, the Properties window is displayed that lists the signal name, the signal type, and the full hierarchical name of the signal.

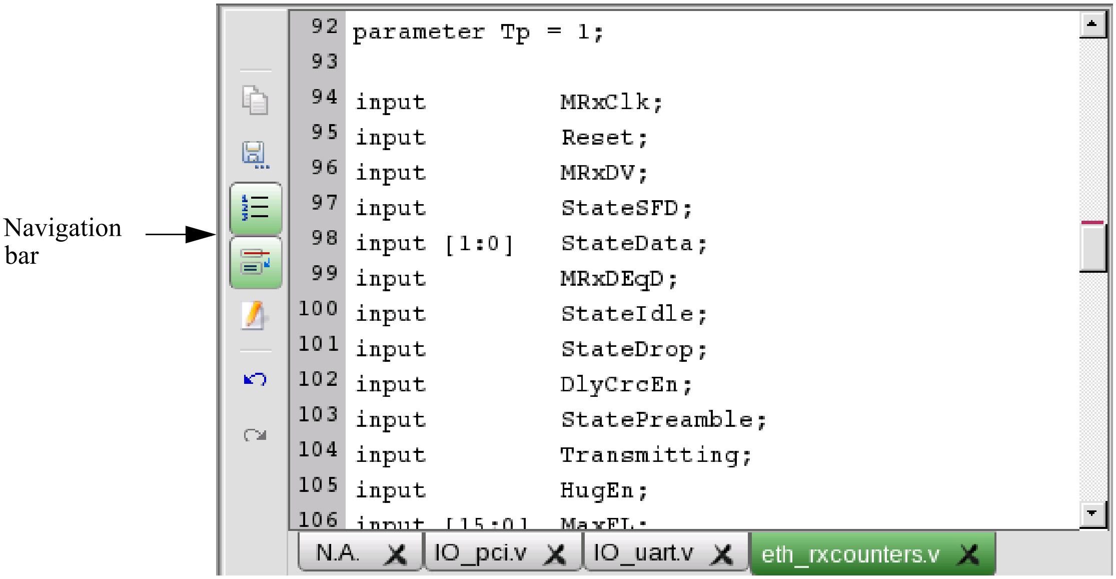
Set SpyGlass Constraints

Use this option to set the SGDC constraints on a signal. For more information on setting SGDC constraints, refer to the *Setting SGDC Constraints* section.

Navigation Bar

A navigation bar is used to edit and print an RTL file, probe between signals, and navigate between loads and drivers declared for a signal.

It is present towards the left side of the *HDL Viewer* pane. The following figure shows the navigation bar:



```
parameter Tp = 1;

input         MRxClk;
input         Reset;
input         MRxDV;
input         StateSFD;
input  [1:0]  StateData;
input         MRxDEqD;
input         StateIdle;
input         StateDrop;
input         DlyCrcEn;
input         StatePreamble;
input         Transmitting;
input         HugEn;
input  [15:0] MaxFL;
```

FIGURE 47. Navigation Bar

To navigate between the loads and drivers declared for a signal, double-click a signal to select it. Next, select from any of the following options:

- *Copy*

  Click this option to copy the selected text to the clipboard.

- *Save As*

  Click this option to save the RTL file with a different name or at a different location.

- *Line Numbers*

  Click this option to toggle on/off line numbers in the RTL file.

- *Open Editor*

Click this option to open the file in a text editor.

- *Prev cross probed line*

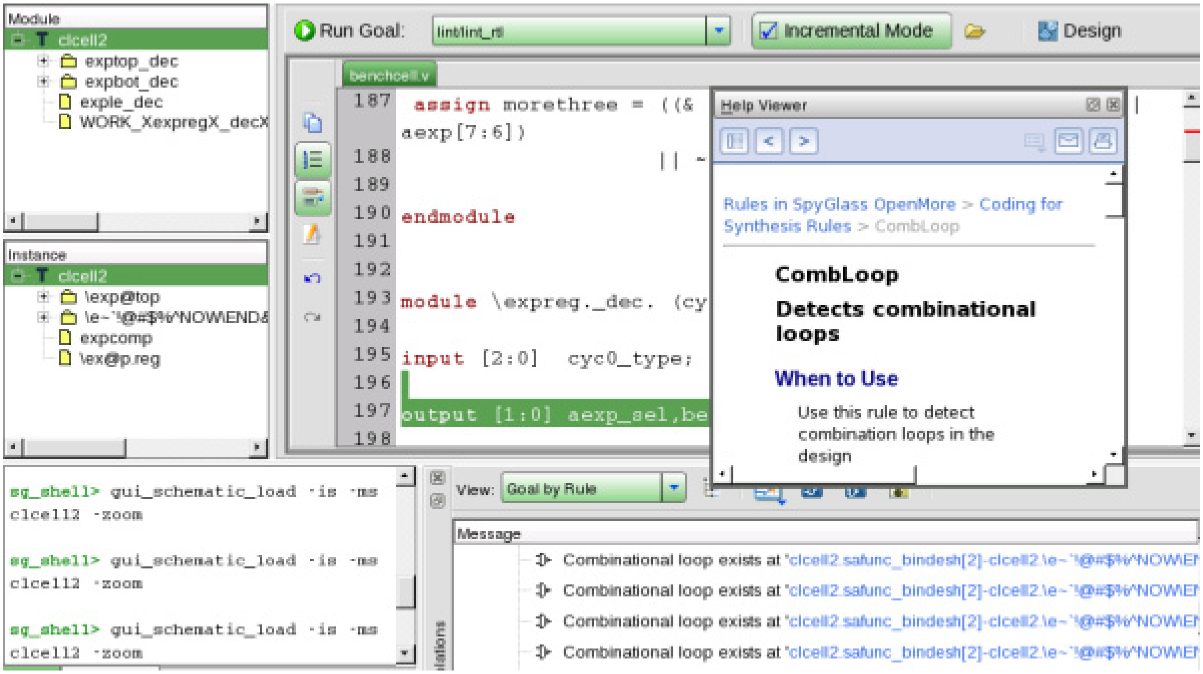
  Click this option to navigate to the previous cross-probed line in the HDL Viewer.

- *Next cross probed line*

  Click this option to navigate to the next cross-probed line in the HDL Viewer.

Help Viewer Pane

Displays help, for the option clicked in the SpyGlass Explorer GUI, in a dockable window. *Figure 48* shows a HTML Viewer pane displaying help for the CombLoop rule:



```
assign morethree = ((&
aexp[7:6])
                    || ~
endmodule

module \expreg._dec. (cy
input [2:0]  cyc0_type;
output [1:0] aexp_sel,be
```

Rules in SpyGlass OpenMore > Coding for Synthesis Rules > CombLoop

CombLoop

Detects combinational loops

When to Use

Use this rule to detect combination loops in the design

```
sg_shell> gui_schematic_load -is -ms
clcell2 -zoom

sg_shell> gui_schematic_load -is -ms
clcell2 -zoom

sg_shell> gui_schematic_load -is -ms
clcell2 -zoom
```

```
Combinational loop exists at 'clcell2.safunc_bindesh[2]-clcell2.\e~`!@#$%^NOW\EN
Combinational loop exists at 'clcell2.safunc_bindesh[2]-clcell2.\e~`!@#$%^NOW\EN
Combinational loop exists at 'clcell2.safunc_bindesh[2]-clcell2.\e~`!@#$%^NOW\EN
Combinational loop exists at 'clcell2.safunc_bindesh[2]-clcell2.\e~`!@#$%^NOW\EN
```

FIGURE 48. Help Viewer Pane

You can view the complete SpyGlass Help documents using the help viewer, without the need of an external browser.

HDL Navigator Pane

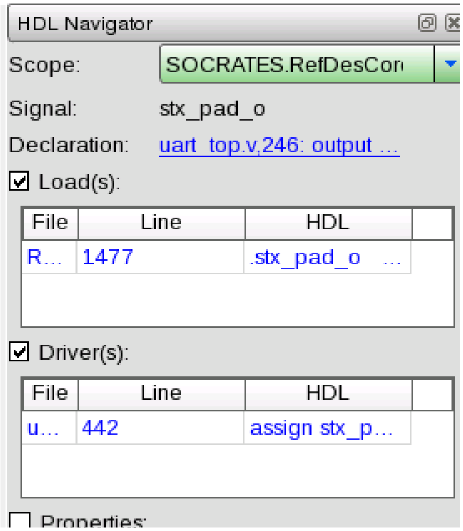
When you double-click on a signal, instance, or a macro in the HDL Viewer pane, the HDL Navigation pane displays the following:

- *Information for Signals*
- *Information for Instances*
- *Information for Macros*

Information for Signals

The *HDL Navigator* pane displays information about loads and drivers declared for a signal, which is double-clicked in the *HDL Viewer Pane*.

The following figure shows the *HDL Navigator* pane:



HDL Navigator

Scope: SOCRATES.RefDesCor

Signal: stx_pad_o

Declaration: uart_top.v,246: output ...

☑ Load(s):

File	Line	HDL
R...	1477	.stx_pad_o ...

☑ Driver(s):

File	Line	HDL
u...	442	assign stx_p...

□ Properties:

FIGURE 49. HDL Navigator

This pane appears only at the *Analyze Setup* stage.

The HDL Navigator pane displays the following information:

Scope *Signal* *Declaration* *Drivers* *Loads*

Scope

A signal may span across a design hierarchy, and therefore, may have different loads and drivers declared at different levels. You can set the scope for a signal in the *Scope* field.

By default, SpyGlass Explorer selects the most relevant scope for you. You can edit this scope depending on your requirement. If a signal has only one scope, the *Scope* field is updated automatically. If the signal has multiple scopes, you can select the relevant scope from the *Scope* drop-down list.

You can set the scope from the *Instance* view or by selecting a violation message.

Signal

This section contains a list of the signals selected in the *Source* section. You can select a signal from the drop-down list to view the drivers and loads declared for that signal.

Declaration

This section contains the file name and line number of an RTL source code where the signal is declared.

Loads

This section contains the loads declared for the signal selected in the *Signal* section. When you click a load in this section, the line where the load is declared is highlighted in the HDL Viewer. You can also find the file and line number of the RTL source code where the load is declared by placing the cursor over the load. Then, the file name and line number of the RTL source is displayed in a balloon window.

Drivers

This section contains the drivers declared for a signal selected in the *Signal* section.

When you click on a driver, the line where the driver is declared is highlighted in the *Source* section. To know the line and line where a driver is declared in the source code, place the cursor over that driver. A balloon appears displaying the line and file information.

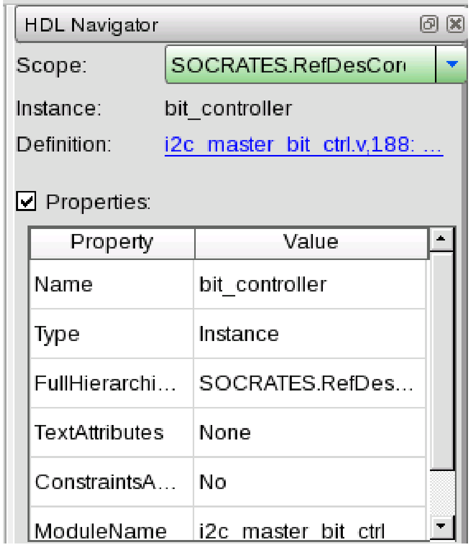
Properties

This section contains information about the signal, such as the signal name, signal type, and the full hierarchical name of the signal.

Information for Instances

The *HDL Navigator* pane displays information about the instance, which is double-clicked in the *HDL Viewer Pane*.

The following figure shows the *HDL Navigator* pane:



HDL Navigator

Scope: SOCRATES.RefDesCon

Instance: bit_controller

Definition: i2c_master_bit_ctrl.v,188: ...

☑ Properties:

Property	Value
Name	bit_controller
Type	Instance
FullHierarchi...	SOCRATES.RefDes...
TextAttributes	None
ConstraintsA...	No
ModuleName	i2c_master_bit_ctrl

FIGURE 50. HDL Navigator - Instance

The HDL Navigator pane displays the following information for Instances:

Scope *Instance* *Definition* *Properties*

Scope

An instance may span across a design hierarchy, and therefore, may have different loads and drivers declared at different levels. You can set the scope for a signal in the *Scope* field.

By default, SpyGlass Explorer selects the most relevant scope for you. You can edit this scope depending on your requirement. If an instance has only one scope, the *Scope* field is updated automatically. If an instance has multiple scopes, you can select the relevant scope from the *Scope* drop-down list.

You can set the scope from the *Instance* view or by selecting a violation message.

Instance

This section contains a list of instances selected in the *Scope* section. You can select an instance from the drop-down list to view the drivers and loads declared for that instance.

Definition

This section contains the file name and line number of an RTL source code where the instance is declared.

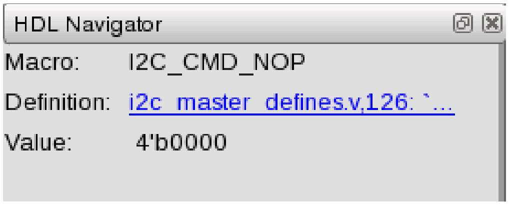
Properties

This section contains information about the instance, such as the instance name, instance type, and the full hierarchical name of the instance.

Information for Macros

The *HDL Navigator* pane displays information about the macro, which is double-clicked in the *HDL Viewer Pane*.

The following figure shows the *HDL Navigator* pane:



HDL Navigator

Macro: I2C_CMD_NOP

Definition: i2c_master_defines.v,126: `...

Value: 4'b0000

FIGURE 51. HDL Navigator Pane - Macros

The HDL Navigator pane displays the following information for Macros:

Macro

This section contains a list of macros selected in the *Source* section. You can select a macro from the drop-down list to view the corresponding definition and value for the selected macro.

Definition

This section contains the file name and line number of an RTL source code where the macro is declared. The information is displayed as a click-able link. Clicking on the link takes you to the respective macro definition in the source.

Value

This section displays the value for the selected macro.

Viewing Declarations of Signals as Loads/Drivers

By default, the search for loads and drivers stops at a module boundary. If the *Stop at module port declarations* option is selected from the *Options* drop-down list, loads and drivers stop at the location where the signal is declared at the start of the module. However, if you deselect the *Stop at module port declarations* option, loads and drivers also show usage of a signal outside/inside the module boundary.

The following may occur when the *Stop at module port declarations* option is selected:

- Clicking the signals in instantiations will display the loads or drivers inside the module being instantiated if the signal is an input or output signal respective to that module (declaration line of the signals/ports).
- Clicking the input port/signal declarations in a module will display the drivers outside the module (location where that module is being instantiated).
- Clicking the output port/signal declarations in a module will display the loads outside the module.
- Clicking any signal that is an input to its module will display the drivers as the port declaration of that signal.
- Clicking any signal that is an output of its module will display the loads as the port declaration of that signal.

The following may occur when the *Stop at module port declarations* option is not selected:

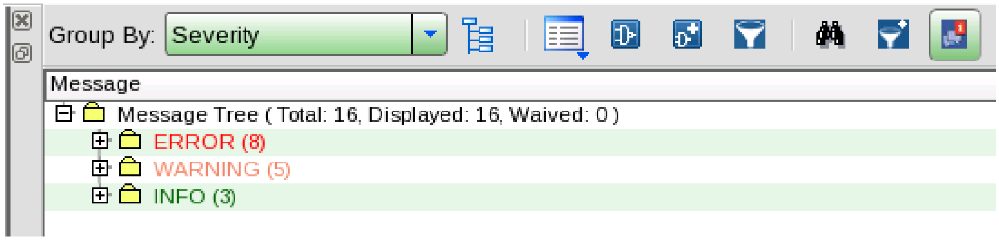
- Clicking the signals in instantiations will display the loads or drivers inside the module being instantiated if the signal is an input or output signal respective to that module (usage of the signal within the module being instantiated).
- Clicking the input port/signal declarations in a module will display the drivers outside the module (location where that module is being instantiated).
- Clicking the output port/signal declarations in a module will display the loads outside the module.
- Clicking any signal that is an input of its module will display the drivers outside the module (location where that module is being instantiated).

- Clicking any signal that is an output of its module will display the loads outside the module.

NOTE: *Clicking the input/output port/signal declarations in a module will always display the loads/drivers outside the module irrespective of whether the* ***Stop at module port declarations*** *option is selected or deselected.*

Violations Pane

The *Violations* pane displays violation messages generated after SpyGlass analysis. The following figure displays the *Violations* pane:



Group By: Severity

Message

Message Tree (Total: 16, Displayed: 16, Waived: 0)

ERROR (8)

WARNING (5)

INFO (3)

FIGURE 52. Results Pane

When you double-click on a violation message in the *Violations* pane, the corresponding design unit is displayed in the *HDL Viewer Pane* and the complete message path is highlighted in *The Modular Schematic Window*. For other rule messages, only the containing design unit is displayed.

SpyGlass displays a specific symbol prefixed with some violation messages. The following table describes the meaning of each symbol:

Symbol	Description
▷	Indicates that the message has a schematic associated with it
T	Indicates that selected rule messages and rule descriptions have associated text data or are multi-line messages
∿	Indicates that the message has a waveform associated with it
G	Indicates that the message has an FSM associated with it

This section describes the following topics:

- *Message Grouping*
- *Advanced Search in the Results Pane*
- *Message Help*

- *Right-click Menu Options in the Violations Pane*
- *Waiver Tree*

Message Grouping

You can group messages in different grouping orders by selecting an appropriate option from the *Group By* drop-down list.

This section describes the following topics:

- *Viewing the Grouping Level*
- *Customizing the Grouping Order*
- *Color Scheme in the Msg Tree Page*

Viewing the Grouping Level

The text displayed in the *Group By* text-box specifies the first level of the currently selected grouping order. To view all levels of the currently selected grouping order, place the cursor over the *Group By* text-box. A tool-tip appears that specifies all levels of the currently selected grouping order. Similarly, you can view all the levels of the grouping orders listed in the *Group By* pull-down list by placing the cursor over the grouping orders listed therein.

The *Group By* pull-down menu contains the pre-defined as well as the custom message grouping order. Selecting any of the options available in the *Group By* pull-down menu changes the message grouping in the corresponding message/waiver/filter tree.

Table 2 describes the pre-defined grouping order for various *Group By* options in the Message Tree:

TABLE 2 Grouping Order in Message Tree

Group By Option	**Grouping Order**
Severity	Severity Class -> Rule
Severity Label	Severity Label -> Rule
Goal by Severity	Goal -> Severity Class -> Rule

TABLE 2 Grouping Order in Message Tree (Continued)

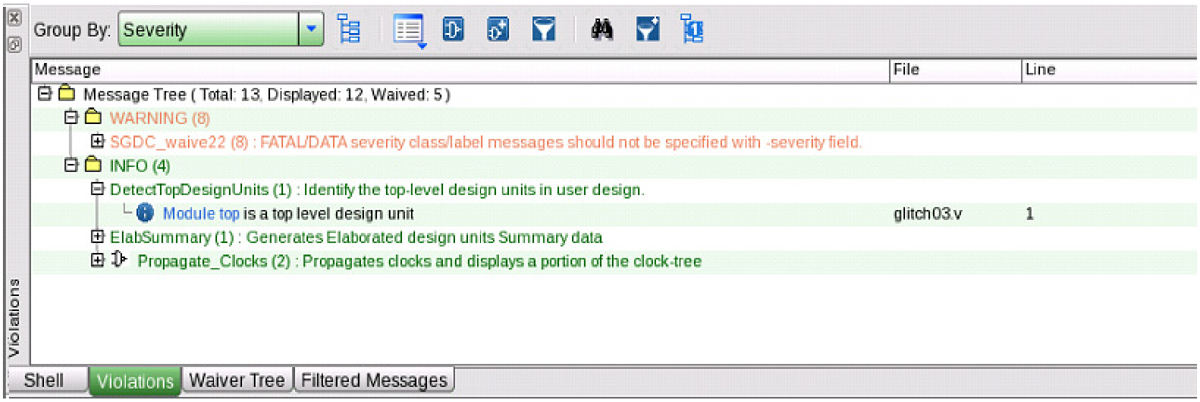
Group By Option	Grouping Order
Built-In	BuiltIn -> Severity Class -> Rule
Goal by Rule	Goal -> Rule (Goal Order)
UserTag	User Tag -> Severity Class -> Rule
Module	Module -> Severity Class -> Rule
Constraints Mode	SDC Mode -> Severity Class -> Rule
Constraints Mode - I	SDC Mode -> Rule

Table 3 describes the pre-defined grouping order for various *Group By* options in the Message Tree:

TABLE 3 Grouping Order in Waiver Tree

Group By Option	Grouping Order
Waiver by Severity	Waiver -> Severity Class -> Rule
Waiver by Module	Waiver -> Module -> Rule
Severity by Waiver	Severity Class -> Waiver -> Rule
Module by Waiver	Module -> Waiver -> Rule

Figure 53 shows the Message Grouping Order in the Message Tree when the Severity Group By option is selected:



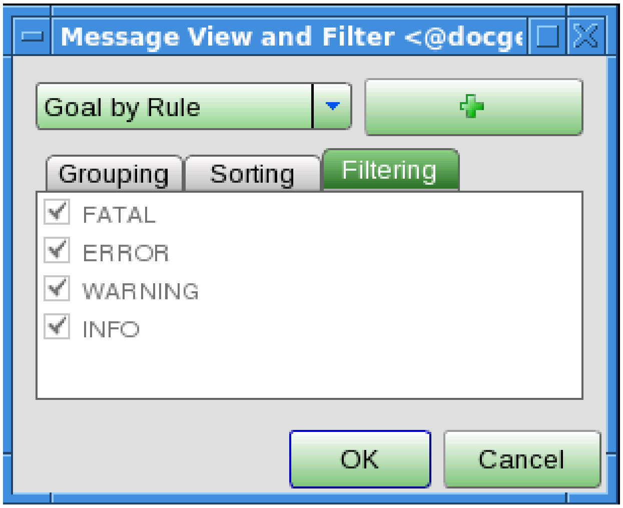
Group By: Severity

Message	File	Line
⊟ Message Tree (Total: 13, Displayed: 12, Waived: 5)		
⊟ WARNING (8)		
⊞ SGDC_waive22 (8) : FATAL/DATA severity class/label messages should not be specified with -severity field.		
⊟ INFO (4)		
⊟ DetectTopDesignUnits (1) : Identify the top-level design units in user design.		
Module top is a top level design unit	glitch03.v	1
⊞ ElabSummary (1) : Generates Elaborated design units Summary data		
⊞ Propagate_Clocks (2) : Propagates clocks and displays a portion of the clock-tree		

FIGURE 53. Group By Severity

Customizing the Grouping Order

In addition, you can also set the message grouping order according to your preference. To do so, select the icon in the Violations pane. The **Message View and Filter** dialog box is displayed as shown in *Figure 54*:



Goal by Rule

☑ FATAL
☑ ERROR
☑ WARNING
☑ INFO

FIGURE 54. Message View and Filter

In the **Message View and Filter** dialog box, you can select the Grouping, Sorting, and Filtering criteria and create your own custom grouping, sorting, and filtering.

If messages are not grouped by severity, the severity of the messages can be identified based on the following severity icons:

Message Severity	Severity Icon
FATAL	f
ERROR	⚠
WARNING	⚠
INFO	ℹ

Color Scheme in the Msg Tree Page

Messages in the *Msg Tree* page are displayed according to the following default color scheme:

Message Severity	Display Color
FATAL	Crimson
ERROR	Red
WARNING	Amber
INFO	Green

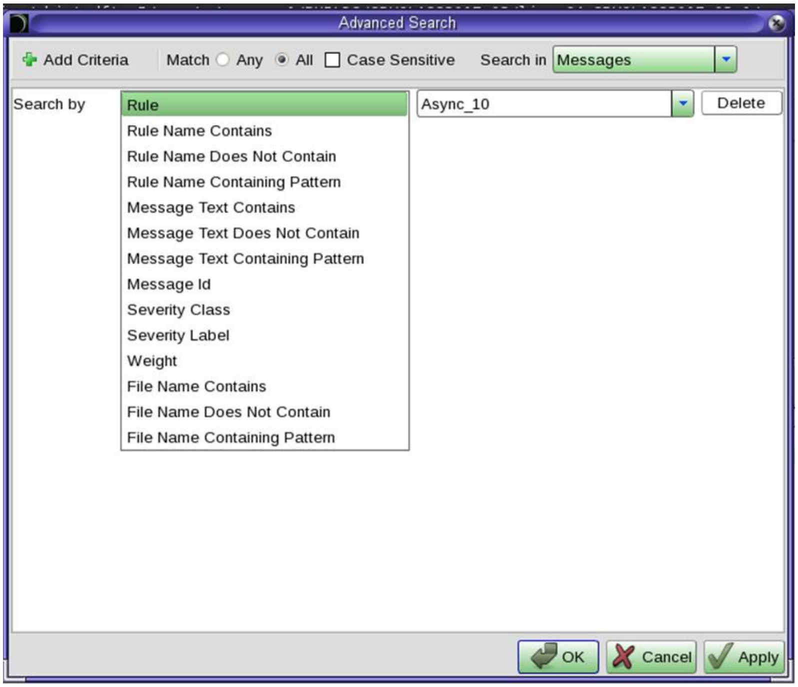
You can customize the above color scheme according to your preference by using the *Message Page* of the *Tools > Preferences* menu option. Alternatively, you can right-click on the message and select the *Preferences* option from the shortcut menu to open the *Preferences* dialog.

Advanced Search in the Results Pane

You can perform an advanced search in the *Msg Tree* page to search for a specific text.

To perform an advanced search, click in the *Violations* pane. This displays the *Advanced Search* dialog.

The following figure shows the *Advanced Search* dialog:



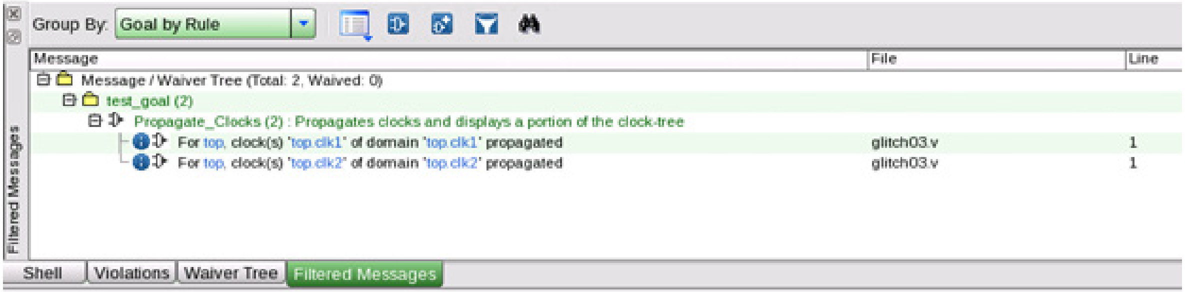
Advanced Search
Add Criteria
Case Sensitive
Rule Name Contains
Rule Name Does Not Contain
Rule Name Containing Pattern
Message Text Contains
Message Text Does Not Contain
Message Text Containing Pattern
Message Id
Severity Class
Severity Label
Weight
File Name Contains
File Name Does Not Contain
File Name Containing Pattern
Delete
OK
Cancel
Apply

Match ○ Any ● All □ Case Sensitive Search in Messages

Search by Rule Async_10

FIGURE 55. Message Tree - Advanced Search

Click on OK will to the message database and display the matched messages under the **Filtered Messages** tab as shown in *Figure 56*:



Message	File	Line
Message / Waiver Tree (Total: 2, Waived: 0)		
test_goal (2)		
Propagate_Clocks (2) : Propagates clocks and displays a portion of the clock-tree		
For top, clock(s) 'top.clk1' of domain 'top.clk1' propagated	glitch03.v	1
For top, clock(s) 'top.clk2' of domain 'top.clk2' propagated	glitch03.v	1

FIGURE 56. Filtered Messages

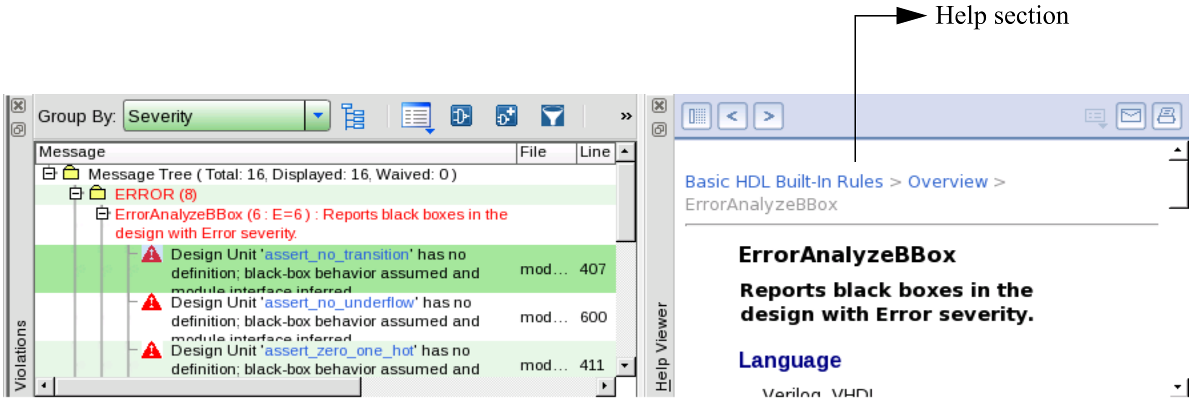
Alternatively, you can also specify the following Tcl command to perform search:

```
gui_show_messages [ get_messages  -of_rule [get_rules
{Propagate_Clocks}] -regexp]
```

Message Help

The *Help* section in the *Violations* pane displays different type of help information, such as message help and goal debug help.

The following figure displays the *Help* section:



Message	File	Line
Message Tree (Total: 16, Displayed: 16, Waived: 0)		
ERROR (8)		
ErrorAnalyzeBBox (6 : E=6) : Reports black boxes in the design with Error severity.		
Design Unit 'assert_no_transition' has no definition; black-box behavior assumed and module interface inferred	mod...	407
Design Unit 'assert_no_underflow' has no definition; black-box behavior assumed and module interface inferred	mod...	600
Design Unit 'assert_zero_one_hot' has no definition; black-box behavior assumed and	mod...	411

Basic HDL Built-In Rules > Overview > ErrorAnalyzeBBox

ErrorAnalyzeBBox

Reports black boxes in the design with Error severity.

Language

Verilog, VHDL

FIGURE 57. Message Help

Right-click Menu Options in the Violations Pane

When you right-click on a violation message, the following options appear in the shortcut menu:

Incremental Schematic

Displays the Incremental Schematic for the selected violation message.

Modular Schematic

Displays the Modular Schematic for the selected violation message.

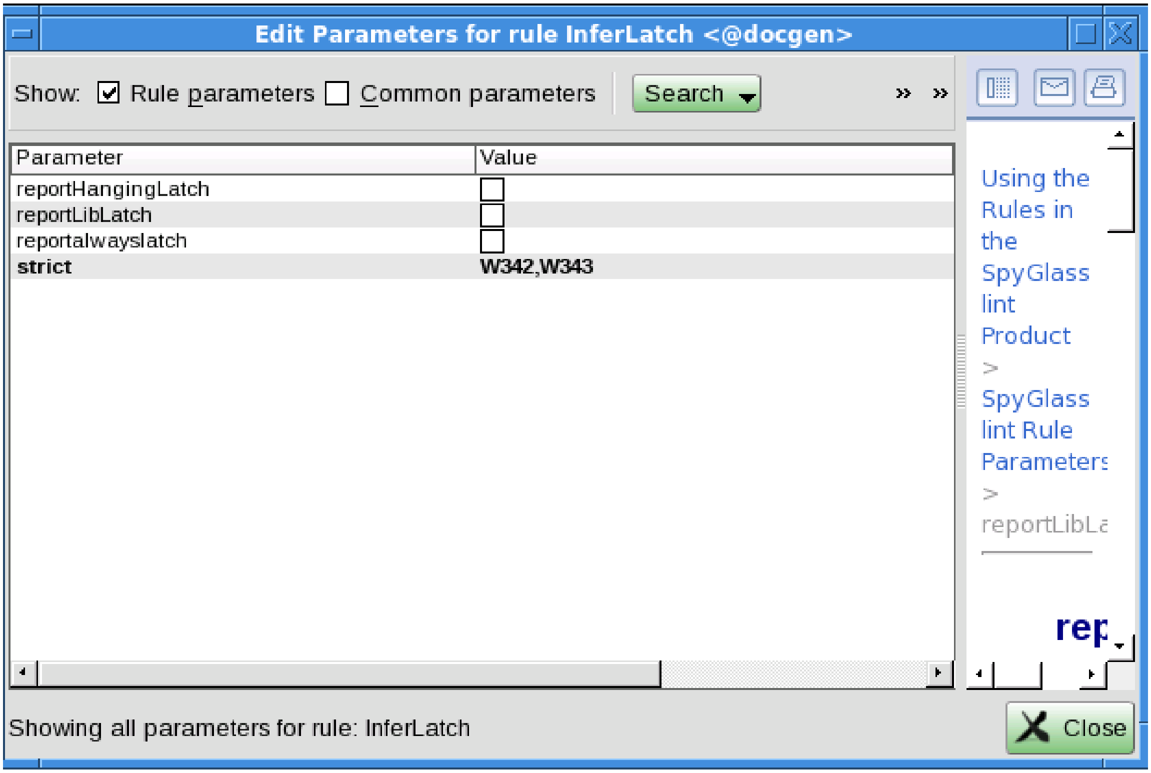
Open Spreadsheet

Opens the Spreadsheet Report.

Edit Parameters

Select this option to view or modify parameters applicable to the rule of the selected violation message.

When you select this option, the Edit Parameter for Rule dialog is displayed as shown in the following figure:



Edit Parameters for rule InferLatch <@docgen>
Show: ☑ Rule parameters □ Common parameters
Search
Using the Rules in the SpyGlass lint Product > SpyGlass lint Rule Parameters > reportLibLa
rep
Close

Parameter	Value
reportHangingLatch	□
reportLibLatch	□
reportalwayslatch	□
strict	**W342,W343**

Showing all parameters for rule: InferLatch

FIGURE 58. Set Parameters

By default, the above dialog displays parameters for the scenario that was last run.

If you want to change parameter values of a different scenario, perform the following steps:

1. Select the required scenario from the Scenario drop-down list.

NOTE: *You can also create a new scenario by selecting the Create New Scenario option from this drop-down list. When you select this option, the Create New Scenario dialog appears in which you can specify the name of the new scenario. By default, this new scenario has the same settings as that of the last modified scenario.*

2. Update parameter values as per your requirement.

3. Select the Run current goal option if you want to save the scenario and automatically run that scenario after the next step.
4. Click the OK button.

After performing the above steps, the selected scenario is saved with the updated parameter values.

While updating parameter values of a particular scenario, if you select another scenario from the Scenario drop-down list, a Warning dialog appears that prompts you to save changes in the current scenario.

If you want to specify additional details (other than parameters) for a scenario, click the Click here link in the above dialog. This displays a page under the Setup Goal tab under which you can specify additional details, such as SGDC files for a scenario.

Select Default Waiver File

Select this option to specify a waiver file to be used by default.

For details, see *Setting Default Waiver File*.

Waive Messages To

Select this option to waive messages to the selected waiver file. You can chose the default waiver file or create a new waiver file.

Waive Selected Messages

Select this option to waive the selected message.

For details on this option, refer to the *Waiving Selected Messages* topic of *SpyGlass Explorer User Guide*.

Save Messages

Select this option to save the contents of the *Msg Tree* page.

Clicking on this option displays the following sub-options:

- **All Hierarchies**: Enables you to save all the violations in the tree.
- **Selected Hierarchies**: Enables you to save the violations for the selected module or instance and its respective hierarchies.

Copy Text

Select this option to copy the message text.

Display Message

Select this option to view the complete message text in a separate balloon window.

You can select this option if a message text is long and does not fit in the available space.

Collapse Siblings

Collapses the category or node for which a violation is selected.

Preferences

Select this option to display the *Edit > Preferences* dialog.

Configure Columns

Allows you to select columns to display in the Violation pane. By default, only the **Message**, **File**, and **Line** columns are displayed in the Violations pane.

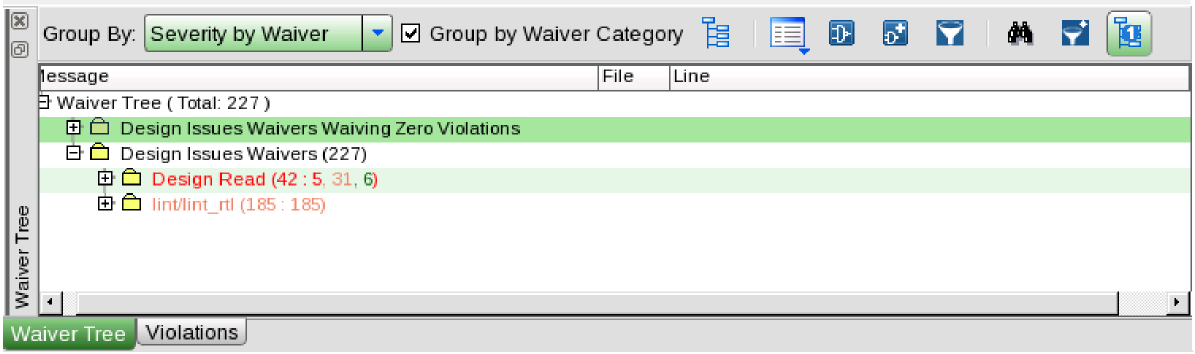
Using the Configure Columns option, you can also display **Severity**, **ID**, **Module**, and **Rule** for each violation in the Violations pane.

Waiver Tree

The *Waiver Tree* page displays the following details:

- A waiver tree containing messages waived specified by the waive constraints
- Comments applied to each of these messages
- Count of total number of waived messages

The following figure displays the *Waiver Tree* page:



Group By: Severity by Waiver

☑ Group by Waiver Category

lessage	File	Line
⊟ Waiver Tree (Total: 227)		
⊞ Design Issues Waivers Waiving Zero Violations		
⊟ Design Issues Waivers (227)		
⊞ Design Read (42 : 5, 31, 6)		
⊞ lint/lint_rtl (185 : 185)		

FIGURE 59. Waiver Tree

By default, messages in the above page are first grouped by waiver expressions and then by severity.

To group messages based on severity first and then waiver expressions, de-select the *Group by Waiver Category* option.

Right-Click Menu Options in the Waiver Tree Page

When you right-click on a waiver expression displayed in the *Waiver Tree* page, the following options appear in the shortcut menu:

Incremental Schematic

Displays the Incremental Schematic for the selected violation message.

Modular Schematic

Displays the Modular Schematic for the selected violation message.

Edit Parameters

Select this option to view or modify parameters applicable to the rule of the selected violation message.

Save Messages

Select this option to save the contents of the *Msg Tree* page.

Clicking on this option displays the following sub-options:

- **All Hierarchies**: Enables you to save all the violations in the tree.
- **Selected Hierarchies**: Enables you to save the violations for the selected module or instance and its respective hierarchies.

Copy Text

Select this option to copy the message text.

Display Message

Select this option to view the complete message text in a separate balloon window.

You can select this option if a message text is long and does not fit in the available space.

Collapse Siblings

Collapses the category or node for which a violation is selected.

Preferences

Select this option to display the *Edit > Preferences* dialog.

Configure Columns

Allows you to select columns to display in the Violation pane.

Interaction with Other Windows

You can probe a message in most pages of the Message window by double-clicking the message. Then, the related objects are highlighted in other windows that are open, as follows:

Object Type	Window Actions	Highlight Details
Message	The source file containing the message is loaded in the HDL Viewer and the related source code line is highlighted.	The source code line is highlighted.
	Modular Schematic window is redrawn to show the design unit containing the message.	Components involved in the message are highlighted.
	Components involved in the message are added to the Incremental Schematic window.	Components involved in the message are highlighted across the hierarchical boundaries.
	One or more entries are added to the Legend window list depending on the number of colors associated with the message (as set in the product).	-

The Modular Schematic Window

The *Modular Schematic* window displays a hierarchical schematic layout of the modules analyzed by SpyGlass. This window displays data based on the violation message selected or a module selected from the *Modules Page*.

To display the *Modular Schematic* window, click the *Modular Schematic icon* in the *Violations* pane.

Figure 60 shows the *Modular Schematic* window:

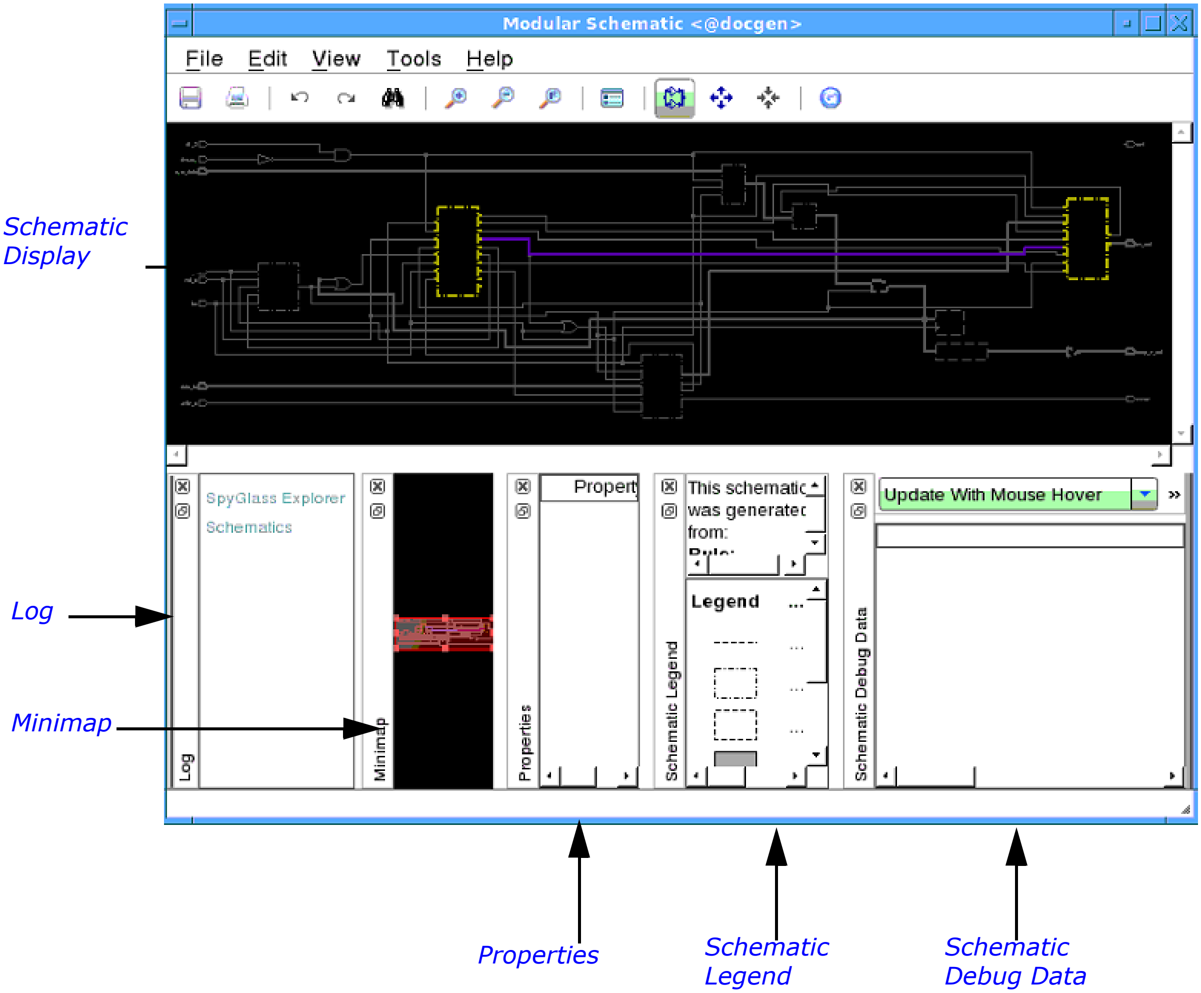


FIGURE 60. The Modular Schematic Window

The *Modular Schematic* window is divided in to the following sections:

Schematic Display	*Schematic Legend*	*Schematic Debug Data*
Log	*Minimap*	*Properties*
Toolbar	*Menu bar*	

Schematic Display

This section displays all the components (across hierarchical boundaries) pertaining to the selected message or a probed component.

You can select multiple messages at a time and view all the components pertaining to the selected messages.

When you try to view a flattened design unit with very large number of objects (instances, nets, and ports) in the main schematic window, a warning is displayed and the schematic window is closed automatically. Please note that this issue relates to flattened design unit size and is not related to the overall design size.

Set the value of the AUTOENABLE_HUGE_SCHEMATIC_DISPLAY configuration key to yes. Once changed to yes, a warning is still displayed but the user can continue the action that may result in a long wait, hanging of SpyGlass, or out of memory situation. This would depend on design size and available memory on the system.

Schematic Legend

This section shows the following information:

- The meaning of colors and text attributes applied on the objects in the schematic.
- The rule name, rule summary, and message of the selected violation.

NOTE: *The information appearing in this section applies to the first selected message, that is, the primary message. It does not apply to the secondary messages.*

Schematic Debug Data

This section shows messages based on the actions performed in the schematic.

Log

Display the log for the actions performed in the Schematic Window

Minimap

Provides a concise view of the whole schematic. This facilitates in debugging very large designs.

Properties

Hovering over any object or clicking any object in the schematic updates the values in the Properties pane

Menu bar

See *The Modular Schematic Window Menu Bar* section.

Toolbar

The toolbar contains buttons, such as *Print*, *Find*, *Undo*, *Redo*, *Zoom In*, *Zoom Out*, *Zoom Fit*, and *Preferences*.

This section explains the following topics:

- *The Modular Schematic Window Menu Bar*
- *Using the Back Annotation Support*
- *Creating RTL Groups*
- *Interaction with Other Windows*
- *Using the Cursor*

The Modular Schematic Window Menu Bar

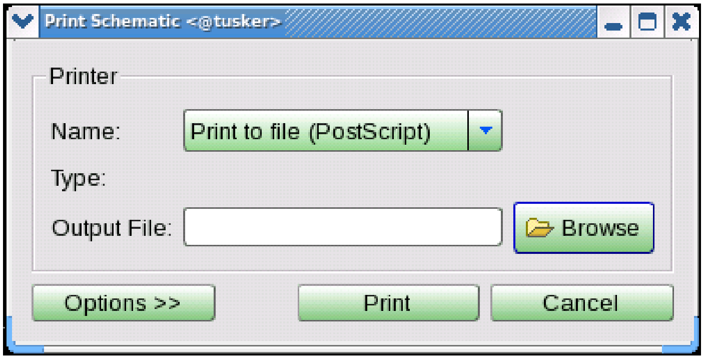
The Modular Schematic window menu bar has the following options:

File > Print	*File > Save As*	*File > Close*
Edit > Undo	*Edit > Redo*	*Edit > Clear All Items*
Edit > Find...	*Edit > Show Case Analysis*	*Edit > Clear Case Analysis*
Edit > Preferences	*View > Zoom*	*View > Pan*
View > Regenerate	*View> Windows*	*Tools > Preferences*

File > Print

Use this menu option to print the schematic or save it as a Postscript file.

When you select this menu option, the Schematic Print dialog appears:



Print Schematic <@tusker>

Printer

Name: Print to file (PostScript)

Type:

Output File:

FIGURE 61. Modular Schematic Window - Schematic Print Dialog

Select or enter as follows:

- *Name*

  Select any of the following option from the drop-down list:

Option	Description
Print to file (PostScript)	Saves the schematic as a postscript file
Print to file (SVG)	Saves the schematic in an SVG (Scalable Vector Graphics) format
Print to file (EPS)	Saves the schematic in an EPS (Encapsulated PostScript) format
Print to file (PDF)	Saves the schematic as a PDF file

- *Type*

  Displays the type of file selected.

- *Output File*

  Specify the full path name of the destination file. Alternatively, click *Browse...* and navigate through the directory structure to set the destination file name. Click Save in the Save File dialog to set the file name.

NOTE: *Clicking Save in Save File dialog does not save the schematic to a postscript file; it only sets the file name. You still need to click OK in the Print Dialog to save the schematic to a postscript file.*

- *Color Mode*

  Select *Inverted Color* to print or save the schematic in a reverse color mode, select *Mono* to print or save the schematic in monochrome mode, or select *Color* to print or save the schematic in a color mode.

- *Sheet Size*

  Select the target paper size from the supported paper sizes.

- *Orientation*

  Select *Landscape* to print or save the schematic in a landscape orientation, select *Portrait* to print or save the schematic in a portrait orientation, or select *Auto* to let SpyGlass decide the best-fit orientation.

- *View to Print*

  Select *full* to print or save the full schematic or select *current* to print or save the currently displayed view of the schematic.

- *Print Highlight Info*

If you have selected to print or save the schematic in Color mode, you can set this option to print the highlighted objects with their respective colors.

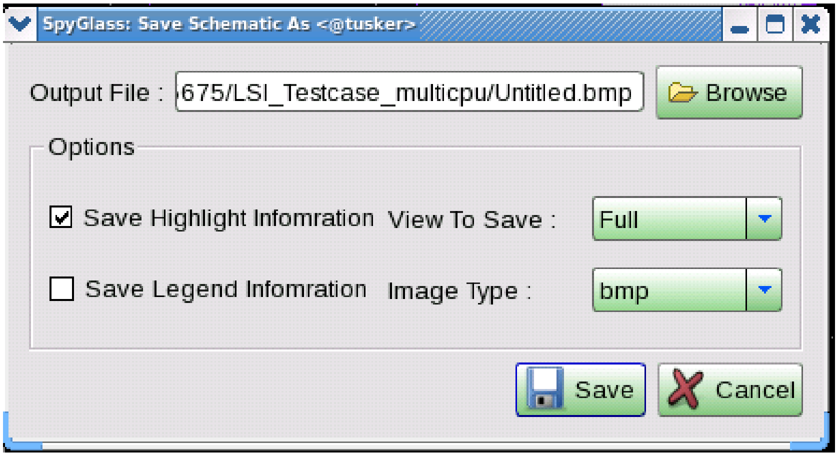
- *Print Legend Info*

  Prints the legend information, if available, for a schematic.

You can also invoke the Schematic Print dialog by using the <*Ctrl*>+*P*> key combination on your keyboard or by clicking the (🖶) button on the Modular Schematic toolbar located below the menu bar.

File > Save As

Use this menu option to save the schematic view as an image. When you select this menu option, the Save Schematic As dialog appears as shown in *Figure 62*:



SpyGlass: Save Schematic As <@tusker>

Output File : 675/LSI_Testcase_multicpu/Untitled.bmp

Options

☑ Save Highlight Infomration View To Save : Full

□ Save Legend Infomration Image Type : bmp

FIGURE 62. Modular Schematic Window - Save Schematic As

Specify the following information:

- *Output File*

  Enter the location where you want to save the image in the *Output File* text field. Alternatively, click (📂) and browse to the location where you want to save the image.

Click *Save* to save the image.

You can also open the **Save Schematic As** dialog by clicking (💾) on the Schematic toolbar.

- *Save Highlight Information*

  Select this option to save the visible portion of the schematic along with the highlighted information as an image.

- *Save Legend Information*

  Select this option to save the visible portion of the schematic along with the legend information as an image.

- *View to Save*

  Select one of the following options from the drop-down list:

  ❒ Full

  ❒ Fullfit

  ❒ Current

- *Image Type*

  Click the *Image Type* drop-down list and select the format in which you want to save the image. The available formats are *bmp*, *gif*, *jpeg*, and *png*.

File > Close

Use this menu option to close the Modular Schematic window.

You can also close the Modular Schematic window by pressing the *<Ctrl>+Q>* key combination on your keyboard.

Edit > Undo

Use this menu option to undo the last probe action in the Modular Schematic window.

You can also undo the last probe action by pressing the *<Ctrl>+<Z>* key combination on your keyboard or by clicking the (↶) button on the Modular Schematic toolbar.

Clicking the (↶) button repeatedly leads to the object from where the probe was initiated.

Edit > Redo

Use this menu option to redo a probe action that was previously undone in the Modular Schematic window.

You can also redo a probe action by pressing the <Ctrl>+<Y> key combination on the keyboard or by clicking the (↷) button on the Modular Schematic toolbar.

NOTE: *The Redo option is enabled only when you have undone a probe action once.*

Edit > Clear All Items

Use this menu option to clear all selections in the Modular Schematic window.

You can also clear all selections by pressing the *<Shift>+C* key combination on your keyboard.

Edit > Find...

Use this menu option to find a net, port, or instance in the schematic.

See *Finding Objects* for more details.

You can also invoke the Find Dialog by pressing the *<Ctrl>+<F>* key combination on the keyboard or by right-clicking anywhere in the Modular Schematic Window and selecting Find... You can also invoke the Find dialog by clicking the (🔍) button on the Modular Schematic toolbar.

Edit > Show Case Analysis

Use this menu option to annotate the related show case analysis message (if any) as an auxiliary message over the currently displayed message.

When you select this menu option, a sub-menu appears. Select an appropriate option from this sub-menu based on the type of violations you want to view.

When you select the required option from the sub-menu, the *Show Case Analysis* dialog appears, as shown in the following figure:

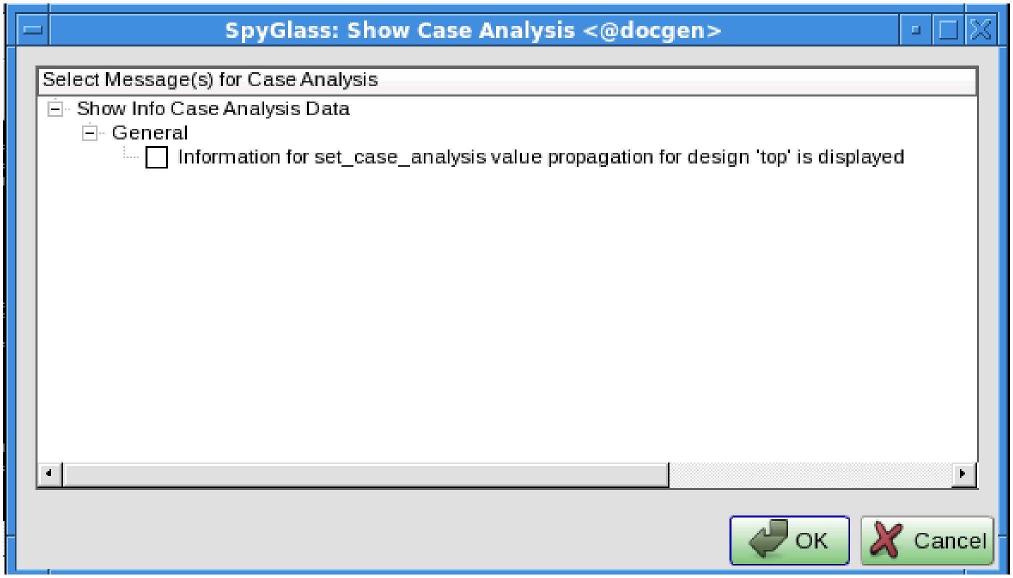

SpyGlass: Show Case Analysis <@docgen>
Select Message(s) for Case Analysis
Show Info Case Analysis Data
General
□ Information for set_case_analysis value propagation for design 'top' is displayed
OK
Cancel

FIGURE 63. Modular Schematic Window - Show Case Analysis

In the above dialog, you can select the message(s) to be annotated.

NOTE: *You cannot select the SHIFT, CAPTURE, and CAPTURE_ATSPEED mode options together. For example, if you have selected the SHIFT mode options and you try to select the CAPTURE options also, the Conflicting Case Analysis dialog appears prompting you to clear the previously selected options (SHIFT mode options) to enable the selection of CAPTURE mode options.*

You can also press the <*Ctrl*>+*A* key combination on the keyboard to invoke the Show Case Analysis dialog.

Edit > Clear Case Analysis

Use this menu option to clear all case analysis annotations in the current view.

You can also press the <*Shift*>+*A* key combination on the keyboard to clear all case analysis annotations.

Edit > Preferences

Use this option to invoke the Preferences dialog. See *Tools > Preferences* for more details.

You can also invoke the Preferences dialog by pressing the *<Ctrl>+<R>* key combination on the keyboard or by clicking the (▤) button on the Modular Schematic toolbar.

View > Zoom

Select one of the following options:

TABLE 4 Zoom Menu Options

Option	**Description**	**Keyboard Shortcut**
In	Zoom in the current schematic view	*<Z>*
Out	Zoom out of the current schematic view	*<O>*
Fit	Display the complete schematic of the current design unit	*<F>*

View > Pan

Select one of the following options:

TABLE 5 Pan Menu Options

Option	Description	Keyboard Shortcut
Left	Pan left in the current schematic view	Left arrow key
Right	Pan right in the current schematic view	Right arrow key
Up	Pan up in the current schematic view	Up arrow key
Down	Pan down in the current schematic view	Down arrow key

View > Regenerate

Use this option to regenerate the Modular Schematic.

View> Windows

Use this option to toggle on/off various views in the Modular Schematic window:

Table 6 describes the various options available under the View >Windows menu:

TABLE 6 View>Windows Menu

Option	Description
Log	Prints appropriate messages based on the actions performed in the schematic
Mini-map	Displays the mini-map pane, which provides a concise view of the whole schematic making it easier to debug very large designs
Properties	Displays the Properties pane. Hovering over any object or clicking any object in the schematic updates the values in the Property pane
Schematic Legend	Displays the Schematic Legend window. This window displays the meaning of highlighted colors and the text attributes applied on the objects in that schematic window
Schematic Debug Data	

Tools > Incremental Schematic

Use this option to display the *The Incremental Schematic Window*.

Tools > Preferences

Use this option to display the Preferences window.

Using the Back Annotation Support

The *Modular Schematic* window enables back-annotation between the schematic and the source code.

For example, when you select a gate in the schematic, SpyGlass highlights the corresponding line in the RTL code (shown in *HDL Viewer Pane*) that generated that gate.

Black boxes appear in a different color in the schematic to distinguish them from other objects.

Creating RTL Groups

An RTL group is a group of a logic, such as:

- *Always/Process Blocks in the RTL Code*
- *Vectored Instances in the RTL Code*
- *Registers and Combinational Logic in the RTL Code*

Creating RTL groups in the *Modular Schematic* window makes it easier to navigate in the schematic and trace a particular logic.

RTL grouping is particularly useful in viewing a module that has many leaf-level instances.

Use the following toolbar buttons in the schematic to work with RTL groups:

Button	Description
⊳	Click this button to enable the RTL grouping.
✥	Click this button to expand all groups and their subgroups in the current module. Other modules remain unaffected.
✥	Click this button to collapse all expanded subgroups and objects in the current module to view consolidated groups. Other modules remain unaffected

The filtered schematic view (Incremental Schematic or IS), which is commonly used for debugging rule violation messages, does not currently utilize the grouping capabilities in order that information does not get hidden while debugging specific schedule instances. Moreover, gate-level (structural) schematics also do not utilize the grouping capabilities

Always/Process Blocks in the RTL Code

The following RTL blocks in a module definition are grouped in the modular schematic view:

- process blocks (used in VHDL)
- always blocks (used in Verilog)

These blocks are further distinguished as sequential and combo blocks in the schematic view, as shown in the following figure:

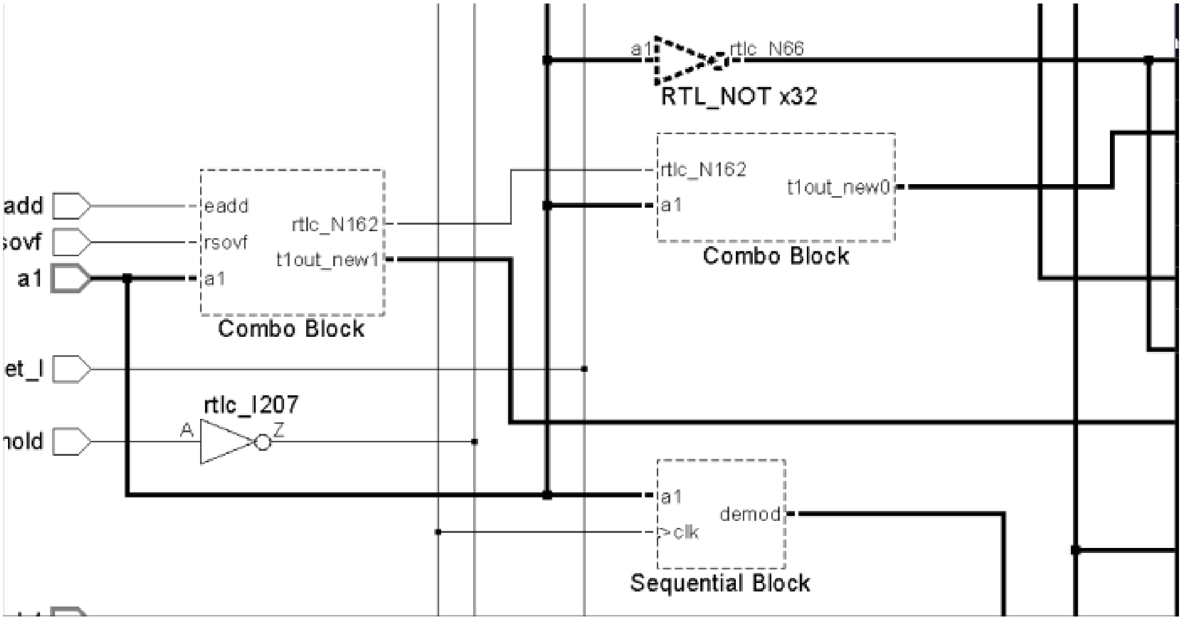


FIGURE 64. Always/Process Blocks

In the above view:

- Each of these blocks is shown in the schematic display as a rectangle with dashed boundary. The boundaries of instances for which you can dive down their hierarchy are shown in a dash-dot pattern.

- The sequential blocks and the combo blocks are named as *Sequential Block* and *Combo Blocks*, respectively in the view.
- These blocks appear as single entities. To view the objects and sub-groups contained in the group, you can expand these entities by double-clicking them.

Vectored Instances in the RTL Code

The *Modular Schematic* window displays the vectored or arrayed instances in the RTL source code as single components with appropriate bus pins. Vectors share the same master module for all the instances in the code, and therefore, the names of the vector groups are derived from the master modules of the instances and the number of instances in that group.

For example, RTL_MUX x32 means that the group contains 32 instances of RTL_MUX.

The vector groups are generally shown in the schematic display as a rectangle with a thicker dotted boundary than normal groups or instances (analogous to net bundles being thicker than single nets). If a vector group has the same input/output mapping as that of the constituent instances, the vector group is shown with the same symbol as that of the instances in the schematic view.

For example, a vector group containing a buffer with a bus input and a bus output is shown by a buffer symbol having the same bus as input/output.

The following illustration shows a vector group representation in the Modular Schematic window. Note that the vector group has the same mapping as that of the constituent instances and therefore it is represented using the same symbol as that of the instances.

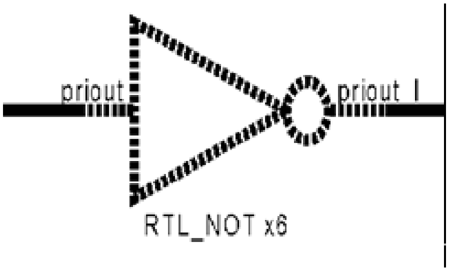


FIGURE 65. Vector Group Representation

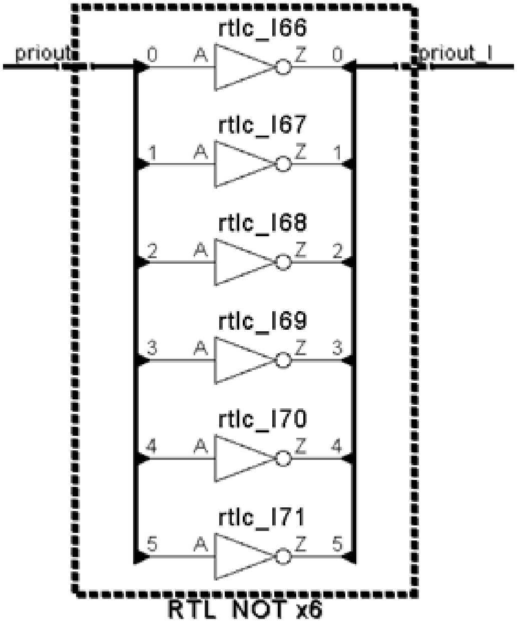


FIGURE 66. Vector Group Representation (Expanded)

Registers and Combinational Logic in the RTL Code

Consider the following figure:

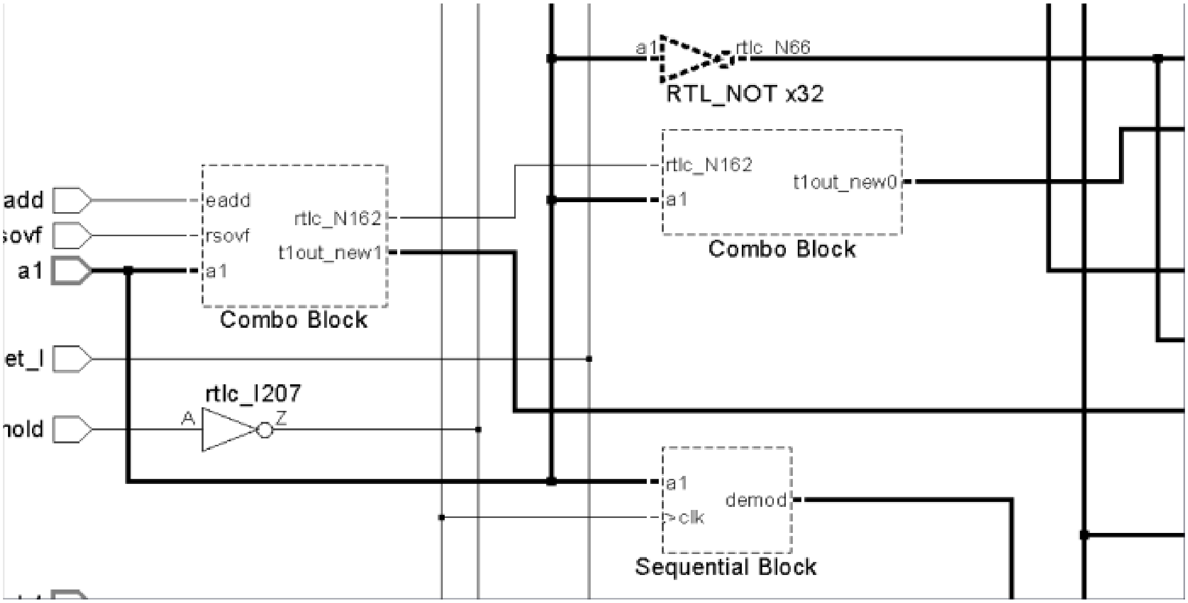


FIGURE 67. Vector Group Representation (Expanded)

The figure displays a group having purely combinational logic as *Combo Block* in the design. You can easily distinguish the combinational logic blocks from the sequential blocks that are shown as *Sequential Block*).

Interaction with Other Windows

When you click an object in the *Modular Schematic* window, SpyGlass cross-probes that object in other windows, as described in the following table:

Object Type	Window Actions	Highlight Details
Nets	HDL Viewer has the source file containing the net displayed.	The line where the net is created or inferred is grooved. If the net name is present in the RTL line, the net name is also highlighted.
	Net and all connected components are added to the Incremental Schematic window.	Net is highlighted across the hierarchical boundaries.
	Net is added to the Legend window list.	-
Ports	HDL Viewer has the source file containing the port displayed.	The line where the port is created is grooved and the port name is highlighted.
	Port and all components connected to the port’s connected net are added to the Incremental Schematic window.	Port and its connected net are highlighted across the hierarchical boundaries.
	The net connected to the port is added to the Legend window list.	-
Instances	HDL Viewer has the source file containing the instance displayed.	The line where the instance is created or inferred is grooved. If the instance name is present in the RTL line, the instance name is also highlighted.
	Instance icon is added to the Incremental Schematic window.	Instance is highlighted.
	Instance is added to the Legend window list.	-

Object Type	Window Actions	Highlight Details
Pins	HDL Viewer has the source file containing the pin displayed.	The line where the pin is created or inferred is grooved. If the name of the net connected to the pin is present in the RTL line, the net name is also highlighted.
	Pin and all components connected to the pin’s connected net are added to the Incremental Schematic window.	Pin and its connected net are highlighted across the hierarchical boundaries.
	The net connected to the pin is added to the Legend window list.	-

NOTE: *When you probe an element of a precompiled VHDL library cell in the Modular Schematic window or the Incremental Schematic window, SpyGlass performs a textual search for the element in the corresponding VHDL library cell source file and displays it.*

Using the Cursor

You can use the cursor in different ways to perform different actions in the schematic.

- *Single-Clicking*
- *Double-Clicking*
- *Click-Drag Combinations*
- *Right-Clicking*

Single-Clicking

You can perform the following actions in the *Modular Schematic* window by through single-clicking:

- Single-click an un-probed schematic component to remove all other probes and to start probing for that component. See *Interaction with Other Windows* for more details.

- Repeatedly single-click on an object to traverse all occurrences of the name of that object in a design by using simple text search of an object name.
- <Ctrl>+single-click on an un-probed object to probe it.
- <Ctrl>+single-click on a probed object to deselect its probe without affecting other probes.
- Select multiple objects by using <Ctrl> + single click.

  You can select only up to 32 objects in the *Modular Schematic* window. If you select more than 32 objects, a warning dialog appears prompting you to deselect some of the selected objects.

By default, single-click probe is disabled. Select the *Enable Single-click Probe* option in the *Schematic* page of the *Preferences* dialog.

Double-Clicking

When you double-click on a schematic component in the *Modular Schematic* window, the module, gate, or net appears (drills-down) to the schematic of that component (if available). In addition, the section of code describing the component in the source file is highlighted in the HDL Viewer.

If the component is a module that has its own description file, the source file is opened in the *Source* section and is highlighted in the *File/Design/Constraints/Instances* page.

Click-Drag Combinations

Keeping the mouse button pressed, when you drag the cursor in a particular direction, the modular schematic view changes accordingly.

You can drag the cursor in any of the following directions to see different views:

Northwest-to-Southeast

Dragging in this direction zooms in the selected area in the schematic.

On releasing the mouse button, the zoomed-in area fills the entire schematic view.

Northeast-to-Southwest

Dragging in this direction and releasing the mouse button displays the details of current module in the entire schematic view. This is called zoom to fit.

Southeast-to-Northwest

Dragging in this direction results in going up the hierarchical tree to display the schematic of the parent module.

If the parent module had already been displayed previously, it will inherit the zoom settings of the prior visit. If the parent module had not been previously viewed, it will open in the zoom to fit mode.

Southwest-to-Northeast

Dragging in this direction decreases the magnification of the schematic view.

The farther you drag the cursor, the magnification reduces accordingly.

A short drag in this direction results in the viewing area to zoom out by a small multiplier, while dragging from one corner of the screen to the other will cause the viewing area to zoom out by a large multiplier.

Right-Clicking

Use the right-click menu options in the *Modular Schematic* window to perform the following actions.

Viewing Object Name	*Finding Objects*	*Viewing Object Properties*
Viewing Object Properties	*Loading or Appending to the Incremental Schematic Window*	*Viewing RTL Source Code of an Object*
Viewing Subgroups	*Hiding Subgroups*	*Viewing Case Analysis Settings on an Object*
Viewing SDC and SGDC Constraints Set on an Object	*Viewing Connected Nets*	*Viewing Library Cell Instance Information*

Viewing SpyGlass-Generated Cell Macro Definitions	*Cross-Referencing to Power Data*	*Viewing Debug Data*
Show Activity and Probability Annotation	*Generating Constraints*	*Viewing Liberty Files*

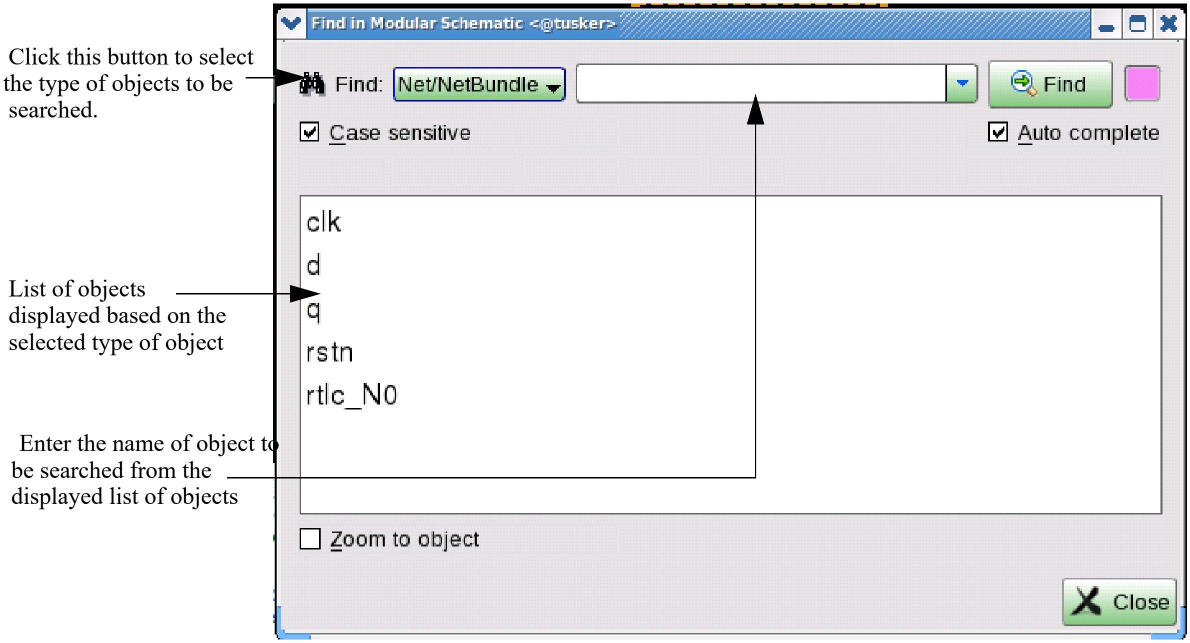
Viewing Object Name

When you right-click on an object in the schematic, the name of that object appears as the first option of the shortcut menu.

Finding Objects

To find an object in the schematic, right-click in the schematic and select the *Find* option from the shortcut menu.

The *Find Dialog* appears, as shown in the following figure:



Click this button to select the type of objects to be searched.

List of objects displayed based on the selected type of object

Enter the name of object to be searched from the displayed list of objects

Find in Modular Schematic <@tusker>

Find: Net/NetBundle

☑ Case sensitive

☑ Auto complete

clk
d
q
rstn
rtlc_N0

□ Zoom to object

FIGURE 68. Modular Schematic Window - Find Dialog

In the above dialog, select the object type to be searched after clicking the button adjacent to Find, as shown in the following figure:

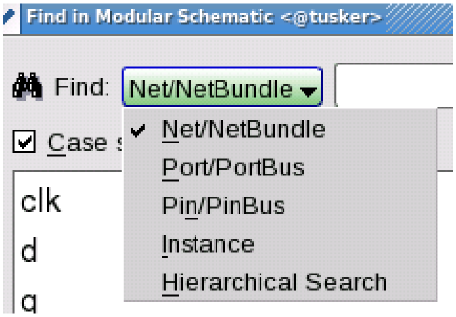


FIGURE 69. Find - Select Object Type

By default, *Net/NetBundle* is selected as the object type.

Once you find the required object in the displayed list in the *Find* dialog, select that object and then click the find icon ( ). When you click this icon, the selected object is:

- Highlighted in the *Modular Schematic* window.

  If you have selected the *Zoom to found object* option in the *Find* dialog, the tool zooms-in the size of the object in the schematic.

- Highlighted in the *HDL Viewer Pane*.

In addition, you can perform the following actions in the *Find* dialog:

Using Automatic Filtering	*Using Advanced Search*
Searching Objects by Their Hierarchical Names	

Using Automatic Filtering

Automatic filtering is the process in which SpyGlass progressively refreshes the displayed list of object names in the *Find* dialog as you enter characters in the textbox in this dialog.

To disable this feature, select the *Disable Auto Filtering* option.

NOTE: *The search is case-insensitive. Therefore, typing* `foo` *will find and show* `foo`*,*

`fOo1`, `FOO`, `Foo23`, *etc.*

Using Advanced Search

In this type of search, you can specify wildcard expressions for searching objects.

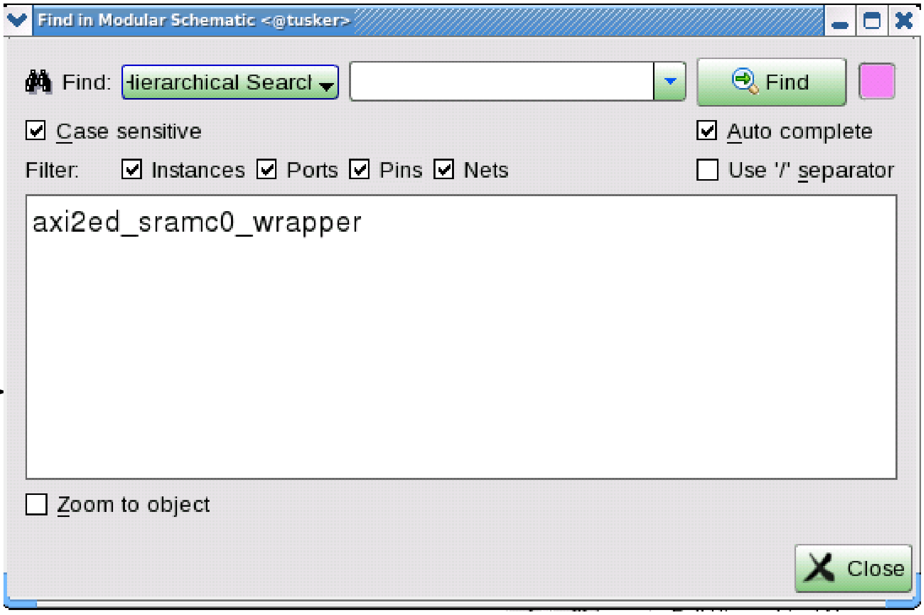
To implement advanced search, perform the following steps:

1. Select the *Use Advanced Search* option in the *Find* dialog.
2. Specify a regular expression containing wildcard characters in the textbox.

Searching Objects by Their Hierarchical Names

To search objects, such as instances, nets, pins, and ports by their hierarchical names, select the *Hierarchical Search* option from the drop-down list.

When you select this option, additional search options appear in the *Find* dialog, as shown in the following figure:



Find in Modular Schematic <@tusker>

Find: Hierarchical Searcl

☑ Case sensitive ☑ Auto complete

Filter: ☑ Instances ☑ Ports ☑ Pins ☑ Nets ☐ Use '/' separator

axi2ed_sramc0_wrapper

☐ Zoom to object

List of top-level design units displayed here, by default

FIGURE 70. Find in Modular Schematic

In the above dialog, you can perform the following actions:

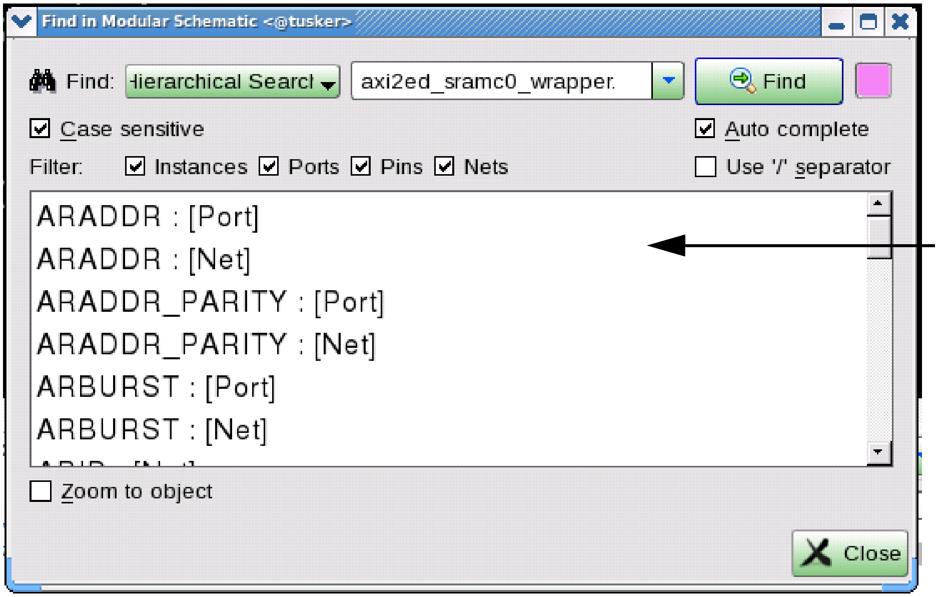
- Specify a search string in the textbox.

  As you start specifying the starting characters of a valid hierarchical name (using the dot hierarchy separator), the list of objects is progressively refreshed in this dialog.

- Dive down the hierarchy of an object displayed after the search.

  To dive down the hierarchy of an object, double-click on the object name in the dialog. This action displays the list of objects within the hierarchy of the object double-clicked.

  In the displayed list, object type (net, net bundle, port, pin, or port bus) appears along with each object name, as shown in the following figure:



Find in Modular Schematic <@tusker>
Find: Hierarchical Search
axi2ed_sramc0_wrapper
Find
☑ Case sensitive
☑ Auto complete
Filter: ☑ Instances ☑ Ports ☑ Pins ☑ Nets
☐ Use '/' separator
☐ Zoom to object
Close

ARADDR : [Port]
ARADDR : [Net]
ARADDR_PARITY : [Port]
ARADDR_PARITY : [Net]
ARBURST : [Port]
ARBURST : [Net]

Object types displayed with each object name

FIGURE 71. Find in Modular Schematic - Results

- Filter the list of objects based on object types, such as instances, ports, pins, nets, or a combination of these types.

  To perform such filtering, select appropriate check boxes, such as *Instances*, *Ports*, *Pins*, and *Nets* in the *Find* dialog.

- Perform case-insensitive search

  You can perform case-insensitive search for finding objects by hierarchical names in the VHDL and mixed-language modes. To implement this search, select the *Use Case-Insensitive search for VHDL* option.

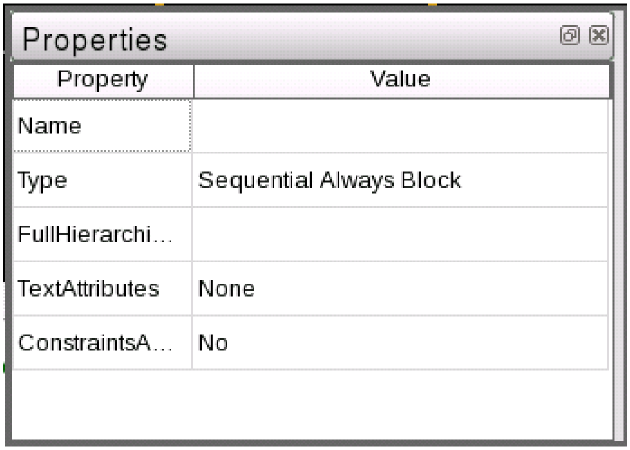
  This option is enabled (only in VHDL and mixed-language modes) when you select the *Hierarchical Instance* search option.

NOTE: *The Use Case-Insensitive search for VHDL option is selected by default in the VHDL/ mixed-language modes. Case insensitive search is performed only inside the VHDL modules. However, this functionality is not available for Verilog modules in the mixed-language mode.*

Viewing Object Properties

To view the basic properties of an object displayed in the schematic, right-click on the object and select *Properties* option from the shortcut menu.

The *Properties* dialog appears, as shown below:



Properties

Property	Value
Name	
Type	Sequential Always Block
FullHierarchi...	
TextAttributes	None
ConstraintsA...	No

FIGURE 72. The Properties Dialog

The above dialog lists object properties, such as object name, object type, complete hierarchical object name, and text attributes attached to the object.

NOTE: *For net bus, port bus, or pin bus, the text attributes are shown in a tabular form, listing only those bits that have some text attributes attached.*

Loading or Appending to the Incremental Schematic Window

You can load or append a port or an instance component appearing in the *Modular Schematic* window to *The Incremental Schematic Window*.

To perform such operation, select a port or instance and select the following options from the shortcut menu:

- *Load to IS*

  Selecting this option clears *The Incremental Schematic Window* and loads the selected component only.

- *Append to IS*

  Selecting this option adds the selected component to *The Incremental Schematic Window*. In this case, existing components remain as it is.

  When a component is added to the *Incremental Schematic* window, only the component icon is shown without connections (for ports) and without pins (for instances). In this case:

  - ❐ Double-click the port component to add the connected net and all its end connections (ports or instances).
  - ❐ Double-click an instance component to add the instance pins.

NOTE: *When more than one instance of different hierarchies is added or cross-probed to the Incremental Schematic window, parent hierarchies of the instances (up to their common ancestor) also get added to the Incremental Schematic. By default, this feature is turned on and can be turned off by clearing the check box for the Maintain Relative Hierarchies for Instances in IS option from the Schematic Page of the Tools > Preferences menu option.*

Viewing RTL Source Code of an Object

To view the RTL source code of an object, right-click the object and select *Cross Probe to RTL* option from the shortcut menu. The source code corresponding to the object is highlighted in *HDL Viewer Pane*.

NOTE: *When you right-click on a signal in the Modular Schematic window, this option may appear as Cross-Reference to RTL/Waveform Viewer (instead of Cross-Reference to RTL). This occurs if the selected violation message has a Waveform Viewer associated with it and that signal also appears in that Waveform Viewer.*

Viewing Subgroups

To expand all subgroups of the selected group, right-click on that group and select the *Expand All Sub Groups* option from the shortcut menu. This displays subgroups of the selected group in the schematic window.

To view or hide the subgroups within the selected group, click the *Enable Groups* button (-▷-) or *Disable Groups* button (▨), respectively, on the schematic toolbar.

Hiding Subgroups

To hide all subgroups of the selected group, right-click on that group and select the *Collapse All Sub Groups* option from the shortcut menu. This hides the subgroups of the selected group in the schematic window.

To view or hide the subgroups within the selected group, click the *Enable Groups* button ( ) or *Disable Groups* button ( ), respectively, on the schematic toolbar.

Viewing Case Analysis Settings on an Object

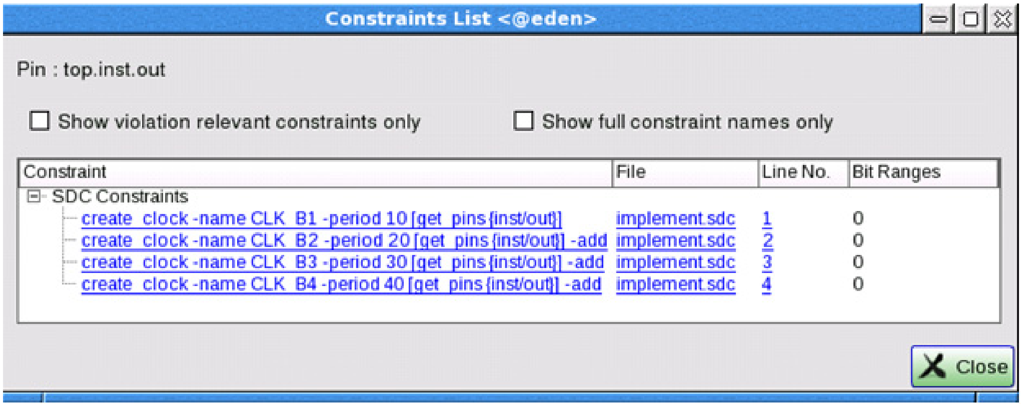
If you have specified case analysis settings on an object by using the `set_case_analysis/testmode` constraints, the object is tagged with a forced value in the schematic.

To view the source of these settings, right-click on the object and select the *Cross Probe Case Analysis* option from the shortcut menu. This displays the SGDC/ SDC file containing the first found case analysis setting in the *HDL Viewer Pane*. In addition, the line containing the constraint is highlighted in this pane.

Viewing SDC and SGDC Constraints Set on an Object

To view SDC/SGDC constraints set on an object, such as port, net, instance, or terminal, right-click on that object and select *Cross Probe to Constraints* option from the shortcut menu.

This displays the *Constraints List* dialog showing constraints (SDC/SGDC) set on the object. The following figure shows the *Constraints List* dialog:



Constraints List <@eden>

Pin : top.inst.out

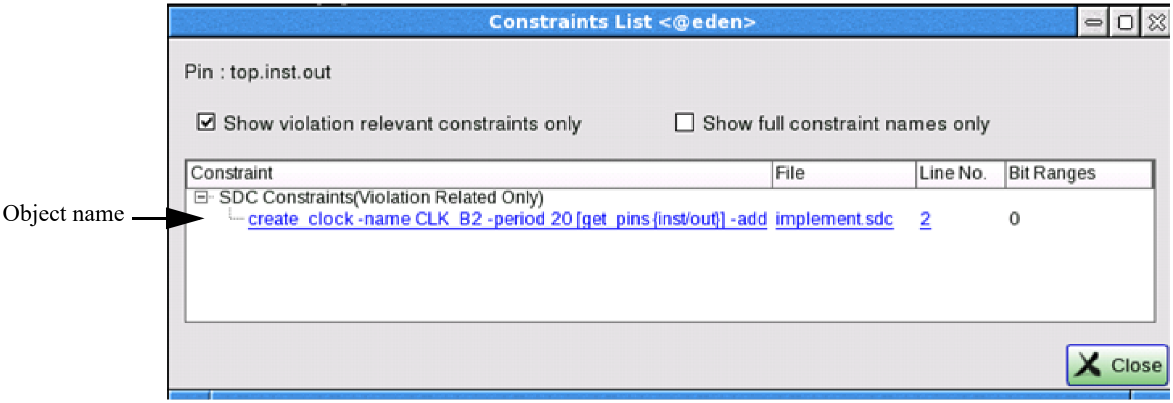
☐ Show violation relevant constraints only ☐ Show full constraint names only

Constraint	File	Line No.	Bit Ranges
⊟ SDC Constraints			
create_clock -name CLK_B1 -period 10 [get_pins {inst/out}]	implement.sdc	1	0
create_clock -name CLK_B2 -period 20 [get_pins {inst/out}] -add	implement.sdc	2	0
create_clock -name CLK_B3 -period 30 [get_pins {inst/out}] -add	implement.sdc	3	0
create_clock -name CLK_B4 -period 40 [get_pins {inst/out}] -add	implement.sdc	4	0

FIGURE 73. Constraints List-Default View

In the above window, the Constraints column specifies the name of the constraint.

You can view the details of the constraints, such as, file name by selecting the **Show only the full constraint names only** option.



Constraints List <@eden>

Pin : top.inst.out

☑ Show violation relevant constraints only   □ Show full constraint names only

Constraint	File	Line No.	Bit Ranges
⊟ SDC Constraints(Violation Related Only)			
create_clock -name CLK_B2 -period 20 [get_pins {inst/out}] -add	implement.sdc	2	0

FIGURE 74. Constraints List

In the above dialog:

- The *Constraint* column specifies the name of the constraint.
- The *File* column specifies the name of the SDC/SGDC file.
- The *Line No.* column specifies the line number in the constraints file where the constraint is defined.
- When you click the name of the object, that object is focused in the *Modular Schematic* window.
- When you click on a row corresponding to a constraint, the corresponding line in the constraints file where the selected constraint is defined is highlighted in *HDL Viewer Pane*.

NOTE: *Please note the following points:*

- *If constraints of a particular type, such as SGDC are not set for an object, they do not appear in the Constraints List dialog.*
- *If no constraints are set on an object, the Cross-Reference to Constraints option appears disabled.*
- *When you right-click a net and select the Cross-Reference to Constraints option from the shortcut menu, you can also see the constraints set on ports of connected nets in addition to the constraints set on the net.*

Viewing Connected Nets

To view nets connected to each bit of a pin bus, port bus, or net bundle, right-click on such an object and select the *List connected nets* option (for pin bus and port bus) or *List Nets* option (for net bundle) from the shortcut menu.

This displays a window listing all nets connected to the pin bus, port bus, or net bundle along with their logic values, if available. For unconnected bits, it is indicated that the bit is hanging.

Viewing Library Cell Instance Information

For instances of SpyGlass-compiled gate library cells, you can view interface and functionality inferred by SpyGlass Library Compiler while compiling gates library.

Right-click on such an instance and select *Cell Information*. A text box appears that lists the cell name, cell type, cell source library (.sglib file), inferred pin interface, and inferred functionality. You can print the displayed information by clicking the *Print* button.

Viewing SpyGlass-Generated Cell Macro Definitions

During synthesis, SpyGlass adds cells from its internal library for inferred objects, such as MUX and select boxes. You can view the corresponding macro definition by right-clicking the macro instance in the Modular Schematic window and then selecting *Show Macro Definition...* from the shortcut menu. A window appears displaying the corresponding macro definition (from the <your-inst-dir>/SPYGLASS_HOME/auxi/target_libs/generic/macro_lib.v* directory), if available.

Cross-Referencing to Power Data

While debugging power-related violations through schematic, you may want to cross-probe to the UPF file in which isolation/level shifter strategies are defined for design elements.

To enable you locate design elements on which such strategies are defined, SpyGlass displays certain symbols adjacent to these elements in the schematic. For example, notice ▶ symbols representing isolation logic in the following schematic:

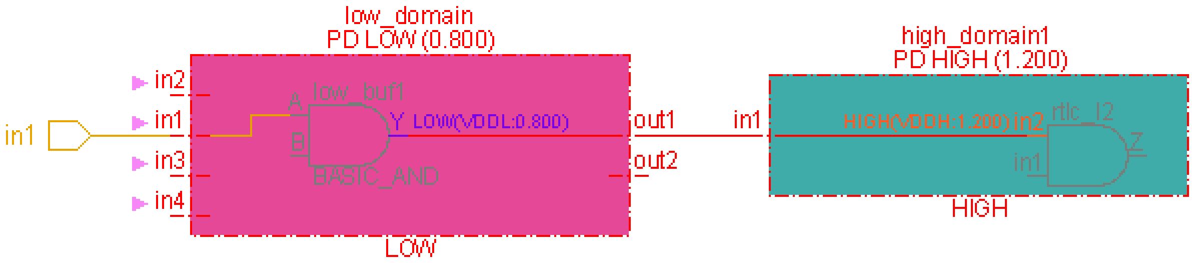


FIGURE 75. Cross-Referencing to Power Data

In the above schematic, the ► symbols indicate that isolation logic is applied on the `in1`, `in2`, `in3`, and `in4` pins.

NOTE: *Different symbols appear to represent a particular type of strategy. For example, ► appears to represent an isolation logic and ʃ appears to represent a level shifter logic. These symbols appear based on the type of the selected rule violation. For example, if both level shifter and isolation logic is applied on an element, and you a select a rule violation that reports information related to only isolation logic, the ► symbol appears adjacent to that element.*

Once you locate such design elements, you can directly cross-probe to the UPF file by right-clicking on an element and selecting the *Cross Probe to Power Data* option from the shortcut menu.

When you select this option, a sub-menu appears displaying the type of strategy, such as *Isolation* and *Level shifter* that is applied on the selected element. On clicking the *Isolation* or *Level shifter* option from the sub-menu, names of strategies appear in another sub-menu.

For example, consider the following figure:

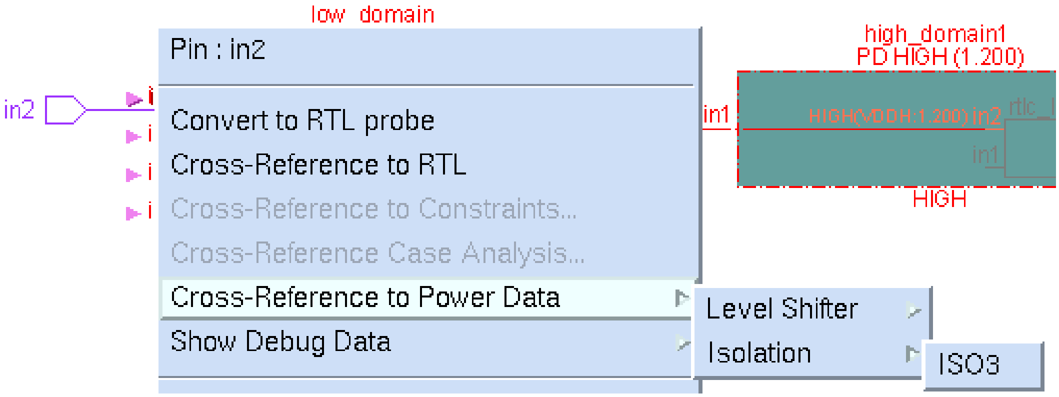


FIGURE 76. Isolation or Level Shifter Options

In the above example, when you click the isolation logic name (`ISO3`) displayed as a sub-menu of the *Isolation* option, SpyGlass cross-probes to the UPF file where this isolation logic is defined.

Viewing Debug Data

To view the debug data, right-click the object and select any of the following options:

- *Show Debug Data-> DFT*: Select this option to view SpyGlass DFT solution debug data.

NOTE: *SpyGlass supports the back annotation feature for SpyGlass DFT solution.*

- *Show Debug Data-> Power Est*: Select this option to view power data.
- *Show Debug Data-> Clock-reset*: Select this option to view SpyGlass CDC solution debug data.

Show Activity and Probability Annotation

Use this option to annotate activity/probability data on the selected instance.

NOTE: *By default, this option is disabled. This option is enabled only when you run a goal of the SpyGlass Power Family solution.*

For more information, refer to the *Viewing Simulation Data in Schematic* section of the *Power Family Rules Reference Guide*.

Generating Constraints

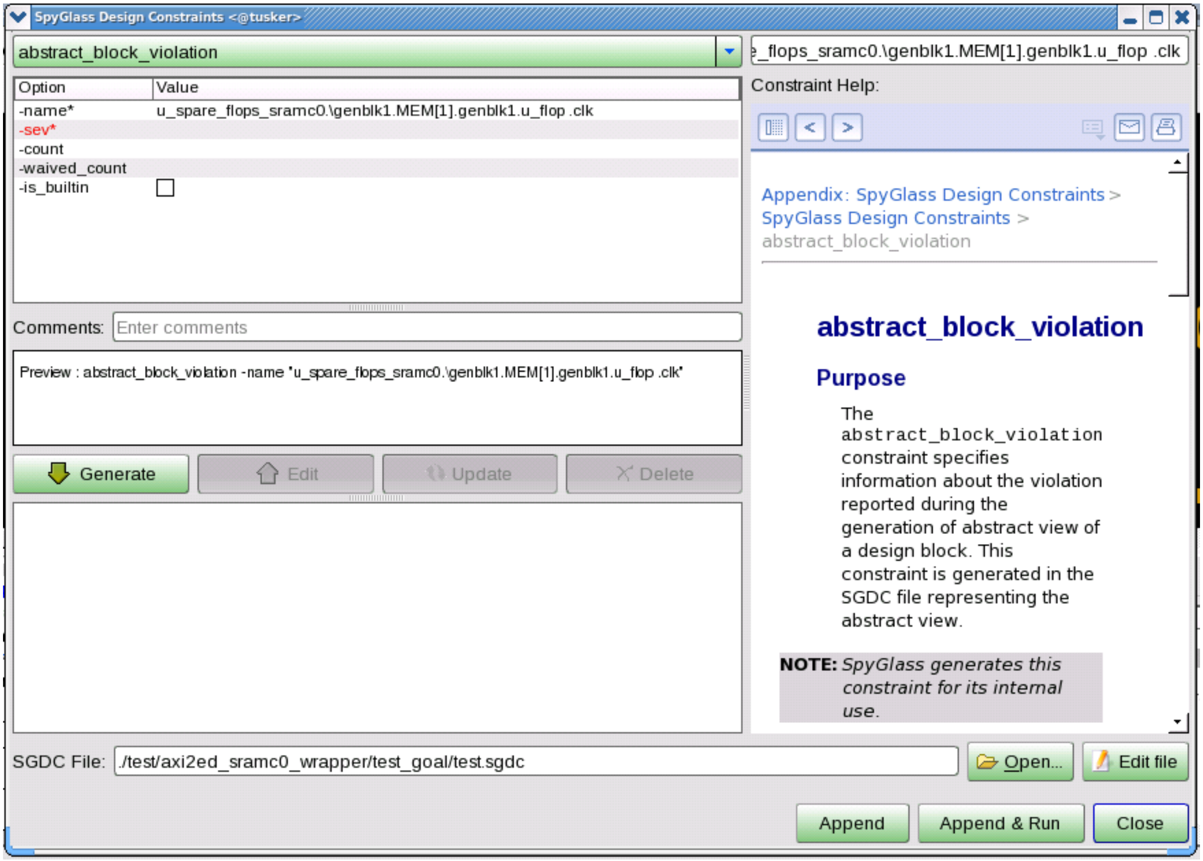
To generate a constraint on an object, such as port, net, and terminal, perform the following steps:

1. Right-click on an object in the schematic, and select the *Set SpyGlass Constraints* option from the shortcut menu.

   When you select this option, the *SpyGlass Design Constraints* window is displayed.

NOTE: *When you **right-click** on an instance of a module, the set SGDC constraint is applied for **all** the instances of a module.*

The following figure displays the *SpyGlass Design Constraints* window:



SpyGlass Design Constraints <@tusker>

abstract_block_violation

Option	Value
-name*	u_spare_flops_sramc0.\genblk1.MEM[1].genblk1.u_flop .clk
-sev*	
-count	
-waived_count	
-is_builtin	□

Comments: Enter comments

Preview : abstract_block_violation -name 'u_spare_flops_sramc0.\genblk1.MEM[1].genblk1.u_flop .clk'

SGDC File: ./test/axi2ed_sramc0_wrapper/test_goal/test.sgdc

Constraint Help:

Appendix: SpyGlass Design Constraints > SpyGlass Design Constraints > abstract_block_violation

abstract_block_violation

Purpose

The abstract_block_violation constraint specifies information about the violation reported during the generation of abstract view of a design block. This constraint is generated in the SGDC file representing the abstract view.

NOTE: *SpyGlass generates this constraint for its internal use.*

FIGURE 77. SpyGlass Design Constraints

2. Select the constraint you want to generate from the drop-down list.

   When you select a constraint, the *Arguments* section displays various fields for different arguments applicable to that constraint. For example, if you select the reset constraint, the *Constraint Arguments* section displays various fields of different arguments of the reset constraint, such as -name, -value, -soft, etc. In addition, the help of that constraint is displayed in the *Help* section.

   You can enter the name of the objects in the from, through, or to fields or right click any field and select the last selected object from the *Select*

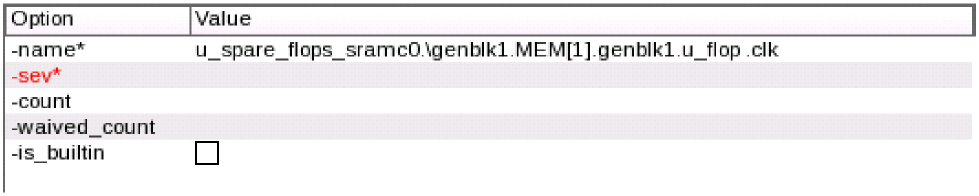
Object from MS/Select Object from IS shortcut menu options.

NOTE: *The* **Set SGDC Constraint** *option is enabled for the instances, except internal instances. When you specify the* `cdc_false_path` *constraint in the* **Select Constraint Type** *field, for such instances, in the* **SGDC Constraint Editor** *window, the name of the master module is populated in the name field.*

NOTE: *You can also add the cdc_false_path constraints for the Ac_unsync01 rule through the spreadsheet viewer.*

3. Specify the argument details in the *Constraint Arguments* section.

   You must specify values for all the mandatory arguments. Such argument names are suffixed with an asterisk (*), as shown in the following figure:



Option	Value
-name*	u_spare_flops_sramc0.\genblk1.MEM[1].genblk1.u_flop .clk
-sev*	
-count	
-waived_count	
-is_builtin	□

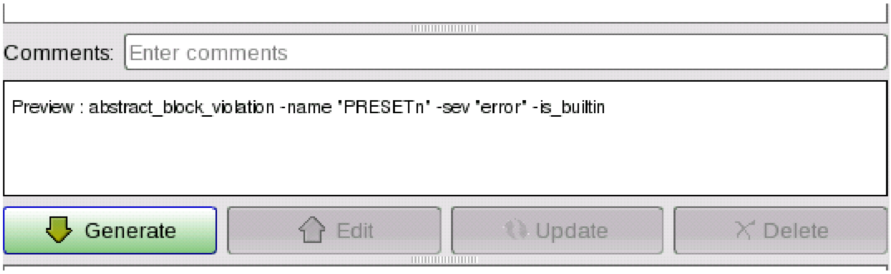
FIGURE 78. Constraint Arguments

In the above case, `-name` is the mandatory argument.

Please note the following points:

- While specifying argument values, you can simultaneously see the preview of the constraint command being generated in the *Preview Constraint* section.

  An example is shown in the following figure:



Comments: Enter comments

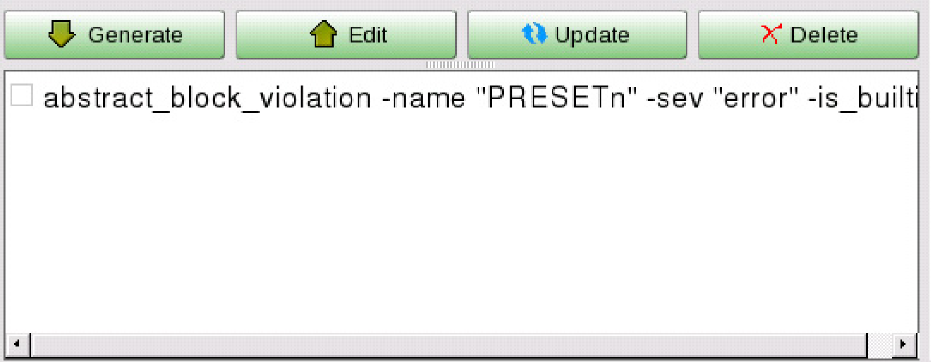
```
Preview : abstract_block_violation -name 'PRESETn' -sev 'error' -is_builtin
```

FIGURE 79. Preview Constraint

- ❒ If you specify an invalid value in the -value text box for a particular constraint, SpyGlass reports an appropriate error message in the *Preview Constraint* section and highlights the text in this section in red.

4. Click the *Generate* button to generate the specified constraint for the selected object in the schematic.

   The generated constraint is displayed in the *Constraints List* section, as shown in the following figure:



□ abstract_block_violation -name "PRESETn" -sev "error" -is_builti

FIGURE 80. Constraints List

NOTE: *If the number of constraints added exceeds the page size, click the (◀ ▶) buttons to go to previous or next pages.*

5. Specify the name of the constraint file in the *Constraint File* textbox. Alternatively, click ⬇ to select the required SGDC file from the drop-down list. If this list does not contain the required file, click 📂 and browse to the directory containing the required file.
6. Click the *Append* button to add the newly generated constraints to the specified SGDC file.
7. Click the *Append & Run* button to apply your settings and run SpyGlass analysis with the new commands.

NOTE: *You need not specify the current design in the SGDC Constraint Editor window. SpyGlass automatically inserts the appropriate current design whenever required.*

8. Click the *Close* button to close the *SGDC Constraint Editor* window.

Viewing Liberty Files

The schematic displays cells defined in .lib or .sglib files specified by using the gateslib or sglib commands, respectively.

To view the definition of such cells, right-click on a cell and select the *Open Liberty File* option from the shortcut menu.

This displays the liberty file (.lib file) in a text editor window showing the definition of the selected cell.

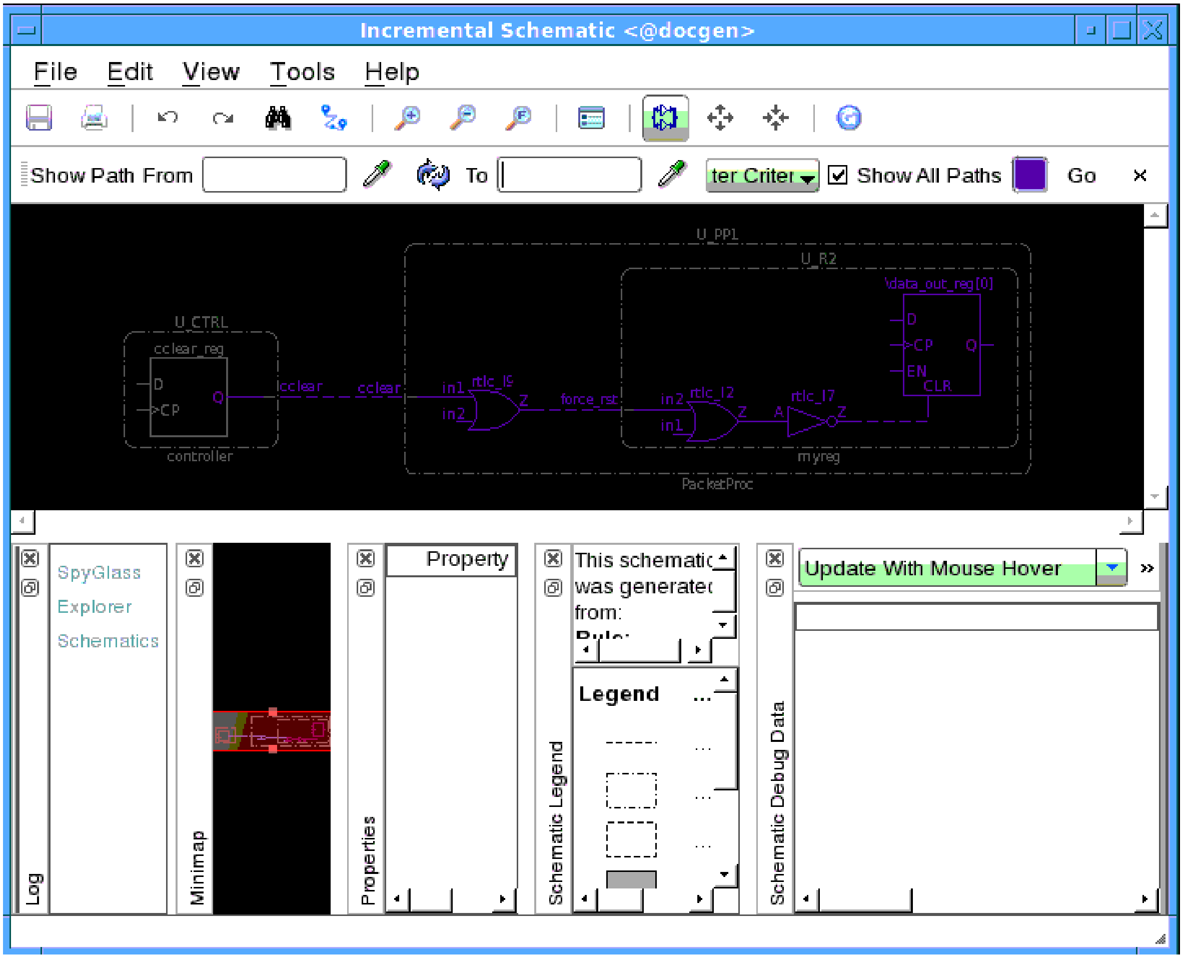
NOTE: *SpyGlass opens the text editor specified by the Text Editor option in the Preferences dialog.*

The text editor always opens the .lib file. So if the selected cell is specified in a .sglib file, SpyGlass opens it's .lib version from which that .sglib is generated.

The Incremental Schematic Window

The Incremental Schematic window displays the selected portions of (flattened) design schematic across hierarchical boundaries.

To display the *Incremental Schematic* window, double-click on the violation message and click in the *Violations* pane.



This schematic was generatec from: Rule:

FIGURE 81. Incremental Schematic Window

The *Incremental Schematic* window is divided into the following sections:

The schematic Display	*Schematic Information*	*Log*
Minimap	*Properties*	*Menu Bar*
Toolbar		

The schematic Display

This section displays all components (across hierarchical boundaries) involved in a selected message or a probed component. You can select more than one message at a time and view all components involved in the selected messages.

Schematic Information

This section displays the meaning of highlighted colors and the text attributes applied on the objects in the schematic window. In addition, if a violation message is selected, it also displays the rule name, short summary of the rule, and the violation message.

NOTE: *The information visible in this section applies to the first selected message, that is, the primary message. It does not apply to secondary messages.*

Log

Display the log for the actions performed in the Incremental Schematic Window.

Minimap

Provides a concise view of the whole schematic. This facilitates in debugging very large designs.

Properties

Hovering over any object or clicking any object in the schematic updates the values in the Properties pane

Menu Bar

Refer to the *Waveform Viewer Window* topic.

Toolbar

Toolbar contains various buttons, such as *Print*, *Find*, *Undo*, *Redo*, *Zoom In*, *Zoom Out*, *Zoom Fit*, and *Preferences*.

This section explains the following topics:

- *Waveform Viewer Window*
- *Interaction with Other Windows*
- *Creating IS Probes*
- *Using the Cursor*
- *Highlighting Path Between Two Reference Points*

The Incremental Schematic Window Menu Bar

The menus available in the Incremental Schematic Window Menu Bar are the same as available in the *The Modular Schematic Window Menu Bar*.

Interaction with Other Windows

Interaction of the *Incremental Schematic* window with other windows is same as it is in the *Modular Schematic* window except that the *HDL Viewer* and the *Modular Schematic* windows are updated when a schematic component of another module is selected.

Creating IS Probes

In the Incremental Schematic window, an IS probe is created when:

- A component is loaded from the Modular Schematic window to the Incremental Schematic window. See *Loading or Appending to the Incremental Schematic Window* for details.
- A component in the Incremental Schematic window is explored further by double-clicking (see *Double-Clicking*) or by tracing input or output cones (see *Tracing Cones*).

When an instance is cross-probed into the Incremental Schematic from any other window, such as from RTL, Instance Browser, Power Browser, or Design Tree, the parent hierarchies of the instances are also added to the Incremental Schematic. By default, this feature is enabled and can be disabled by clearing the check box for the *Maintain Relative Hierarchies for Instances in IS* option from the *Schematic Page* of the *Tools > Preferences* menu option.

NOTE: *SpyGlass does not allow you to cross-probe to the RTL files of encrypted design units. If you try to cross-probe to the RTL of such design units, SpyGlass displays a message in the RTL viewer specifying that the file is encrypted.*

Clearing IS Probes

To clear IS probes, right click anywhere in the *Incremental Schematic* window and select the *Clear All IS Probes* option from the shortcut menu.

Using the Cursor

You can change view in the Incremental Schematic window in the following different ways:

Single-Clicking

See *Single-Clicking* section in the Modular Schematic window.

Double-Clicking

Double-clicking on different objects displays different results in the schematic, as discussed in the following points:

- Double-clicking on a port or pin schematic component

  When you double-click a port or pin schematic component in the Incremental Schematic window, the schematic is expanded to show the connected net and all its end connections (instances or ports).

  When only one component is connected to the IS Probed port or pin, the connected component is displayed with a solid line connection. However, if there are multiple components connected to the IS Probed port or pin, the first found component is displayed with a dotted line connection. Double-click the dotted line connection and the next found component connected to the IS Probed port or pin is displayed. If it is the only other component connected to the IS Probed port or pin, the component is displayed with a solid line connection. Otherwise, the component is displayed with a dotted line connection. Continue double-clicking the dotted line connection until all connected components are added with solid line connections.

- Double-clicking on an instance schematic component

  When you double-click an instance schematic component in the Incremental Schematic window, the schematic is expanded to show the pins of that instance.

- Double-clicking on a pinbus schematic component

  When you double-click a pin bus schematic component in the Incremental Schematic window, the schematic is expanded to show the probed signals and objects connected to these probed signals.

  Set the *Expand all signals of the Pinbus on Double Clicking* option of the *Preferences* dialog to have the schematic expanded to show all connected nets and the objects connected to these connected nets whenever you double-click a pin bus.

Please note the following points:

- Double-clicking works as a toggle. For example, when you double-click an instance that is currently showing only the pins involved in the probe, the schematic is expanded to show the pins of that instance. Now, double-clicking again will bring back the original view. Corresponding messages are displayed in the Log section of the Incremental Schematic window.
- When one or more pins of an instance have case analysis settings and you have selected the *View > Show Case Analysis* menu option, double-clicking on the instance for the first time expands the schematic to show the currently showing pins and those pins with case analysis settings. Double-clicking again expands the schematic to display all pins of the instance and their case analysis settings, if any. Double-clicking the third time returns the schematic to the original display. Corresponding messages are displayed in the Log section of the Incremental Schematic window.

Click-Drag Combinations

Refer to the *Click-Drag Combinations* section in the Modular Schematic window.

Right-Clicking

Use the right-click menu options in the *Incremental Schematic* window to perform the following actions:

Viewing the Object Name

For details on this option, see *Viewing Object Name*.

RTL Probing

RTL probe (or normal probe) is a process in which when you click an object in the schematic, the following actions occur:

- The object is cross-probed to other windows, such as *HDL Viewer Pane*.

However, in some cases, you may need to locate an object first in the *Incremental Schematic* window before creating RTL probe for that object. In such cases, perform the following steps:

1. Create IS probe for pins (by double-clicking the pins) to add the objects connected to the pins in the *Incremental Schematic* window.

   For details, see *Creating IS Probes*.

NOTE: *IS probes do not update other SpyGlass windows.*

2. When the desired object is found in the *Incremental Schematic* window after multiple IS probes, convert the IS probe into an RTL probe by right-clicking the desired object and selecting the *Convert to RTL probe* option from the shortcut menu.

After performing the above steps, the object is cross-probed across other SpyGlass windows and the corresponding entry is added to the Legend window.

NOTE: *Please note the following points:*

- *If you clear all IS probes then only RTL probes are retained in the Incremental Schematic window. See Clearing IS Probes.*
- *When you right-click on a signal in the Incremental Schematic window, this option may appear as Cross-Probe Object (instead of Convert to RTL probe). This occurs if the selected violation message has a Waveform Viewer associated with it and that signal appears in the Waveform Viewer.*
- *SpyGlass does not allow you to cross-probe to the RTL of encrypted design units. If you try to cross-probe to RTL of such design units, SpyGlass displays a message in the RTL viewer specifying that the file is encrypted.*

Viewing the RTL Source Code of an Object

To view the RTL source code of an object, right-click on the object and select the *Cross-Reference to RTL* option from the shortcut menu. The source code corresponding to the object is highlighted in *HDL Viewer Pane*.

NOTE: *When you right-click on a signal in the Incremental Schematic window, this option may appear as Cross-Probe Object (instead of Cross-Reference to RTL). This occurs if the selected violation message has a Waveform Viewer associated with it and that signal appears in that Waveform Viewer.*

Viewing Case Analysis Settings on an Object

For details on this option, see *Viewing Case Analysis Settings on an Object*.

Viewing SDC/SGDC Constraints Set on an Object

For details on this option, see *Viewing SDC and SGDC Constraints Set on an Object*.

Cross-Referencing to Power Data

For details on this option, see *Cross-Referencing to Power Data*.

Viewing Debug Data

To view debug data, right-click on an object and select any of the following options based on your requirement:

- *Show Debug Data-> DFT* option to view SpyGlass DFT solution debug data.

NOTE: *SpyGlass supports the back annotation feature for SpyGlass DFT solution.*

- *Show Debug Data-> Power Est* option to view power data
- *Show Debug Data-> Clock-reset* option to view SpyGlass CDC solution data

Viewing Object Properties

For details on this option, see *Viewing Object Properties*.

Show Activity and Probability Annotation

Use this option to annotate activity/probability data on the selected instance.

NOTE: *By default, this option is disabled. This option is enabled only when you run a goal of the SpyGlass Power Family solution.*

For more information, refer to the *Viewing Simulation Data in Schematic* section of the *Power Family Rules Reference Guide*.

Viewing/Tracing Connected Nets

To view/trace the nets connected to each bit of a pin bus, net bundle, or port bus, right-click on the net and select the *List/Trace Connected Nets* option from the shortcut menu.

A dialog appears listing all nets connected to the bits of a pin bus, net bundle, and port bus. For unconnected bits, it is indicated that the bit is hanging.

The *List/Trace Connected Nets* dialog automatically appears if there are no probed signals and you double-click a pin bus, net bundle, or port bus.

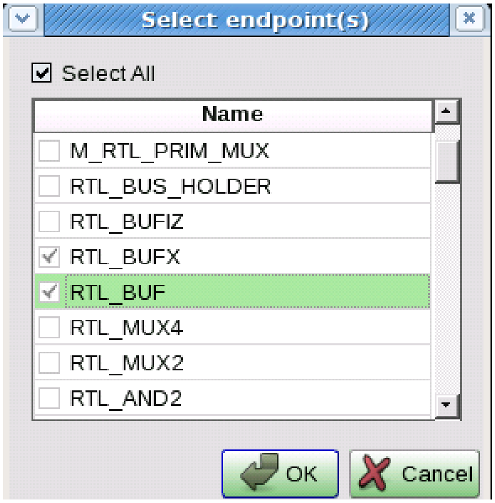
You can also set the *Show List/Trace Connected Nets Dialog on Double Clicking* option in the *Schematic Page* of the *Tools > Preferences* menu option to have the *List/Trace Connected Nets* dialog appear whenever you double-click a pin bus, net bundle, or port bus.

Tracing Cones

You can trace the input cone (for input or inout pins) or output cone (for output or inout pins) of any pin in the schematic portion displayed in the *Incremental Schematic* window.

To trace the cone, right-click on a pin and select any of the options from the shortcut menu:

- *Show Input Cone > To Primary Inputs* to view all components in the fan-in cone of the selected pin up to primary inputs.
- *Show Input Cone > To Flops* to view all components in the fan-in cone of the selected pin up to flip-flops.
- *Show Input Cone > To Flops/Latches* to view all components in the fan-in cone of the selected pin up to flip-flops and latches.
- *Show Input Cone > Define End Point...* to set the end points (instances) up to which you want to view the fan-in components. Then, the *Define End Point* dialog appears listing all the master modules instantiated in the design:



☑ Select All

Name
□ M_RTL_PRIM_MUX
□ RTL_BUS_HOLDER
□ RTL_BUFIZ
☑ RTL_BUFX
☑ RTL_BUF
□ RTL_MUX4
□ RTL_MUX2
□ RTL_AND2

FIGURE 82. Define Endpoint Dialog

Select the master module(s) and then click *Show Cone*. The *Incremental Schematic* window is updated to show all components in the fan-in of the selected pin up the instances of the selected master module(s). You can search for an instance by entering its name in the *Find Text* field and clicking *Search*. As you type the starting letters of a valid module name, the results are updated accordingly.

- Select *Show Input Cone > To Module Boundary* to display the input cone till the boundary of the current design unit in which the cone is being seen.

The procedure for tracing output cones for output or inout pins is same using the *Show Output Cone* option in the shortcut menu.

You can undo the last Show Cone actions using the *Undo Show Cone* option in the shortcut menu.

Highlighting Path for the Selected Objects

Perform the following actions to highlight the path between the selected objects:

1. Select and right-click on an object in the *Incremental Schematic.*
2. Click Show Path from the right-click menu.

A sub-menu is displayed as shown in the following figure:

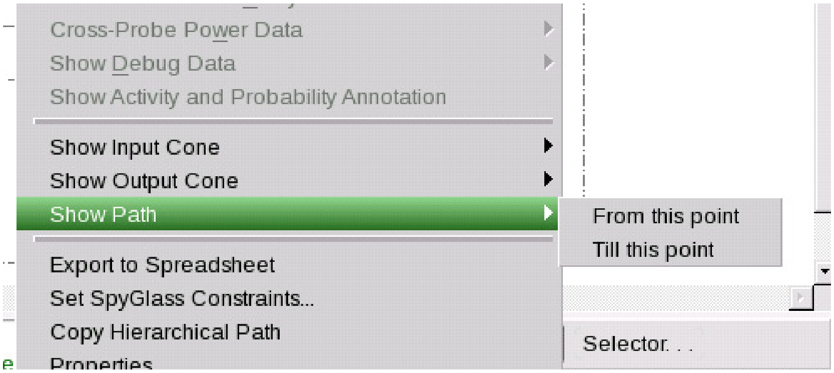


FIGURE 83. Show Path

3. Click **From this point** to make the selected object as the source object.
4. Click **Till this point** to make the selected object as the destination object.
5. Click **Go** on the **Show Path** toolbar to highlight the path for the selected objects.

NOTE: *You specify hierarchical names of instance/pin or port as an input to the **From** and **To** fields. However, you can not specify net names.*

For more information on highlighting the paths for the selected objects, see *Highlighting Path Between Two Reference Points*.

Setting SGDC Constraints

For details on this option, see *Generating Constraints*.

Searching the Incremental Schematic Window

For details on this option, see *Finding Objects*.

Viewing Library Cell Instance Information

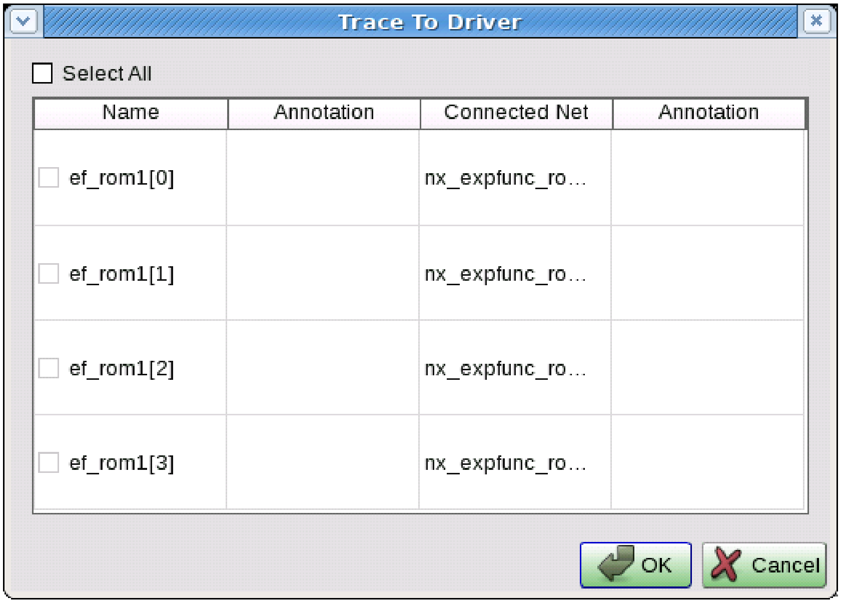
For details on this option, see *Viewing Library Cell Instance Information*.

Tracing the Driver of an Input Pin

In large designs, the driver of an input pin can be present quite far from the pin. In such cases, to view the driver of such input pins, right-click on the required pin and select the *Trace to Driver* option from the shortcut menu.

When you select this option, the pin driver is shown at the centre of the schematic.

In case if you select a pin that is a bus, a dialog appears in which all signals associated with that pin are shown. The following figure shows such dialog:



Trace To Driver

□ Select All

Name	Annotation	Connected Net	Annotation
□ ef_rom1[0]		nx_expfunc_ro...	
□ ef_rom1[1]		nx_expfunc_ro...	
□ ef_rom1[2]		nx_expfunc_ro...	
□ ef_rom1[3]		nx_expfunc_ro...	

FIGURE 84. Trace to Driver Option

In the above dialog, select the required pin to bring its driver into view.

If you select multiple pins, driver of the first selected pin is shown in the schematic.

If a pin has multiple drivers, the first driver appears in the schematic.

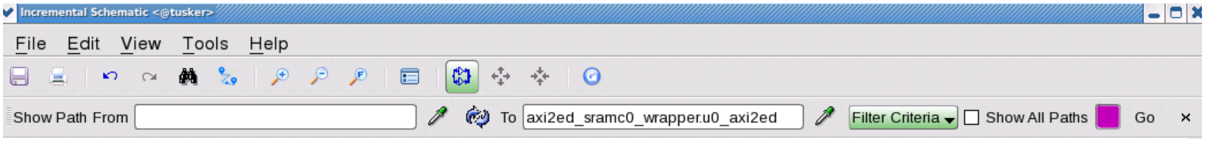
Viewing Liberty Files

For details on this option, see *Viewing Liberty Files*.

Highlighting Path Between Two Reference Points

To highlight the path between two reference points, for debugging, in the Incremental Schematic, click the **Path** button ( )in the Incremental Schematic toolbar.

Clicking the Path button displays the expanded Show Path toolbar as shown in *Figure 85*:



Show Path From [ ] To axi2ed_sramc0_wrapper.u0_axi2ed Filter Criteria ☐ Show All Paths Go ×

FIGURE 85. Show Path Toolbar

The Show Path toolbar has two editable fields, namely, **From** and **To**, to input the hierarchical names of the objects. You can either type in the object name or use the dropper button to pick the name of the object from Incremental Schematic.

NOTE: *You specify hierarchical names of instance/pin or port as an input to the **From** and **To** fields. However, you can not specify net names.*

However, if no path is found for the selected objects, an error message is displayed.

Also, if wrong hierarchial name is provided for the object, an error message is displayed.

This feature is violation independent. That is, the path for the specified objects is searched throughout the synthesized design, independent of whether the object is part of the violation or not.

You can perform the following actions from the Show Path toolbar:

- *Specify the Filter Criteria*
- *Display One or All Paths*

- *Change Color of the Displayed Path*

Specify the Filter Criteria

You can select one or more of the following options from the **Filter Criteria** drop-down list to neglect the paths containing the specified option from being displayed:

- Flop
- Latch
- BlackBox

Therefore, if you have selected Flop in the Filter Criteria drop-down list, then any path that encounters a flop will not be explored further and will not be displayed. Same behavior is applicable to other options in the Filter Criteria drop-down list.

Alternatively, to highlight the path between the selected objects using the right-click menu, see *Highlighting Path for the Selected Objects*.

Display One or All Paths

You can display single or all paths available in the design for the selected objects using the **Max Paths** drop-down list.

Select **One** from the drop-down list to display only one path for the selected objects.

Select **All** from the drop-down list to display all the paths available for the selected objects.

NOTE: *Selecting All to search for all the paths can be time-consuming for larger designs.*

Change Color of the Displayed Path

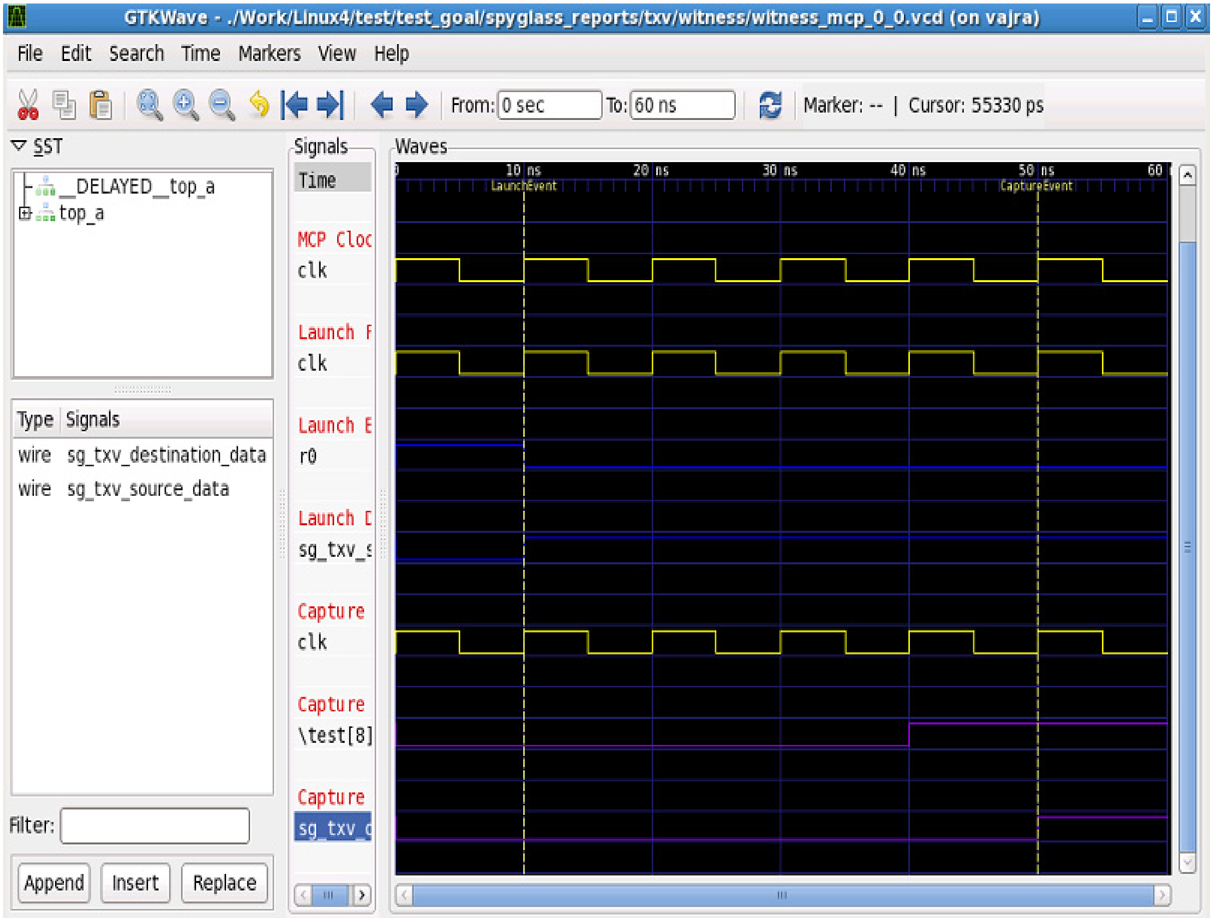
Click the Color button adjacent to the Max Paths drown-down list to display the **Show Color** dialog box. You can select any color from the color palette to highlight the selected path with that color.

Waveform Viewer Window

To debug complex bugs in your design, viewing message, RTL, schematic, and FSM may not be sufficient to find the exact cause of bugs. For such cases, you can use the Waveform Viewer window.

Waveform Viewer is an analysis tool that enables you to find the root cause of a functional bug in the design. This tool illustrates a sequence of events leading to the functional problem. These events are illustrated from an initial state of a design/sub-design to the time when the bug appears. These events can be generated as a set of simulation vectors in the VCD format. Each event or time frame in VCD corresponds to an edge of a clock relevant to the violation. The waveform viewer launched for a failure contains this VCD content.

The presence of a waveform is indicated by the waveform icon, ∿, in front of the violation message. To view the waveform for the violation, click the ∿ button in the toolbar. This displays the *Waveform Viewer* window, as shown in the following figure:



GTKWave - ./Work/Linux4/test/test_goal/spyglass_reports/txv/witness/witness_mcp_0_0.vcd (on vajra)

From: 0 sec To: 60 ns Marker: -- | Cursor: 55330 ps

Type	Signals
wire	sg_txv_destination_data
wire	sg_txv_source_data

FIGURE 86. Waveform Viewer Window

Initially, SpyGlass loads a small set of signals in the waveform viewer. These signals are believed to be a good start for the debugging process.

As the debugging progresses, you can gradually load more signals in any of the following ways:

- By using cross-probing capabilities

  By looking at loaded signals, RTL code, and schematic, if you can identify relevant signals, values of which may unveil the cause of a bug, click on such signals under RTL/schematic to load the waveform of the signal and analyze the transitions.

For details, refer to the *Cross-Probing in Waveform Viewer* topic.

- By selecting the fan-in option from the right-click menu displayed for the selected signal in the waveform viewer

  This action loads signals in the fan-in cone of the selected signal for which a waveform is available. This is limited to one signal in any path in the fan-in cone (the first signal visited in the path for which a waveform is available). By default, a waveform is only generated for registers and primary inputs in the cone of influence of a violation. You can use the watch point constraint to generate waveform for intermediate signals.

- By selecting the fan-in cone option from the right-click menu displayed for the selected signal in the waveform viewer

  This action reports total number of signals in the cone headed by selected signal as well as all the signals names. Signals for which waveforms are available are highlighted; by clicking on these signals, you can load the corresponding waveform.

Following are main features and capabilities provided by SpyGlass for easy debugging of a functional violation:

- Signal hiding capability

  While loading signals during debug process, there may be too many signals loaded which can make the search and tracing difficult. You can elect signals such signals that are not required, and hide them. You can later unhide them as and when required.

  For details, refer to the *Signal Hiding Capability in Waveform Viewer* topic.

- Signals ordering

  The signals appear in topological ordering by default. You can reorder signals in alphabetical order and revert using provided buttons added for this purpose.

- Moving signals around

  You can select a signal(s) and move it to a different location in the waveform viewer. To do this, select the signal(s) that you want to move and place the cursor at the signal boundary. Next, drag the signal(s) to the required position. The automatic ordering feature can be used to restore the original ordering or alphabetical ordering.

- An internal signal, autoverify_state[0:31], is always generated that changes value each time the design transitions from one state to another state (at least one register changes value). When large numbers of signals are loaded into the waveform viewer, this signal can be used to track any changes caused by a previous transition by adding a vertical marker on the next autoverify_state transition and scrolling through all signals in the waveform viewer.

- By default, single selection mode is used in waveform viewer. However, you can select/deselect multiple signals by using Ctrl+Click.

Given all the above capabilities and user knowledge of the design, the root cause of a bug can be identified. However, if more analysis is required, the generated VCD file can be analyzed under your preferred simulation environment and more simulation can be performed to analyze further the context of the failure. Alternatively, you may add new assertions and analyze them to understand further the cause of a given failure.

While techniques to identify the cause of a violation are the same, no matter if that violation is due to a RTL bug, erroneous assertion, or missing constraints, you may focus on identifying the category of the violation to find a quick answer to this question. For this purpose, instead of analyzing the failure through tracing signals in an intelligent way (selecting signals believed relevant from RTL/ schematic) or in topological order, you may focus on some set of signals that are prune to cause constraint-related violations:

- Reset: A reset not recognized in a design will participate in falsifying an assertion. By a quick look at the reset signal waveform, you may determine that a reset is making an assertion fail. To fix this problem provide the reset as a constraint in the SpyGlass Auto Verify Design Constraints file.
- Clock: A wrong clock waveform can cause a violation; furthermore, wrong relative clocks frequencies in multi-clock system can cause a violation. By examining the clocks waveforms, you may be able to find such problems. To fix clock definition related problem you can provide the correct definition of the clock in the constraint file.
- Other special ports: Ports such as test mode should be properly constrained. By looking into the waveform signals, the failure may be attributed to such signals.
- Initial state: A wrong initial state may cause a false violation. Specifically, if registers are left free (not assigned to neither a “0” nor “1”), SpyGlass Auto Verify solution will assign them to appropriate values to make assertions fail. By looking into registers file report “.reg”

file and/or the waveform one can quickly determine if a violation is caused by such un-initialized registers.

Cross-Probing in Waveform Viewer

SpyGlass provides the following cross-probing capabilities for Waveform Viewer:

- RTL to waveform probing

  Clicking on a signal in the RTL code results in loading (if it is not already loaded) and highlight of the corresponding waveform, if a waveform for the signal is available. The waveform for the given signal is placed in its position according to a predefined ordering (alphabetical or topological). This action is performed if a waveform window is already open. Note that the same action (right-click on RTL signal) also highlights the corresponding schematic wire if a schematic window is already open.

- Schematic to waveform probing

  Selecting the *Cross-Probe Objec*t option from the right-click menu for any signal in the schematic window loads and highlights the corresponding waveform. This is similar to RTL to waveform probing.

- Waveform to RTL and schematic probing

  Selecting Cross-*probe to Schematic/RTL* option for any signal in the waveform viewer, results in corresponding signals highlight in the RTL code and schematic viewer.

Signal Hiding Capability in Waveform Viewer

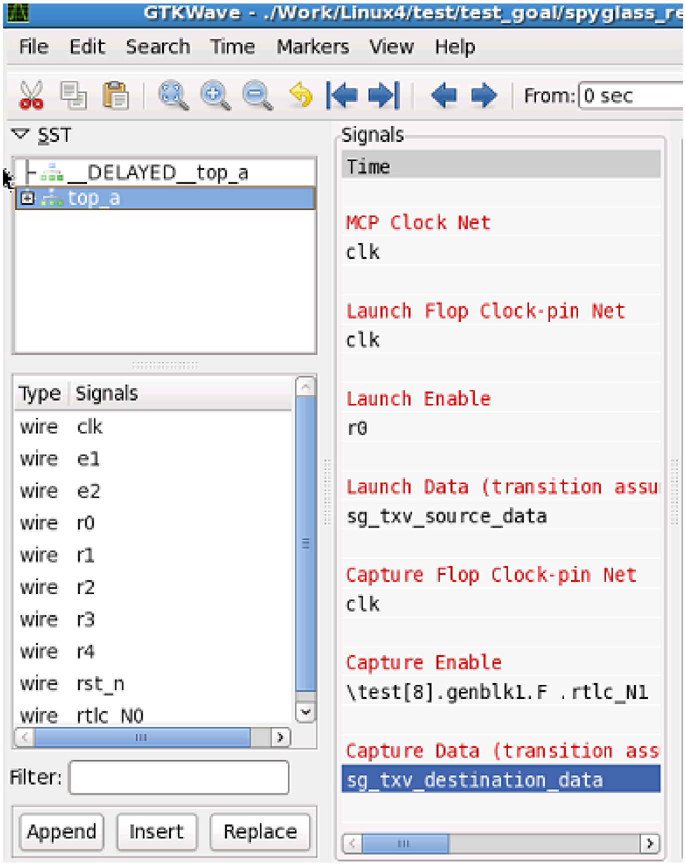
SpyGlass enables you to determine the signals that you want to hide/ unhide. You can chose to *Show a Signal* or Hide a Signal.

Show a Signal

To show a signal, perform the following steps from the GTKWave widow

1. Navigate the hierarchy in the SST pane.
2. Select the preferred signal from the list of available signals.

3. Select one of the following options:
   - Click the **Append** button to append the signal at the end of the visible signal list
   - Click the **Insert** button to insert the signal in the selected location in visible signal list
   - Click the **Replace** button to replace the selected signal in visible signal list



Type	Signals
wire	clk
wire	e1
wire	e2
wire	r0
wire	r1
wire	r2
wire	r3
wire	r4
wire	rst_n
wire	rtlc_N0

```
Signals
Time

MCP Clock Net
clk

Launch Flop Clock-pin Net
clk

Launch Enable
r0

Launch Data (transition assu
sg_txv_source_data

Capture Flop Clock-pin Net
clk

Capture Enable
\test[8].genblk1.F .rtlc_N1

Capture Data (transition ass
sg_txv_destination_data
```

FIGURE 87. Show or Hide Signals

You can also use drag and drop signals to move the signals to the visible signal list.

Hide a Signal

To hide a signal, perform the following steps from the GTKWave widow

1. Select and right-click on the signal in the signal pane
2. Click Cut to delete the signal
3. Alternatively, use <Crtl+X> keyboard shortcut

Right-Click Menu Options of the Waveform Viewer Window

When you right-click on a signal in the *Incremental Schematic* window, this option may appear as Cross-Probe object (instead of Convert to RTL probe). This occurs if the selected violation message has a Waveform Viewer associated with it and that signal appears in the Waveform Viewer.

When you right-click on any signal in the *Waveform Viewer* window, SpyGlass displays a shortcut menu. Various options of this shortcut menu are described in the following table:

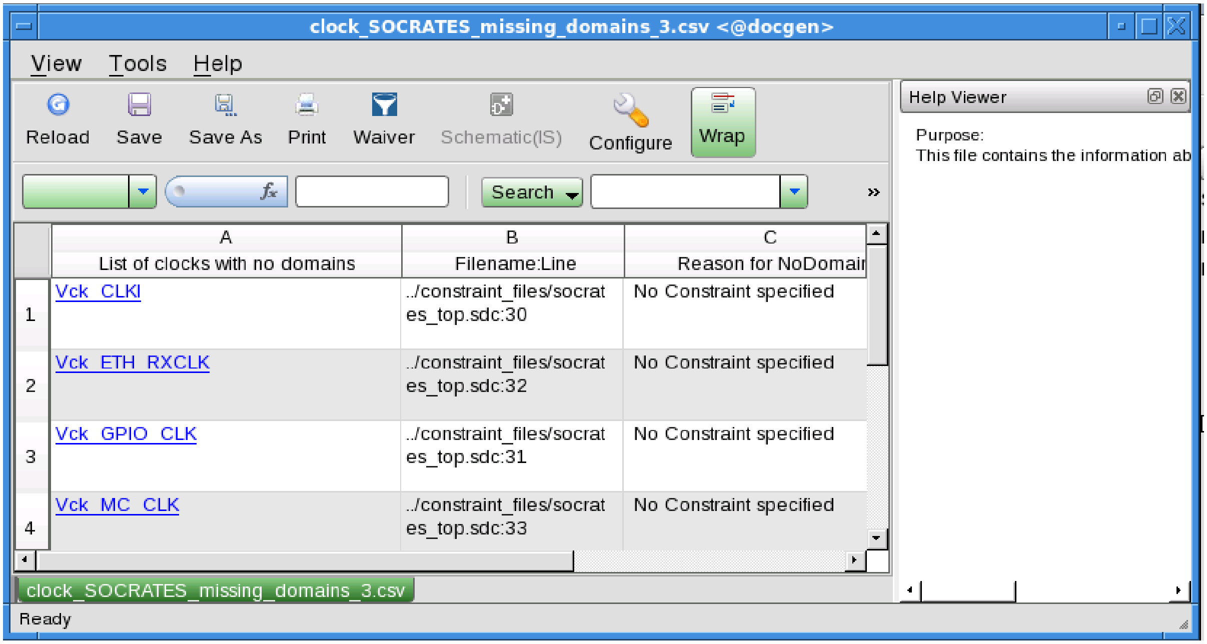
Option	Description
Fanin	Loads the selected signal in the fan-in cone of the selected signal for which a waveform is available. The fan-in signals for the selected signal are highlighted in the Schematic Viewer/RTL viewer.
Fanin Cone	Displays a complete set of signals in the fan-in cone
Fanout	Loads the selected signal in the fan-out cone of the selected signal for which a waveform is available. The fan-out signals for the selected signal are highlighted in the Schematic Viewer/RTL viewer.
Fanout Cone	Displays a complete set of signals in the fan-out cone
Cross Probe to RTL/Schematic	Highlights the signal in the Schematic Viewer pane and the HDL Viewer pane.

Option	Description
Transition Cause	
Help	Displays the short help for the signal in a popup window.

Spreadsheet Viewer Window

The *Spreadsheet Viewer* window contains all the violation messages reported by a particular rule in a tabular format. The header of such rules is prefixed with the spreadsheet icon (▤).

To open a spreadsheet, select the rule header (under which all the violation messages of that rule are listed) and select the *Spreadsheet* link in the *Results* pane. This displays the *Spreadsheet Viewer* window, as shown in the following figure:



	A	B	C
	List of clocks with no domains	Filename:Line	Reason for NoDomair
1	Vck_CLKI	../constraint_files/socrates_top.sdc:30	No Constraint specified
2	Vck_ETH_RXCLK	../constraint_files/socrates_top.sdc:32	No Constraint specified
3	Vck_GPIO_CLK	../constraint_files/socrates_top.sdc:31	No Constraint specified
4	Vck_MC_CLK	../constraint_files/socrates_top.sdc:33	No Constraint specified

Purpose:
This file contains the information ab

FIGURE 88. Spreadsheet Viewer Window

The above window displays the total number of violation messages in the status bar.

You can cross-probe from the *Spreadsheet Viewer* window to other windows/panes, such as *Schematic Window*, *HDL Viewer*, and File/Design Constraints/Instance views.

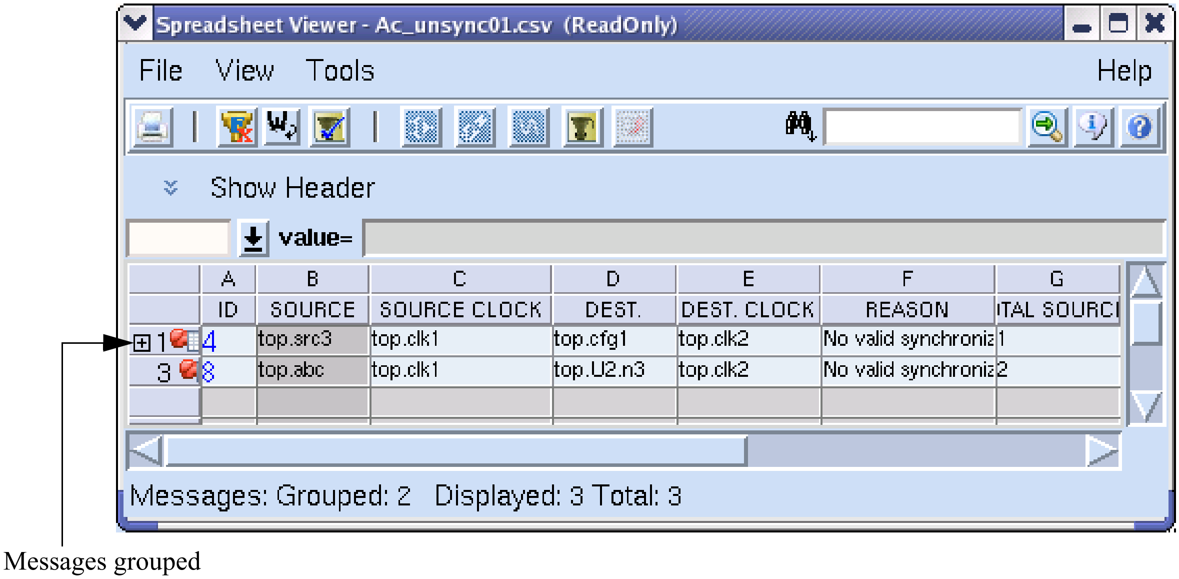
To enable cross-probing from the *Spreadsheet Viewer* window, SpyGlass provides hexadecimal violation *ID*, which corresponds to the hexadecimal violation *ID* generated for individual violation messages.

You can select multiple rows in the above spreadsheet by using Ctrl + click, and view a cumulative schematic showing all violations reported in the selected rows. These violations are shown in the Complete Display mode only.

NOTE: *If you de-select a primary violation in the spreadsheet, all its corresponding secondary violations are also de-selected.*

Viewing Grouped Messages in a Spreadsheet

Grouped messages in a spreadsheet appear as shown in the following figure:

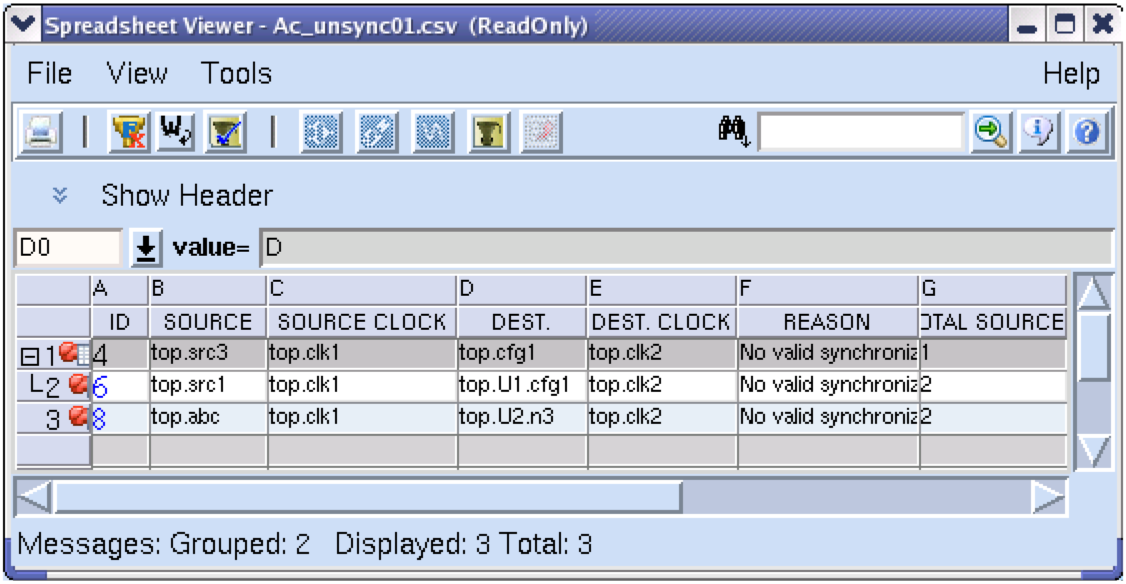


	A	B	C	D	E	F	G
	ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	REASON	ITAL SOURCI
⊞ 1	4	top.src3	top.clk1	top.cfg1	top.clk2	No valid synchroniz	1
3	8	top.abc	top.clk1	top.U2.n3	top.clk2	No valid synchroniz	2

Messages: Grouped: 2 Displayed: 3 Total: 3

FIGURE 89. Viewing Grouped Messages - Collapsed View

In the above figure, click the + sign to view all grouped messages, as shown in the following figure:



	A	B	C	D	E	F	G
	ID	SOURCE	SOURCE CLOCK	DEST.	DEST. CLOCK	REASON	DTAL SOURCE
⊟1	4	top.src3	top.clk1	top.cfg1	top.clk2	No valid synchroniz	1
L2	6	top.src1	top.clk1	top.U1.cfg1	top.clk2	No valid synchroniz	2
3	8	top.abc	top.clk1	top.U2.n3	top.clk2	No valid synchroniz	2

Messages: Grouped: 2 Displayed: 3 Total: 3

FIGURE 90. Viewing Grouped Messages - Expanded View

Waiving Rows in a Spreadsheet

The *Spreadsheet Viewer* in SpyGlass contains all the violation messages and related information for a particular rule in a tabular format. Sometimes, voluminous information is reported for a rule. The Waiver feature allows you to view the relevant messages and remove the irrelevant rows from the spreadsheet.

For waived messages, the value in the **Waived column** is **Yes**.

Selecting the **Show Waived rows** option in the **View** menu, displays all the violation messages for a rule, including the Waived messages. To hide the waived messages, de-select the **Show Waived rows** option.

This section explains the following topics related to the waiver feature of the Spreadsheet Viewer:

- *Specifying Product-Specific and User-Specific Waivers*

- *Applying the Waiver Command*
- *Example of Applying the Waiver Command*
- *Editing the Waiver Commands Using Waiver Editor*

Specifying Product-Specific and User-Specific Waivers

You can waive rows in a spreadsheet to remove the irrelevant information by specifying following types of waivers:

- **Product-specific waivers**: Product-specific waivers are used to specify initial set of waiver commands. These waivers are specified in the product-defined waiver file. Commands in the product-defined waiver file gets overwritten on re-run.

  Syntax

  The syntax for defining a product-specific waiver file is as follows:

  ```
  spreadsheet_waiver_info/<CSV_NAME>_waiverinfo_product
  ```

  User-specific waivers: You can specify user-specific waiver commands for any CSV file to remove the specific information from being displayed. Right-click a row in the spreadsheet and select **Waive Row(s)** option to waive a row from the spreadsheet. This waiver command gets added in the user-specified waiver file. The user-specified waiver file does not get overwritten on re-run. Instead, new waiver commands get appended to existing commands in the file.

  Syntax

  The syntax for defining a user-specific waiver file is as follows:

  ```
  spreadsheet_waiver_info/<CSV_NAME>_waiverinfo_user
  ```

Applying the Waiver Command

The *sg_waive* command is used to waive a row from the spreadsheet. The syntax of the sg_waive command is as follows:

```
sg_waive -col {<column name/column_number>} -criteria
{<waiving_criteria>} -value {<value_for_criteria>}
```

Arguments

The *sg_waive* command takes the following arguments:

- **<column name/column number>**: name of the column or column number in the spreadsheet

- **<waiving_criteria>**: Waiving criteria for pattern matching based on which the row would be removed. See valid waiving criterion for a list of valid expressions that you can use as an input to this argument.

  Following table lists the valid waiver criteria that you can specify for the -criteria argument:

TABLE 7 Valid Criterion

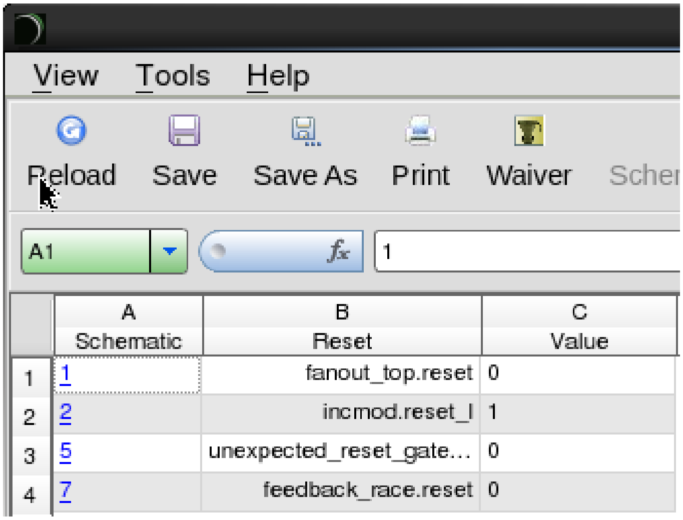
Syntax	**Description**
contains	Contains
! contains	Does not contain
==	Equals
!=	Not equal to
=~	Regular expression
begins	Begins with
!begins	Does not begin with
ends	Ends with
!ends	Does not end with

- **<value_for_criteria>**: Value to match with the value in the spreadsheet. Apart from strings and integers, you can also specify regular expression for pattern matching.

NOTE: *Do not include the columns, which contain the dynamic information, for generating the waiver commands. For example, Schematic ID,* in the *Example of Applying the Waiver Command* section described below, can change in each run. Therefore, it is not recommended to specify Schematic ID in a waiver command.

Example of Applying the Waiver Command

Consider following CSV file, which lists the asynchronous resets in the design:



	A Schematic	B Reset	C Value
1	1	fanout_top.reset	0
2	2	incmod.reset_l	1
3	5	unexpected_reset_gate...	0
4	7	feedback_race.reset	0

FIGURE 91. Example: Waiving Rows

Consider the following *sg_waive* command:

```
sg_waive -col {RESET} -criteria {Equals} -value
{incmod.reset_l} -col {VALUE} -criteria {Equals} -value {1}
```

The above command waives 2nd row from the CSV file.

To waive multiple messages, you can also use regular expression as shown in the following example:

```
sg_waive -col {RESET} -criteria {Contains} -value {*reset} -
col {VALUE} -criteria {==} -value {0}
```

The above command waives 1st, 2nd, and 3rd rows from the CSV file.

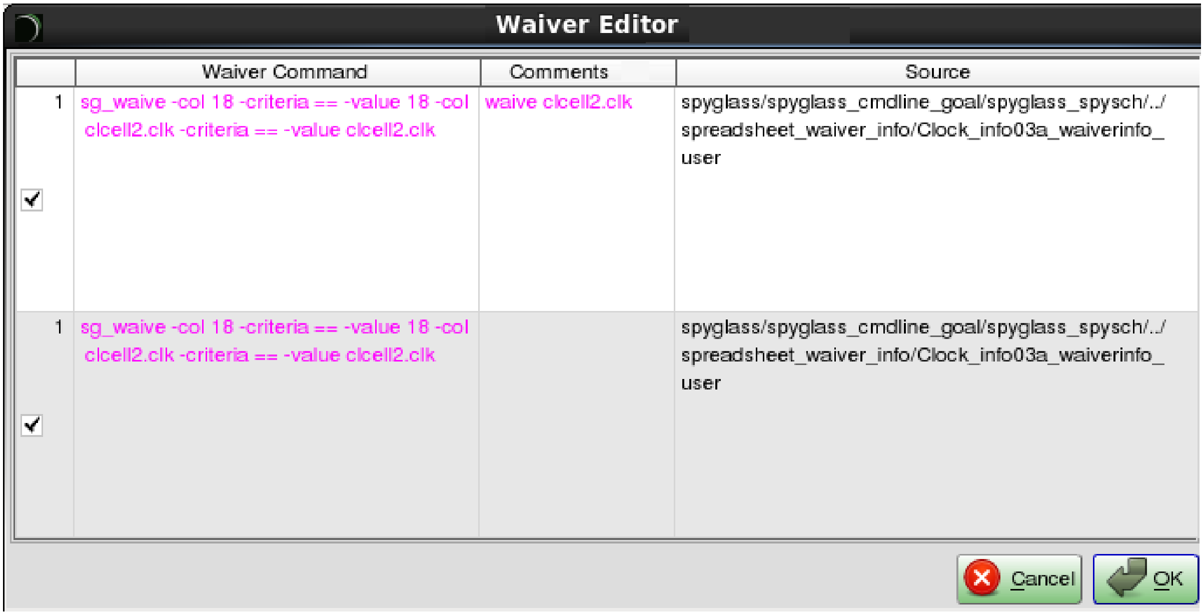
Editing the Waiver Commands Using Waiver Editor

You can edit the waiver command (sg_waive) for the waived row using the Waiver Editor window.

To edit a waiver command using the Waiver Editor window, perform the following steps from the spreadsheet viewer:

1. Click the Waiver option from the toolbar in the spreadsheet viewer.

   The Waiver Editor window is displayed as shown in *Figure 92*:



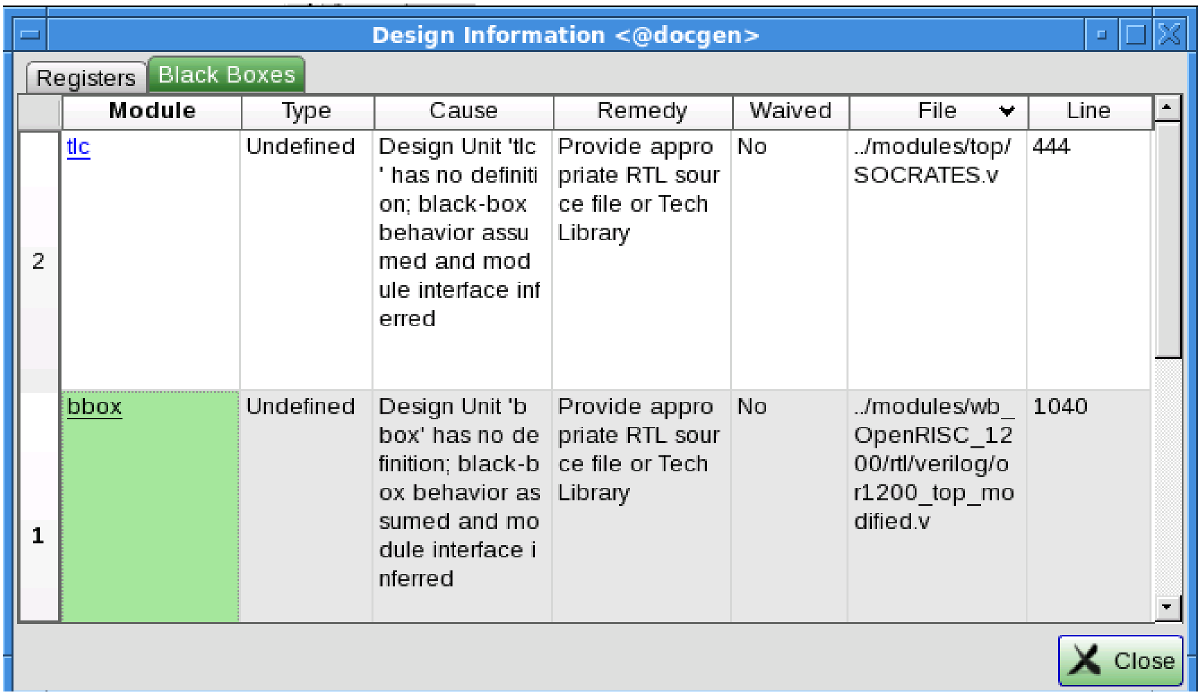
	Waiver Command	Comments	Source
1 ☑	sg_waive -col 18 -criteria == -value 18 -col clcell2.clk -criteria == -value clcell2.clk	waive clcell2.clk	spyglass/spyglass_cmdline_goal/spyglass_spysch/../spreadsheet_waiver_info/Clock_info03a_waiverinfo_user
1 ☑	sg_waive -col 18 -criteria == -value 18 -col clcell2.clk -criteria == -value clcell2.clk		spyglass/spyglass_cmdline_goal/spyglass_spysch/../spreadsheet_waiver_info/Clock_info03a_waiverinfo_user

FIGURE 92. Waiver Editor

Black Box Viewer Window

The *Black Box Viewer* window displays comprehensive information about all black box modules in the currently loaded design. This information appears in the form of a spreadsheet.

The following figure shows the *Black Box Viewer* window:



	Module	Type	Cause	Remedy	Waived	File	Line
2	tlc	Undefined	Design Unit 'tlc' has no definition; black-box behavior assumed and module interface inferred	Provide appropriate RTL source file or Tech Library	No	../modules/top/SOCRATES.v	444
1	bbox	Undefined	Design Unit 'bbox' has no definition; black-box behavior assumed and module interface inferred	Provide appropriate RTL source file or Tech Library	No	../modules/wb_OpenRISC_1200/rtl/verilog/or1200_top_modified.v	1040

FIGURE 93. Black Box Viewer Window

In the above window, each row represents information about a particular black box module. These details are mentioned in the following table:

Column	Description
Module	Displays the name of the black box module
Type	Displays the type of the black box module
Cause	Displays the cause of the module being considered as a black box
Remedy	Displays the suggested remedy for converting the black box module to a regular module
File	Displays the design file name in which the black box module is present
Line	Displays the line number in the design file where the black box module is present

In this window, you can:

- Click a column header to sort the information in that column in ascending or descending order.
- Copy information of a cell to the clipboard.

  You can copy information by right-clicking the cell and selecting *Copy* option from the shortcut menu. The copied information can then be pasted in any other application or location.

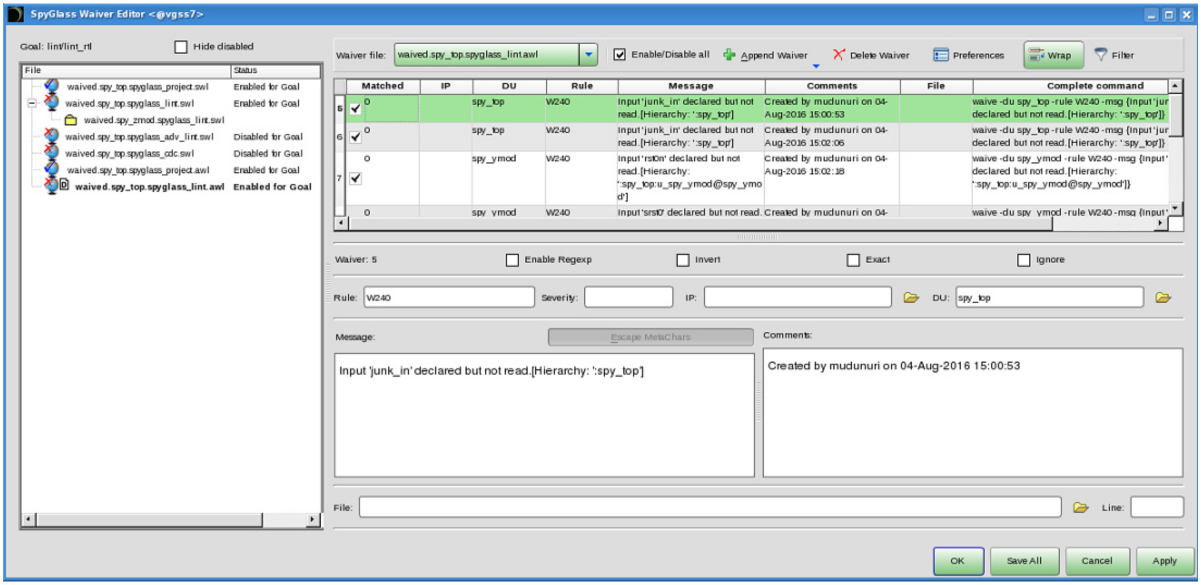
The Waiver Editor Window

If a violation message does not indicate a serious problem, you can waive that message so that it does not appear in the reported messages list in the *Violations Pane*. This way, you can focus on fixing other violations that indicate serious design issues.

Details of all waived messages are saved in waiver files (.awl files). SpyGlass Explorer can't write/save/edit contents in to .swl files. Pre-existing .swl waiver files can be used only in read-only mode.

Details of the Waiver Editor Window

The following figure displays the *Waiver Editor window*:



Goal: lint/lint_rtl □ Hide disabled

File	Status
waived.spy_top.spyglass_project.swl	Enabled for Goal
waived.spy_top.spyglass_lint.swl	Enabled for Goal
waived.spy_zmod.spyglass_lint.swl	
waived.spy_top.spyglass_adv_lint.swl	Disabled for Goal
waived.spy_top.spyglass_cdc.swl	Disabled for Goal
waived.spy_top.spyglass_project.awl	Enabled for Goal
waived.spy_top.spyglass_lint.awl	**Enabled for Goal**

Waiver file: waived.spy_top.spyglass_lint.awl ☑ Enable/Disable all

		Matched	IP	DU	Rule	Message	Comments	File	Complete command
5	✓	0		spy_top	W240	Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']	Created by mudunuri on 04-Aug-2016 15:00:53		waive -du spy_top -rule W240 -msg {Input 'jur declared but not read.[Hierarchy: ':spy_top']}
6	✓	0		spy_top	W240	Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']	Created by mudunuri on 04-Aug-2016 15:02:06		waive -du spy_top -rule W240 -msg {Input 'jur declared but not read.[Hierarchy: ':spy_top']}
7	✓	0		spy_ymod	W240	Input 'rst0n' declared but not read.[Hierarchy: ':spy_top:u_spy_ymod@spy_ymod']	Created by mudunuri on 04-Aug-2016 15:02:18		waive -du spy_ymod -rule W240 -msg {Input ' declared but not read.[Hierarchy: ':spy_top:u_spy_ymod@spy_ymod']}
		0		spy_ymod	W240	Input 'srst0' declared but not read.	Created by mudunuri on 04-		waive -du spy_ymod -rule W240 -msg {Input '

Waiver: 5 □ Enable Regexp □ Invert □ Exact □ Ignore

Rule: W240 Severity: IP: DU: spy_top

Message: Escape MetaChars

Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']

Comments:

Created by mudunuri on 04-Aug-2016 15:00:53

File: Line:

FIGURE 94. SpyGlass Waiver Editor Window

The *Waiver Editor window* contains the following four buttons at the bottom of the window:

Button	Description
OK	Click this button to apply the current waiver settings and exit the *Waiver Editor window* NOTE: SpyGlass prompts you for a re-run of SpyGlass when edited block waiver files, which have changed from the last run of SpyGlass are applied, or when some new hierarchical waive import commands are added in the waiver files.
Apply	Click this button to apply the current waiver settings (that is, any new waiver expressions or any changes made to the existing waiver expressions) in the *Waiver Editor window* and continue. NOTE: If the *Apply Waiver When Created* option is selected in the *Waiver* page in the *Preferences* dialog, any message selected for waiving is added to the *Waiver Editor window* and waived immediately. When you select the *Apply Waiver From Spread Sheet* option, the message is just added to the *Waiver Editor window* and is waived only when you click *Apply* or *OK* in the *Waiver Editor window*
Cancel	Click this button to close the *Waiver Editor window* to abandon the unapplied changes and exit. If there are any unapplied changes, SpyGlass prompts you to either cancel the action (takes you back to the Waiver Editor window so that you can apply changes) or continue with the action (closes the Waiver Editor window abandoning the unapplied modifications).
Save All	Click this button to save the modifications/changes made in the waiver file. If you have any read-only files with unsaved changes, then pressing the **Save All** button prompts you to save the file by displaying the read-only Files dialog. Clicking the *Continue* button in this dialog saves the waiver files (other than read-only ones) to the disk. Clicking the *Cancel* button does not save any file to the disk.

This window is divided into the following sections:

- *Tree-View Section*
- *Grid View Section*
- *Details Section*

You can resize all the above sections by dragging the edges of the required section.

Tree-View Section

This section lists the waiver files for the currently loaded goal. The name of the currently loaded goal appears on the header of this section.

The following figure illustrates the sample tree-view section:

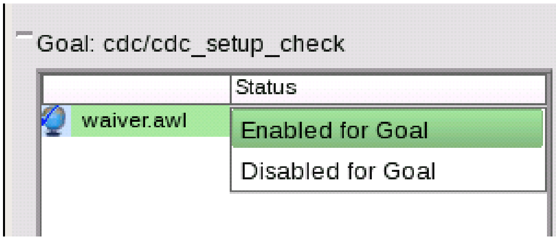


FIGURE 95. Tree -View Section

The above figure shows three waiver files for the currently loaded goal, **initial_rtl/lint/synthesis**. To enable or disable a waiver file for this goal, select the waiver file and then select the **Enabled for Goal** or **Disabled for Goal** option in the **Status** column.

Symbols Appearing for Different Waiver Files

The following table provides the description of each symbol:

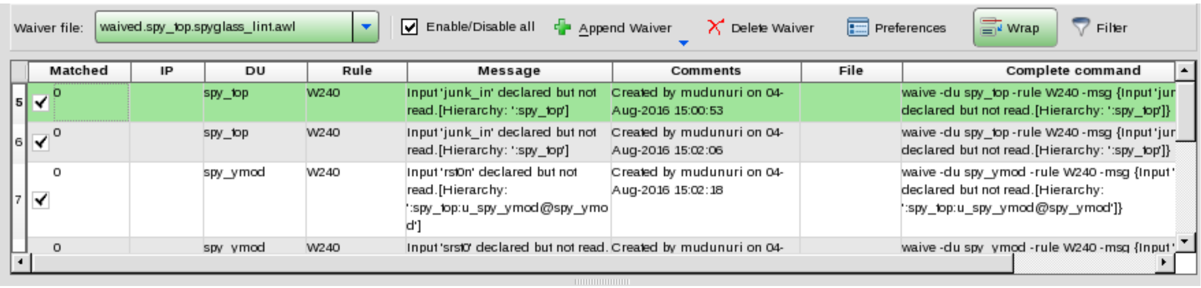
Symbol	Description
🟥	Specifies that the waiver file is read-only. This icon also appears for hierarchical migrated waiver files and pragma-based waiver files.
🟩🟥	Specifies that the waiver file contains partially-migrated waiver commands. When you move the cursor over such waiver files, a tool-tip appears specifying the name of that waiver file and the number of migrated commands over the total number commands.
🗎	Specifies the default waiver file. For details on a default file, see *Set File As Default*.

Symbol	Description
✓	Specifies that the waiver file is *Enabled for Goal*.
✗	Specifies that the waiver file is *Disabled for Goal*.

Grid View Section

This section contains waiver information in the form of a spreadsheet.

The following figure illustrates a sample grid-view:



Waiver file: waived.spy_top.spyglass_lint.awl

☑ Enable/Disable all

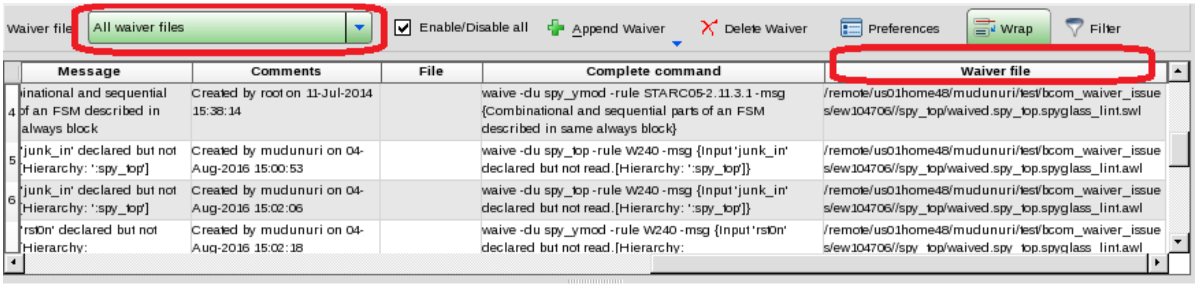
	Matched	IP	DU	Rule	Message	Comments	File	Complete command
5 ✓	0		spy_top	W240	Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']	Created by mudunuri on 04-Aug-2016 15:00:53		waive -du spy_top -rule W240 -msg {Input 'jur declared but not read.[Hierarchy: ':spy_top']}
6 ✓	0		spy_top	W240	Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']	Created by mudunuri on 04-Aug-2016 15:02:06		waive -du spy_top -rule W240 -msg {Input 'jur declared but not read.[Hierarchy: ':spy_top']}
7 ✓	0		spy_ymod	W240	Input 'rst0n' declared but not read.[Hierarchy: ':spy_top:u_spy_ymod@spy_ymod']	Created by mudunuri on 04-Aug-2016 15:02:18		waive -du spy_ymod -rule W240 -msg {Input ' declared but not read.[Hierarchy: ':spy_top:u_spy_ymod@spy_ymod']}
	0		spy_ymod	W240	Input 'srst0' declared but not read.	Created by mudunuri on 04-		waive -du spy_ymod -rule W240 -msg {Input '

FIGURE 96. Grid View Section

When you double-click on a waiver file appearing in the tree-view section, the individual waiver details of that file appear in separate rows and columns in the grid-view. In addition, the name of the file get updated in the file selection combo box on the title bar of the *Waiver Editor Window*.

By default, during the initial display, SpyGlass displays the waiver content from the default waiver file. If there is no default waiver file set, then it displays all wavers from all waiver files.

You can change the grid-view content by choosing one of the waiver files from the **Waiver File** selection combo box. You can also select the **All waiver files** option from the **Waiver File** selection combo box to load all the waivers from all the waiver files. In this case, the grid-view displays the additional **Waiver File** column, which lists the appropriate waiver file name for each corresponding row (*Figure 97*).



	Message	Comments	File	Complete command	Waiver file
4	inational and sequential of an FSM described in always block	Created by root on 11-Jul-2014 15:38:14		waive -du spy_ymod -rule STARC05-2.11.3.1 -msg {Combinational and sequential parts of an FSM described in same always block}	/remote/us01home48/mudunuri/test/bcom_waiver_issues/ew104706//spy_top/waived.spy_top.spyglass_lint.swl
5	'junk_in' declared but not [Hierarchy: ':spy_top']	Created by mudunuri on 04-Aug-2016 15:00:53		waive -du spy_top -rule W240 -msg {Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']}	/remote/us01home48/mudunuri/test/bcom_waiver_issues/ew104706//spy_top/waived.spy_top.spyglass_lint.awl
6	'junk_in' declared but not [Hierarchy: ':spy_top']	Created by mudunuri on 04-Aug-2016 15:02:06		waive -du spy_top -rule W240 -msg {Input 'junk_in' declared but not read.[Hierarchy: ':spy_top']}	/remote/us01home48/mudunuri/test/bcom_waiver_issues/ew104706//spy_top/waived.spy_top.spyglass_lint.awl
	'rst0n' declared but not [Hierarchy:	Created by mudunuri on 04-Aug-2016 15:02:18		waive -du spy_ymod -rule W240 -msg {Input 'rst0n' declared but not read.[Hierarchy:	/remote/us01home48/mudunuri/test/bcom_waiver_issues/ew104706//spy_top/waived.spy_top.spyglass_lint.awl

FIGURE 97. Viewing Content from All Waiver Files

In the grid-view, you can perform the following actions:

- Select the available waiver file from the selection combo box.
- Enable or disable individual waivers by selecting or deselecting the corresponding buttons ( ) in each row.
- Choose **Enable/Disable all** check button on the tool bar to update all displayed waivers in one go.
- Sort the available waiver expressions by a rule or a file. To sort, right-click on the waiver file header.

  Sorting the waivers disables the **Details** section of the Waiver Editor window. To re-enable the **Details** section, select the **Restore** option.
- Enable or disable individual waivers by selecting or deselecting the corresponding buttons (□) in each row.
- To modify the settings of a waiver item, select □ button corresponding to that waiver. Details of the waiver is displayed in the *Details Section*. Update the required fields in the details section as required.
- Resize rows and columns of the grid by dragging the edges of rows and columns.
- Specify new waivers by clicking the *Append Waiver* option.
- Delete an existing waiver setting by clicking the *Delete Waiver* option.
- Filter waivers based on the waiver comments. Press the **Filter** button on the toolbar to display the **Comment Filter** field. The field accepts regular expression format for pattern matching.

  When you specify any text or string in the Comment Filter field, only the matched results are displayed in the grid.

You can view the list of rules for which waivers have been applied. The count of such rules is displayed as a hyperlink in the *Matched* column of the spreadsheet. When you click the count in the *Matched* column a list rules for which the waiver has been applied is displayed in the shortcut menu. You can also click a rule to locate the message in the Waived tree.

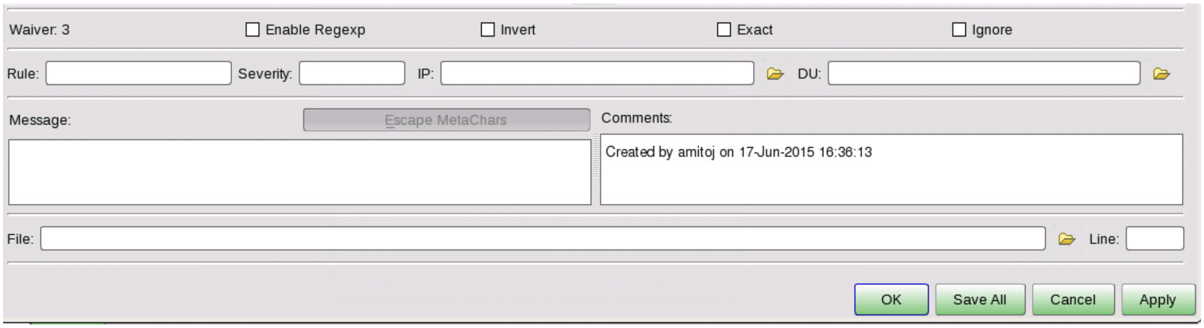
Details Section

This section enables you to specify details for the currently selected waiver file in the tree-view section. However, for un-editable waiver files, options and controls are disabled.

If some of the waiver files are not editable, for example, swl format files or include/import files, the corresponding waiver rows will not display the check button to modify the enable/disable status.

Also if the current grid-view is displaying a read-only, that is, not editable file, the **Enable/Disable all** check button on tool bar will become inactive.

The following figure illustrates the details-view section:



Waiver: 3 □ Enable Regexp □ Invert □ Exact □ Ignore

Rule: Severity: IP: DU:

Message: Comments: Created by amitoj on 17-Jun-2015 16:36:13

File: Line:

FIGURE 98. Waiver Details

In the above section, you can specify the required waiver information in different fields.

The following points discuss various fields of this section:

- *File:* Use this field to specify files for which rule messages should be waived. You can also click ... and select a file from the *File Selection* dialog.

- *Line*: Use this field to specify the line for which rule messages should be waived.
- *IP*: Use this field to waive rule messages for the specified design unit (IP blocks) or all design units specified in the IP library. You can also click ... and select a file from the *IP Selection* dialog.
- *DU:* Use this field to waive rule messages for the specified design unit or all design units in the specified library. You can also click ... and select a file from the *DU Selection* dialog.
- *Rule*: Use this field to specify the rules, rule groups, or products for which the messages should be waived.
- *Severity*: Use this field to specify the severity class or severity label for which the rule messages should be waived.
- *Except*: Use this field to specify the rules, rule groups, or products for which the messages need to be excluded.
- *Weight*: Use this field to specify the weight of rules that need to be waived.
- *Comment*: Use this field to add a comment for the specified waiver. By default, when a row is added in the spreadsheet, the comment text box displays information about the user who created the waiver and the time when the waiver was created.
- *Message:* Use this field to specify the message that should be waived.

  In this section, at least one of the fields must be specified from one of the following field groups:

  Group 1: *File*/*Line*, *IP*, *DU*

  Group 2: *Rule*, *Severity*

  When more than one field is specified from Group 1, a message is waived if any one of the Group 1 field criteria is met. When more than one field is specified from Group 2, a message is waived only if all the Group 2 field criteria are met. If fields from both Group 1 and Group 2 are specified, a message is waived only if any one of the Group 1 field criteria is met and all Group 2 field criteria are met.

  When you type a value in the *File*, *Line*, *IP*, *DU*, *Rule*, *Severity*, *Except*, or *Weight* fields, the *Waiver Expression* column is dynamically updated with the value entered by you. For example, if you enter the value “NoXInCase-ML” in the *Rule* field, the *Waiver Expression* column is dynamically updated as follows:

```
waive -rule "NoXInCase-ML"
```

- *Enable Regexp*: Use this option to enable regular expression support in the selected fields. When you select this option, the *Escape MetaChars* option is also enabled.
- *Escape MetaChars*: As you enter a message expression in the *Message* field, any meta-characters entered are shown in red color. You can escape these meta-characters yourself or select the *Escape MetaChars* option to escape meta-characters.

  Escaped meta-characters are displayed in black color. The message expression entered in the *Message* field appears in the waiver description in the *Waiver Expression* field of the table.
- *User Comment*: Use this field to add/edit comments for waivers.
- *Do not itemize matched messages*: Select this option to report only the number of messages waived (instead of individual messages) in the waiver report.

Right-Click Menu Options in the Waiver Window

Different right-click menu options appear in the tree-view and grid-view section of the *Waiver* window. They are discussed in the following sections:

- *Right-Click Options of Tree-View Section*
- *Right-Click Options of Grid-View Section*

Right-Click Options of Tree-View Section

When you right-click on the *Waiver Files* node or a file under this node, the following options appear in the shortcut menu (*Figure 99*):

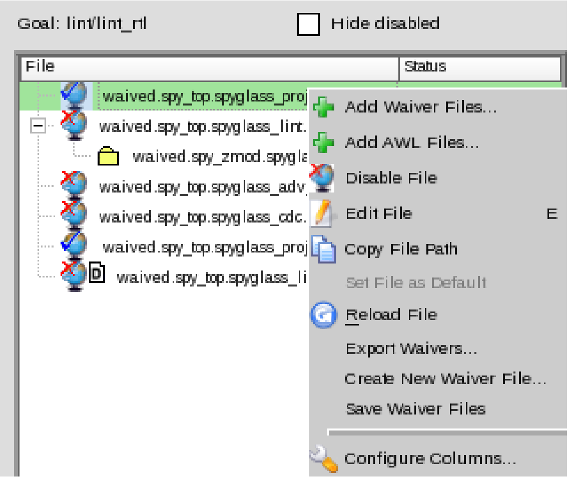


FIGURE 99. Right-Click Options for Tree-View Section

Add Waiver Files

Select this option to add existing waiver files, SWL files, in the tree-view.

Add AWL Files

Select this option to add existing AWL files in the tree-view.

Disable Files

Select this option to enable or disable the file for the current goal.

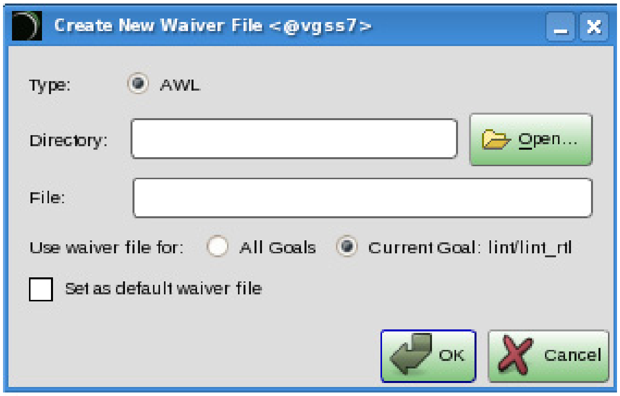
Copy File Path

Select this option to copy the file path to the clip-board.

Create New Waiver File

Select this option to add a new waiver file in the tree-view.

When you select this option, the *Create new waiver file* dialog appears, as shown in the following figure:



Create New Waiver File <@vgss7>

Type: ● AWL

Directory:

File:

Use waiver file for: ○ All Goals ● Current Goal: lint/lint_rtl

□ Set as default waiver file

FIGURE 100. Create New Waiver File

Waiver files that are not yet saved on the disk are indicated with a red-colored icon in the tree-view.

Details of this dialog are given in the following points:

- *Directory* text box enables you to specify the directory in which the waiver file is present.
- *Browse* button is used to browse to the directory of the waiver file.
- *File Name* text box enables you to specify the name of the waiver file (preferably with the .awl extension).
- *Set as default* option is used to set the waiver file as the default file.
- *OK* button is used to create the specified waiver file and add it in the tree-view.

NOTE: *Waiver files can have any extension. When you are creating a new waivers file, SpyGlass suggests the .awl extension to facilitate better recognition and handling. If you do not specify the file extension for a waiver file, then by default, the .awl file extension is appended to the waiver file name.*

Save Waiver Files

Select this option to save the selected file with its current waiver settings.

Set File As Default

Select this option to set the selected waiver file as the default waiver file for all SpyGlass sessions until you select another waiver file to be the default file.

The default waiver file name is highlighted in blue color in the tree-view, and the 🗎 icon appears before such file name.

Edit File

Select this option to edit the selected waiver file.

When you select this option, a text editor appears in which you can make the required changes.

Save File

Select this option to save the selected file with its current waiver settings.

Export Waiver Files

Select this option to save waiver information in a tab-separated format. This option appears only when you right-click on the *Waiver Files* node.

This format contains various tab-separated columns, such as FileName, Rule, Comment, No of matches, DU, IP, and Waiver Expression. Appropriate information is present under each column.

Saving waivers in this format is useful when you load this format in a csv file from a different editor window. In any editor window, waiver information in the csv file appears in appropriate columns with proper headings.

NOTE: *You cannot open this csv file through SpyGlass GUI.*

Reload File

Select this option to reload a waiver file.

It is not recommended to edit the same file inside as well as outside the *Waiver Editor* window simultaneously. One set of changes will be lost whenever the file is saved and/or reloaded.

It is recommended to edit waiver files in the *Waiver Editor* window only. If you edit waiver files outside this window (for example by using a text editor), such edits are not effective until you reload that waiver file in the *Waiver Editor* window.

Configure Column(s)

Use this option to select/de-select additional tree view columns.

Right-Click Options of Grid-View Section

When you right-click one or more rows in the grid, following options appear in the right-click shortcut menu:

Insert Waiver > Row Above

Select this option to insert a new waiver row above the currently selected row.

Insert Waiver > Row Below

Select this option to insert a new waiver row below the currently selected row.

Insert Import Waiver > Row Above

Select this option to import a new hierarchical waiver row above the currently selected row.

Insert Import Waiver > Row Below

Select this option to import a new hierarchical waiver row below the currently selected row.

Delete Waiver(s)

Select this option to delete the currently selected row(s).

Copy

Select this option to copy waiver commands in the selected row.

Paste

Select this option to paste the copied waiver commands in the currently selected row.

Paste Below

Select this option to paste the copied waiver commands in the next row of the currently selected row.

Undo

Select this option to undo the last change.

Redo

Select this option to redo the last undo change.

Display Comment

Select this option to view the complete text of the User Comment field in a pop-up box.

Set Row Height

Select this option to set the row height between 1 (default) to 5 units for the entire table. This option is useful if the table contains long message expressions that do not fit in the default row height.

Insert Common Comment

Select this option to add a common comment to all the selected waiver commands. When you select this option, SpyGlass displays the *Add Common Comment* dialog. In this dialog, add the required comment in the textbox, and click the *OK* button. This adds the specified comment to all the selected waiver commands.

Cross Probe to File

Select this option to cross probe between the selected waiver file and the Waiver Editor window.

Waiver Settings Affecting the Msg Tree Results

If SpyGlass analysis has already been run on the current design, any modifications in the waiver settings are reflected in the *Msg Tree*. However, the *Msg Tree* itself is persistent and retains its expanded state.

NOTE: *It is recommended that you save (or discard) the settings made in the Waiver Editor window before starting a new SpyGlass analysis. Otherwise, there can be a mismatch between the number of messages actually displayed in SpyGlass (this is based on the current settings of Waiver Editor window) and the number of messages that should be displayed based on the saved (physical) waiver files. SpyGlass also prompts you if such a mismatch exists at the time of starting SpyGlass analysis run.*

Hierarchy Traversal across SpyGlass Windows

For the purpose of hierarchy traversal, the following types of design units are assumed:

- RTL Module: Refers to a module that contains line number dumped by the message
- Schematic Top Module: Refers to a top hierarchy for which schematic back-annotation data is dumped by a message
- Schematic Module: Refers to a module for which schematic back-annotation data is dumped by a message

When a message is selected, the RTL module is always shown in *HDL Viewer Pane*. However, the Modular Schematic window and Incremental Schematic window can show a Schematic Module depending on the schematic highlight data dumped by the message.

There can be the following three cases where the ‘Instance hierarchies’ can differ:

1. RTL Module and Schematic Top Module are same.

   Schematic Top Module is shown in the schematic windows and Schematic Top Module is picked as the ‘Instance hierarchy’ for schematic windows.

   Highlighting is visible in both Modular Schematic window and Incremental Schematic window.

2. RTL Module is different from Schematic Top Module but there is a Schematic Module same as RTL Module.

   Schematic Module is shown in Modular Schematic window and hierarchy of Schematic Module is picked for Modular Schematic window. For example, if Schematic Top Module is A.b.c and the RTL module is D, an instance of D is searched in the schematic data and shown. A.b.c.d will be considered as ‘Instance hierarchy’ for Modular Schematic window.

Both Modular Schematic window and Incremental Schematic window show the schematic highlighting. The case in which no instance of D is found is covered in Case 3 below.

3. No Schematic module for RTL Module

   RTL Module is shown in Modular Schematic window and any ‘Instance hierarchy’ is picked for that RTL Module from the Design view page. This behavior is also valid for the non-schematic messages also. There is no highlighting in the Modular Schematic window. However, if the top module of the picked ‘Instance hierarchy’ is same as that of the Schematic Top module, the Incremental Schematic window shows the back-annotation data.

Understanding Design Stages in SpyGlass

This section discusses the following stages of SpyGlass:

- *Stage 1: Setting up the Design (Design Setup)*
- *Stage 2: Selecting a Goal (Goal Setup & Run)*
- *Stage 3: Analyzing a Design (Analyze Results)*

Stage 1: Setting up the Design (Design Setup)

This is the first stage when you start a new session in SpyGlass.

During this stage, you create the basic design setup by specifying information, such as design files and design options. In addition, you check for some basic design issues before proceeding to the next stage.

To complete this stage, perform the following tasks:

1. *Adding Design Files*
2. *Viewing and Changing Design Read Options*
3. *Running Design Read*

Adding Design Files

Select the *Add Design Files* tab to add design files for SpyGlass analysis. You can use this tab to perform the following tasks:

- *Adding Files in GUI*
- *Editing Files*
- *Setting Stop Files*
- *Ignoring Files from SpyGlass Analysis*
- *Configuring Columns*

Adding Files in GUI

In the *Add Design Files* tab, you can add various types of files, as described in the following table:

TABLE 8 File Types Under the Add Design Files Tab

File	Description	Tcl Shell Usage

HDL files	These files include Verilog (.v, .verilog, and .sv) files, VHDL (.vhdl and .vhd) files, Design Exchange Format (.def) files, and Library Exchange Format (.lef) files.
	NOTE: *If you specify a file of an unknown type, it appears in red color. You cannot proceed to the next step unless you specify the correct file type*.
		`read_file [-type <verilog|vhdl|hdl>] <filename>`
		Note: *read_file <filename> is equivalent to read_file -type hdl <filename>*
		Example:
		`read_file -type verilog test.v`

SGDC files	The files are .sgdc files that contain SpyGlass design constraints. These design constraints are used to provide additional design information that is not apparent in the RTL.	`read_file -type sgdc <filename>`
		Example:
		`read_file -type sgdc constraints.sgdc`

HDL libraries	Contain precompiled VHDL or Verilog files.	`set_option lib <logical_name> <physical_path>`
		Example:
		`set_option lib MyLib /a/b/mylib_path`

Technology libraries	Include Synopsys .lib, .sglib, .plib, and .gateslib files that are required for structured netlists using those library cells.	`read_file -type <gateslib | sglib | plib> <lib name>`
		Example:
		`read_file -type sglib library.sglib`

Source list files	Include files with a .spp or .f extension. These files contain the design files, design options, or a combination of both.	`read_file -type sourcelist <file>`
		Example:
		`read_file -type sourcelist sources.f`

Steps for Adding Files

To add a file, perform the following steps from the Design Setup pane:

1. Click the *Add File(s)* button:

   Alternatively, right-click in the *HDL Files*, *SGDC Files*, *HDL Libraries*, or *Tech Libraries* sections and select the *Add HDL File(s)*, *Add Constraint File(s)*, *Add HDL Lib File(s)*, or *Add Tech Lib File(s)* options, respectively, from the shortcut menu.

   This displays the *Add File(s)* dialog, as shown in the following figure:

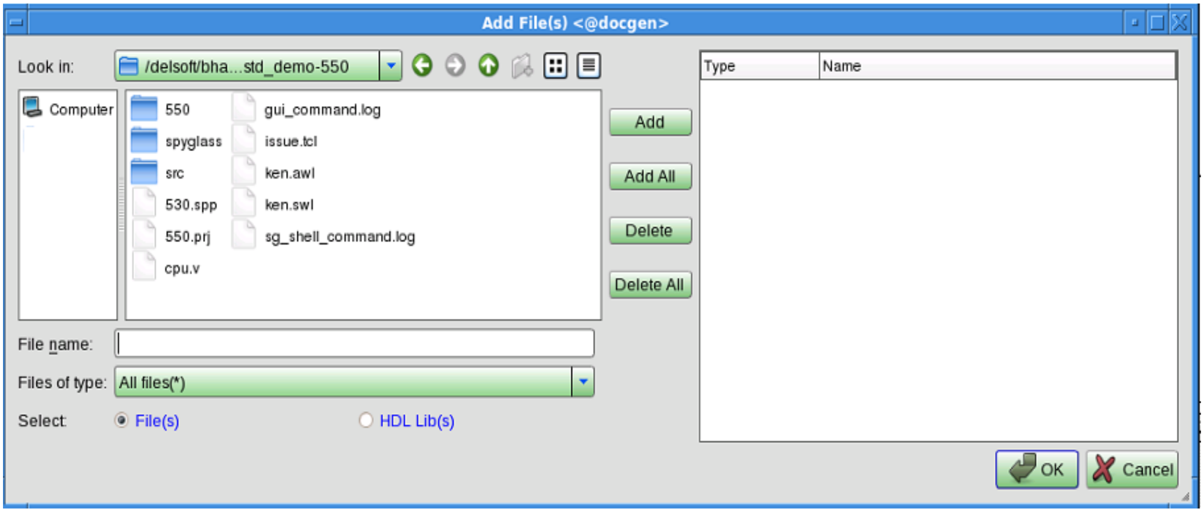


FIGURE 101. Add File(s) Dialog Box

2. Click the *File(s)* option to add the HDL files or the *HDL Lib(s)* option to add HDL libraries.
3. Click the *Look In* drop-down list to select a directory containing the required files.
4. For HDL files, select the file type from the *Filter* drop-down list.

   For HDL library files, select the directory containing the files and specify a logical library name in the *Logical Library Name* text field.

NOTE: *The Logical Library Name text field appears only when you click the HDL Lib(s)* *option.*

5. Select the required file, and click the *Add* button to add that file.

   With HDL files, you can add all files present in the directory by clicking the *Add All* button.

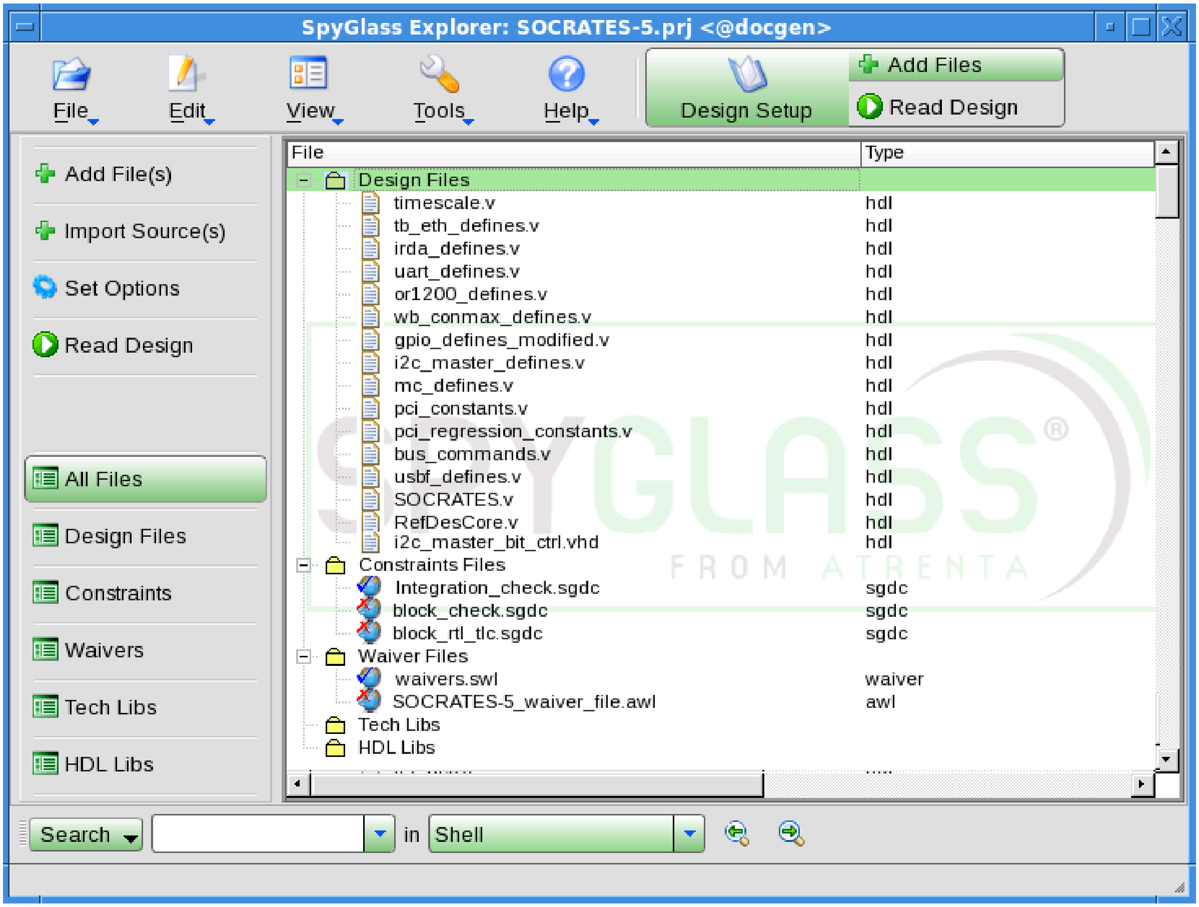
NOTE: *If you are adding HDL library files, you can add the directory that contains the HDL files.*

6. Click the *OK* button to close the dialog.

NOTE: *If you want to remove a file that you have added in the Add File(s) dialog, select that file and click the Delete button. To remove all the added files, click the Delete All button.*

After performing the above steps, the specified files appear under appropriate sections, such as *HDL Files*, *Constraints Files*, *HDL Libraries*, and *Tech Libraries*, depending upon the type of the file selected.

The following figure shows various sections containing different files:



File	Type
Design Files	
timescale.v	hdl
tb_eth_defines.v	hdl
irda_defines.v	hdl
uart_defines.v	hdl
or1200_defines.v	hdl
wb_conmax_defines.v	hdl
gpio_defines_modified.v	hdl
i2c_master_defines.v	hdl
mc_defines.v	hdl
pci_constants.v	hdl
pci_regression_constants.v	hdl
bus_commands.v	hdl
usbf_defines.v	hdl
SOCRATES.v	hdl
RefDesCore.v	hdl
i2c_master_bit_ctrl.vhd	hdl
Constraints Files	
Integration_check.sgdc	sgdc
block_check.sgdc	sgdc
block_rtl_tlc.sgdc	sgdc
Waiver Files	
waivers.swl	waiver
SOCRATES-5_waiver_file.awl	awl
Tech Libs	
HDL Libs	

FIGURE 102. Add Files Pane

Placing the mouse pointer over a file in any of these sections displays a tool-tip showing the path, size, and last modified date of that file.

Adding Source Files

It is recommended that you specify a list of all HDL files in a single source file (.spp or .f), and then add that source file in SpyGlass.

Use the following command in a project file to add a source file:

```
read_file -type sourcelist <file>
```

When you add a source file, design files specified in that source file appear in the *HDL Files*, *SGDC Files*, *HDL Libraries*, and *Tech Libraries* sections depending upon the file type.

A source file should contain HDL files and HDL-related directives. For example, you can specify a Verilog library directory by using the `-y <path>` command.

SpyGlass accepts many of the same HDL directives as some popular simulation tools in the source file, and HDL code directly supports some directives. For a list of supported HDL directives, see *Supported HDL Directives*. For HDL directives that differ by tool, see *Re-using Simulation Scripts* for the conversion to SpyGlass project format.

NOTE: *If an option (design file or other design option) added through a source file is already present in a project file, the option added through the source file gets higher precedence. This means that the option specified through the source file overrides the option present in the project file.*

Changing the Status of Constraints Files

By default, constraint files appearing under the *Constraints Files* section are *Globally Enabled*. This means that these files are enabled for all goals and all such files are passed in the design read run.

You can change this status by selecting the *Globally Disabled* option from the drop-down list appearing adjacent to a constraint file name.

if in a project file, specify the following command after selecting a goal using the current_goal command:

```
read_file -type sgdc
```

This command makes the specified Constraints file specific to that goal only. However, if you have changed the status of the Constraints file before selecting any goal, then it becomes a global Constraints file and is applied to all the goals of that project.

Excluding Files from SpyGlass Analysis

To exclude a file from SpyGlass analysis, perform any of the following actions:

- Right-click on the file, and select the *Delete* option from the shortcut menu.
- Select the file, and click the *Delete File(s)* link.
- If you want to remove the file from a project file, delete the corresponding *read_file* command that you specified to add this file.
- If you are working in sg_shell and the file is already added, then use the remove_file command. For example, the following command would remove all HDL files:

  ```
  remove_file -type hdl
  ```

  You cannot remove a single HDL file in the sg_shell mode.

To remove a source file, right-click on the file name and select the *Delete Source List File* shortcut menu option.

To remove multiple files from the *HDL Files*, *HDL Libraries*, and *Tech Libraries* sections, perform the following steps:

1. Press and hold the <*Ctrl*> key on the keyboard and select the files that you want to delete.
2. Click the *Delete File(s)* link or press the <*Delete*> key on the keyboard.

Importing sourcelist files

Instead of adding a sourcelist file to the project, you can import the file. When importing, all the files added through the sourcelist file would become a part of the project.

The subsequent changes to the imported file will not get reflected in the project. Therefore, you must import the sourcelist file again, if you want to override any setting in the current sourcelist file.

Specifying Functionality Information of Gate Cells

If your design contains instantiated gates (cells), you should tell SpyGlass how to interpret these cells so that SpyGlass can analyze them.

By default, SpyGlass treats such cells as black boxes and does not analyze them.

To enable SpyGlass analyze such cells, specify the functionality information including structure and parametric data of these cells in any of the following ways:

- *Specifying Functionality Information through .lib Files*
- *Specifying Functionality Information through Verilog design or Library File*
- *Specifying Functionality Information through a VHDL File*

NOTE: *The actual interpretation of gate cells may be limited or incorrect for complex cells. Also, the SpyGlass Logic Evaluator engine (mainly used by the SpyGlass DFT solution) does not work for libraries specified with the gateslib command.*

Specifying Functionality Information through .lib Files

In this case, perform the following steps:

1. Specify the .lib file by using the `gateslib` option, as shown below:

   ```
   read_file -type gateslib <lib-file>
   ```

2. Specify the following command in a project file:

   ```
   set_option enable_gateslib_autocompile yes
   ```

   When you specify the above command, SpyGlass compiles the specified .lib file into its corresponding .sglib file. From this .sglib file, SpyGlass extracts the functionality information of the gate cells.

   If you do not perform this step, SpyGlass is unable to extract information from the specified .lib file. As a result, SpyGlass treats the gate cells as black boxes.

Specifying Functionality Information through Verilog design or Library File

If the functionality information of gate cells is present in a Verilog design file or a Verilog library file, perform the following steps:

1. Specify the files by using the following commands:
   - ❐ For Verilog library file:

     ```
     set_option v <file-name> command
     ```

   - ❐ For Verilog design file:

     ```
     read_file -type verilog <file-name>
     ```

2. Specify the following command in a project file:

   ```
   set_option enable_precompile_vlog yes
   ```

NOTE: *If you also specify a .lib file by using the read_file -type gateslib <lib-file>*

command, SpyGlass ignores that .lib file and picks functionality information of gate cells from the specified Verilog library file or Verilog design file.

Specifying Functionality Information through a VHDL File

If the functionality information of gate cells is present in a VHDL file, specify that file by using the following command:

```
read_file -type vhdl <file-name>
```

NOTE: *If you also specify a .lib file by using the gateslib command, SpyGlass ignores that .lib file and picks functionality information of gate cells from the specified VHDL file.*

Specifying a List of .sglib Files

Use the following command in a project file to specify a list of .sglib files:

```
read_file -type sourcelist <file-name>.f
```

Where, <file-name>.f contains a list of .sglib files, as shown in the following example:

```
-sglib path/to/sglibs/k1.sglib
-sglib path/to/sglibs/k2.sglib
-sglib path/to/sglibs/k3.sglib
-sglib path/to/sglibs/k4.sglib
```

Specifying Compressed Verilog Designs

You can directly specify compressed Verilog netlist/RTL files (.gz files) to SpyGlass. This avoids the task of uncompressing netlist/RTL files that are typically huge in size, thereby occupying large disk space.

To specify a compressed netlist/RTL file in SpyGlass, perform any of the following actions:

- Click the *Add Files* option under the *Add Design Files* tab, and select the required compressed file.
- Specify the compressed file by using the following project file command:

  ```
  read_file -type <verilog | vhdl | hdl> <compressed-file-
  ```

```
name>
```

- Specify the name of the compressed file in a source file (.f file), and specify that source file in SpyGlass by using the following project file command:

  ```
  read_file -type sourcelist <source-file-name>
  ```

Editing Files

You can edit files, such as HDL files, libraries, and source files.

Editing HDL Files/Technology Libraries

To edit a file in the *HDL Files* or *Tech Libraries* section, perform the following steps:

1. Right-click on the file, and select the *Edit File* option from the shortcut menu or press the *E* button on the keyboard

   The selected file appears in a text editor.
2. Make the required modifications in the file.
3. Save the file.
4. Close the text editor.

Setting Stop Files

SpyGlass does not perform rule-checking on design units specified in the files marked as stopped.

To set a file as a stop file, right-click on the file in the *HDL Files* section and select the *Stop file* option from the shortcut menu. The 🟥 icon appears before the file name to indicate that file as a stopped file.

To remove a file from the files list that is stopped, right-click on that file and select the *Remove Stop File* option from the shortcut menu.

Usage in sg_shell or the project file

- To specify a stop file.When working in sg_shell or project file, specify the following command:

  ```
  set_option stopfile <file>
  ```

- To remove the specified stopfile, specify the following command:

  ```
  remove_option stopfile
  ```

- To ignore a file from processing, specify the following command:

  ```
  set_option ignorefile <file>
  ```

- To remove the file marked as ignored, specify the following command:

  ```
  remove_option ignorefile
  ```

Ignoring Files from SpyGlass Analysis

Ignoring a file means ignoring all design units specified in that file from SpyGlass analysis. SpyGlass considers such design units as black boxes during analysis.

To ignore a file, right-click on the file name in the *HDL Files* section and select the *Ignore File* option from the shortcut menu. The ℹ icon appears before the file name to indicate that this file will be ignored during SpyGlass analysis.

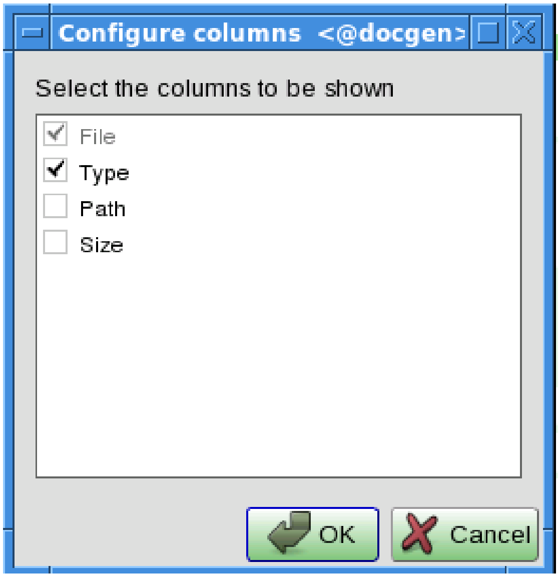
If you later want to consider such a file for SpyGlass analysis, right-click on that file name and select the *Remove Ignore File* option from the shortcut menu.

Configuring Columns

To display or hide columns appearing in the *HDL Files*, *SGDC Files*, and *Tech Libraries* sections, perform the following steps:

1. Right-click on any section, and select the *Configure Columns* option from the shortcut menu.

   The *Configure Columns* dialog appears as shown below.



Configure columns <@docgen>

Select the columns to be shown

☑ File

☑ Type

☐ Path

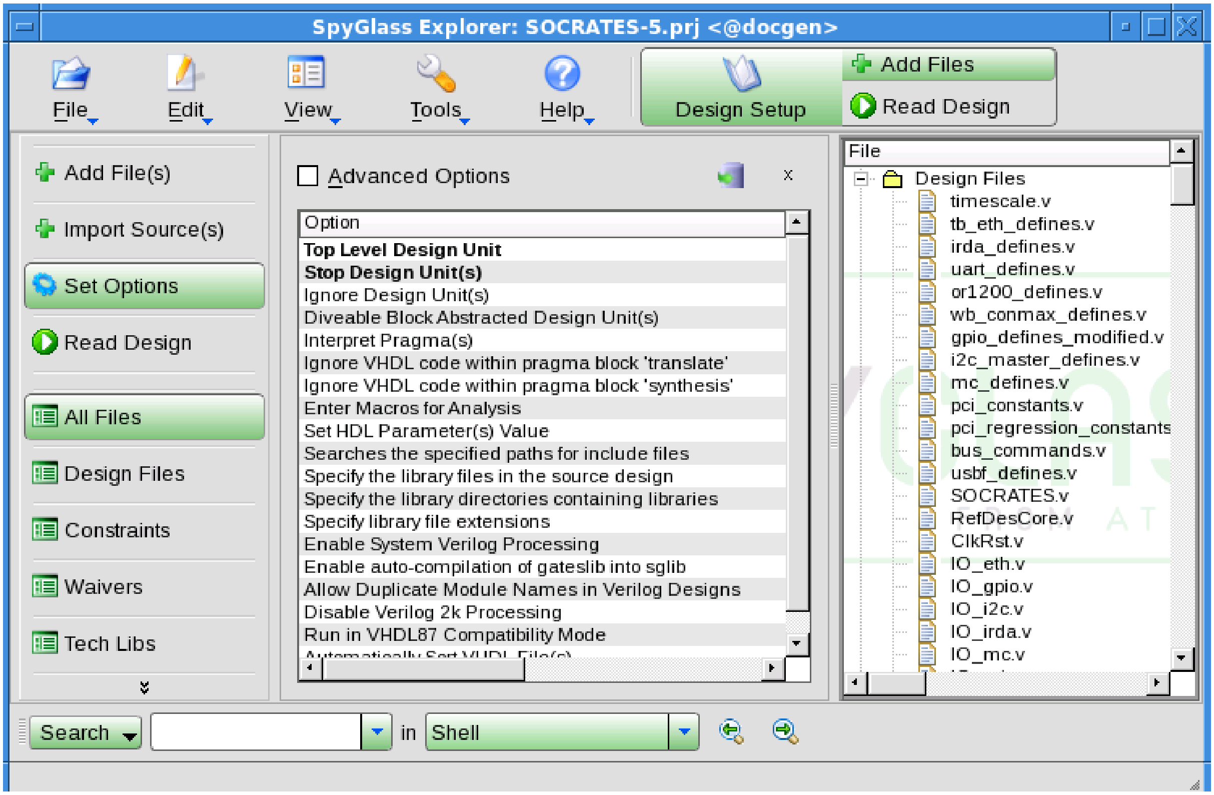
☐ Size

FIGURE 103. Configure Columns

2. Select the column name to be displayed.
3. Click OK to return to the File view pane.
4. Click Cancel to cancel the selection.

Viewing and Changing Design Read Options

Click the *Set Options* tab to specify the design-related options that affects the SpyGlass run. When you click this tab, the read options appear, as shown in the following figure:



□ Advanced Options

Option
Top Level Design Unit
Stop Design Unit(s)
Ignore Design Unit(s)
Diveable Block Abstracted Design Unit(s)
Interpret Pragma(s)
Ignore VHDL code within pragma block 'translate'
Ignore VHDL code within pragma block 'synthesis'
Enter Macros for Analysis
Set HDL Parameter(s) Value
Searches the specified paths for include files
Specify the library files in the source design
Specify the library directories containing libraries
Specify library file extensions
Enable System Verilog Processing
Enable auto-compilation of gateslib into sglib
Allow Duplicate Module Names in Verilog Designs
Disable Verilog 2k Processing
Run in VHDL87 Compatibility Mode

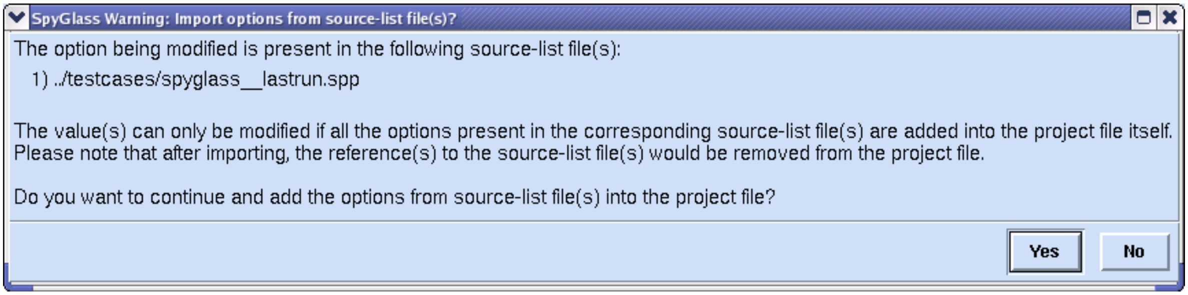
File
Design Files
timescale.v
tb_eth_defines.v
irda_defines.v
uart_defines.v
or1200_defines.v
wb_conmax_defines.v
gpio_defines_modified.v
i2c_master_defines.v
mc_defines.v
pci_constants.v
pci_regression_constants
bus_commands.v
usbf_defines.v
SOCRATES.v
RefDesCore.v
ClkRst.v
IO_eth.v
IO_gpio.v
IO_i2c.v
IO_irda.v
IO_mc.v

FIGURE 104. Design Read Options

By default, SpyGlass displays only the commonly-used read options. To view all the options, select the **Advanced Options** check box.

When you select an option, the help related to that option appears in the *Option Help* section.

NOTE: *The design read option that is specified using a source file (.spp or .f) appears as grayed out. You can edit this option by double-clicking the option. However, editing a design read option that is specified through the source file breaks its reference to the source file and is treated as a normal design option. A warning dialog is also displayed, as shown below.*



The option being modified is present in the following source-list file(s):

1) ../testcases/spyglass__lastrun.spp

The value(s) can only be modified if all the options present in the corresponding source-list file(s) are added into the project file itself. Please note that after importing, the reference(s) to the source-list file(s) would be removed from the project file.

Do you want to continue and add the options from source-list file(s) into the project file?

FIGURE 105. Warning Message

Using Verilog Constructs

By default, SpyGlass assumes that you are using the Verilog 1364-2001 constructs.

SpyGlass provides synthesis support for the following Verilog 2001 constructs:

TABLE 9 Verilog 2001 Constructs

Combined port and data type declarations	ANSI C style module declaration	Module port parameter list
ANSI C style UDP declarations	Variable initial value at declaration (Initial value is ignored)	ANSI C style task/ function declaration
Constant functions	Comma-separated sensitivity list	Combinational logic sensitivity lists
Implicit nets for continuous assignments	Disabling implicit net declarations	Variable vector part selects
Multidimensional arrays	Array bit and part selects	Signed-up, net and port declarations
Signed based integer numbers	Signed functions	Sign conversion system functions
Arithmetic shift operators	Assignment width extension past 32 bits	Power operators

TABLE 9 Verilog 2001 Constructs

Sized parameter constants	Explicit in-line parameter definition	Fixed local parameters
Enhanced conditional compilation	Source file and line compiler directive	Generate blocks

If you are using Verilog 1364-1995 constructs, specify the following command in the project file:

```
set_option disablev2k yes
```

Using SystemVerilog Constructs

If you want to analyze a design containing SystemVerilog language constructs, specify the following command in the project file:

```
set_option enableSV yes
```

NOTE: *You can find the details of supported SV constructs in the xls sheet, SpyGlass SystemVerilog Support, located in the* $SPYGLASS_HOME/doc *directory.*

Running Design Read

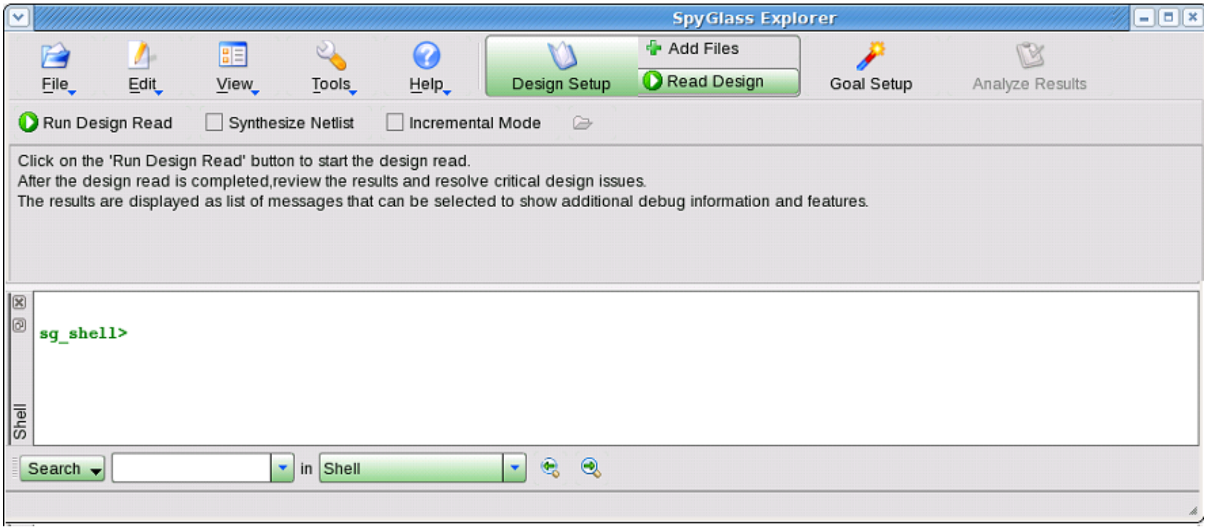
Running design read performs the first level of HDL analysis.

Running Design Read in GUI

To run the design read process in GUI, perform the following steps:

1. Click the *Read Design* tab.

   The following page appears:



Click on the 'Run Design Read' button to start the design read.
After the design read is completed,review the results and resolve critical design issues.
The results are displayed as list of messages that can be selected to show additional debug information and features.

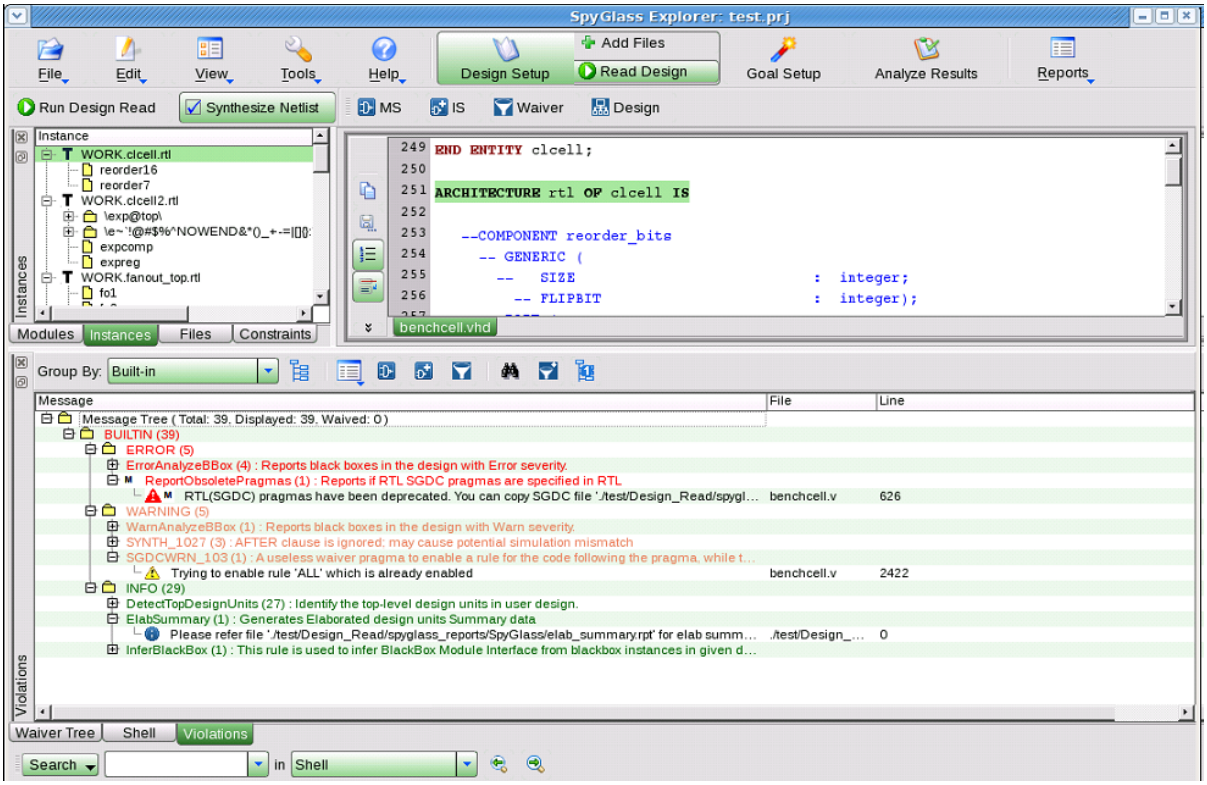
```
sg_shell>
```

FIGURE 106. Read Design Tab

2. If you want to perform analysis only up to elaboration, click the *Run Design Read* link.

   However, if you also want to perform synthesis, select the *Synthesize Netlist* option and then click the *Run Design Read* link. In this case, you can specify the type of synthesis you wish to perform by setting the *Design Read Synthesis Flavor* option to an appropriate value under the *Set Read Options* tab.

When the HDL analysis is complete, SpyGlass displays the message information in the *Message Window* as shown in the following figure:



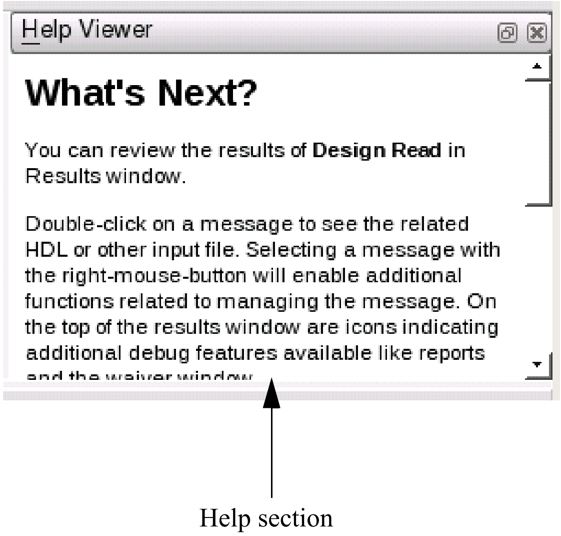
```
END ENTITY clcell;

ARCHITECTURE rtl OF clcell IS

   --COMPONENT reorder_bits
     -- GENERIC (
       --   SIZE                                :  integer;
         -- FLIPBIT                             :  integer);
```

```
Message                                                                                  File              Line
Message Tree ( Total: 39. Displayed: 39. Waived: 0 )
  BUILTIN (39)
    ERROR (5)
      ErrorAnalyzeBBox (4) : Reports black boxes in the design with Error severity.
      M ReportObsoletePragmas (1) : Reports if RTL SGDC pragmas are specified in RTL
        M RTL(SGDC) pragmas have been deprecated. You can copy SGDC file './test/Design_Read/spygl...  benchcell.v  626
    WARNING (5)
      WarnAnalyzeBBox (1) : Reports black boxes in the design with Warn severity.
      SYNTH_1027 (3) : AFTER clause is ignored; may cause potential simulation mismatch
      SGDCWRN_103 (1) : A useless waiver pragma to enable a rule for the code following the pragma, while t...
        Trying to enable rule 'ALL' which is already enabled                             benchcell.v       2422
    INFO (29)
      DetectTopDesignUnits (27) : Identify the top-level design units in user design.
      ElabSummary (1) : Generates Elaborated design units Summary data
        Please refer file './test/Design_Read/spyglass_reports/SpyGlass/elab_summary.rpt' for elab summ...  ./test/Design_...  0
      InferBlackBox (1) : This rule is used to infer BlackBox Module Interface from blackbox instances in given d...
```

FIGURE 107. Run Design Read-Messages Window

When SpyGlass analysis is complete, The *Help* section displays the additional steps that you need to perform to clean the design.

The following figure shows the Help section:



What's Next?

You can review the results of **Design Read** in Results window.

Double-click on a message to see the related HDL or other input file. Selecting a message with the right-mouse-button will enable additional functions related to managing the message. On the top of the results window are icons indicating additional debug features available like reports and the waiver window

FIGURE 108. Run Design Read-Help Section

Running Design Read in Batch

Once you have specified the required commands in the project file, run the design-read process by specifying the following command:

```
spyglass -project <file.prj> -batch -designread
```

Checks Performed During the Design Read Process

SpyGlass performs the following actions during design read:

- Checks whether the design is syntactically correct and complete, including checking for missing macros, inconsistent or undefined parameter/generic values, missing include files and so on.
- Reports basic data for the design, including the number of design units, number of source files, design hierarchy, and configuration parameter settings by using `top` and `stop` options.
- Reports all potential top-level design units in the design.

- Creates a black box model for it and reports a warning or error message, when SpyGlass cannot find a model for a design unit. For details on black box handling and resolving issues, refer to the *Working with Black Boxes* section.

NOTE: *Syntax errors must be resolved to continue. All other errors should be resolved to avoid problems during later analysis steps.*

For a more complete example of a project file, refer to the *Example of a Tcl-based Project File* section.

Viewing Messages after Running Design Read

After the design read process, SpyGlass reports various violation messages categorized by severity. The following table describes each type of severity:

Severity	Description
FATAL	Indicates a critical problem preventing further processing. You must fix such violations before proceeding to the next stage.
ERROR	Indicates a serious problem affecting design quality or analysis and should be resolved. It is highly recommended to resolve or reconcile all such messages.
WARNING	Indicates a problem that might be serious. It is highly recommended to review all such messages to check for possible problems.
INFO	Indicates an informational message about the design. Such messages can provide further design and analysis information that helps in the debugging of various errors and warnings.

While reviewing the output of design read, check the bottom of the run log, spyglass.log, for a summary of messages by severity. Each run contains the moresimple.rpt report that contains details of each message.

Identifying Common Syntax Errors and Issues

SpyGlass reports syntax errors in the following cases:

- If a macro is referenced in a design before (or without) being declared, SpyGlass reports the `STX_VE_533` syntax error.
- If you do not correctly specify a Verilog include directory, SpyGlass reports the `STX_VE_485` syntax error.
- If you specify an incorrect language standard

  By default, the Verilog standard is Verilog (IEEE Std 2000) and the VHDL standard is VHDL (IEEE Std 1993).

In addition, SpyGlass checks for various other issues described below:

- If you do not specify library files/directories, you get black boxes corresponding to the instances of the library cells in the design.
- If you specify library files without the `set_option v` command, the *DetectTopDesignUnits* rule may report multiple top modules.
- You must sort VHDL files according to the way they are referenced in the design HDL. If you do not sort them, SpyGlass may report errors or warnings. Unless you know that the order is correct, it is recommended you use the `set_option sortmethod <du | lexical>` project file command.

Tips for Debugging Syntax Errors

Design read primarily reports issues related to design HDL syntax. Following are the recommended steps to debug syntax errors:

1. Review the syntax error reported and resolve the error if possible.
2. Specify a valid language.
3. Specify a top-level module.
4. Check if synthesis pragmas are used adjacent to the code where the syntax error is reported.

If the issue is still not resolved, you can temporarily avoid that issue by stopping that design unit by specifying the following command:

```
set_option stop <design-unit>
```

Viewing Reports

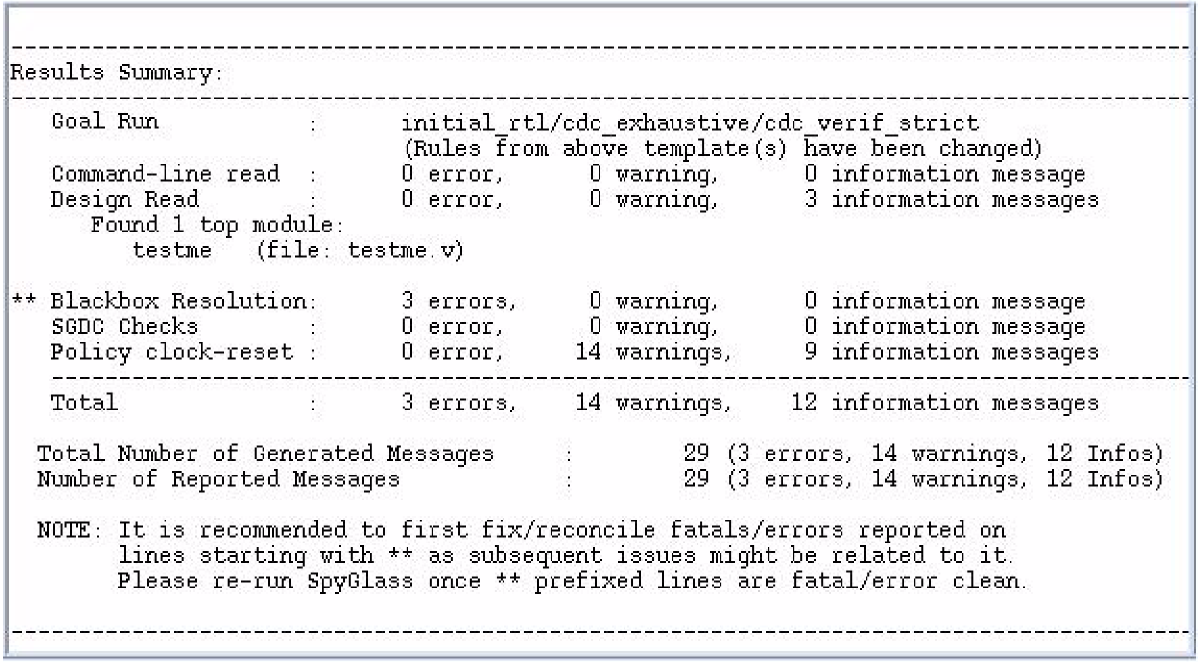
SpyGlass provides several predefined report formats to display violation messages or redirect reports to files for later review.

You can view these reports in any of the following ways:

- By selecting the required report from the *Tools -> Report* menu option.
- By selecting the required report from the *Reports* option in the *Results* pane.

Results Summary of SpyGlass Run

SpyGlass also generates a results summary in the *Session Log* pane at the end of the analysis. The following figure shows the results summary of a SpyGlass run:



```
------------------------------------------------------------------------------------------
Results Summary:
------------------------------------------------------------------------------------------
   Goal Run             :      initial_rtl/cdc_exhaustive/cdc_verif_strict
                               (Rules from above template(s) have been changed)
   Command-line read    :      0 error,      0 warning,      0 information message
   Design Read          :      0 error,      0 warning,      3 information messages
      Found 1 top module:
         testme   (file: testme.v)

** Blackbox Resolution:        3 errors,     0 warning,      0 information message
   SGDC Checks          :      0 error,      0 warning,      0 information message
   Policy clock-reset   :      0 error,     14 warnings,     9 information messages
   ---------------------------------------------------------------------------------------
   Total                :      3 errors,    14 warnings,    12 information messages

  Total Number of Generated Messages     :        29 (3 errors, 14 warnings, 12 Infos)
  Number of Reported Messages            :        29 (3 errors, 14 warnings, 12 Infos)

  NOTE: It is recommended to first fix/reconcile fatals/errors reported on
        lines starting with ** as subsequent issues might be related to it.
        Please re-run SpyGlass once ** prefixed lines are fatal/error clean.

------------------------------------------------------------------------------------------
```

FIGURE 109. Results Summary

The results summary displays the number of error messages found in the design.

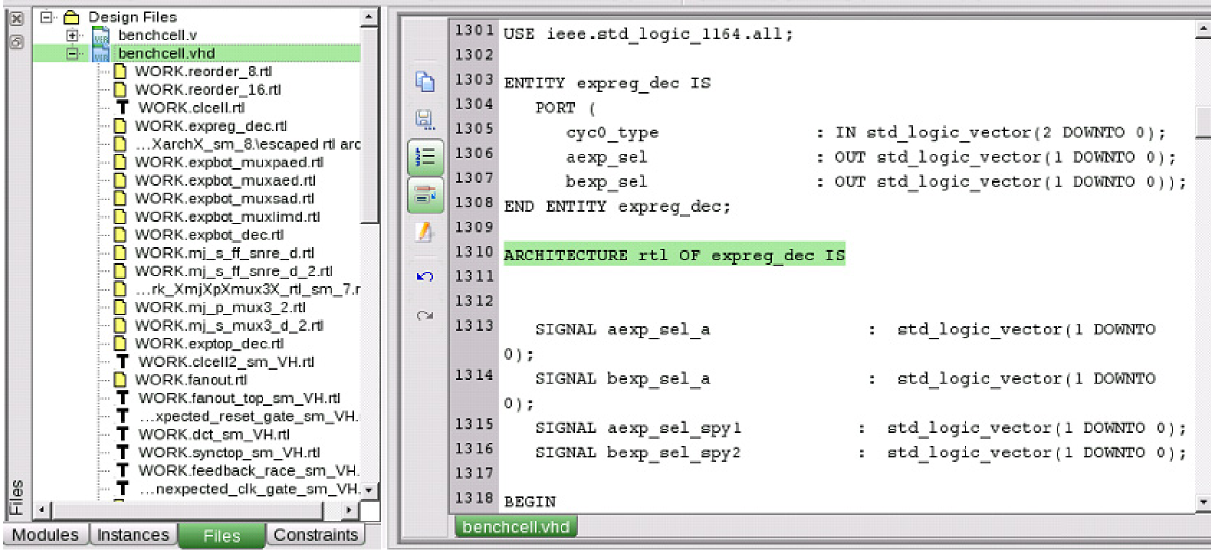
Certain error messages in the results summary are prefixed with two asterisks (* *). It is recommended that these errors are fixed before proceeding further.

To fix the issues in the design, load the source files in the *Source* section as explained in the *Viewing Source Files* section.

Viewing Source Files

When you double-click on a message on the *Msg Tree* page, the source file appears in the *source* section and the design unit instances appear on the *Instance Tab* and the *Module Tab*. In addition, the code corresponding to the message appears in the source section.

The following figure shows the contents of the source file, blocking1.v:



Design Files
- benchcell.v
- benchcell.vhd
  - WORK.reorder_8.rtl
  - WORK.reorder_16.rtl
  - WORK.clcell.rtl
  - WORK.expreg_dec.rtl
  - ...XarchX_sm_8.\escaped rtl arc
  - WORK.expbot_muxpaed.rtl
  - WORK.expbot_muxaed.rtl
  - WORK.expbot_muxsad.rtl
  - WORK.expbot_muxlimd.rtl
  - WORK.expbot_dec.rtl
  - WORK.mj_s_ff_snre_d.rtl
  - WORK.mj_s_ff_snre_d_2.rtl
  - ...rk_XmjXpXmux3X_rtl_sm_7.r
  - WORK.mj_p_mux3_2.rtl
  - WORK.mj_s_mux3_d_2.rtl
  - WORK.exptop_dec.rtl
  - WORK.clcell2_sm_VH.rtl
  - WORK.fanout.rtl
  - WORK.fanout_top_sm_VH.rtl
  - ...xpected_reset_gate_sm_VH.
  - WORK.dct_sm_VH.rtl
  - WORK.synctop_sm_VH.rtl
  - WORK.feedback_race_sm_VH.
  - ...nexpected_clk_gate_sm_VH.

```
USE ieee.std_logic_1164.all;

ENTITY expreg_dec IS
   PORT (
      cyc0_type                   : IN std_logic_vector(2 DOWNTO 0);
      aexp_sel                    : OUT std_logic_vector(1 DOWNTO 0);
      bexp_sel                    : OUT std_logic_vector(1 DOWNTO 0));
END ENTITY expreg_dec;

ARCHITECTURE rtl OF expreg_dec IS


   SIGNAL aexp_sel_a                 :  std_logic_vector(1 DOWNTO
0);
   SIGNAL bexp_sel_a                 :  std_logic_vector(1 DOWNTO
0);
   SIGNAL aexp_sel_spy1             :  std_logic_vector(1 DOWNTO 0);
   SIGNAL bexp_sel_spy2             :  std_logic_vector(1 DOWNTO 0);

BEGIN
```

FIGURE 110. View Source Files

In the above page, double-click on the required file (source file or include file) from the *File View* page to load the contents of that file in the *Source* section.

You can edit a file by clicking the *Edit File* link on the navigation bar located at the left of the Source window.

After you have cleaned the design from any fatal errors, re-run design read.

Searching Instances

To search an instance in the *Instances View* page, perform the following steps:

1. In the *Search* section parallel to the menu bar, select the *Instance View* option from the pull-down list, as shown in the following figure:

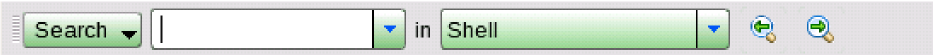


FIGURE 111. Search

2. Click Search to specify the required search option(s). For details, refer to the *Specifying Search Options* topic.
3. Specify the search text in the *Search* textbox.
4. Click the *Go* link.

After performing the above steps, the first module/instance whose name contains the search string appears in the *Instance View* page. Continue clicking the *Go* link to find more module/instances names containing the search string.

Specifying Search Options

When you click Search, various search options appear, as shown in the following figure:

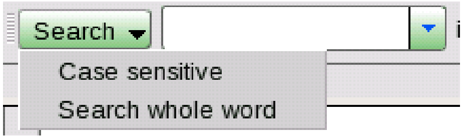


FIGURE 112. Search Options

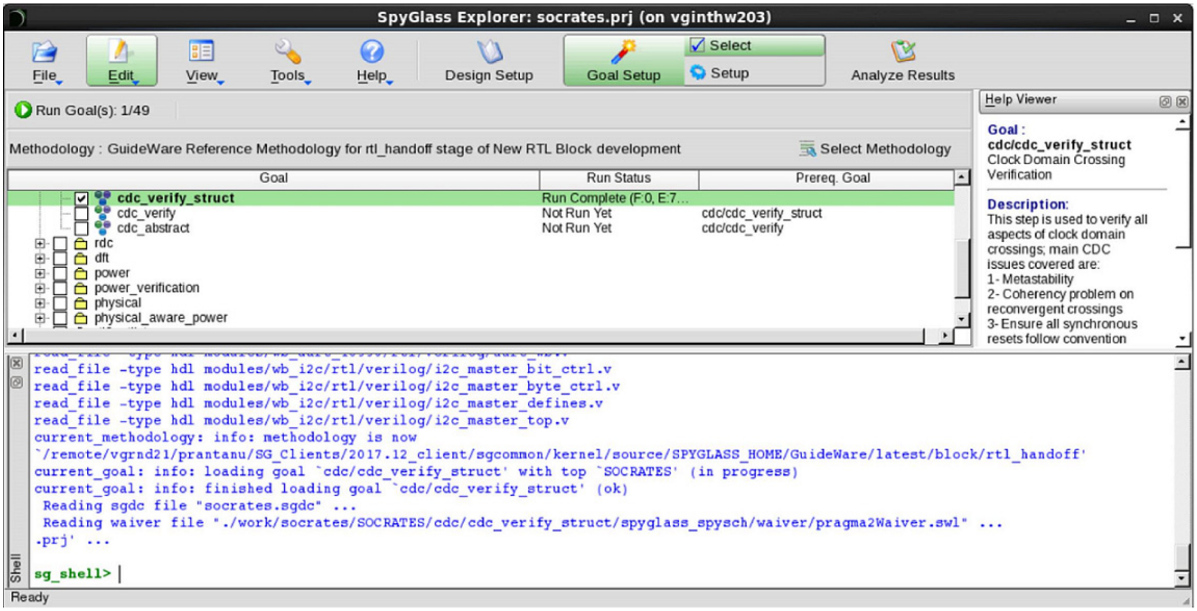
You can select appropriate option(s) from the above list to qualify your search.

NOTE: *When you select the Search in Hierarchical Path option, only Search Backwards option is enabled.*

Stage 2: Selecting a Goal (Goal Setup & Run)

After completing the *Design Setup* stage successfully, click the *Goal Setup & Run* tab to proceed to the next stage.

When you select the *Goal Setup & Run* tab, the following page appears:



Methodology : GuideWare Reference Methodology for rtl_handoff stage of New RTL Block development

Goal	Run Status	Prereq. Goal
☑ cdc_verify_struct	Run Complete (F:0, E:7...	
□ cdc_verify	Not Run Yet	cdc/cdc_verify_struct
□ cdc_abstract	Not Run Yet	cdc/cdc_verify
□ rdc		
□ dft		
□ power		
□ power_verification		
□ physical		
□ physical_aware_power		

Goal :
cdc/cdc_verify_struct
Clock Domain Crossing Verification

Description:
This step is used to verify all aspects of clock domain crossings; main CDC issues covered are:
1- Metastability
2- Coherency problem on reconvergent crossings
3- Ensure all synchronous resets follow convention

```
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_bit_ctrl.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_byte_ctrl.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_defines.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_top.v
current_methodology: info: methodology is now
`/remote/vgrnd21/prantanu/SG_Clients/2017.12_client/sgcommon/kernel/source/SPYGLASS_HOME/GuideWare/latest/block/rtl_handoff'
current_goal: info: loading goal `cdc/cdc_verify_struct' with top `SOCRATES' (in progress)
current_goal: info: finished loading goal `cdc/cdc_verify_struct' (ok)
 Reading sgdc file "socrates.sgdc" ...
 Reading waiver file "./work/socrates/SOCRATES/cdc/cdc_verify_struct/spyglass_spysch/waiver/pragma2Waiver.swl" ...
.prj' ...

sg_shell>
```

FIGURE 113. Goal Setup and Run

NOTE: *Specify a top-level module in Stage 1: Setting up the Design (Design Setup) before proceeding to the Goal Setup & Run stage.*

In this tab, you can perform various actions, such as selecting goals and specifying parameters and constraints for the selected goals. Finally, run the selected goals before proceeding to the next stage, *Stage 3: Analyzing a Design (Analyze Results)*.

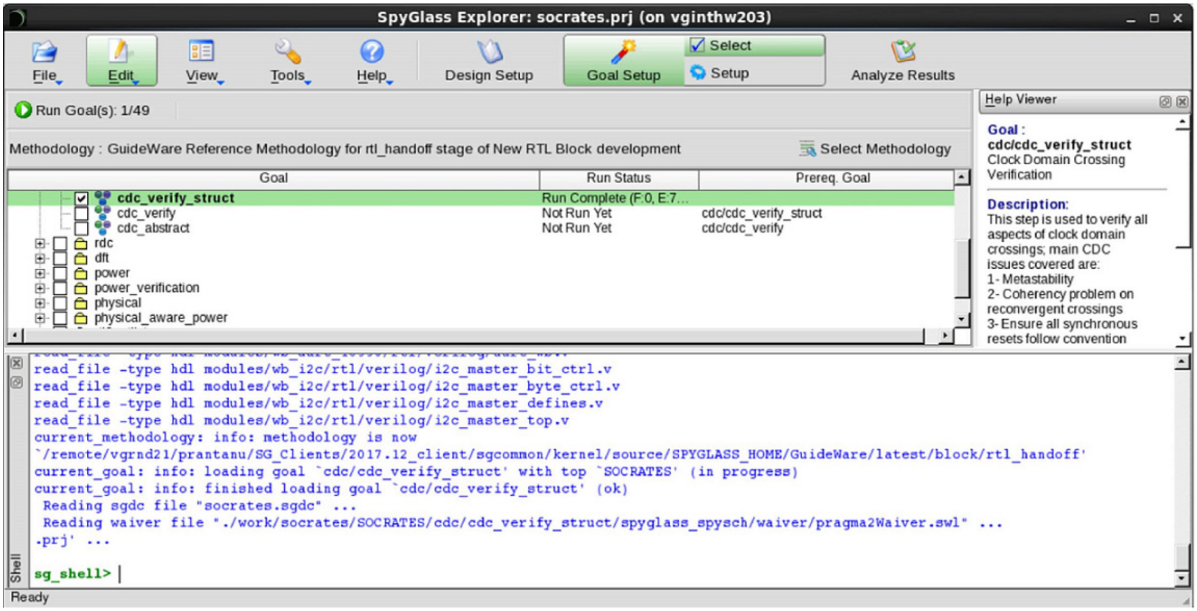
This stage is divided into the following steps:

1. *Selecting a Goal*
2. *Modifying a Goal*

Selecting a Goal

To select a goal, click the *Select Goal* tab.

Under this tab, various goals appear in the order based on the selected methodology, as shown in the following figure:



Methodology : GuideWare Reference Methodology for rtl_handoff stage of New RTL Block development

Goal	Run Status	Prereq. Goal
☑ cdc_verify_struct	Run Complete (F:0, E:7...	
□ cdc_verify	Not Run Yet	cdc/cdc_verify_struct
□ cdc_abstract	Not Run Yet	cdc/cdc_verify
⊞ □ rdc		
⊞ □ dft		
⊞ □ power		
⊞ □ power_verification		
⊞ □ physical		
⊞ □ physical_aware_power		

Goal :
cdc/cdc_verify_struct
Clock Domain Crossing Verification

Description:
This step is used to verify all aspects of clock domain crossings; main CDC issues covered are:
1- Metastability
2- Coherency problem on reconvergent crossings
3- Ensure all synchronous resets follow convention

```
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_bit_ctrl.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_byte_ctrl.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_defines.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_top.v
current_methodology: info: methodology is now
`/remote/vgrnd21/prantanu/SG_Clients/2017.12_client/sgcommon/kernel/source/SPYGLASS_HOME/GuideWare/latest/block/rtl_handoff'
current_goal: info: loading goal `cdc/cdc_verify_struct' with top `SOCRATES' (in progress)
current_goal: info: finished loading goal `cdc/cdc_verify_struct' (ok)
 Reading sgdc file "socrates.sgdc" ...
 Reading waiver file "./work/socrates/SOCRATES/cdc/cdc_verify_struct/spyglass_spysch/waiver/pragma2Waiver.swl" ...
.prj' ...

sg_shell> |
```

FIGURE 114. Goal Selection

NOTE: *The default methodology is available at $SPYGLASS_HOME/GuideWare/latest directory.*

NOTE: *When you click on a sub-methodology/goal, the help related to that sub-methodology/goal appears in the* Help *window.*

The *Select Goal* tab also displays additional information in various fields, as described in the following table:

TABLE 10

Field Name	Description
Setup Status	Displays whether you need to set up the parameters and constraints for the selected goal. If the setup status of a goal is *Setup Optional*, you may or may not set the parameters for that goal. However, if the setup status of a goal is *Setup Recommended*, this means that the goal requires some additional steps. SpyGlass enables you to perform these steps through a setup wizard (see *Modifying a Goal* section for details).
Run Status	Displays whether the selected goal was run. When you click the *Run Selected Goal(s)* link, the status of the selected goal changes to *Running*. However, when the run is completed, then the run status changes to *Completed*. In addition, the run status displays the total violation count based on severity. The Run Status for the sub-methodology displays the goals selected for the methodology **NOTE:** *The run status also appears on the status bar*.
Prereq. Goals	Displays the goal that should run before the selected goal runs.

Under the *Select Goal* tab, you can perform the following actions:

- Select goal(s) from the available list.
- Select the *All* option to select all the goals of the current methodology.
- Select the *None* option to deselect all the goals of the current methodology.

After selecting the required goal(s), click the *Run Selected Goal(s)* option to run the selected goal(s).

SpyGlass Explorer displays the *Sequential Mode* dialog, if you have selected multiple goals.

When you select goals for analysis and place the cursor on the *Run Selected Goal(s)* link, a tool-tip appears displaying the total number of selected goals, name of goals, and the methodology/sub-methodology from which you selected the goals.

If multiple goals of different products are run together, the behavior of the run depends on their respective rule's synthesis modes. For more information, see to the *design-read Synthesis Flavor*.

Selecting a Goal in Batch Mode

You can run a goal in batch mode by specifying the following command:

```
spyglass -batch -project <project_file> -goal <goal_name>
```

You can also run multiple goals by specifying the following command:

```
spyglass -batch -project <project_file>
-goal <goal_name1>, <goal_name2>
```

Modifying a Goal

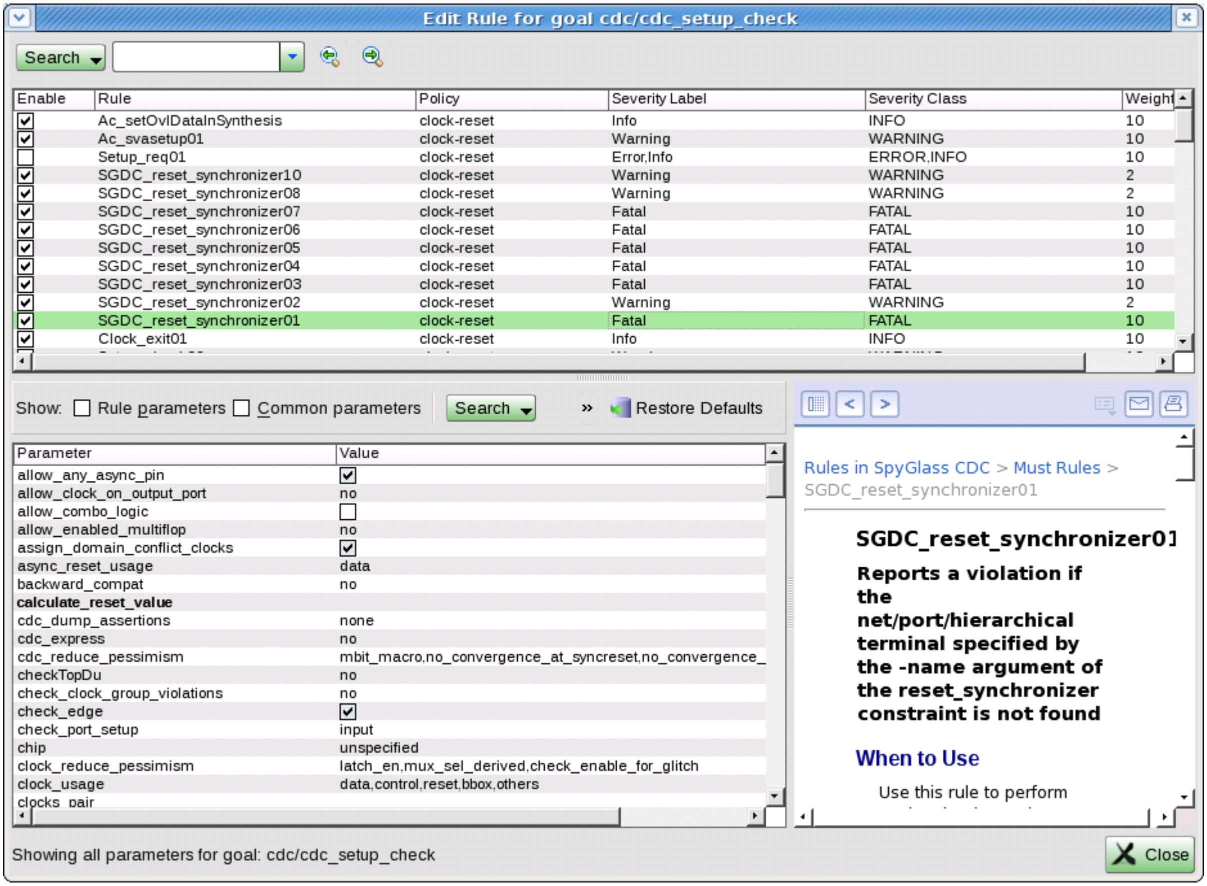
You can modify a goal by:

- *Enabling or Disabling Rules of a Goal*
- *Adding Rules in a Goal*
- *Editing Parameter Values of a Goal*

Enabling or Disabling Rules of a Goal

To enable or disable rules of a goal, right-click on a goal appearing under the *Select Goal* tab, and select the *Edit Rules and Parameters* option from the shortcut menu. This displays the *Edit Rule for goal <goal_name>* dialog containing all rules (enabled and disabled) for the selected goal.

The following figure shows the *Edit Rule for goal <goal_name>* dialog for the `connectivity` goal:



Edit Rule for goal cdc/cdc_setup_check

Enable	Rule	Policy	Severity Label	Severity Class	Weight
☑	Ac_setOvlDataInSynthesis	clock-reset	Info	INFO	10
☑	Ac_svasetup01	clock-reset	Warning	WARNING	10
□	Setup_req01	clock-reset	Error,Info	ERROR,INFO	10
☑	SGDC_reset_synchronizer10	clock-reset	Warning	WARNING	2
☑	SGDC_reset_synchronizer08	clock-reset	Warning	WARNING	2
☑	SGDC_reset_synchronizer07	clock-reset	Fatal	FATAL	10
☑	SGDC_reset_synchronizer06	clock-reset	Fatal	FATAL	10
☑	SGDC_reset_synchronizer05	clock-reset	Fatal	FATAL	10
☑	SGDC_reset_synchronizer04	clock-reset	Fatal	FATAL	10
☑	SGDC_reset_synchronizer03	clock-reset	Fatal	FATAL	10
☑	SGDC_reset_synchronizer02	clock-reset	Warning	WARNING	2
☑	SGDC_reset_synchronizer01	clock-reset	Fatal	FATAL	10
☑	Clock_exit01	clock-reset	Info	INFO	10

Show: □ Rule parameters □ Common parameters

Parameter	Value
allow_any_async_pin	☑
allow_clock_on_output_port	no
allow_combo_logic	□
allow_enabled_multiflop	no
assign_domain_conflict_clocks	☑
async_reset_usage	data
backward_compat	no
calculate_reset_value	
cdc_dump_assertions	none
cdc_express	no
cdc_reduce_pessimism	mbit_macro,no_convergence_at_syncreset,no_convergence_
checkTopDu	no
check_clock_group_violations	no
check_edge	☑
check_port_setup	input
chip	unspecified
clock_reduce_pessimism	latch_en,mux_sel_derived,check_enable_for_glitch
clock_usage	data,control,reset,bbox,others
clocks_pair	

Rules in SpyGlass CDC > Must Rules >
SGDC_reset_synchronizer01

SGDC_reset_synchronizer01

Reports a violation if the net/port/hierarchical terminal specified by the -name argument of the reset_synchronizer constraint is not found

When to Use

Use this rule to perform

Showing all parameters for goal: cdc/cdc_setup_check

FIGURE 115. Edit Rules for Connectivity

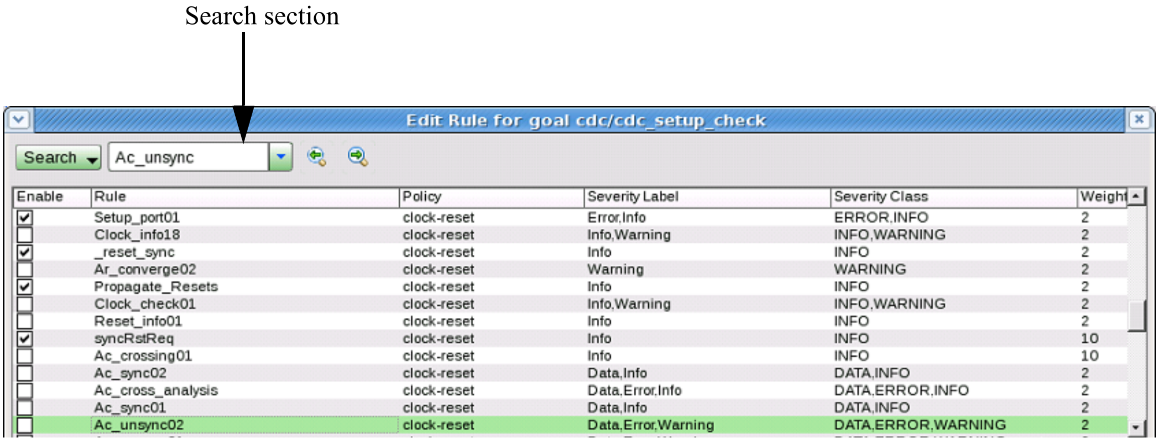
In the above dialog, you can:

- Enable a goal by clicking ✗ adjacent to a goal name. This enables the goal and ✓ appears adjacent to the goal name.
- Disable a goal by clicking ✓ adjacent to a goal name. This disables the goal and ✗ appears adjacent to the goal name.

Adding Rules in a Goal

To add rules in a goal, right-click on the goal and select the *Add Rule(s)* option from the shortcut menu. This displays the *Edit Rules* dialog for the goal.

The following figure shows the *Edit Rules* dialog for the `connectivity` goal:



Ac_unsync

Enable	Rule	Policy	Severity Label	Severity Class	Weight
☑	Setup_port01	clock-reset	Error,Info	ERROR,INFO	2
☐	Clock_info18	clock-reset	Info,Warning	INFO,WARNING	2
☑	_reset_sync	clock-reset	Info	INFO	2
☐	Ar_converge02	clock-reset	Warning	WARNING	2
☑	Propagate_Resets	clock-reset	Info	INFO	2
☐	Clock_check01	clock-reset	Info,Warning	INFO,WARNING	2
☐	Reset_info01	clock-reset	Info	INFO	2
☑	syncRstReq	clock-reset	Info	INFO	10
☐	Ac_crossing01	clock-reset	Info	INFO	10
☐	Ac_sync02	clock-reset	Data,Info	DATA,INFO	2
☐	Ac_cross_analysis	clock-reset	Data,Error,Info	DATA,ERROR,INFO	2
☐	Ac_sync01	clock-reset	Data,Info	DATA,INFO	2
☐	Ac_unsync02	clock-reset	Data,Error,Warning	DATA,ERROR,WARNING	2

FIGURE 116. Edit Rules for Connectivity-Search

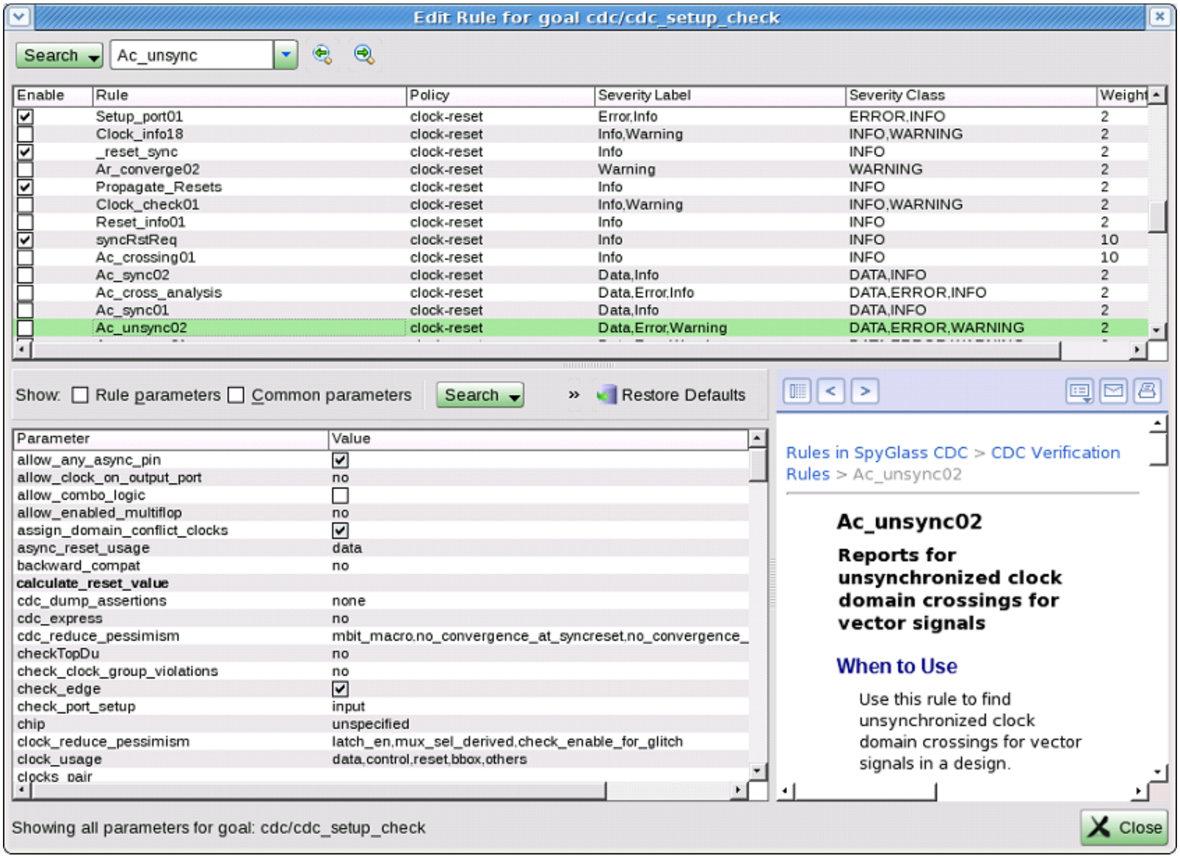
In the above dialog, the right-most section displays rules for the selected goal and the left-most section is the *Search* section used for searching rules to be added in a goal.

To add rules in a goal, perform the following steps:

1. Search for a rule that you want to add in a goal in the *Search* section.

   This step is similar to searching rules in the *MCS* window. For details, see *Searching Rules*.

Depending upon the specified search criteria, rules appear in the *Search* section, as shown in the following figure:



Enable	Rule	Policy	Severity Label	Severity Class	Weight
☑	Setup_port01	clock-reset	Error,Info	ERROR,INFO	2
□	Clock_info18	clock-reset	Info,Warning	INFO,WARNING	2
☑	_reset_sync	clock-reset	Info	INFO	2
□	Ar_converge02	clock-reset	Warning	WARNING	2
☑	Propagate_Resets	clock-reset	Info	INFO	2
□	Clock_check01	clock-reset	Info,Warning	INFO,WARNING	2
□	Reset_info01	clock-reset	Info	INFO	2
☑	syncRstReq	clock-reset	Info	INFO	10
□	Ac_crossing01	clock-reset	Info	INFO	10
□	Ac_sync02	clock-reset	Data,Info	DATA,INFO	2
□	Ac_cross_analysis	clock-reset	Data,Error,Info	DATA,ERROR,INFO	2
□	Ac_sync01	clock-reset	Data,Info	DATA,INFO	2
□	Ac_unsync02	clock-reset	Data,Error,Warning	DATA,ERROR,WARNING	2

Parameter	Value
allow_any_async_pin	☑
allow_clock_on_output_port	no
allow_combo_logic	□
allow_enabled_multiflop	no
assign_domain_conflict_clocks	☑
async_reset_usage	data
backward_compat	no
calculate_reset_value	
cdc_dump_assertions	none
cdc_express	no
cdc_reduce_pessimism	mbit_macro,no_convergence_at_syncreset,no_convergence_
checkTopDu	no
check_clock_group_violations	no
check_edge	☑
check_port_setup	input
chip	unspecified
clock_reduce_pessimism	latch_en,mux_sel_derived,check_enable_for_glitch
clock_usage	data,control,reset,bbox,others
clocks_pair	

Rules in SpyGlass CDC > CDC Verification Rules > Ac_unsync02

Ac_unsync02

Reports for unsynchronized clock domain crossings for vector signals

When to Use

Use this rule to find unsynchronized clock domain crossings for vector signals in a design.

Showing all parameters for goal: cdc/cdc_setup_check

FIGURE 117. Search Results

2. Select the required rules from the *Search* section.

   You can select multiple rules by pressing the <*Ctrl*> key and clicking the required rules.

3. Right-click on the selected rules, and select the *Add Rule(s) to Goal* option from the shortcut menu.

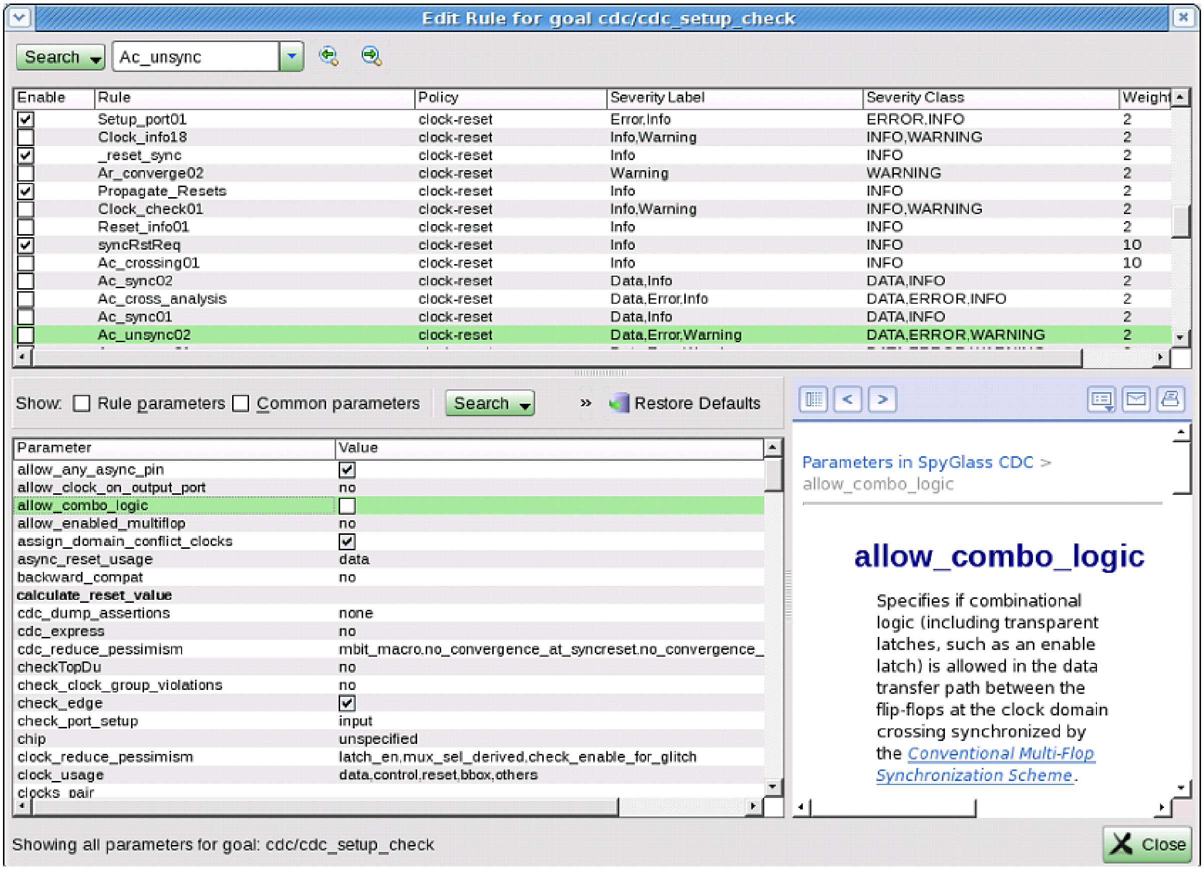
   Alternatively, you can select the *Add Rule(s) to Goal* option in the *Search* section.

After performing the above steps, the specified rules appear in the selected goal.

Editing Parameter Values of a Goal

To edit parameter values for various rules of a goal, right-click on that goal and select the *Edit Parameter(s)* option from the shortcut menu. This displays the *Edit Parameter(s)* dialog for that goal.

The following figure shows the *Edit Parameter(s)* dialog for the `connectivity` goal:



Enable	Rule	Policy	Severity Label	Severity Class	Weight
☑	Setup_port01	clock-reset	Error,Info	ERROR,INFO	2
☐	Clock_info18	clock-reset	Info,Warning	INFO,WARNING	2
☑	_reset_sync	clock-reset	Info	INFO	2
☐	Ar_converge02	clock-reset	Warning	WARNING	2
☑	Propagate_Resets	clock-reset	Info	INFO	2
☐	Clock_check01	clock-reset	Info,Warning	INFO,WARNING	2
☐	Reset_info01	clock-reset	Info	INFO	2
☑	syncRstReq	clock-reset	Info	INFO	10
☐	Ac_crossing01	clock-reset	Info	INFO	10
☐	Ac_sync02	clock-reset	Data,Info	DATA,INFO	2
☐	Ac_cross_analysis	clock-reset	Data,Error,Info	DATA,ERROR,INFO	2
☐	Ac_sync01	clock-reset	Data,Info	DATA,INFO	2
☐	Ac_unsync02	clock-reset	Data,Error,Warning	DATA,ERROR,WARNING	2

Parameter	Value
allow_any_async_pin	☑
allow_clock_on_output_port	no
allow_combo_logic	☐
allow_enabled_multiflop	no
assign_domain_conflict_clocks	☑
async_reset_usage	data
backward_compat	no
calculate_reset_value	
cdc_dump_assertions	none
cdc_express	no
cdc_reduce_pessimism	mbit_macro,no_convergence_at_syncreset,no_convergence_
checkTopDu	no
check_clock_group_violations	no
check_edge	☑
check_port_setup	input
chip	unspecified
clock_reduce_pessimism	latch_en,mux_sel_derived,check_enable_for_glitch
clock_usage	data,control,reset,bbox,others
clocks_pair	

Parameters in SpyGlass CDC > allow_combo_logic

allow_combo_logic

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

Showing all parameters for goal: cdc/cdc_setup_check

FIGURE 118. Edit Parameters for Connectivity

The above dialog displays parameters and their values for the current goal.

Viewing Different Types of Parameters List

The above dialog contains the *Show* drop-down list from which you can select any of the following options:

Option name	Purpose
Common Parameters	(Default) Select this option to view commonly used parameters of the selected goal.
Other Parameters	Select this option to view parameters that are not commonly used for the selected goal.
All Parameters	Select this option to view all parameters (common and un-common) of the selected goal.

In addition to the above options, the *Show* drop-down list also contains rule names that are enabled for the selected goal. You can select the required rule to view all parameters applicable for that rule.

Editing Parameter Values

Modify a value for a parameter in the *Value* column adjacent to a parameter.

If you want to assign all the parameters to their respective default values, click the *Restore Defaults* option.

Running Custom Goals

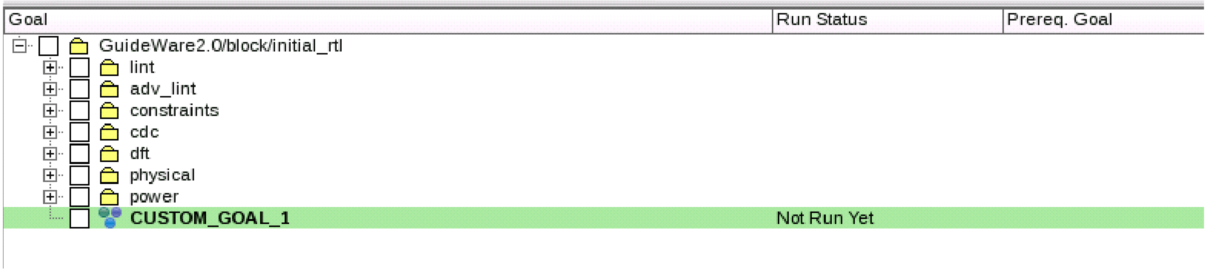
Custom goals are user-defined goals that you can create by using the `define_goal` command in a project file.

Custom goals appear under the *Select Goal* tab parallel to the existing goals list.

For example, consider that you specify the following command in a project file:

```
define_goal CUSTOM_GOAL_1 -policy { lint } {
  set_parameter abc def
}
```

When you load this project file, the `CUSTOM_GOAL_1` goal appears under the *Select Goal* tab as shown in the following figure:



Goal	Run Status	Prereq. Goal
□ GuideWare2.0/block/initial_rtl		
□ lint		
□ adv_lint		
□ constraints		
□ cdc		
□ dft		
□ physical		
□ power		
□ CUSTOM_GOAL_1	Not Run Yet	

FIGURE 119. Select Goal

Running Goals in Parallel

You can run multiple goals in parallel on different machines.

To enable parallel execution of goals, perform the following actions:

- Set the value of `ENABLE_PARALLEL_RUN` key to `goal` in the .spyglass.setup file:

  ```
  ENABLE_PARALLEL_RUN = goal
  ```

  By default, this key is set to `none` and parallel execution of goals is disabled.

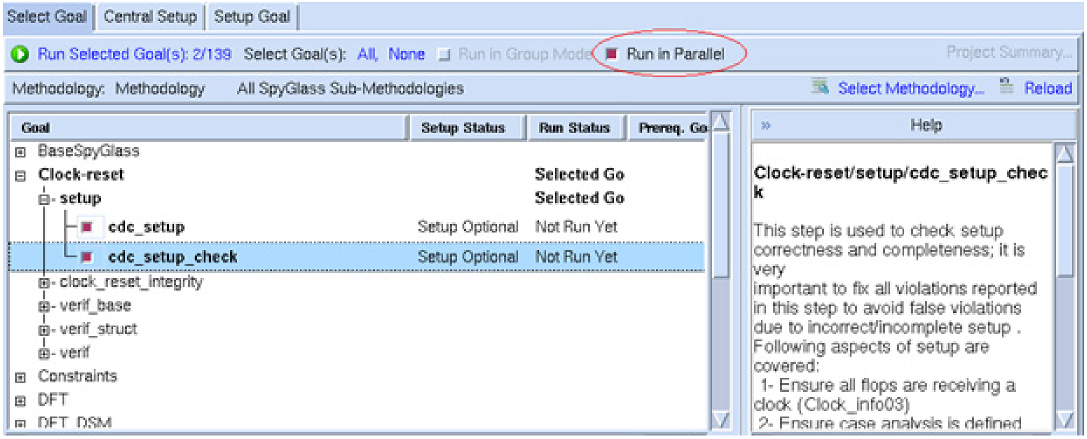
- Specify a host configuration file by using the `-host_config_file` command-line option. For details on this option, see *SpyGlass Explorer-Specific Options*.

  You can specify this file by using the `HOST_CONFIG_FILE` key in .spyglass.setup file:

  ```
  HOST_CONFIG_FILE = <path-of-config-file>
  ```

  Alternatively, you can specify or edit this file in the *Specify host_config_file* field in the *Miscellaneous* page of the *Preferences* dialog.

When you set the `ENABLE_PARALLEL_RUN` key to `goal`, the *Run in Parallel* option appears under the *Select Goal* tab, as shown in the following figure:



Goal	Setup Status	Run Status	Prereq. Go
⊞ BaseSpyGlass			
⊟ Clock-reset		Selected Go	
⊟- setup		Selected Go	
■ cdc_setup	Setup Optional	Not Run Yet	
■ cdc_setup_check	Setup Optional	Not Run Yet	
⊞- clock_reset_integrity			
⊞- verif_base			
⊞- verif_struct			
⊞- verif			
⊞ Constraints			
⊞ DFT			
⊞ DFT_DSM			

Clock-reset/setup/cdc_setup_check

This step is used to check setup correctness and completeness; it is very important to fix all violations reported in this step to avoid false violations due to incorrect/incomplete setup . Following aspects of setup are covered:

1- Ensure all flops are receiving a clock (Clock_info03)

2- Ensure case analysis is defined

FIGURE 120. Run Selected Goal(s)

To run the selected goals in parallel, select the *Run in Parallel* option and click the *Run Selected Goal(s)* option.

Please note the following points:

- If the login type in the specified host configuration file is `lsf`, do not specify the `-I` option of `bsub` command in the `LSF_CMD` keyword.
- Parallel goal run is not supported in the DEF mode.

When you run goals in parallel, following project files are created:

- **<project_name>_modified.prj**: This is the modified project file, using which you can run the goals in parallel. This file contains all the user settings and data required for all the goals.
- **<project_name>_temp.prj**: This file contains the default settings. Initially, the default settings are saved and goals are run with the new (modified) project <project_name>_modified.prj. Once the parallel run is complete, the settings specified in the *<project_name>_temp.prj* file are used for the subsequent runs.

Peak Memory Reduction During Parallel Goal Run

Normally, during parallel goal run, design read happens during first goal run. You can use the `parallel_run_options` option to specify the mode in which parallel run design read is performed. Currently, the `separate_design_read` value is supported for this option.

Consider the following example:

```
sg_shell> set_option parallel_run_options
separate_design_read
```

When you set the value of this option to `separate_design_read`, a separate design read is performed and NOM DB is saved, during design read. This can help in reducing the PEAK memory requirement.

Goals are then run in parallel in NOM restore mode.

NOTE: *Currently, separate design read is done for goals that use the classic view of synthesis and is not available for the goals that need techmapped or optimized synthesis.*

Running Goals in Parallel in Batch

To run goals in parallel, perform the following actions:

- Set the `ENABLE_PARALLEL_RUN` configuration key to `goal` in .spyglass.setup.
- Specify the `-parallel_run` command-line option and a list of goals to run in parallel in the batch mode.

  In addition, specify a host configuration file in batch by using the `-host_config_file` command-line option if the file is not specified by using the `HOST_CONFIG_FILE` key in .spyglass.setup.

Please note the following points:

- Parallel goal run in batch is enabled only if you have specified the `-parallel_run` command-line option as well as set the `ENABLE_PARALLEL_RUN` key to `goal` in .spyglass.setup.
- If you specify the `-host_config_file` option but do not specify the `-parallel_run` option, the `-host_config_file` option is ignored and parallel goal run is not enabled. In addition, a warning appears in a tool-tip.

- SpyGlass run aborts if you do not specify a host configuration file either by using the `-host_config_file` option in batch or setting the `HOST_CONFIG_FILE` key in .spyglass.setup.
- SpyGlass ignores the `HOST_CONFIG_FILE` key if the `ENABLE_PARALLEL_RUN` key is set to none.

Format of Configuration File Containing Parallel Goal Run Settings

The format of the configuration file containing parallel goal run settings in is same as that of the configuration file used to perform distributed runs of advanced SpyGlass CDC solution rules on several machines.

This configuration file is an ASCII text file that contains specific lines for different methods, as discussed below:

- For LSF method

  The LSF method contains the following lines:

  ```
  LOGIN_TYPE: lsf
  MAX_PROCESSES: <num>
  LSF_CMD: <bsub-command>
  ```

  Syntax Rules

  ❐ You must add a space after `LOGIN_TYPE:` and `MAX_PROCESSES`

  ❐ The # and // symbols are supported as comments in the config file

NOTE: *Currently, only the* `qsub` *command is supported for LSF protocol. To use the* `qsub` *command, see Using the qsub Command During Parallel Goal Run through LSF.*

Following table contains details of various arguments of the above lines:

Argument	Description
<num>	Specifies the maximum number of processes to be spawned. This is a mandatory argument.
<bsub-command>	Specifies the LSF invocation command. By default, the value of this argument is bsub.

Following is an example of the LSF method:

```
LOGIN_TYPE: lsf
```

```
//LOGIN_TYPE: rsh
MAX_PROCESSES: 5
LSF_CMD: bsub -q "normal | priority"
//LSF_CMD: bsub -q bsub
```

In the above example, the `-q` option is used to specify the queue as normal or priority.

- For RSH and SSH methods

  The RSH and SSH methods contain the following lines:

```
LOGIN_TYPE: rsh | ssh
MAX_PROCESSES: <num>
MACHINES:
<machine1-name>[:<num-processes>]
<machine2-name>[:<num-processes>]
...
```

Syntax Rules

❒ You must add a space after `LOGIN_TYPE:` and `MAX_PROCESSES`

❒ The # and // symbols are supported as comments in the config file.

The following table contains details of various arguments of the above lines:

Argument	Description
<num>	Specifies the maximum number of processes to be spawned. This is a mandatory argument.
<machine1-name>, <machine2-name>	Specifies machine names If you want to run goals on the current machine itself, do not specify the MACHINES keyword and machine names. In this case, SpyGlass ignores the LOGIN_TYPE keyword.
<num-processes>	Specifies the number of processes to be spawned on the specified machine By default, the value of this argument is 1.

NOTE: *If the login type is SSH, the SSH_ID_FILE file contains login details for the SSH login so that login does not require user name and password. Check the man page for the ssh login on how to generate this file.*

Following is an example of the SSH method:

```
LOGIN_TYPE: ssh
#LOGIN_TYPE: rsh
MAX_PROCESSES: 5
MACHINES:
engr1: 1
#engr2: 1
ae3: 1
condor1: 4
```

Using the qsub Command During Parallel Goal Run through LSF

The qsub command is not inherently supported while running goals in parallel through LSF protocol. You can still, however, use qsub by writing a wrapper script (say qsub_wrapper) over qsub and specifying it as an LSF command in the parallel run configuration file. This wrapper script would dissect the inputs sent to it by SpyGlass and create a command line suited to qsub.

Parallel run configuration file appears as the following:

```
LOGIN_TYPE: lsf
MAX_PROCESSES: <num>
LSF_CMD: qsub_wrapper
```

Ensure that the directory containing qsub_wrapper script is present in your path variable.

The qsub_wrapper script appears as follows:

```
#!/bin/sh
script=/tmp/my_script$$
outputfile=
spyglass_cmd=
while [ $# -gt 0 ];
do
    case $1 in
    -o)
        shift;
        outputfile=$1
        ;;
    -K) ;;
    *)
        if [ "X${spyglass_cmd}" != "X" ]; then
            spyglass_cmd="${spyglass_cmd} $1"
        else
            spyglass_cmd=$1
        fi
    esac
    shift;
done
\rm -f ${script}
echo "#!/bin/sh" > ${script}
echo "#PBS -o ${outputfile}" >> ${script}
echo "${spyglass_cmd}" >> ${script}
qsub -V ${script}
\rm -f ${script}
```

The above `qsub_wrapper` script generates the /tmp/my_script<process_id> file, which is used as an input to the `qsub` LSF command. This file appears like the following:

```
#!/bin/sh
#PBS -o output.txt
$SPYGLASS_HOME/bin/sg_shell -32bit -tcl test.tcl
```

Enabling Parallel or Sequential Goal Run

Specifying the `-parallel_run` command-line option enables parallel goal run flow. However, depending upon whether you specify a parallel configuration file and set the `ENABLE_PARALLEL_RUN` configuration key, SpyGlass may report errors or run goals serially or in parallel based on the following situations:

- If you set the `ENABLE_PARALLEL_RUN` configuration key to `none`, the specified goals are run sequentially on the current machine.
- If you set the `ENABLE_PARALLEL_RUN` key to `goal`, SpyGlass reports an error message and prompts you to specify a parallel run configuration file, if not specified.

In this case, use the `-host_config_file` command-line option to specify a parallel run configuration file.

- If you set the `ENABLE_PARALLEL_RUN` key to `none`, SpyGlass runs goals sequentially on the current machine after prompting that the `-host_config_file` command is ignored.

Sanity Checks Performed During Parallel Goal Run

SpyGlass performs certain checks during parallel goal run and reports a violation in the following cases:

- If parse errors are found in the parallel run configuration file.
- If an invalid login type is specified in the `LOGIN_TYPE` keyword.
- If LSF run is unsuccessful with the specified command.
- If process count is not a positive integer value.
- If none of the machines specified in the RSH/ SSH protocol is accessible.
- If certain machines are not accessible.
- If the `LOGIN_TYPE` keyword is not specified in the parallel file.
- If an error occurs while executing the LSF bsub command.
- If no goal list is specified in the run_goal command, but the `HOST_CONFIG_FILE` key is set, SpyGlass ignores the -host_config_file command, as parallel goal run is not enabled. In addition, SpyGlass runs the currently selected goal on the current machine.
- If the `ENABLE_PARALLEL_RUN` key is set to `none` and a goal list is specified in the `run_goal` command, but the `HOST_CONFIG_FILE` key is set.

  In this case, SpyGlass ignores the `-host_config_file` command as parallel goal run is not enabled. In addition, SpyGlass runs the goals sequentially on the current machine.
- If the `ENABLE_PARALLEL_RUN` key is set to `none`, but goal list is specified in the `run_goal` command.

  In this case, parallel run is not enabled and goals are run sequentially on the current machine.

- If the ENABLE_PARALLEL_RUN key is set to goal, but neither the host configuration file is supplied in the run_goal command nor the HOST_CONFIG_FILE key is set.

  In this case, SpyGlass aborts the parallel goal run because parallel run settings are not specified.

  To fix this issue, you must specify parallel run settings by using the -host_config_file command or the HOST_CONFIG_FILE configuration key.
- If the run_goal command fails because goals pertaining to different synthesis modes cannot be run together.
- If all goals pertain to the same synthesis mode, they may contain design options or goal options that can cause NOMDB or netlist object model database to be saved again. Such mixing of goals is not allowed.

Run-Time Advantage from a Parallel Goal Run

Parallel goal run should give significant time improvement over running the goals sequentially. In an ideal scenario, if all goals are run in parallel, we should see the overall parallel run time equal to the runtime of the goal that takes maximum time when run individually.

However, in actual parallel run environment, the runtime is more than the ideal situation because of the following factors:

- Parallel goal run is limited by the number of available machines, and also by the number of processes allowed to be run on a given machine.

  If you want to reduce the parallel runtime further, increase the machine pool available for parallel goal run, and update your parallel run configuration file accordingly.
- There may be some interdependencies among the goals specified for parallel run, which could delay running of a goal until its dependent goals have been run.
- Parallel goal run requires some initial setup stage where goals are checked for their synthesis view requirement and any disk write operations, such as design precompilation and design save, are performed. Such setup activities are critical to ensure there are no disk read/write operations at the same time from different goals when these are running in parallel.

In parallel goal run, each goal loads policies/design independently. However, the time spent in parallel run setup (described in the last point above) plus the time taken by policies/design load for each parallel goal, should get offset in parallel goal run if rule-checking time is significant, because rules are running in parallel.

Viewing Directories Created After Goal Run

Once the goal run is complete, SpyGlass creates the following directories:

- A directory by the name of the selected methodology under the *<project-name>* directory.
- A sub-directory by the name of the selected goal.
- Equal number of directories by goal names if you have selected multiple goals.

Incremental Mode Analysis

The Incremental Mode Analysis is useful if you want to compare the results of two goal runs.

In some situations, you may want to compare the results that are generated before and after some changes were made to the design. For example, after making certain changes to the design, you may want to check if these changes cause some new violation message to appear, which were not present before the changes.

When the Incremental Mode option is set, SpyGlass compares the results of the current goals run against the results of a previously run goals (which could be a previous run in the current session of SpyGlass or an isolated run previously saved in some other area by same/different user).

To perform Incremental Mode Analysis, select the *Incremental Mode* button on the left section of the Setup Summary page.

After the analysis is performed (in the Analyze Results stage) with the *Incremental Mode* option set, the violation messages in the *Message Tree* are categorized into the following categories:

- New Messages

  These are the additional violation messages that were not reported in the previous run but have been reported in the current run.

- PreExisting Messages

  These are the violation messages that were reported in the previous run as well as the current run.

NOTE: *You can disable the grouping of messages after analysis (in the Analyze Results stage) by deselecting the Incremental Mode button. Then, the messages will no longer appear as grouped in the above two categories.*

To specify a previous version of the reference VDB file that is present in its respective run directory, click (📂) and browse to the location containing both VDB file and corresponding spysch directory.

If you do not specify a reference VDB file with the incremental mode and the goal has any previously run VDB file, that file will be used as the reference VDB file. If the VDB file does not exist and you run a goal in the Analyze Results stage, a warning message appears indicating that the incremental mode does not apply and the Incremental Mode button is automatically turned off.

NOTE: *The Incremental Mode button is also available in the Analyze Results stage.*

Setting Up the Goal in Batch Mode

You can setup a goal in the project file by specifying the following commands on the command-line:

```
current_goal <goal_name> [-top<top_name>]
set_parameter <parameter> <value>
set_goal_option <option> <value>
```

Following example explains setting up a goal for SpyGlass CDC product:

```
current_goal cdc/cdc_verify_struct -top myTop
set_parameter use_inferred_clocks yes
set_goal_option addrule W110
```

The Methodology Configuration System

SpyGlass provides the Methodology Configuration System (MCS) feature that enables you to modify the existing methodologies and create your own custom methodologies. For more information on MCS, see *Working with Methodologies*.

Running Prerequisite Goals

A goal may have one or more prerequisite goals that should run first, before executing the selected goal.

You can enforce the execution of prerequisite goals to ensure that all the prerequisite goals are run first, before the selected goal.

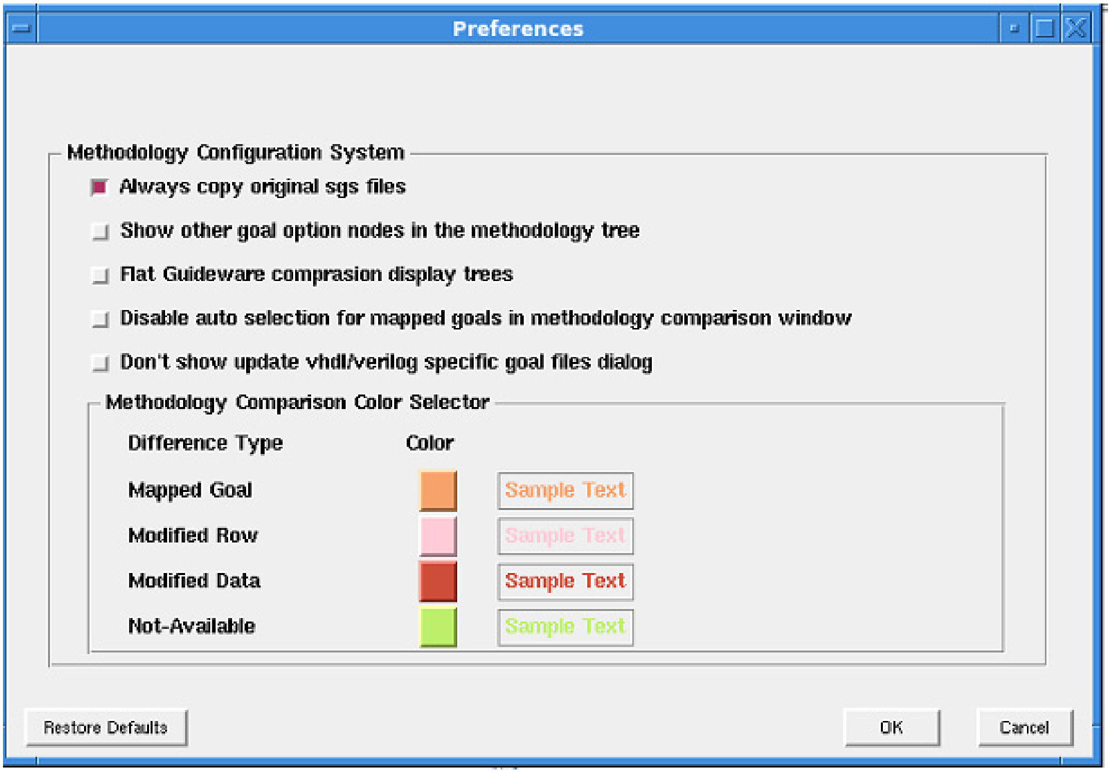
To enforce the execution of prerequisite goals, perform the following steps:

1. Select *Tools -> Preferences* menu option.

   This step displays the *Preferences* dialog.

2. In the *Preferences* dialog, select the *Miscellaneous* option.

   This displays a list of miscellaneous options, as shown in the following figure:



Methodology Configuration System

- ☑ Always copy original sgs files
- □ Show other goal option nodes in the methodology tree
- □ Flat Guideware comprasion display trees
- □ Disable auto selection for mapped goals in methodology comparison window
- □ Don't show update vhdl/verilog specific goal files dialog

Methodology Comparison Color Selector

Difference Type	Color	
Mapped Goal	🟧	Sample Text
Modified Row	⬜	Sample Text
Modified Data	🟥	Sample Text
Not-Available	🟩	Sample Text

FIGURE 121. Methodology Configuration System - Preferences

3. Select the *Enforce Execution of Prerequisite Goals* option appearing under the *General* category.
4. Click the *OK* button.

After performing the above steps, if you try running a goal without first running its prerequisite goal(s), SpyGlass displays a *Warning* dialog. This dialog lists the prerequisite goal(s) that have not yet been run for the current run.

In this dialog, you can select any of the following options, based on your requirement:

- *Select all required prerequisite goal(s)* option to select all the prerequisite goals.

- *Do not unfree the execution of prerequisite goals* option if you do not want to run prerequisite goal(s).

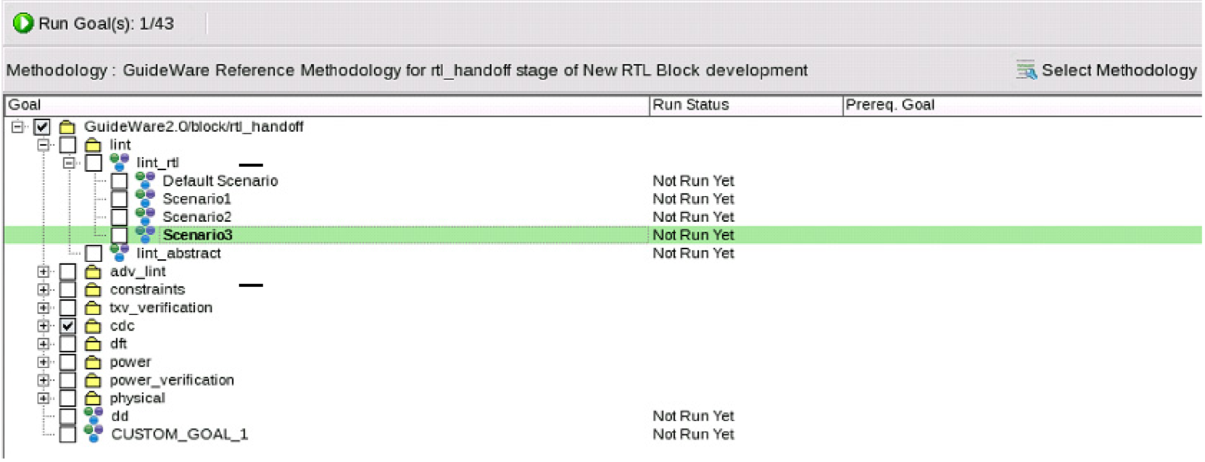
Working with Scenarios

A scenario is a goal that contains modified settings of a goal. You can create multiple scenarios for a goal where each scenario represents different settings for that goal.

For example, you create the scenario, `Scenario1`, for the `connectivity` goal in which you change values of some parameters. Similarly, you can create another scenario, `Scenario2`, for the same `connectivity` goal in which you can specify certain files, such as VCD or SGDC files. You can run these scenarios like any other goal.

The advantage of using scenarios is that you can save different settings (in the form of scenarios) made for a particular goal.

All scenarios for a goal are displayed under the *Select Goal* tab, as shown in the following figure:



Methodology : GuideWare Reference Methodology for rtl_handoff stage of New RTL Block development

```
Goal                                         Run Status      Prereq. Goal
☑ GuideWare2.0/block/rtl_handoff
  □ lint
    □ lint_rtl
      □ Default Scenario                     Not Run Yet
      □ Scenario1                            Not Run Yet
      □ Scenario2                            Not Run Yet
      □ Scenario3                            Not Run Yet
    □ lint_abstract                          Not Run Yet
  □ adv_lint
  □ constraints
  □ txv_verification
  ☑ cdc
  □ dft
  □ power
  □ power_verification
  □ physical
  □ dd                                       Not Run Yet
  □ CUSTOM_GOAL_1                            Not Run Yet
```

FIGURE 122. Run Selected Goal(s)

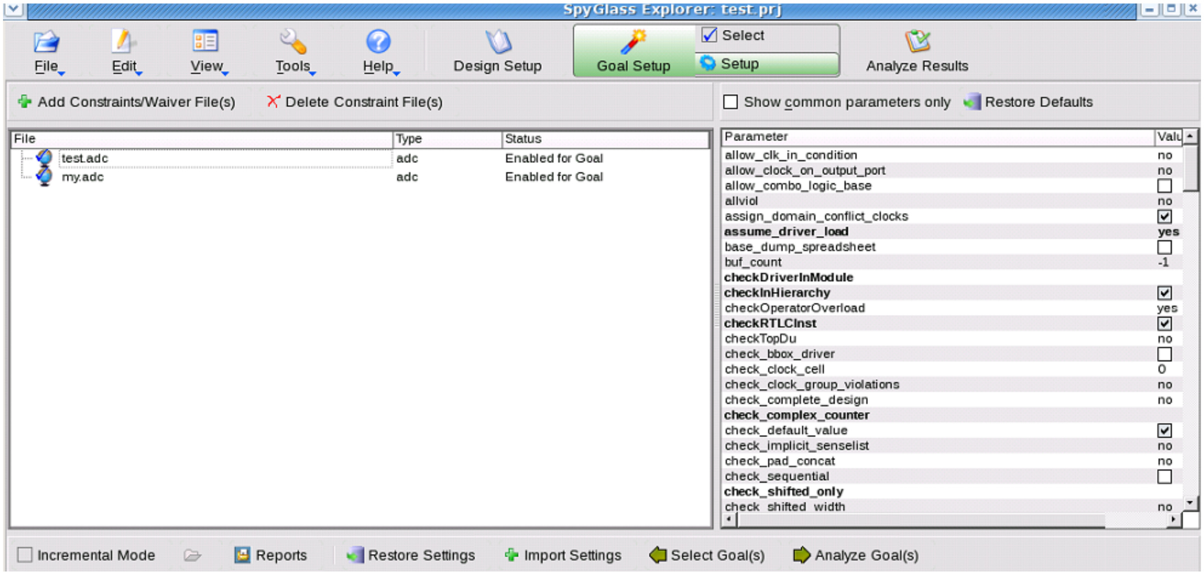
To work with scenarios, you must first enable scenario support by selecting the *Enable Scenario Support* option in the *Miscellaneous* page of the *Preferences* dialog.

Creating Scenarios

To create a scenario for a goal, perform the following steps:

1. Right-click on a goal displayed under the *Select Goal* tab.
2. Select the *Create New Scenario* option from the shortcut menu.

   The page under the *Setup Goal* tab appears, as shown in the following figure:



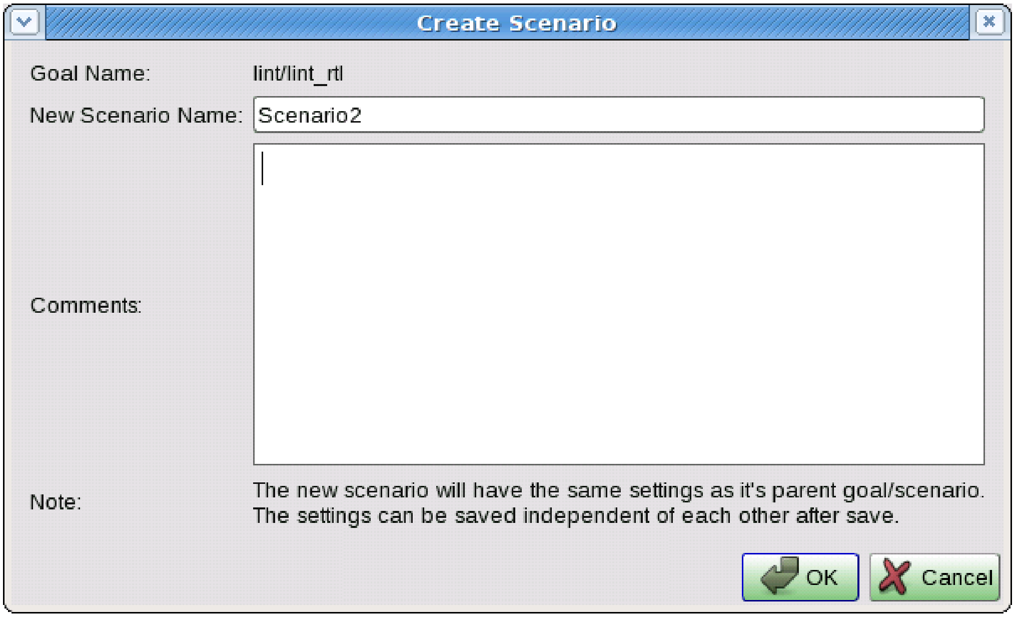
File	Type	Status
test.adc	adc	Enabled for Goal
my.adc	adc	Enabled for Goal

Parameter	Valu
allow_clk_in_condition	no
allow_clock_on_output_port	no
allow_combo_logic_base	□
allviol	no
assign_domain_conflict_clocks	☑
assume_driver_load	**yes**
base_dump_spreadsheet	□
buf_count	-1
checkDriverInModule	
checkInHierarchy	☑
checkOperatorOverload	yes
checkRTLCInst	☑
checkTopDu	no
check_bbox_driver	□
check_clock_cell	0
check_clock_group_violations	no
check_complete_design	no
check_complex_counter	
check_default_value	☑
check_implicit_senselist	no
check_pad_concat	no
check_sequential	□
check_shifted_only	
check_shifted_width	no

FIGURE 123. Creating Scenarios

3. Specify the required settings in the above page. For example, you can change parameter values or add/remove certain files.
4. Click the *Create Scenario* button.

   The *Create Scenario* dialog appears, as shown in the following figure:



Goal Name: lint/lint_rtl

New Scenario Name: Scenario2

Comments:

Note: The new scenario will have the same settings as it's parent goal/scenario. The settings can be saved independent of each other after save.

FIGURE 124. Create Scenario - New Scenario

5. In the above dialog, specify the name of the new scenario to be created.

NOTE: *Scenario names can contain only alphanumeric characters, underscore, minus.*

6. Click the *OK* button to close the above dialog.

   After performing the above steps, the name of the newly created scenario appears in the *Scenario* drop-down list under the *Setup Goal* tab.

Modifying and/or Deleting Scenarios

You can modify scenario details, updating parameter values and adding/deleting files. In addition, you can delete the required scenarios.

Modifying a Scenario

To modify a scenario, perform the following steps:

1. Right-click on a scenario appearing under the *Select Goal* tab.
2. Select the *Edit Scenario* option from the shortcut menu.

The page under the *Setup Goal* tab appears. In this page, the name of the scenario being modified appears in the *Scenarios* drop-down list.

3. Change the settings for the scenario as per your requirement under the *Setup Goal* tab.

After performing the above steps, the selected scenario is updated.

If you want to modify another scenario, select the scenario name from the *Scenarios* drop-down list. The settings related to that scenario get loaded. You can then modify these settings as per your requirement.

Deleting a Scenario

To delete a scenario, right-click on that scenario appearing under the *Select Goal* tab and select the *Delete Scenario* option from the shortcut menu.

Running Scenarios

You can run scenarios in both GUI and batch.

Running Scenarios In GUI

To run a scenario in GUI, select that scenario appearing under the *Select Goal* tab, and click the *Run Selected Goal(s)* option. You can also select multiple scenarios.

Please note the following points:

- If you only want to run a goal from which all its scenarios have been created, select the *Default Scenario* option appearing adjacent to that goal.
- If you want to run a goal as well as all its scenarios, select the goal instead of the *Default Scenario* option of that goal.

Running Scenarios in Batch

To run a scenario in batch, specify the following command:

```
spyglass -project <project-name>.prj -goal
<goal-name>@<scenario-name> -batch
```

For example, to run the scenario, `s1`, of the `connectivity` goal, specify the following command in batch:

```
-project Project-1.prj -goal initial_rtl/lint/
connectivity@s1 -batch
```

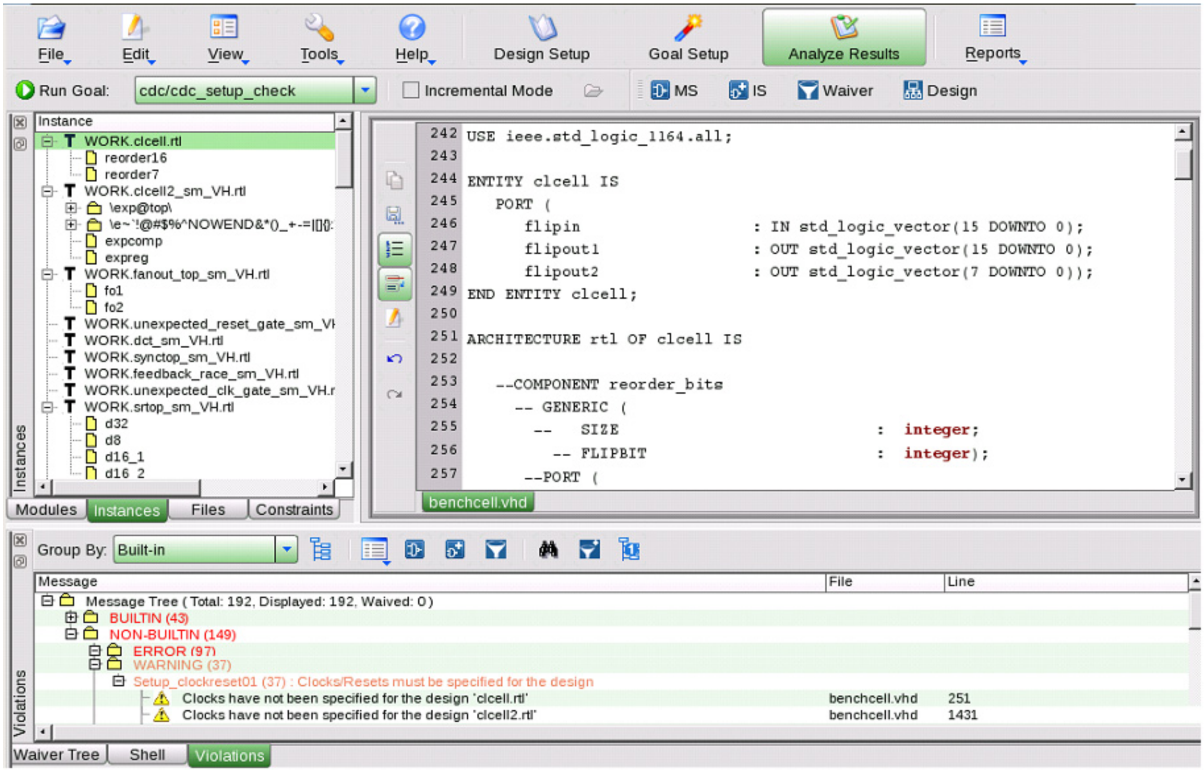
Directory Structure Created After Running a Scenario

The following directory structure is created to store results of a particular scenario run:

```
<project-name>/<top-module-name>/<goal-name>/
<scenario-name>
```

Stage 3: Analyzing a Design (Analyze Results)

This stage enables you to analyze results of a goal run. To view the results, click the *Analyze Results* tab. The following figure shows the page under this tab:



```
USE ieee.std_logic_1164.all;

ENTITY clcell IS
   PORT (
      flipin              : IN std_logic_vector(15 DOWNTO 0);
      flipout1            : OUT std_logic_vector(15 DOWNTO 0);
      flipout2            : OUT std_logic_vector(7 DOWNTO 0));
END ENTITY clcell;

ARCHITECTURE rtl OF clcell IS

   --COMPONENT reorder_bits
     -- GENERIC (
       --   SIZE                          :  integer;
         -- FLIPBIT                       :  integer);
      --PORT (
```

Message	File	Line
Message Tree (Total: 192, Displayed: 192, Waived: 0)		
BUILTIN (43)		
NON-BUILTIN (149)		
ERROR (97)		
WARNING (37)		
Setup_clockreset01 (37) : Clocks/Resets must be specified for the design		
Clocks have not been specified for the design 'clcell.rtl'	benchcell.vhd	251
Clocks have not been specified for the design 'clcell2.rtl'	benchcell.vhd	1431

FIGURE 125. Analyze Results Tab

The above page shows information, such as reported messages related to a particular goal run. If you have run multiple goals, select a goal (whose run results you want to view) from the drop-down list adjacent to the *Run Goal* link in the above page.

Based on the reported messages, perform different actions, such as fixing violations, waiving a message, or tagging a message. For details, see *Working with SpyGlass Messages*.

Editing Source Files

To edit a source file for which a message is reported, perform the following steps:

1. Double-click on the message.

   After this step:

   - The source file containing violation reported by that message appears in the *HDL Viewer* pane.
   - The violating line is highlighted in the source file.

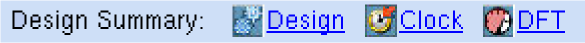
2. Click the Edit File in the *Navigation* bar.

   The file appears in an editor window.

3. Edit the file to fix the violation.

Viewing Goal Summary

SpyGlass displays a high-level goal summary of selected goals, as shown in the following figure:



Design Summary: Design Clock DFT

You can view the result information in the following formats:

- Balloon View: This view provides the complete design-related information in a balloon window.
- Detailed View: This view provides a detailed explanation of the information displayed in the balloon view.

Returning Back to the Goal Setup & Run Stage

After analyzing results, you go back to the *Goal Setup & Run* stage to select a different set of goals and run them to see different analysis results.

Comparing Results of Multiple SpyGlass Runs

Comparing results of two SpyGlass runs enables you to:

- View violations reported in the first run and not reported in the second run.

  This may happen if you have made necessary fixes in the source code or waived unwanted messages before the second run.
- View new violations that were not reported in the first run but are reported in the second run.

  This may happen if the fixes that you made have errors or you have run another set of goals.

Use the incremental mode feature to compare results of two SpyGlass runs.

Introducing the Incremental Mode Feature

Under this feature, SpyGlass compares messages of the current goal(s) run against a set of messages (in the specified .vdb file) of an earlier goal(s) run.

Based on the comparison, messages are displayed under any of the following categories:

- *PreExisting Messages*: Messages that exist in both .vdb files.
- *New Messages*: Messages that exist in the .vdb file of the current run only.
- *Fixed/Missing Messages*: Messages that exist in the .vdb file of the previous run only. Such messages are considered as fixed.

Using the Incremental Mode Feature

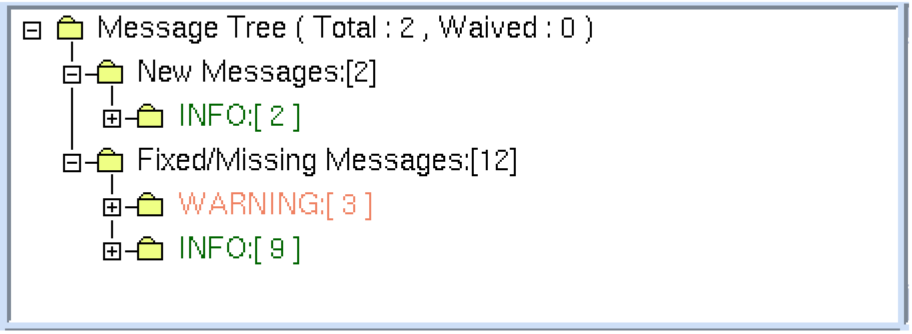
To use the incremental mode feature, perform the following steps:

1. Run the required goals.
2. Based on the reported violations, perform appropriate actions, such as modify source files or waive unwanted messages.

   This step is optional.
3. Under the *Analyze Results* tab, select the *Incremental Mode* option.
4. Click 📂 to specify the reference VDB file with which you want to compare results. For details, see *Using the vdb File*.
5. Select and run the required goals.

   You may re-run the previous goals and/or new goals.

After performing the above steps, SpyGlass reports violations under appropriate categories. An example is shown in the following figure:



```
⊟ Message Tree ( Total : 2 , Waived : 0 )
    ⊟ New Messages:[2]
        ⊞ INFO:[ 2 ]
    ⊟ Fixed/Missing Messages:[12]
        ⊞ WARNING:[ 3 ]
        ⊞ INFO:[ 9 ]
```

FIGURE 126. SpyGlass Violations

Using the vdb File

SpyGlass picks the required vdb file in the following ways:

- If you specify a vdb file, that file is considered for comparing results.
- If you do not specify a reference vdb file with incremental mode and the goal has any previously run vdb file, that file is used as the reference vdb file.

SpyGlass automatically disables the incremental mode features in the following cases:

- If the vdb file does not exist and you run a goal in the *Analyze Results* stage, a warning message appears indicating that the incremental mode does not apply and the *Incremental Mode* option is automatically disabled.
- If no vdb file of the same name is available in the current working directory, the incremental mode is automatically disabled.

Comparison Reported in Batch

The comparison reported in batch mode is shown in the following examples:

Summary of Original Run (Without Incremental Mode)

```
 Total Number of Generated Messages      :       93
  Number of Waived Messages              :       11
  Number of Reported Messages            :       82
```

Summary for Second Run (With Incremental Mode Set)

```
Total Number of Generated Messages         :      161
 Number of Waived Messages                 :       16
 Number of Reported Messages (New)         :       63
 Number of Reported Messages (PreExisting):        82
```

In the above example, the original run generated 93 messages, out of which 11 messages had been waived. The second run (with incremental mode set and run with an extra product) generated 161 messages, out of which 16 messages had been waived and among the reported messages, 82 messages were the same as those reported in the previous run and 63 new messages were reported because of the extra goal being run.

Viewing Different Type of Results

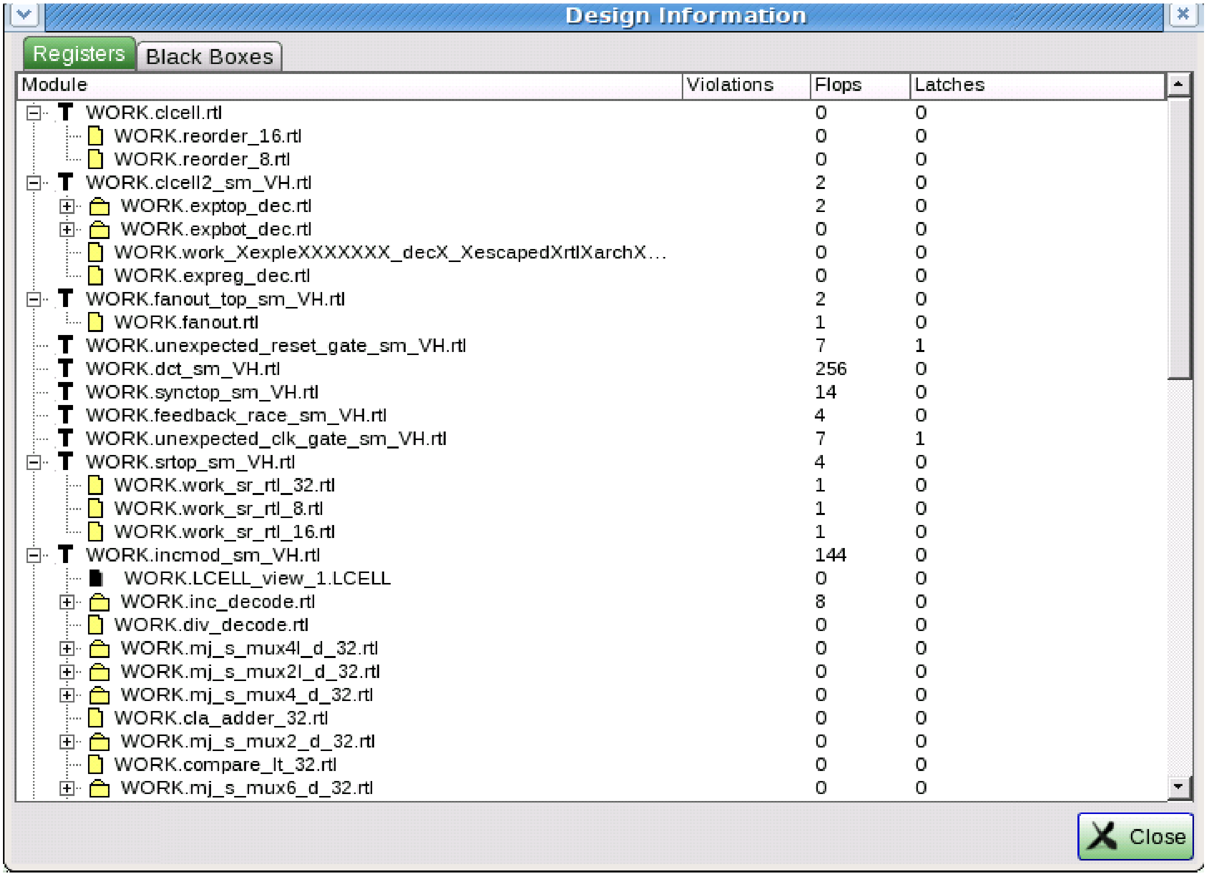
Based on the goal run, SpyGlass displays appropriate options to view different results, such as *Design Results*, *SpyGlass CDC Solution Results*, *SpyGlass Constraints Solution Results*, *SpyGlass TXV Solution Results*, *SpyGlass DFT Solution Results*, and *Power Results*.

Design Results

When you point the mouse on the *Design* option, a tool-tip appears displaying the following information:

- Total number of black boxes in the design
- Total number of latches in the design
- Total number of flip-flops in the design

When you click the *Design* option, the *Design Information* dialog appears as shown in the following figure:



Module	Violations	Flops	Latches
WORK.clcell.rtl		0	0
WORK.reorder_16.rtl		0	0
WORK.reorder_8.rtl		0	0
WORK.clcell2_sm_VH.rtl		2	0
WORK.exptop_dec.rtl		2	0
WORK.expbot_dec.rtl		0	0
WORK.work_XexpleXXXXXXX_decX_XescapedXrtlXarchX...		0	0
WORK.expreg_dec.rtl		0	0
WORK.fanout_top_sm_VH.rtl		2	0
WORK.fanout.rtl		1	0
WORK.unexpected_reset_gate_sm_VH.rtl		7	1
WORK.dct_sm_VH.rtl		256	0
WORK.synctop_sm_VH.rtl		14	0
WORK.feedback_race_sm_VH.rtl		4	0
WORK.unexpected_clk_gate_sm_VH.rtl		7	1
WORK.srtop_sm_VH.rtl		4	0
WORK.work_sr_rtl_32.rtl		1	0
WORK.work_sr_rtl_8.rtl		1	0
WORK.work_sr_rtl_16.rtl		1	0
WORK.incmod_sm_VH.rtl		144	0
WORK.LCELL_view_1.LCELL		0	0
WORK.inc_decode.rtl		8	0
WORK.div_decode.rtl		0	0
WORK.mj_s_mux4l_d_32.rtl		0	0
WORK.mj_s_mux2l_d_32.rtl		0	0
WORK.mj_s_mux4_d_32.rtl		0	0
WORK.cla_adder_32.rtl		0	0
WORK.mj_s_mux2_d_32.rtl		0	0
WORK.compare_lt_32.rtl		0	0
WORK.mj_s_mux6_d_32.rtl		0	0

FIGURE 127. Design Information

In the above dialog, you can perform the following actions:

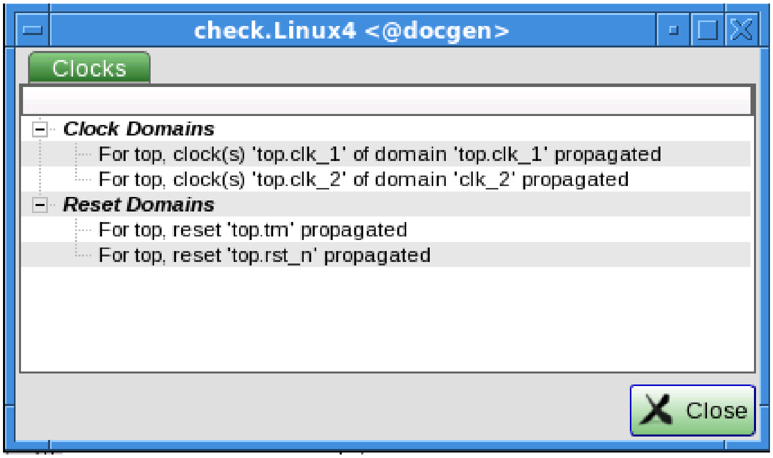
- Click the *Registers* tab to view the information about the modules, latches, and flip-flops present in the design.
- Double-click on a module under the *Registers* tab to highlight that module in the *Module View Page*.
- Click the *BlackBoxes* tab to view the black box information.

  For more information, see *Working with Black Boxes*.

SpyGlass CDC Solution Results

When you point the mouse on the *Clock-Reset* option, a tool-tip appears displaying the total number of unsynchronized clocks, resets, and clock domains.

When you click the *Clock-Reset* option, the *Clock-Reset Information* dialog appears as shown below:



Clock Domains
- For top, clock(s) 'top.clk_1' of domain 'top.clk_1' propagated
- For top, clock(s) 'top.clk_2' of domain 'clk_2' propagated

Reset Domains
- For top, reset 'top.trn' propagated
- For top, reset 'top.rst_n' propagated

FIGURE 128. Clock-Reset Information

SpyGlass Constraints Solution Results

When you point the mouse on the *Constraints* option, a tool-tip appears displaying the total number SDC and SGDC files.

The detailed view is not available for the *Constraints* option.

SpyGlass TXV Solution Results

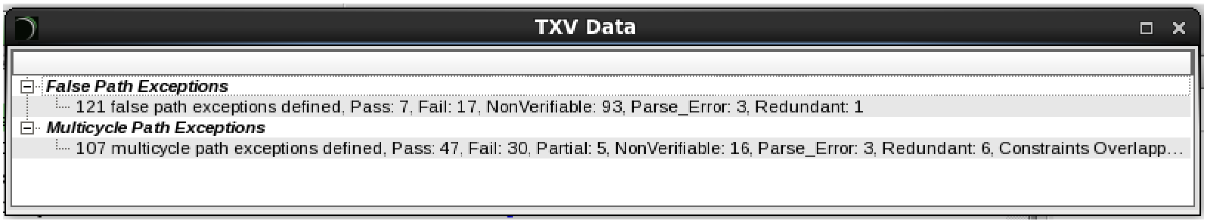
When you point the mouse on the *TXV* option, a tool-tip appears displaying the following information in a tabular format:

- False paths and multi-cycle paths declared as passed
- False paths and multi-cycle paths declared as failed
- False paths and multi-cycle paths declared as incomplete
- False paths and multi-cycle paths declared as inconclusive

When you click the *TXV* option, false paths and multi-cycle paths appear in a tree format. The tree format has the following parent nodes:

- Node for false paths
- Node for the multi-cycle paths

The following figure shows the node for false paths:



False Path Exceptions

121 false path exceptions defined, Pass: 7, Fail: 17, NonVerifiable: 93, Parse_Error: 3, Redundant: 1

Multicycle Path Exceptions

107 multicycle path exceptions defined, Pass: 47, Fail: 30, Partial: 5, NonVerifiable: 16, Parse_Error: 3, Redundant: 6, Constraints Overlapp…

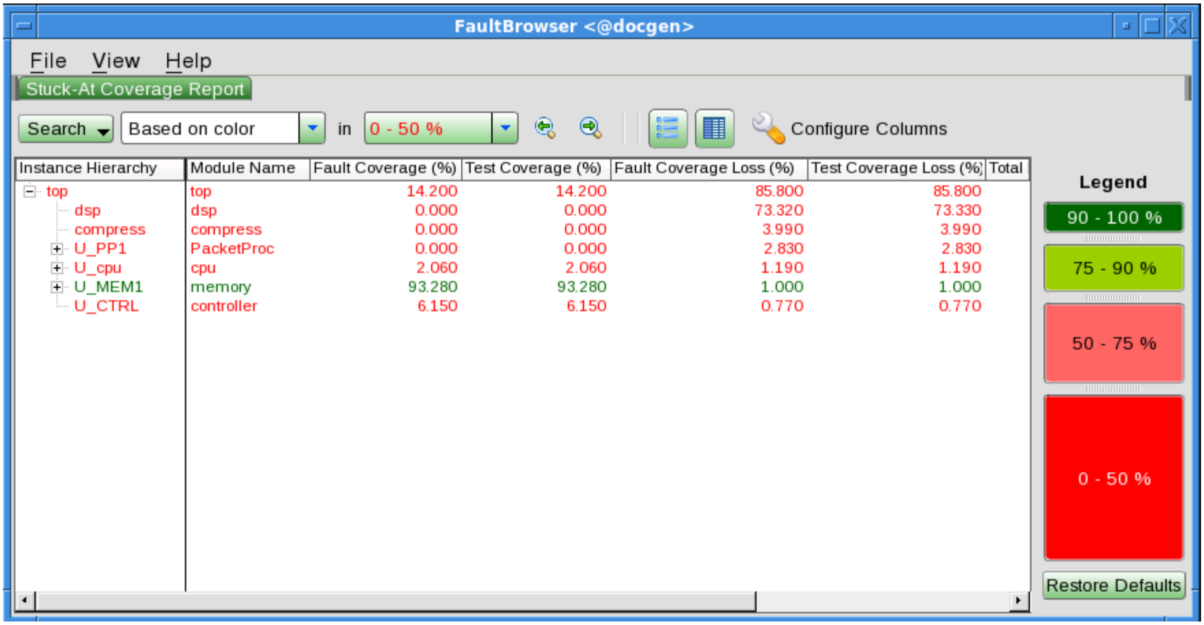
FIGURE 129. False Path Nodes

The False Path and Multi-cycle Path nodes are further categorized into Passed, Failed, Incomplete, and Inconclusive sub-nodes that contain the violation messages.

SpyGlass DFT Solution Results

When you point the mouse on the *DFT* option, a tool-tip appears displaying the test coverage and fault coverage for the top modules.

When you click the *DFT* option, the *DFT Information* dialog appears, as shown in the following figure:



Instance Hierarchy	Module Name	Fault Coverage (%)	Test Coverage (%)	Fault Coverage Loss (%)	Test Coverage Loss (%)	Total
top	top	14.200	14.200	85.800	85.800	
dsp	dsp	0.000	0.000	73.320	73.330	
compress	compress	0.000	0.000	3.990	3.990	
U_PP1	PacketProc	0.000	0.000	2.830	2.830	
U_cpu	cpu	2.060	2.060	1.190	1.190	
U_MEM1	memory	93.280	93.280	1.000	1.000	
U_CTRL	controller	6.150	6.150	0.770	0.770	

FIGURE 130. DFT Information

The information in the *DFT Information* dialog is categorized into the following columns:

- *Instances*: Displays instances from the *Instance View Page*.
- *Module*: Displays modules present in the design.
- *Fault Coverage*: Displays fault-coverage data.
- *Test Coverage*: Displays the test coverage data.

Power Results

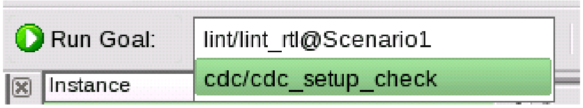
When you point the mouse on the *Power* option, a tool-tip appears displaying the leakage, internal, switching, and total power of the top-level design unit.

When you click the *Power* option, the *Power Browser* option appears. Click this option to open the *SpyGlass Power Browser* window that displays results related to estimating power.

Refer to the *Viewing Power Estimation Results* section of SpyGlass *Power Product Family Rules Reference User Guide* for more details.

Viewing Results of Different Scenarios and Goals

To view results of a scenario or a particular goal, select the required scenario, group name, or goal from the drop-down list, as shown in the following figure:



lint/lint_rtl@Scenario1
cdc/cdc_setup_check

FIGURE 131. Analyze Results - Run Goal

After selecting the required option from the list, click the *Run Goal* option. This step runs SpyGlass analysis again and results are loaded under the *Analyze Results* tab.

Cross-probing from the Msg Tree Page

The messages listed in *Msg Tree* page can be cross-probed to other SpyGlass windows. To do so, double-click a message in the *Msg Tree* and the following cross-probes are created:

- The source file in which the message is reported is highlighted in the *Module View* page.
- The source code of the file in which the message is reported is highlighted in the *Source Window*.
- The corresponding schematic information (if available, indicated by the schematic icon ⊳) is highlighted in the *Modular Schematic* Window and the *Incremental Schematic* Window.

NOTE: *SpyGlass does not allow you to cross-probe to the RTL of encrypted design units. If you try to cross-probe to RTL of such design units, SpyGlass displays a message in the RTL viewer specifying that the file is encrypted.*

Understanding SpyGlass Concepts

SpyGlass functional model is summarized in the following figure:

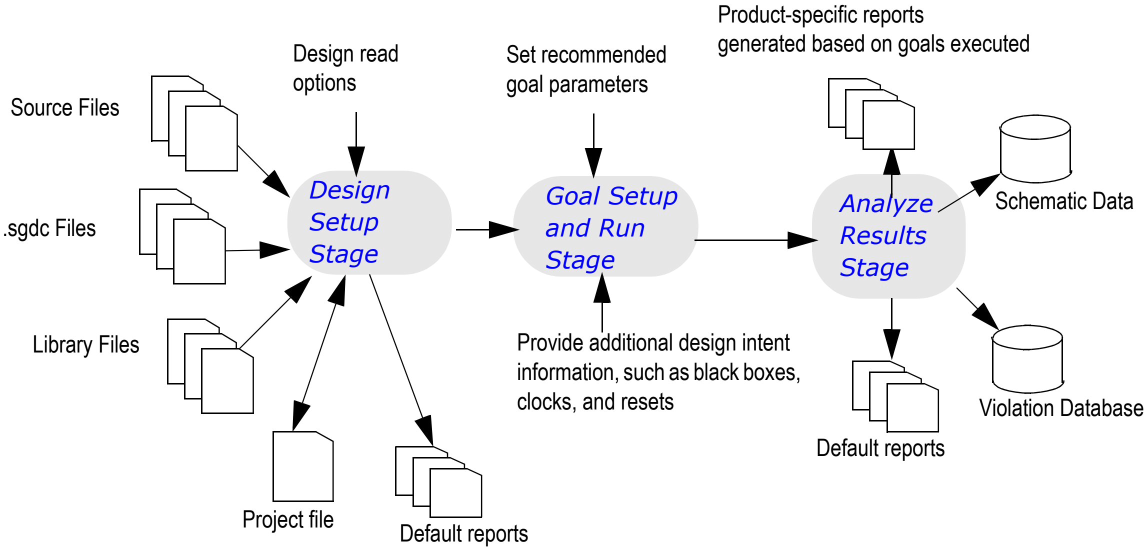


FIGURE 132. SpyGlass Functional Model

Design Setup Stage

This is the first stage in which you can:

- Add design files, SGDC files, precompiled files, and technology files. For details, refer to the *Adding Design Files* topic.
- Specify various design-read options that affect SpyGlass run. For example, you can specify top-level modules in your design, change the language, specify macros, etc.

  For details, refer to the *Viewing and Changing Design Read Options* topic.
- Run the design-read process to perform first level of HDL analysis. For details, refer to the *Running Design Read* topic.

  You must resolve FATAL errors, if any, reported during this stage before proceeding to the next stage.

  You can skip the task of running the design-read process.

Goal Setup and Run Stage

During this stage, you select and run goal(s). A goal is a collection of rules.

During this stage, you can:

- Select and run goal(s). For details, see *Selecting a Goal*.

  You can also specify the order in which goals should run. You can specify this order in an order file. For details, see *Order File*.
- Provide additional design intent information, such as black boxes, clocks, and resets in your design.
- Set the recommended parameters and required constraints for the selected goal.

Analyze Results Stage

This stage enables you to analyze the results of a goal run.

During this stage, you can:

- View violation messages to identify the design issues. For details, see *Working with SpyGlass Messages*.
- Debug the reported issues by referring to schematics, reports, and rule help.
- Waive some violation messages if they do not indicate potential design issues. For details, refer to the *Waiving Messages* topic.

SpyGlass Built-in Checking

While analyzing or synthesizing RTL designs, SpyGlass performs checks on the HDL syntax and structure. These checks are always performed automatically, independently of which SpyGlass rules are requested to be checked.

If any syntax or structure issues are found, SpyGlass generates the corresponding standard error or warning messages (known as *built-in messages*). These built-in messages are different from the rule messages generated during rule-checking.

There are the following classes of such built-in messages:

TABLE 11 Built-in Messages Classes

Message Type	**Message Prefix**
Syntax errors	STX_
Language Warnings	WRN_
Synthesis warnings	SYNTH_
Synthesis errors	SYNTH_
Post-elaboration syntax errors (VHDL only)	ELAB_

Syntax Errors and Language Warning messages are language-specific, that is, there are separate message sets for Verilog, VHDL, and Mixed-Language respectively. Synthesis warnings and errors are language-neutral for the most part. See the *SpyGlass Built-In Messages Reference* for details of these message sets.

The `ELAB_` messages may appear when a VHDL design is being processed after elaboration. These messages relate to syntax issues and have the same number and content as the syntax messages except they have the `ELAB_` prefix.

NOTE: *Some of the rules in SpyGlass products are mapped to the SpyGlass built-in messages. Thus, when you run SpyGlass with rule-checking, some of the standard warning and error messages are suppressed and equivalent rule messages are generated in their place.*

NOTE: *If SpyGlass detects an internal software error, it reports the error as a* `WARNING` *message. These errors are normally associated with a specific implementation of a* *rule. Any such error messages should be reported to Atrenta Customer Support.*

Processing the HDL Designs

When you run SpyGlass on a design, the following process is followed:

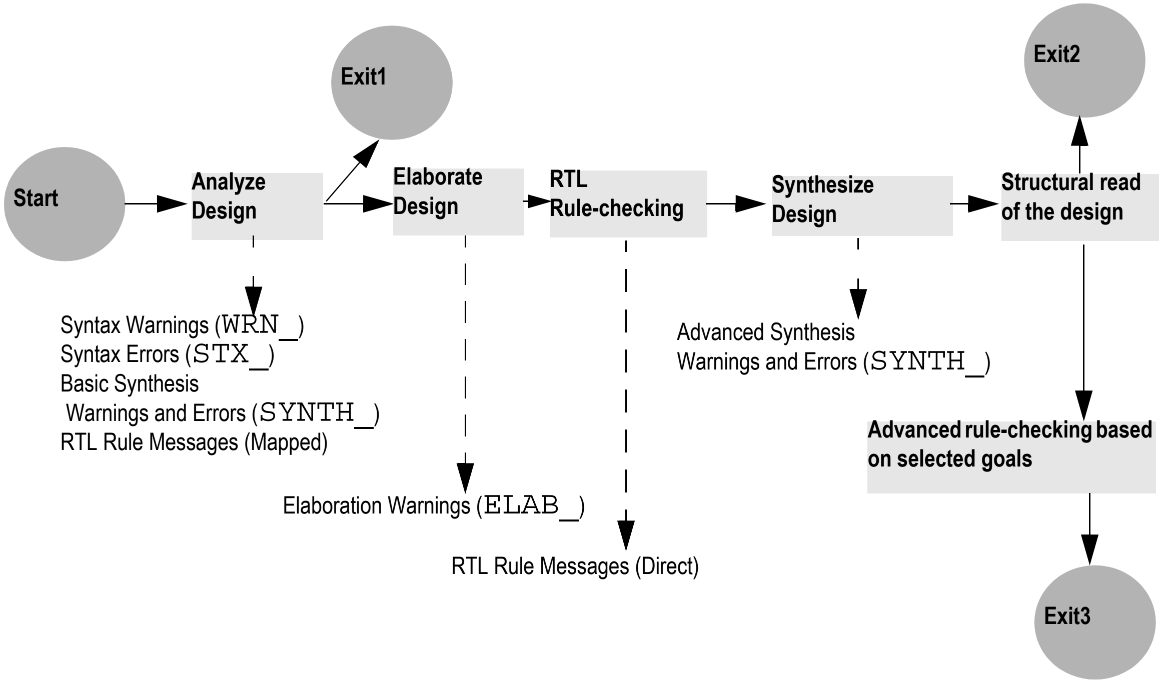

Exit1
Exit2
Start
Analyze Design
Elaborate Design
RTL Rule-checking
Synthesize Design
Structural read of the design
Syntax Warnings (WRN_)
Syntax Errors (STX_)
Basic Synthesis
Warnings and Errors (SYNTH_)
RTL Rule Messages (Mapped)
Advanced Synthesis
Warnings and Errors (SYNTH_)
Advanced rule-checking based on selected goals
Elaboration Warnings (ELAB_)
RTL Rule Messages (Direct)
Exit3

FIGURE 133. SpyGlass Run Process

SpyGlass processing occurs in the following steps:

1. SpyGlass analyzes the design and generates the following types of standard built-in messages:

   ❒ Syntax warning messages (WRN_ messages)

   ❒ Syntax error messages (STX_ messages)

   ❒ Basic synthesis error and warning messages (SYNTH_ messages)

   SpyGlass also generates rule messages (instead of built-in messages mapped to the rules in the selected products)

2. If the design has syntax errors (`STX_` messages) after initial processing, SpyGlass exits (shown as **Exit1**).
3. If the design does not have any syntax errors, SpyGlass may perform elaboration of the RTL design and generate elaboration time errors or warnings (`ELAB_` messages)
4. SpyGlass performs RTL rule-checking and generates rule messages as applicable.
5. SpyGlass synthesizes the design and generates the advanced synthesis error and warning messages (`SYNTH_` messages).
6. SpyGlass performs structural read of the design and then SpyGlass exits (shown as **Exit2**) if no goals are selected.
7. If you have selected goals, SpyGlass runs the rules of the goals, and reports appropriate violation messages.
8. SpyGlass exists (shown as **Exit3**).

SpyGlass Rule Environment

Understanding SpyGlass Rule Definitions

Each SpyGlass product whether standard or custom, is a PERL source file that has the rule definition, the rule group definitions, and PERL subroutines, if any.

Each rule has a number of attributes that decide how the rule will function. Some of the important attributes are as follows:

- *Rule name*, which you use to specify that the particular rule should be run
- *Language* applicable for a rule. This attribute enables you to know if the rule works on Verilog, VHDL, or mixed-language designs.
- *Severity* of the rule, which indicates how important it is to fix your source code
- *Message* displayed if your source code violates the rule
- *Rule primitive* that is called for the rule

Rule Primitives

Rule primitives are C functions that are present either in SpyGlass core or in a shared library. They perform the real work of checking rules against your design and can be parameterized to produce different checks. The primitive called for each rule is specified in the product. However, in normal use, the rule primitives and the rule attributes are transparent to the user. The rule primitive extracts object data from the design database, runs checks as defined by the parameters, and generates a message as defined in the rule, if an error is found.

Rule Types and Order of Execution

SpyGlass rules are categorized to work on a particular design view of your source file. There are rules that work on the source RTL description, rules that work on the synthesized hierarchical netlists, and rules that work on the flattened netlists. Consequently, the rules work in the same order in which the corresponding design becomes available.

The SpyGlass rule types by their order of execution are as follows:

1. SETUP type rules

   The SETUP type rules are checked just before design analysis phase.

   This category contains rules that are independent of HDL source code (Verilog or VHDL) or rules that are checked inside the analysis/synthesis engine.

   The SETUP type rules are as follows:

   ❒ Rules that check for commands (including the product-specific rule parameters) and existence of different custom data requirements.

   ❒ Rules that check for liberty library files (.lib files)

   ❒ Rules that check for SGDC files

   ❒ Enabling of analyzer built-in rules and synthesis built-in rules.

NOTE: *The analyzer built-in rules and synthesis built-in rules are only enabled by standard built-in rule-primitives. Actual checking and rule message generation occurs during actual analysis or synthesis.*

2. RTLALLDULIST type rules

   The RTLALLDULIST type rules operate on all RTL design unit including those from precompiled libraries. For Verilog, a design unit is a module, UDP, or macro-module. For VHDL, a design unit is an entity, architecture, configuration, package, or package body.

   An RTLALLDULIST type rule is checked once on each RTL/Library design unit in the design. These rules can access only the local design unit data. They cannot access hierarchy information.

3. RTLDU type rules

   The RTLDU type rules operate on a single RTL design unit. For Verilog, a design unit is a module, UDP, or macro-module. For VHDL, a design unit is an entity, architecture, configuration, package, or package body.

   An RTLDU type rule is checked once on each design unit in the design. These rules can access only the local design unit data. They cannot access hierarchy information.

   By default, the RTLDU type rules do not run on precompiled design units instantiated in a design. To enable rule checking on such design units, specify the following command in a project file:

   ```
   set_option hdllibdu yes
   ```

4. RTLDULIST type rules

The RTLDULIST type rules work on the complete RTL design (not including the precompiled library design unit), without elaboration.

An RTLDULIST type rule is checked once on the complete design. These rules can access only the local design unit data. They cannot access hierarchy information.

By default, the RTLDULIST type rules do not run on precompiled design units instantiated in a design. To enable rule checking on such design units, specify the following command in a project file:

```
set_option hdllibdu yes
```

5. ELABDU type rules

   The ELABDU type rules work on the elaborated design units.

   An ELABDU type rule is checked once on each design unit, elaborated with unique set of parameters/generics with which it has been instantiated. For example, if a module is instantiated 4 times with a parameter value of 5 and 2 times with a parameter value of 7, the module would be elaborated only twice, (instead of 4+2) once with 5 as the parameter value and once with 7 as the parameter value.

6. RTLTOPDU type rules

   The RTLTOPDU type rules are run on the top design unit, determined by SpyGlass after design elaboration.

7. LEXICAL type rules

   The LEXICAL type rules check on each design file, one at a time. This category generally contains non-electronic and text-based rules, such as line-length, tabs, indentation, naming conventions, etc. By default, the LEXICAL type rules do not run on precompiled RTL files.

   To enable rule checking on such design units, specify the following command in a project file:

   ```
   set_option hdllibdu yes
   ```

8. VSDU type rules

   The VSDU type rules check upon a synthesized object model of a design unit.

   A VSDU type rule is checked once on each synthesized design unit.

9. VSTOPDU type rules

   The VSTOPDU type rules are run on the synthesized object model for each top module hierarchy.

Since the RTL view is not available, such rules' checking is limited to only the synthesized object model.

10. BLOCKDU_CD type rules

The BLOCKDU_CD type rules run on user-specified design units flattened down to the specified flattened partitions (called *blocks*) and the blocks themselves.

In this case, design units are flattened only till block boundaries and each block in turn is flattened till boundaries of its sub-blocks, if any.

The BLOCKDU_CD type rules run on each specified design unit and each block once.

Consider the following example where the design unit `top` has a partition named `blk1`:

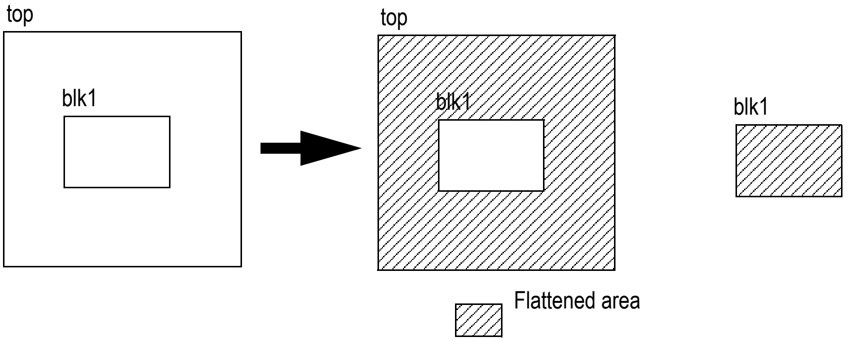


FIGURE 134. Processing with BLOCKDU_CD type rules

In this case, each BLOCKDU_CD type rule is run once on the design unit `top` flattened up to the block `blk1` boundary and once on the flattened block `blk1`.

11. FLATBLOCKDU type rules

The FLATBLOCKDU type rules run on user-specified flattened design units and only the specified flattened partitions (or blocks) within these design units.

In this case, design units and blocks are completely flattened until leaf-level.

The FLATBLOCKDU type rules run on each user-specified flattened design unit and each specified block under the design unit once.

Consider the following example where the design unit `top` has a partition named `blk1`:

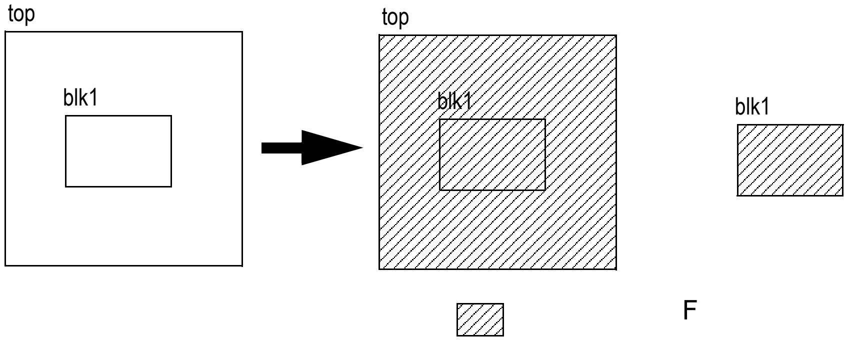


FIGURE 135. Processing with FLATBLOCKDU type rules

In this case, each FLATBLOCKDU type rule is run on the design unit `top` flattened down to leaf-level and on block `blk1` flattened down to leaf-level.

12. FLATDU type rules

    The FLATDU type rules run on the Flat Object Model for each top module in design.

13. FLATDU2_WOL type rules

NOTE: *The FLATDU2_WOL type has been deprecated. Rules registered with this view are internally moved to the FLATDU2_WL view.*

14. FLATDU2_WL type rules

    The FLATDU2_WL type rules are designed to work with SpyGlass Logic Evaluator.

    In this view, .lib instances are not flattened. In addition, .lib functionality is not visible to these rules. Unlike the FLATBLOCKDU type rules, flattening of such rules is applied to the top-level design units only.

    SpyGlass ERC solution has rules of this type, which just use electrical information of a cell but not cell functionality.

15. ALLVIEWS_WL type rules

The ALLVIEWS_WL type rule category is a special deprecated case. The rules of this category require all design views (RTL, Synthesized, and Flattened) in memory and are run after FLATDU2_WL type of rules and before FLATDU type of rules. From flattening point of view, they are similar to FLATDU2_WL type of rules.

16. ALLVIEWS type rules

    The ALLVIEWS type rule category is a special deprecated case. The rules of this category require all design views (RTL, Synthesized, and Flattened) in memory and are run only after rules of all other categories have been run.

Atrenta Standard Products

SpyGlass analyzes HDL source files using pre-defined rules contained in product files. A number of Atrenta Standard products are installed with SpyGlass.

The Atrenta Base products that are available with SpyGlass are:

- **SpyGlass lint solution**, which is based on the commonly accepted set of HDL rules for detecting syntax errors and common connectivity errors such as unused signals and undriven inputs.
- **SpyGlass OpenMore solution**, which is based on the OpenMORE standard developed by Synopsys and Mentor Graphics, and which sets forth certain coding styles and practices that enhance the reusability of HDL design modules.
- **SpyGlass STARC/STARC02/STARC05 solution**, which is based on IP Reuse guidelines compiled by Semiconductor Technology Academic Research Center (STARC).
- **SpyGlass area solution**, which includes rules used to identify HDL that may cause potential problems in silicon area downstream.
- **SpyGlass latch solution**, which includes rules related to latch based designs
- **SpyGlass miscellaneous solution**, which includes useful rules that cannot be classified in any other default products.
- **SpyGlass timing solution**, which includes rules to identify potential timing issues in the design.

- **SpyGlass ERC solution**, which checks a design for correct electrical connectivity at the gate level.

In addition, there are Atrenta Advanced products available that can be purchased separately:

- **SpyGlass DFT Solution**, which checks a design for testability issues at the RT level.
- **SpyGlass Constraints Solution**, which checks the suitability of the Synopsys Design Constraints files
- **SpyGlass Power Verify Solution**, which checks the design for the structural, architectural, and system level issues in order to reduce power.
- **SpyGlass CDC Solution**, which include rules to check clocks, resets, and clock-domain-crossings in the design
- **SpyGlass Power Estimate Solution**, which include rules for power estimation and power reduction
- **SpyGlass TXV Solution**, which is used to verify correctness of timing exceptions and/or check if critical paths can be ignored or relaxed during timing sign-off.

Customizing SpyGlass Rules

SpyGlass allows you to customize a number of its functions to meet your company’s needs. For example:

- Add your own rules or modify the pre-defined rules making use of the pre-defined primitives.

  See the *SpyGlass Policy Customization Guide* for more details.
- Modify the SpyGlass defaults for rule severities, messages, allowed name syntax, or other parameters.

  See the *SpyGlass Policy Customization Guide* for more details.
- Create custom reports

Using Rule Mnemonics

Rule mnemonics refer to the custom rule names assigned to the SpyGlass rule. These custom rule names are more descriptive and convey additional details about a rule.

This section describes the following topics:

- *Enabling the Support for Rule Mnemonics*
- *Converting the Rule Names*
- *Viewing the Rule Help*

Enabling the Support for Rule Mnemonics

By default, the support for rule mnemonics is disabled. To enable this feature, specify the *enable_rule_mnemonic* option in the project file, as shown below:

```
set_option enable_rule_mnemonic <yes | no | 0 | 1>
```

When this option is enabled, SpyGlass uses the custom names specified in the rules_mapping.csv file, throughout the SpyGlass run.

The mapping of the existing SpyGlass rules and the rule mnemonics is specified in the rules_mapping.csv file, which is available at the following location:

```
SPYGLASS_HOME/auxi/rules_mapping.csv
```

NOTE: *If the enable_rule_mnemonic option is enabled, you must only specify the rule mnemonic throughout the SpyGlass run. That is, you must use the rule mnemonics in all the input files, such as, project files, Tcl files, and so on. Otherwise, SpyGlass will not process the rule.*

Following is the format for a rules_mapping.csv file:

```
<existing_rule_name>,<descriptive_rule_name>,<product_name>,
"<comment>"
```

Converting the Rule Names

To ease the process of replacing SpyGlass rule names with the corresponding rule mnemonics in the input files, you can use convert_rule_names.pl file, which is available at the following location:

```
SPYGLASS_HOME/auxi/convert_rule_names.pl
```

This script uses the rules_mapping.csv file to check the existing rule mapping and replaces the SpyGlass rule names with the rule mnemonics accordingly.

Specify the following command to run the convert_rule_names.pl script:

```
perl SPYGLASS_HOME/auxi/convert_rule_names.pl  -i
<input_file>  -o <output_file>
```

Where,

- `<input_file>` refers to any input file in the SpyGlass run, such as, .prj file.
- `<output_file>` refers to the output file, containing the converted rule names.

For more information on the covert_rule_names.pl script, specify the following command:

```
perl SPYGLASS_HOME/auxi/convert_rule_names.pl  --help
```

Viewing the Rule Help

You can view the help of the SpyGlass rules and the corresponding rule mnemonics using the following ways:

- **HTML Help**: Enables you to search for a rule mnemonic and displays the help for the corresponding rule
- **The spyexplain Utility**: Enables you to view the short and long help for both the SpyGlass rules and rule mnemonics

Processing Messages and Displaying Reports

Based on the selected goals, SpyGlass reports appropriate violation messages to indicate various design issues. These messages are written to in report files and SpyGlass log file.

About Rule Severity

SpyGlass supports two levels of rule severities - severity label and severity class.

Each SpyGlass rule has a severity label and each severity label is classified under one of the predefined severity classes. Severity labels are product-specific and severity classes are predefined for SpyGlass.

After SpyGlass analysis run, a rule-checking summary is printed that reports total rule messages found under each rule severity-label and total rule messages found under each predefined rule severity-class that is, the sum of total rule messages found under each rule severity-label classified under that rule-severity class.

Predefined Rule Severity-Classes

SpyGlass supports the following five rule severity-classes:

FATAL Rule Severity-Class

The FATAL rule severity-class is primarily used for a stopper situation related to design rule-checking. Occurrence of a rule message of this class aborts further rule-checking by SpyGlass and requires immediate attention from the user so that rule-checking can be resumed.

Examples of messages of the FATAL rule severity-class are as follows:

1. Syntax Error in the input design source

   Almost all syntax errors cause immediate termination of further rule-checking by SpyGlass since syntax error indicates inability to create a consistent design view that is required by later stages of rule-checking by SpyGlass. Hence, such messages are identified as FATAL class messages. However, certain simulation-related syntax errors may be classified as non-FATAL messages because such errors do not affect the synthesis-oriented design view that is required by SpyGlass even when the constructs are syntactically incorrect as per LRM specifications.

2. Custom product assumptions about the design that are found to be invalidated at runtime

   One example of such assumption is when a custom product requires that `assign` statements are not used in the Verilog design source. Then, there would be a rule in the product that will run first and flag all uses of `assign` statements. If this rule is violated, that is, if such statement are actually found in a design, further rule-checking cannot be performed because all subsequent rules do not take the Verilog `assign` statement functionality into account.

In a large majority of SpyGlass runs, FATAL rule severity-class messages are not expected. Occurrence of such messages indicates an interrupted run and would normally require re-run of SpyGlass after the reported problem has been fixed.

A rule of FATAL rule severity-class is always run completely. However, if a rule message is found for a rule of FATAL rule severity-class, no further rule-checking is performed.

ERROR Rule Severity-Class

The ERROR rule severity-class is normally used to indicate a design error that would cause design functionality to be compromised. Normally, such messages indicate an immediate bug in the design from the perspective of design-integrity aspect being analyzed by the SpyGlass run. Presence of such ERROR rule severity-class messages require user to fix the design after suitable analysis. Applying waiver on an ERROR rule severity-class message may require detailed justification and approval in the SpyGlass use-model adopted by a typical user.

In contrast, other lower-precedence messages indicate only a potential error situation or non-compliance with design development guidelines that do not directly affect design functionality.

Normally a few of these ERROR rule severity-class messages would be expected on a typical in-development design. For a known good design, ERROR rule severity-class messages will be few and typically report only those design errors that are not verifiable by traditional simulation and other verification procedures.

WARNING Rule Severity-Class

The WARNING rule severity-class is the next less-severe message class after the ERROR rule severity-class. As described earlier, a key characteristic of such messages is that these messages indicate only a potential error situation or non-compliance with design development guidelines that do not directly affect design functionality.

Therefore, you can apply selective waivers on such messages with relatively more freedom (and confidence) than say, an ERROR rule severity-class message.

However, from SpyGlass perspective, such messages still carry a hint of caution. Therefore, you should not ignore such messages without proper analysis.

INFO Rule Severity-Class

The INFO rule severity-class represents all SpyGlass rule-checking output that is of informative nature. Such output either may be general design statistics or may represent any kind of design query that a user/product may wish to perform.

Hence, while the count of FATAL, ERROR, or WARNING rule severity-class messages can be used as a measure of design quality, reporting of INFO rule severity-class output is not expected to be a measure of design quality. Instead, it will depend on the design size and amount of design statistics and query data that a user is intending to extract.

DATA Rule Severity-Class

The DATA rule severity-class represents SpyGlass output not belonging to any of other rule severity-classes. Such cases may be secondary data to debug any of preceding rule messages or may represent information that is not reported as part of usual SpyGlass message reports or GUI display.

Typically, you would not be aware of this output. Such output is intended for better diagnostics and usability support within SpyGlass environment. You should not assume anything about the existence or other details of this data.

Rule messages of DATA severity class are not displayed in SpyGlass reports except in the *Session Log Page* of *The Message Window*, where the total reported messages are categorized according to the above-predefined rule severity classes (a category is shown in the Session Log Page only if any message is reported for that category).

SpyGlass Results Summary

At the end of SpyGlass analysis run, a results summary is generated as shown in the following example:

```
------------------------------------------------------------------
Results Summary:
-----------------------------------------------------------------
Goal Run  : spyglass_cmdline_goal
                     (Rules from above goal(s) have been changed)
Command-line read  :   0 error,    1 warning,  0 information message
Design Read        :   0 error,     0 warning,  3 information messages
Found 2 top modules: cdc   (file: ../../Ac_cdc08/cdc.v)
                     ovl   (file: ../../Ac_cdc08/ovl.v)

** black box Resolution: 50 errors, 0 warning, 0 information message
SGDC Checks        :      0 error,  3 warnings, 0 information message
Policy clock-reset :     1 error,  4 warnings, 0 information message
-----------------------------------------------------------------
Total              :    51 errors, 8 warnings, 3 information messages

Total Number of Generated
Messages                    : 63 (51 errors, 8 warnings, 4 Infos)
Number of Waived Messages   : 1  (0 error, 0 warning, 1 Info)
Number of Reported Messages : 62 (51 errors, 8 warnings, 3 Infos)

NOTE: It is recommended to first fix/reconcile fatals/errors
reported on lines starting with ** as subsequent issues might be
related to it.
Please re-run SpyGlass once ** prefixed lines are fatal/error
clean.


-----------------------------------------------------------------
SpyGlass Exit Code 0 (Rule-checking completed with errors)
```

The results summary contains the following information:

- Count of FATAL, ERROR, WARNING, and INFO severity class messages.

The count of FATAL severity class message is displayed first in the results summary. This column is not printed if there are no FATAL severity-class messages.

If there is no message for any of the ERROR, WARNING, and INFO severity classes, the corresponding count set is reported as zero.

- Count of total number generated messages, the waived messages, and the reported messages is displayed as in the following example:

```
Total Number of Generated Messages : 3 (2 errors, 0 warning, 1 Info)
Number of Reported Messages        : 3 (2 errors, 0 warning, 1 Info)
Number of Waived Messages          : 1(0 errors, 1 warning, 0 Info)
```

If you have set a rule message-reporting limit by using the *lvpr* command and some messages are suppressed, the count of suppressed messages is reported. An example is shown below:

```
...
Number of Overlimit Messages :  25
(2 errors, 11 warnings, 12 Infos
...
```

- SpyGlass exit status

For details, see *SpyGlass Exit Status*.

SpyGlass Exit Status

For better integration with other stream tools, SpyGlass generates exit status code that gives you the exact status of SpyGlass run.

By default, SpyGlass reports an exit code of 0 for a successful run and prints one of the following messages:

- `SpyGlass Exit Code 0 (Rule-checking completed without errors or warnings)`
- `SpyGlass Exit Code 0 (Rule-checking completed with warnings)`
- `SpyGlass Exit Code 0 (Rule-checking completed with errors)`
- `SpyGlass Exit Code 0 (Informational command executed, rule-checking not done)`

You can set the *enable_pass_exit_codes* command to `yes` in a project file to report different exit codes for each of the above cases depending on the type of message generated in the current run. For details, see *Messages Printed When SpyGlass Run is Complete*.

NOTE: *Waived messages are not considered while deciding the exit status. Only reported messages are considered.*

SpyGlass may exit abruptly if it is unable to set the stack size to unlimited.

By default, Spyglass internally sets the stack size to unlimited. However, if it is unable to do so due to insufficient permissions then the following warning message appears:

```
WARNING: Stacksize could not be set as unlimited through ulimit
command.This may lead to nondeterministic tool behavior
```

In such cases, contact system administrator to check for permissions to change stack size.

Messages Printed When SpyGlass Run is Complete

This section describes the exit codes when the SpyGlass run is complete.

These exit codes appear when you set the *enable_pass_exit_codes* command to `yes`. If you set this command to `no`, SpyGlass reports the default exit code of 0 in place of these exit codes, as described below.

- `SpyGlass Exit Code 0 (Rule-checking completed without errors or warnings)`

  The above message appears when a SpyGlass run is complete without any error or warning messages.

  Typically, this situation indicates that your design is clean with respect to the executed goals.

- `SpyGlass Exit Code 11 (Rule-checking completed with warnings)`

  The above message appears when a SpyGlass run is complete without any error message but with warning messages.

  This status indicates that some rules of the warning severity are reported. See *WARNING Rule Severity-Class* for understanding and handling warning messages.

- `SpyGlass Exit Code 12 (Rule-checking completed with errors)`

  The above message appears when a SpyGlass run is complete with error messages (and possibly warning messages).

  This status indicates that some rules of the error severity are reported. See *ERROR Rule Severity-Class* for understanding and handling error messages.

- `SpyGlass Exit Code 20 (Informational command executed, rule-checking not done)`

  The above message appears if you have executed an informational option of SpyGlass.

  Typically, no further action is required for this exit status.

Messages Printed When SpyGlass Run is not Complete

This section describes the exit codes when the SpyGlass run is not complete.

SpyGlass reports these exit codes irrespective of the value of the *enable_pass_exit_codes* command.

- `SpyGlass Exit Code 1 (Abnormal termination - termination not trapped by software)`

  The above message appears when SpyGlass run is terminated because of an abnormal error that is not trapped by SpyGlass.

  This exit status indicates some operating system-related problem, such as sack overflow and out of memory issue.

  In such cases, the SpyGlass log file may also be incomplete as SpyGlass was not able to trap an error signal and report suitably in the log file.

  In such cases, check available machine resources and do the required correction. If the problem persists, report the problem to Atrenta Support.

- `SpyGlass Exit Code 3 (Abnormal termination trapped by software)`

  The above message appears when SpyGlass run is terminated because of an error that is trapped by SpyGlass.

  This exit status indicates that some operating system-related problem, such as segmentation fault or memory corruption has occurred but SpyGlass was able to trap the error signal and report suitably in log file.

  In such cases, check the gdb trace and stack trace printed in the SpyGlass log file and take corrective action.

  After SpyGlass traps the error signal, it tries to generate the moresimple report that contains error messages and/or rule violations reported before the end of run. If the moresimple report is generated, it may contain useful information, which may help in debugging the situation. Along with the error details given in the SpyGlass log file, send the moresimple report to Atrenta Support.

- `SpyGlass Exit Code 4 (License failure, rule-checking aborted)`

  The above message appears when a SpyGlass run is terminated because of a license failure.

  Check the license status and take a corrective action.

- `SpyGlass Exit Code 5 (Rule-checking interrupted by User)`

  The above message appears when you forcibly terminate a SpyGlass run by killing the corresponding process.

In this case, SpyGlass-generated results may be incomplete and should not be used.

- `SpyGlass Exit Code 6 (Rule-checking terminated due to FATAL errors - design syntax error)`

  The above message appears when a SpyGlass run is terminated because of a fatal design error (syntax errors). See *FATAL Rule Severity-Class* for understanding and handling fatal messages.

  In this case, check design inputs and take a corrective action.

- `SpyGlass Exit Code 7 (Rule-checking terminated due to FATAL errors - usage or run error)`

  The above message appears when a SpyGlass run is terminated because of incorrect usage and incorrect/incomplete inputs. See *FATAL Rule Severity-Class* for understanding and handling fatal messages.

  In this case, check the specified inputs and take a corrective action.

- `SpyGlass Exit Code 8 (Design database save failure, rule-checking aborted)`

  The above message appears when a design cannot be saved because of reasons, such as incorrect save database directory.

  In this case, rectify the save and restore-related commands.

- `SpyGlass Exit Code 137 (run killed by user, or by system due to lack of resources like memory etc.)`

  The above message appears when a signal is sent to SpyGlass process to terminate the session immediately.

  The signal is sent either by the user or by the operating system because of lack of resources, such as memory.

  In this case, run SpyGlass on a machine that has higher memory.

- `SpyGlass Exit Code 153 (File size limit exceeded)`

  The above message appears when a signal is sent to SpyGlass process to terminate the session if size of files generated by SpyGlass exceeds the maximum limit allowed by the operating system.

  In this case, increase the maximum allowed size of a file to enable SpyGlass process to complete successfully.

Working with Input Design and Libraries

This section describes all aspects of reading a design in SpyGlass. This includes reading in design HDL and technology libraries, understanding and debugging results, and dealing with special HDL aspects, such as pragmas.

Working with Precompiled Libraries

SpyGlass provides the feature of precompiling Synopsys Liberty™ files (.lib files) to SpyGlass-compatible format library files (.sglib files) that you can use as an input for SpyGlass analysis.

The .sglib files contain cell information, including functional view for library cells. Precompiling .lib files to .sglib files enables you to check and fix errors in libraries before using them.

NOTE: *SpyGlass-compatible format library files (.sglib files) are specific to a SpyGlass release. You must recompile libraries for each version. SpyGlass provides a feature of automatically compiling libraries at the time of analysis itself. For details, see Automatically Compiling Gate Libraries.*

Advantages of Using Precompiling Libraries

Precompiling libraries provide the following benefits:

- By compiling sub-blocks, you can find syntax issues and fix them more quickly than running the entire design together.
- You can share HDL blocks and code with other blocks without involving the actual source code.
- If multiple SpyGlass licenses are available, precompilation can reduce the overall runtime of SpyGlass by parallelizing the design-read.

When you precompile an HDL file into a library, SpyGlass creates a library directory of design blocks. Higher-level blocks can later refer to these design blocks.

Specifying Modes in Which Libraries Should be Compiled

You must compile each version on a supported platform of a corresponding architecture and use them on all supported platforms of the same architecture. For example, you can compile your VHDL libraries on 64-bit Solaris platform and use them on 64-bit Solaris or Linux platforms.

By default, SpyGlass compiles libraries and runs in 64-bit mode.

Compiling HDL Files into a Library

The process of compiling HDL files into a library consists of the following tasks:

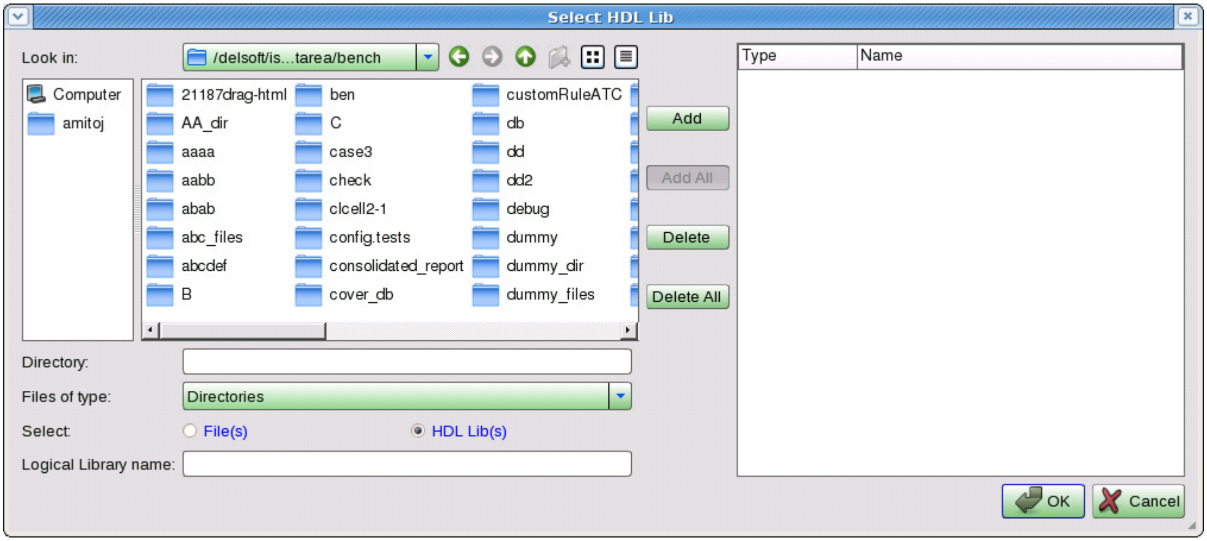
1. *Defining a Logical Library*
2. *Including HDL Files in the Logical Library*
3. *Generating a Precompiled Library*

Defining a Logical Library

To define a logical library, perform the following steps:

1. Right-click in the *HDL Libraries* section under the *Add Design Files* tab, and select the *Add HDL Lib File(s)* option from the shortcut menu.

   The *Add File(s)* dialog appears, as shown in the following figure:



Look in: /delsoft/is...tarea/bench

Directory:

Files of type: Directories

Select: ○ File(s) ● HDL Lib(s)

Logical Library name:

FIGURE 136. Add File(s)

2. Specify the logical library name in the *Logical Library Name* textbox.

   This step is equivalent to specifying the following command in the project file:

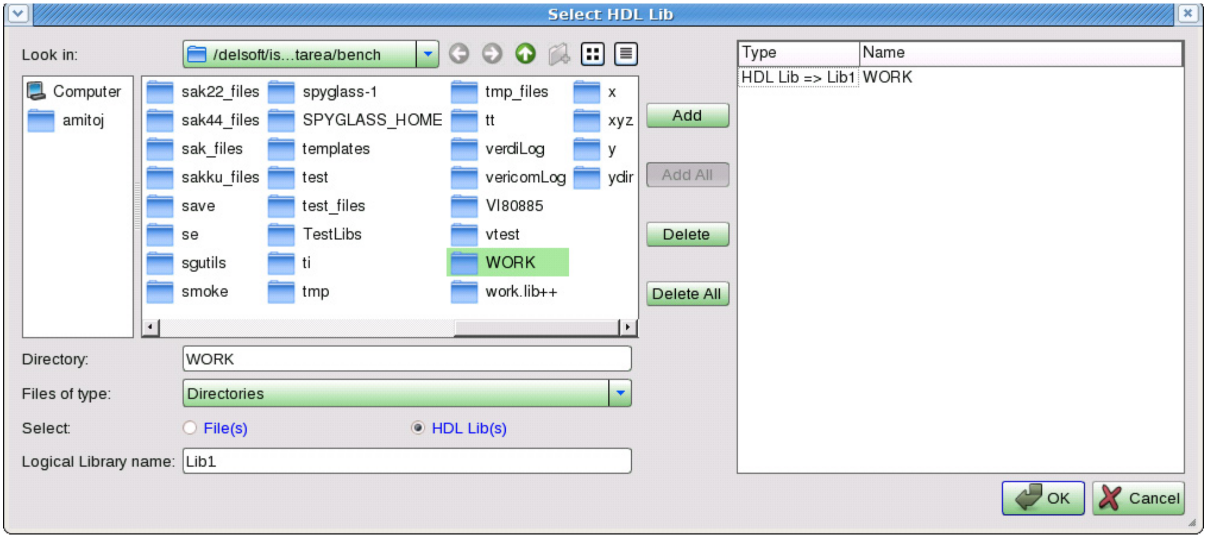
```
set_option work <value>
```

By default, the *<current-working-dir>*/WORK directory is the working directory.

The 32-bit or 64-bit version of user-compiled libraries is created in sub-directories, 32 or 64, respectively, under the specified working directory.

3. Click ⬇ to browse to the physical working directory corresponding to the specified logical library name.
4. Select the required physical working directory.
5. Click the *Add* button.

   A mapping between a logical library to a physical working directory appears in the right-most section, as shown in the following figure:



Look in: /delsoft/is...tarea/bench

Directory: WORK

Files of type: Directories

Select: ○ File(s) ● HDL Lib(s)

Logical Library name: Lib1

Type	Name
HDL Lib => Lib1	WORK

FIGURE 137. Add File(s)

6. Click the *OK* button.

   The *HDL Libraries* section now appears, as shown in the following figure:

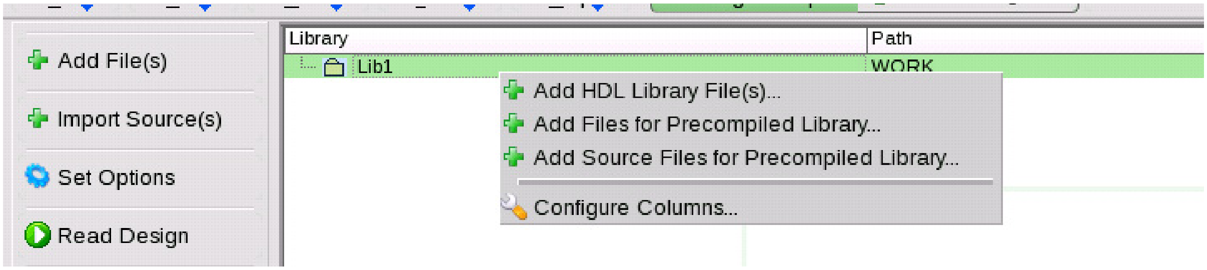

Add File(s)
Import Source(s)
Set Options
Read Design
Library
Path
Lib1
WORK
Add HDL Library File(s)...
Add Files for Precompiled Library...
Add Source Files for Precompiled Library...
Configure Columns...

FIGURE 138. HDL Libraries

This task is equivalent to specifying the following command in the project file:

```
set_option lib <library-name> <working-directory>
```

Please note the following points:

- The logical library is a library used while creating a precompiled library and a physical library refers to the complete path.
- You must create a library mapping correctly. The most common issues with library compilation are related to mapping, and typically these are detected at a later stage when the libraries are used.
- You can specify the same physical path for multiple logical libraries. You cannot, however, map a logical library to multiple physical paths.
- You can map multiple logical libraries to a single intermediate library, which can then allow you to change library bindings at runtime. For details, refer to the *Compiling Libraries in Mixed-Language Designs* topic.

Including HDL Files in the Logical Library

To include HDL files in the logical library, perform the following steps:

1. Right-click on any logical library (*Figure 139*).

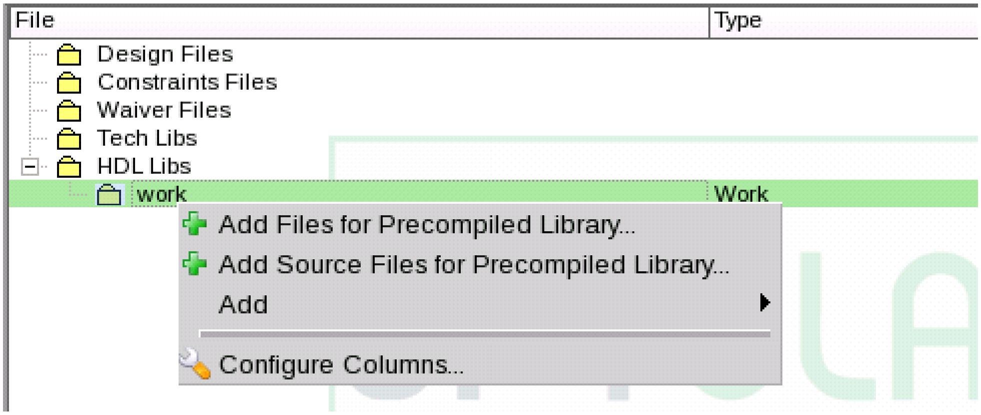


FIGURE 139. Right-Click Men for Logical Library

2. Select **Add Files for Precompiled Library** option from the right-click menu.

   The **Open File** dialog box is displayed.

3. Select the file to be precompiled.

   The selected file is displayed in the File pane under the corresponding logical directory as shown in *Figure 140*.

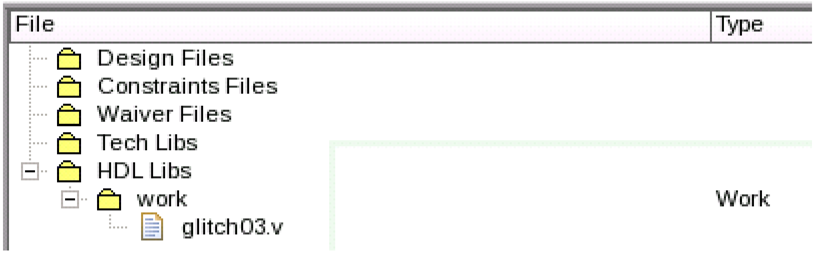


FIGURE 140. Precompile File Mapping

NOTE: *Please note the following points:*

- *Ensure that you define libraries in the order they are used. Undefined dependencies or cyclic dependencies result in errors. For example, if a design unit is instantiated by another design unit, you must first define the lower-level design unit.*
- *The VHDL sort function does not affect the order of precompiled files.*

Tcl Command

You can also add the HDL files in a logical library using the following Tcl command:

```
set_option libhdlfiles <library-name> {space-separated file
list}
```

Including Source Files in a Library

To include HDL files in the logical library, perform the following steps:

1. Right-click on any logical library (*Figure 139*).
2. Select **Add source files for Precompiled Library** option from the right-click menu.

   The **Open File** dialog box is displayed.
3. Select the file to be precompiled.

   The selected file is displayed in the File pane under the corresponding logical directory.

Alternatively, you can add source files in a library by specifying the following Tcl command:

```
set_option libhdlf <library-name> {space-separated source
file list}
```

Enabling Elaboration

If you want to enable elaboration during the precompilation process, specify the following command in the project file:

```
set_option elab_precompile yes
```

Generating a Precompiled Library

To generate a precompiled library, perform the following steps:

1. Click the *Run Design Read* tab.
2. Click the *Run Design Read* option.

Once the analysis is complete, the precompiled files are stored in the directory mentioned in the *HDL Libraries* section.

Alternatively, you can generate a precompiled library through a project file by specifying the following command while invoking SpyGlass:

```
spyglass -project mylib.prj -batch -designread
```

By default, SpyGlass writes the compiled library into a default directory, WORK, in the current working directory.

NOTE: *Please note the following points:*

- *Ensure that the lower-level precompiled libraries are syntactically clean and error-free before using them at higher level.*
- *Fatal issues found during precompilation of a library aborts the run, and subsequent libraries are not compiled.*

Automatically Compiling Gate Libraries

A compiled library format is specific to a particular SpyGlass release. Therefore, you must compile libraries in every release.

You can automatically compile libraries (.lib files) to SpyGlass-compatible format (.sglib files) in any of the following ways:

- *Using the GUI to Automatically Compile Libraries*
- *Using the enable_gateslib_autocompile Option*
- *Using the force_gateslib_autocompile Option*
- *Using the AUTOENABLE_GATESLIB_AUTOCOMPILE Key*

When you perform automatic compilation, SpyGlass compiles .lib files into .sglib files unless there is an up-to-date copy in the cache directory. For details on specifying a cache directory, see *Specifying a Cache Directory*.

You can also forcefully compile libraries so that SpyGlass always compiles the .lib files and overwrite the existing .sglib files present in the cache directory. For details, see *Using the force_gateslib_autocompile Option*.

Using the GUI to Automatically Compile Libraries

To automatically compile gate libraries through GUI, perform the following steps:

1. Click the *Set Read Options* tab under the *Design Setup* tab.
2. Set the *Enable auto-compilation of gateslib into sglib* option to *Yes*.
3. Click the *Run Design Read* tab.
4. Click the *Run Design Read* option to run the design read process.

Every such run generates the following:

- A SpyGlass-compatible format library file, <libfile-name>.sglib

  For example, the R123.lib library file is converted into R123.sglib. Similarly, the a45.slflib library file is converted into a45.sglib.

- Following SpyGlass output files:
  - The spyglass_lc_<libfile>.log file and the name spyglass_lc_<libfile>.vdb file
  - Standard SpyGlass automatic report (moresimple report)

Fixing Violations After Running Design Read

After the design read process, if SpyGlass reports violations for a library, fix the violations by:

- Checking the automatic report.
- Referring to the log file.

However, if the library compiles without errors, use the generated .sglib file for SpyGlass analysis. For details, see *Specifying Precompiled Libraries for SpyGlass Analysis*.

Using the enable_gateslib_autocompile Option

Specify the following command in the project file to compile gate libraries automatically:

```
set_option enable_gateslib_autocompile yes
```

Along with the above command, you can also specify gate libraries (.lib files) and .sglib files by using the gateslib and sglib options, respectively, of the read_file project file command.

However, if you do not set the enable_gateslib_autocompile command to yes but specify .lib files and .sglib files together, SpyGlass reports an error message. In this case, perform any of the following actions:

- Compile the .lib file to .sglib file by using the spyglass_lc utility and then specify the generated .sglib file by using the following project file command:

  ```
  read_file -type sglib <sglib-file>
  ```

- Specify the following project file command:

  ```
  set_option enable_gateslib_autocompile yes.
  ```

Using the force_gateslib_autocompile Option

To compile gate libraries forcefully, specify the following command in the project file:

```
set_option force_gateslib_autocompile yes
```

Using the AUTOENABLE_GATESLIB_AUTOCOMPILE Key

You can set the value of the AUTOENABLE_GATESLIB_AUTOCOMPILE configuration key to yes or yes_forced.

Setting the value of this key to yes is equivalent to specifying the enable_gateslib_autocompile option. Similarly, setting the value of this key to yes_forced is equivalent to specifying the force_gateslib_autocompile option.

Specifying a Cache Directory

By default, SpyGlass compiles gate library in the spyglass_cache directory. You can specify a different directory by specifying the following command in the project file:

```
set_option cachedir <directory-name>
```

If the specified cache directory does not exist, SpyGlass creates that directory (only the leaf most directory). The cache directory allows you to reuse the results of any previous .lib compilation done in the same SpyGlass run. It holds only single .sglib file corresponding to the .lib files used in the last SpyGlass run.

Conditions for Auto-Compilation of Gate Libraries

It is possible that in subsequent SpyGlass runs, you use the `enable_gateslib_autocompile` option to compile gate libraries that you had already compiled earlier by using this option.

In this case, auto-compilation of such gate libraries occurs only if any of the following conditions hold true:

- md5sum of any of the specified .lib file has changed.
- Order of .lib file specification on command-line has changed.

  SpyGlass performs this check only when the specified libraries contain duplicate cells across different libraries.

NOTE: *SpyGlass ignores duplicate cells within the same library. However, such cells are retained across different libraries and the checkDupCells rule flags a violation for such cases.*

- SpyGlass version has changed

If all the above conditions are false, the already compiled .sglib files are considered to be up to date in the cache directory. Therefore, no recompilation occurs in such case and the existing .sglib libraries are picked from the cache directory.

If you forcefully compile gate libraries (discussed in the next topic), SpyGlass does not evaluate the specified conditions. In this case, the .lib files are always compiled irrespective of what is present in the cache directory.

If automatic compilation of technology libraries is successful, the AutoGenerateSglib rule reports an appropriate message. The severity of this message is “Error,” if there are errors in the library compilation run. Otherwise, its severity is “Info.” You can also view the moresimple.rpt report of the library compiler run in GUI by clicking the AutoGenerateSglib rule message.

NOTE: *The AutoGenerateSglib rule does not report a violation when the cache directory of auto compilation is reused or auto compilation has failed due to fatal violations in the library compiler run.*

Built-in VHDL Libraries That Do Not Require Any Mapping

SpyGlass VHDL environment comes with the following precompiled logical libraries:

- IEEE
- STD
- SYNOPSYS

There is no need to provide mapping for the above-mentioned libraries.

Precompiling Verilog Libraries

To precompile Verilog library files, perform the following steps:

1. Specify Verilog files by using the `read_file` command in the project file, as shown in the following example:

   ```
   read_file -type verilog RTL/top.v
   ```

2. Specify the following command in the project file:

   ```
   set_option enable_precompile_vlog yes
   ```

3. Specify a logical working directory in which SpyGlass should generate the precompiled library by using the following command in a project file:

   ```
   set_option work <directory-name>
   ```

4. Specify a mapping of logical library name to the physical library path by using the following command in a project file:

   ```
   set_option lib <logical-name> <physical-path>
   ```

5. Run the design-read process. For details, see *Running Design Read*.

NOTE: *Please note the following points:*

- *When you precompile a Verilog module that contains gates instances, SpyGlass does not compile those gate instances. You should compile gates library separately into a SpyGlass-format gates library (*.sglib *file).*

- *Modules in the library files specified by using the v and y options of the* `set_option` *command are compiled along with the design modules.*
- *Verilog modules compiled with one version of SpyGlass may not be compatible with another version of SpyGlass and may require recompilation with the other version.*

Naming and Mapping Verilog Libraries

Modules or User-Defined Primitives (UDPs) missing in your Verilog source code are normally present in a single library file or in files stored in a library directory.

You must specify where to find the library by providing either of the following:

- Verilog library file names by specifying the following command in the project file:

  ```
  set_option v {space-separated list of lib names}
  ```

- Directory containing libraries by specifying the following command in the project file:

  ```
  set_option y { space-separated path names of directories }
  ```

SpyGlass first checks the current directory for the specified libraries. If the specified libraries are not present in the current directory, SpyGlass searches in the specified path.

Like all standard Verilog EDA tools, SpyGlass requires you to specify the file extension for files located in library directories specified by using the v or y options. You can specify library file extension by specifying the following command in the project file:

```
set_option libext {space-separated list of extensions}
```

Structure of Precompiled Verilog Libraries

The following points describe the structure of a precompiled Verilog library:

- For each precompiled Verilog module, SpyGlass creates a sub-directory, <module-name>.mod.

- Each such sub-directory has the corresponding <module-name>.dmp file, which is a binary dump of the module.
- The module sub-directory also has a <module-name>.dep file, which has dependency information.
- If you choose to encrypt the Verilog modules, an additional file, .encrypt, is also created in the module's sub-directory.

NOTE: *UDPs are also precompiled and used by SpyGlass.*

NOTE: *Do not mix encrypted and un-encrypted modules in the same library.*

Library Searching Mechanism

To search for a library, SpyGlass performs a case-sensitive search in the following order:

1. Library defined by the 'uselib directives
2. All Verilog libraries specified by using the v/y option of the `set_option` command
3. Work library
4. Libraries listed by using the lib option of the `set_option` command. SpyGlass searches libraries in the order specified by this command

Working with Precompiled Verilog Libraries in Mixed Language Mode

Consider the following mixed-language design:

```
//top.v
module top;
  middle m1();
endmodule

--middle.vhd
entity middle is
end middle;

architecture mid of middle is
begin
  M1: entity work.bottom(module);
```

```
end mid;

//bottom.v
module bottom;
endmodule
```

In this example, Verilog module top instantiates VHDL DU middle that, in turn, instantiates Verilog module bottom.

To perform a multiple step compilation, perform the following steps:

1. Compile bottom.v.

   ```
   set_option work mylib1
   set_option lib mylib1 ./MYLIB1
   set_option libhdlfiles mylib1 {bottom.v}
   set_option elab_precompile yes
   ```

2. Compile middle.vhd.

   ```
   set_option work mylib1
   set_option lib mylib1 ./MYLIB1
   set_option libhdlfiles mylib1 {middle.vhd}
   ```

3. Invoke SpyGlass on top.v.

   ```
   set_option work mylib1
   set_option lib mylib1 ./MYLIB1
   set_option libhdlfiles mylib1 {top.v}
   set_option elab_precompile yes
   ```

Different instantiations in this example are resolved as follows:

- Search the design unit named `middle` in the source file, top.v.
- Search in the WORK library (and any other libraries specified by the lib option) among Verilog DUs, as the design unit `middle` does not exist in top.v.
- Search VHDL DUs in the WORK library, as the design unit named `middle` does not exist in Verilog DUs in the specified libraries.
- The design unit named `middle` is found in the VHDL DUs and is resolved.
- Search among VHDL source files, as the VHDL design unit named `middle` contains an instantiation of DU named `bottom`.

If not found, search VHDL DUs in the WORK library (and any other libraries specified with the lib option of the set_option command).

- Search Verilog source files, as the design unit named bottom does not exist in VHDL DUs in the specified libraries.

  If not found, search the Verilog DUs in the WORK library (and any other libraries specified with the lib option of the set_option command).

  The design unit named bottom is found in the Verilog DUs and is resolved.

Please note the following:

- You do not need to compile the bottom.v file (Step 1 above). Just supply it using the v option as follows:

  ```
  set_option work mylib1
  set_option lib mylib1 ./MYLIB1
  set_option libhdlfiles mylib1 {top.v}
  set_option elab_precompile yes
  set_option v bottom.v
  ```

  Then, the bottom.v file is also searched in addition to the Verilog DUs in the WORK library (and any other libraries specified with the lib option of the set_option command). Thus, the DU named bottom is found and resolved. Once the bottom.v file is analyzed, the DUs in the file are also compiled and stored in the WORK library for future use.

- You can also compile and store all the DUs of the above example in a single command as follows:

  ```
  set_option work mylib1
  set_option lib mylib1 ./MYLIB1
  set_option libhdlfiles mylib1 {top.v middle.vhd}
  set_option elab_precompile yes
  set_option v bottom.v
  ```

  Then, all DUs are compiled and stored in the mylib1 directory.

  Now, suppose you have used the precompiled module named top in another design file named mytop.v. You can compile the complete hierarchy as follows:

  ```
  set_option work mylib2
  set_option lib mylib1 ./MYLIB1
  ```

```
set_option lib mylib2 ./mylib2
set_option libhdlfiles mylib1 {mytop.v}
set_option elab_precompile yes
```

The above command would use the set_option lib mylib1 ./mylib1 part to find and bind the instantiation of the DU named `top`.

- It is not required to specify the lib command for various parts of the sub-hierarchy. Instantiation information is picked from the .dep file for each compiled DU.

Support for Foreign Attributes

SpyGlass supports foreign attributes in the following syntax:

```
ATTRIBUTE FOREIGN OF <architecture-name> :
  ARCHITECTURE IS "VERILOG : <module-name>
       -lib <library-name>";
```

The architecture containing a foreign attribute is not a part of the design hierarchy. Therefore, the tool does not elaborate and synthesize the architecture if search for Verilog master is successful.

If you have not specified any Verilog library name, then SpyGlass searches for Verilog master in current Verilog source files only.

Specifying Verilog Libraries by Using the 'uselib Statement

The 'uselib statement is used to specify a Verilog source library file or a directory in which SpyGlass should search for definitions of modules or UDPs instantiated in a design.

You can use the 'uselib statement in the following ways:

1. Specify the source library file directly using the following syntax:

   ```
   'uselib file=<file-name>
   ```

   Where <file-name> is the name of the source file containing the module/UDP description.

2. Specify the directory containing the source library file (and the file extension) using the following syntax:

   ```
   'uselib dir=<dir-name> libext=<ext-list>
   ```

Where <dir-name> is the name of the directory containing the source library files and <ext-list> is the plus character-separated list of file extensions (including the dot[.] character).

```
'uselib dir=/usr/john/myvlibs libext=.v+.vlog+.vlg
```

3. Specify a precompiled library using the following syntax:

```
'uselib lib=<lib-name>
```

   Where <lib-name> is the logical name of the precompiled Verilog library containing the module/UDP description.

NOTE: *In this case, you also need to specify the precompiled Verilog library to SpyGlass as described in the Precompiling Verilog Libraries topic.*

Compiling Libraries in Mixed-Language Designs

Your design may contain VHDL design units instantiated in Verilog modules or Verilog modules instantiated in VHDL design units. The following topics describe steps to compile libraries in such cases:

- *VHDL Library Design Units Instantiated in Verilog Modules*
- *Verilog Modules Instantiated in VHDL Design Units*

VHDL Library Design Units Instantiated in Verilog Modules

To analyze mixed-language designs with VHDL design units instantiated in Verilog modules, perform the following steps:

1. Specify the *Language Mode* as mixed under the *Set Read Options* tab.
2. Precompile VHDL design units into a library as described in the *Compiling HDL Files into a Library* topic.
3. Specify Verilog source files and VHDL library compiled earlier under the *Add Design Files* tab.
4. Run the design read process.

Verilog Modules Instantiated in VHDL Design Units

To analyze mixed-language designs with Verilog modules instantiated in VHDL design units, perform the following steps:

1. Specify the *Language Mode* as mixed under the *Set Read Options* tab.
2. Precompile Verilog libraries as described in the *Precompiling Verilog Libraries* topic.
3. Specify VHDL source files and Verilog library under the *Add Design Files* tab.
4. Run the design read process.

Searching Master Instance in Mixed-Language Mode

While working in the mixed-language mode with precompiled gate libraries, SpyGlass searches for the master of an instance in the following order:

1. First searches in the parent domain.

   For example, if the instance is in the Verilog source file, SpyGlass searches for a master in the Verilog domain (source files and precompiled Verilog libraries, if any).
2. Next, SpyGlass searches in the domain of the other language.

   In the above example, SpyGlass searches in the VHDL domain.
3. Next, SpyGlass searches in the SpyGlass-compatible format library files.

If the master is still not found, the instance is considered as a black box.

However, if cell definition is present in both in sglib and HDL, SpyGlass ignores the cell definition present in sglib. In addition, SpyGlass reports the IgnoredLibCells warning message that points to a report containing the source sglib name and HDL back-reference information of all the ignored library cells.

If you want to give higher preference to technology library definition present in .lib/.sglib over user-specified definition present in source HDL files, precompiled libraries, and simulation models while searching for the master of an instance, use the `set_option prefer_tech_lib yes` command in the project file.

NOTE: *When you specify the* `prefer_tech_lib` *option, then irrespective of whether a functional view exists for a .lib cell definition or not, higher priority is given to technology library definitions. If you intend to overwrite or provide functional view of the cell from HDL and use other properties of that cell from sglib, ensure that you pass HDL descriptions of that cell during library compilation stage, that is, during sglib creation.*

Debugging Issues in Gate Libraries

If it is necessary to debug a problem related to a precompiled gate library, perform any of the following actions:

- Set the enable_sglib_debug option in the *Other Command Line Options(s)* field to *Yes* under the *Set Read Options* tab.
- Specify the following command in the project file:

```
set_option enable_sglib_debug yes
```

This creates the debug_sglib report that contains the following information:

- Inferred functionality for each gate that was successfully synthesized by SpyGlass library compiler.
- Details of cells that could not be synthesized along with the reason of failure.

If you are using .sglib files in your SpyGlass run, SpyGlass generates the additional report, sglib_version_summary.rpt, which lists sglib names, SpyGlass Library Compiler version with which they were compiled, and their status. The report also lists enhancements made in the subsequent SpyGlass library compiler releases starting from the oldest version of library files used in the current SpyGlass run.

The report is generated whenever any .sglib file is present in the current SpyGlass run. But this report is not generated if you have specified the `set_option enable_gateslib_autocompile yes` command in the project file.

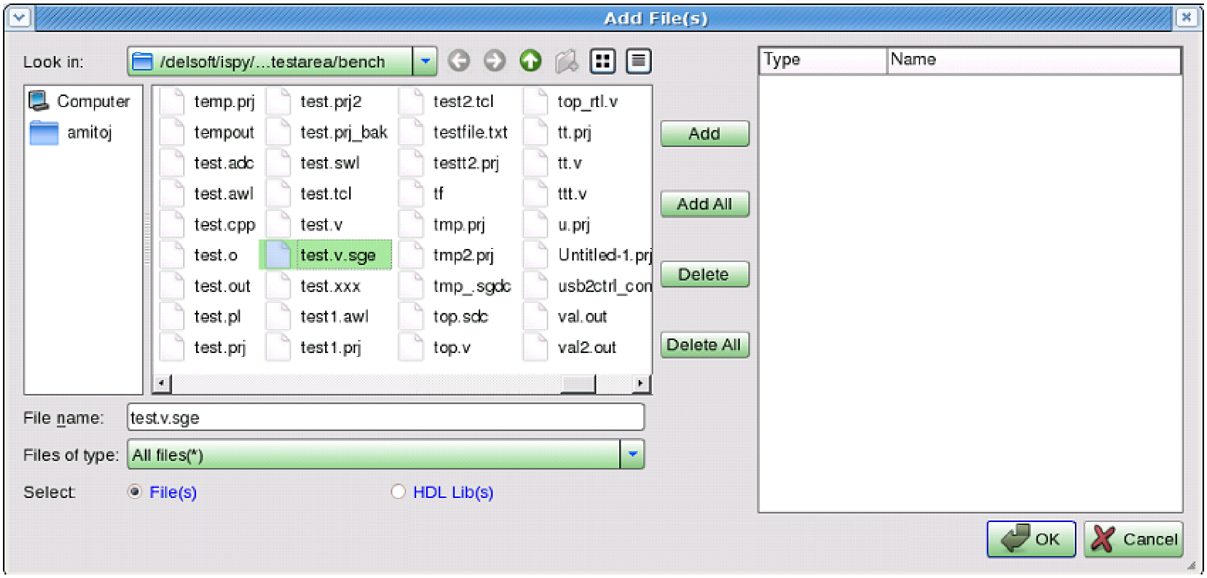
However, if you specify the `set_option enable_sglib_debug yes` command, it overrides the above specified commands and the report is generated. In addition, if you are using library files that are compiled with a library compiler version that is higher than the version of SpyGlass that you are using, SpyGlass reports a fatal violation and generates the sglib_version_summary.rpt report.

Specifying Precompiled Libraries for SpyGlass Analysis

To use a precompiled library for a higher-level block or design, perform the following steps:

1. Select the *Add Design Files* tab under the *Design Setup* tab.
2. Right-click in the area under the *Tech Libraries* section.
3. Select the *Add Tech Lib File(s)* option from the shortcut menu.

   The *Add File(s)* dialog appears, as shown in the following figure:



Add File(s)

Look in: /delsoft/ispy/...testarea/bench

File name: test.v.sge

Files of type: All files(*)

Select: ● File(s) ○ HDL Lib(s)

FIGURE 141. Specifying Precompiled Libraries

4. Select the required .sglib files from the *Add File(s)* dialog.

NOTE: *In this step, you can also specify .plib and/or .lef files.*

5. Click the *Add* button.
6. Click the *OK* button.

   The selected .sglib files appear under the *Tech Libraries* section.

Alternatively, use the following command in the project file to use the precompiled libraries in SpyGlass run:

```
read_file -type sglib <filename.sglib>
```

After specifying precompiled libraries during SpyGlass analysis, click the *Set Read Options* tab and perform the following actions:

- Specify a top-level module in the *Top Level Design Unit* field under this tab.

  Alternatively, you can specify the following command in the project file:

  ```
  set_option top <top-module>
  ```

  It is highly recommended to specify a top-level module to avoid analyzing modules in the library that are not used in the design.

- Set the value of the *Enable RTL Checking of precompiled HDL Libraries* field to *Yes* to enable the analysis of precompiled libraries.

  Alternatively, specify the following command in the project file:

  ```
  set_option hdllibdu yes
  ```

NOTE: *Specifying precompiled library paths incorrectly may result in built-in messages, such as STX_11, STX_464, WRN_384, and DetectBlackBoxes.*

Specifying Multiple Technology Libraries of the Same Name

If you specify multiple technology libraries of the same name, SpyGlass considers the first occurrence of the library and ignores the rest. In such a scenario, if you are compiling technology libraries with SpyGlass library compiler, SpyGlass reports the LIBERROR_313 violation. However, if you are using such libraries during SpyGlass run, SpyGlass reports the ReportDuplicateLibrary violation.

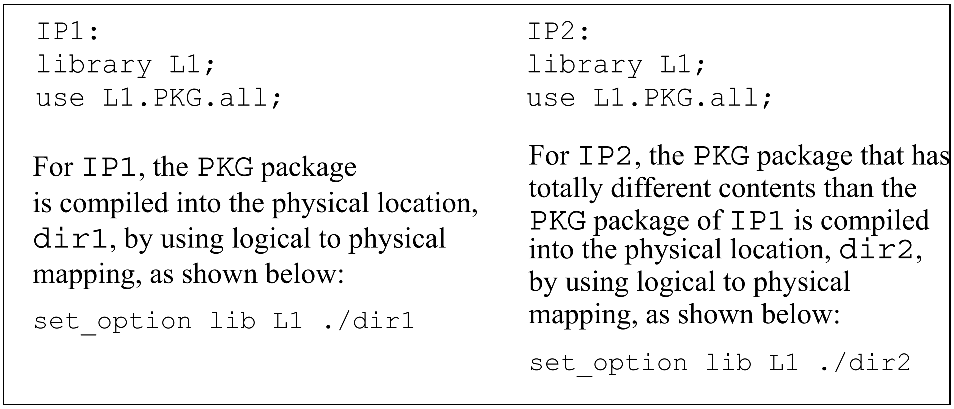
Using Intermediate Logical Library Name Support in VHDL

As discussed in the *Compiling HDL Files into a Library* topic, to use a user-defined VHDL library, you need to provide a logical to physical mapping by using the lib command, as shown below:

```
set_option lib <logical_name> <physical_path>
```

However, the above use model has a limitation in cases where two different blocks of a hierarchical design are using the same package name from a library and they are brought together to the top-level. In such cases, you can use only one library with that logical name in the top-level.

For example, consider two hierarchical blocks, IP1 and IP2, which use the same package, PKG, as shown below:



```
IP1:
library L1;
use L1.PKG.all;
```

For `IP1`, the PKG package is compiled into the physical location, `dir1`, by using logical to physical mapping, as shown below:

```
set_option lib L1 ./dir1
```

```
IP2:
library L1;
use L1.PKG.all;
```

For `IP2`, the PKG package that has totally different contents than the PKG package of `IP1` is compiled into the physical location, `dir2`, by using logical to physical mapping, as shown below:

```
set_option lib L1 ./dir2
```

The top-level design, TOP, has instances of IP1 and IP2. Now if you want to specify logical to physical mapping for the L1 library, you can specify only one mapping (that is, either L1 to dir1 or L1 to dir2), as shown below:

```
set_option lib L1 ./dir1
```

In the above case, SpyGlass picks up only one package, that is, PKG from IP1. However, the second PKG package is not available for IP2 in this case.

NOTE: *Because the contents of both the packages are different, the design is incomplete.*

Using Intermediate Library Support

You can solve the above problem (without modifying library or package names by using intermediate library support.

Intermediate library support enables you to map multiple logical libraries to a single intermediate library by using the libmap option of the `set_option` command. This intermediate library is then mapped to a physical location by using the lib option of the `set_option` command.

```
set_option libmap <logical-name> <intermediate-logical-name>

set_option lib <intermediate-logical-name> <physical-name>
```

By using the libmap option, both the IP1 and IP2 can refer to their own packages, as shown below:

```
IP1:                           IP2:
Library L1;                    Library L1;
Use L1.PKG.all;                Use L1.PKG.all;

set_option libmap L1 IP1       set_option libmap L1 IP2
set_option lib IP1 ./dir1      set_option lib IP2 ./dir2
```

Now in the TOP design, IP1 and IP2 are picked from the T1 and T2 libraries, as shown below:

```
Library T1;
Use T1.all;
Library T2;
Use T2.all;
```

In the above case, there is no reference to L1. Therefore, correct packages are picked from IP1 and IP2 intermediate libraries, as shown below:

```
set_option libmap T1 IP1
set_option lib IP1 ../dir1
set_option libmap T2 IP2
set_option lib IP2 ../dir2
```

NOTE: *Please note the following points for the intermediate library support:*

- Do not specify intermediate library name in the libhdlfiles/libhdlf option, as shown in the following example:

  ```
  set_option libmap L1 IP1
  set_option lib IP1 ./P1
  set_option libhdlfiles IP1 {case1.vhd case2.v}
  ```

  In the above example, an intermediate library name, IP1, is specified in the libhdlfiles option. As a result, SpyGlass reports a FATAL violation.

- Do not specify logical library name in the lib command, as shown in the following example:

  ```
  set_option libmap L1 IP1
  set_option lib L1 ./P1
  ```

  In the above example, logical library name, L1, is specified in the lib option. As a result, SpyGlass reports a FATAL violation.

- Do not specify logical library name in the work option, as shown in the following example:

  ```
  set_option libmap L1 IP1
  set_option lib IP1 ./P1
  set_option work L1
  ```

  In the above example, logical library name, L1, is specified in the work option. As a result, SpyGlass reports a FATAL violation.

Working with Compressed Gate Library Files

The SpyGlass library compiler can accept files compressed by using the UNIX gzip utility.

NOTE: *Currently, only those compressed library files that have been generated using the gzip utility and have .lib.gz extension are supported.*

Please note the following:

- The gzip-compressed file should have the .gz extension.
- You can specify both un-compressed and compressed library files together.
- You cannot specify a concatenated compressed library file, because SpyGlass does not support concatenation of compressed files.
- SpyGlass Library Compiler exits with a FATAL message if there are problems with the specified compressed library file.
- SpyGlass format library files (.sglib files) generated from compressed library files are already compressed files. Thus, you do not need to compress them again.
- SpyGlass reports messages related to library files contained in the compressed library file with the compressed library file name and the un-compressed library file line number.

  SpyGlass un-compresses and shows the un-compressed library files contained in the specified compressed library file, so you can check the reported problems easily.
- Specify file-based waivers with the name of the compressed library file only.

Just like gate cells in un-compressed library files, you cannot cross-probe from the instances of compressed library gate cells in the RTL to the schematic windows.

Working with Encrypted Compiled Libraries

SpyGlass enables you to encrypt VHDL/Verilog libraries during compilation. You can then use the created encrypted precompiled dump in the same way as the normal precompile dump.

Creating Encrypted Library Dump

You can create encrypted precompiled libraries by setting the value of the *Enable HDL Encryption* option to *yes* under the *Set Read Options* tab. The dump so created is in traditional SpyGlass precompiled dump format with some additional information embedded into it that specifies that the dump is encrypted.

Please note the following points while creating encrypted library dump:

- Library compiled with one version of SpyGlass may not be compatible with another version of SpyGlass. Therefore, you should create encrypted dump with the appropriate SpyGlass release so that it can be used with the SpyGlass version used by the IP user.
- While compiling VHDL files, specify the files in a proper order to take care of any dependencies. Alternatively, set the value of the *Automatically Sort VHDL File(s)* option to *yes* under the *Set Read Options* tab.

You may set the value of the dump_all_modes option of the `set_option` command to `yes` in the project file when the *Enable HDL Encryption* option is already set to *yes*. In such cases, when you create encrypted library dump, SpyGlass creates a precompile dump on both 32-bit and 64-bit platforms, irrespective of the platform on which you run SpyGlass.

Using Encrypted Library Dump

Use the encrypted precompiled dump in the same way as the normal precompile dump. You can use the normal precompile dump by specifying a logical library to the physical path mapping of dump by using the `lib` option of the `set_option` command in the project file.

Please note the following points while using encrypted libraries:

- It is recommended you to keep the encrypted IPs in their respective logical libraries and not merge them in a single precompiled library.
- Encrypted libraries and their design units should be referred in an encrypted IP user's design like any other normal precompiled library with appropriate VHDL/Verilog constructs, as shown in the following example:

  ```
  --VHDL
  library L1;
  use L1.all;


  //Verilog
  uselib lib=L1
  ```

  For more information about using precompiled libraries in a design, refer to the following bullets:
- If two or more encrypted libraries have the same design unit, use the fully qualified name or appropriate VHDL/Verilog constructs (as mentioned above) in the instantiation to pick the design unit from a specific library. Otherwise, SpyGlass would consider it based on the order specified by using the lib option of the `set_option` command. Please note that usage of encrypted design units follows the same paradigm as for a normal precompiled library
- Rule-checking behavior for encrypted libraries

  When you use the encrypted precompiled Verilog/VHDL modules with SpyGlass, all rule-checking on these modules is enabled by default. Any highlighting information inside such modules is shown on the module boundary only. Please note that all the messages are reported on the original file and line of encrypted IP.

  You can disable RTL rule-checking on these modules by setting the value of the *Disable Encrypted HDL Checks* option to *yes* under the *Set Read Options* tab. If you specify this option, SpyGlass disables RTL rule-checking on encrypted modules. In addition, SpyGlass internally removes any messages from that point inside an encrypted IP.
- SpyGlass behavior on specifying the waive -ip command on encrypted IPs

When you specify the waive -ip <IP-name> command, SpyGlass waives any violation coming on the file and line of an encrypted IP or any design unit instantiated inside encrypted IP.

NOTE: *The hdllibdu option of the set_option command does not have any effect on the above rule-checking behavior for encrypted modules. In addition, SpyGlass treats all design units instantiated under an encrypted design unit as encrypted even if they are not encrypted.*

NOTE: *The LEXICAL type rules do not run on encrypted RTL files.*

Viewing Built-In Messages for Precompiled Libraries

While parsing RTL files to generate a precompiled RTL dump, SpyGlass performs various checks, such as:

- Parsing-related checks performed by Verilog and VHDL analyzers
- Synthesis-related checks that are performed before hand during RTL parsing itself

By default, SpyGlass displays violations of the above checks in the following stages as the *Dump BuiltIn Rules in Precompile Flow* option (under *Set Read Options* tab) is set to *Yes* by default (that is, `set_option dump_precompile_builtin yes`):

- During RTL parsing in the precompilation step

  You can fix the parsing and synthesis-related issues in the precompilation step itself before using the precompiled dump in the top-level run.

- During usage of the precompiled dump

  If you do not fix the parsing and synthesis-related issues in the precompilation step, you can view these messages while using the precompiled dump in the top-level run by setting the *Dump BuiltIn Rules in Precompile Flow* option (under *Set Read Options* tab) to *Yes* (that is, `set_option hdllibdu yes`).

  These messages are stored as a part of that RTL dump and are restored while using the precompiled dump.

Example

Consider the following VHDL code (mixed.vhd) that result in the WRN_405 violations:

```
-- pragma synthesis_off
  -- synopsys translate_off
-- some VHDL code
  -- synopsys translate_on
  -- pragma synthesis_on
```

The following scenarios explain the usage of the `dump_precompile_builtin` and hdllibdu options:

Scenario 1
The WRN_405 violations are not reported during the usage of the precompiled dump as the dump_precomile_builtin option is set to no during the creation of that precompiled dump.
	Creating a precompiled dump:
	`read_file -type vhdl test.vhd`
	`set_option dump_precompile_builtin no`
	`set_option noelab yes`
	`set_option lib L1 P1`
	`set_option work L1`

	Using the precompiled dump:
	`set_option top top`
	`set_option lib L1 P1`
	`set_option hdllibdu yes`

Scenario 2
The WRN_405 violations are reported during the usage of the precompiled dump as the dump_precomile_builtin option is set to yes during the creation of the precompiled dump, and the hdllibdu option is specified during the usage of the precompiled dump.
	Creating a precompiled dump:
	`read_file -type vhdl test.vhd`
	`set_option dump_precompile_builtin yes`
	`set_option noelab yes`
	`set_option lib L1 P1`
	`set_option work L1`

	Using the precompiled dump:
	`set_option top top`
	`set_option hdllibdu yes`
	`set_option lib L1 P1`

NOTE: *Please note the following points:*

- As built-in messages may change across releases, you must precompile your libraries for each release in which you want to use the precompiled dump. This is because SpyGlass restores built-in messages of precompiled units compiled in the current version.
- If you want some product-specific built-in checks to be reported on the usage of precompiled design units, then during the RTL precompilation step, you must set the value of the AUTOENABLE_BUILTIN_CHECKS_FOR_POLICY configuration key to an appropriate product name in the .spyglass.setup file.
- SpyGlass restores the built-in messages for only those precompiled design units that are being checked in the current run. For example, if a precompiled design unit was stopped, the corresponding built-in messages are not restored.
- By default, SpyGlass performs rule-checking on encrypted precompiled design units (design units picked from a precompile dump created with the `set_option enable_hdl_encryption yes` command) even if you do not specify the `set_option hdllibdu yes` command. To disable rule-checking on such design units, set the *Disable Encrypted HDL Checks* field (under *Set Read Options* tab) to *Yes*.

  This option works independent of the `set_option hdllibdu yes` command. In conformance with this behavior, in case a design is precompiled with the `set_option dump_precompile_builtin yes` command, built-in messages on design units picked from such an encrypted precompiled dump would be reported by default unless the `set_option disable_encrypted_hdl_checks yes` command is specified.

Impact of the addrules Option While Using Pre-compiled Dump

While using precompiled RTL dump, if you enable a built-in rule by using the `set_goal_option addrules` *`<rule-name>`* command, but that rule was disabled while generating that precompiled dump, the corresponding built-in message is not reported during the precompiled dump usage even if you specify the `set_option hdllibdu yes` command.

In such cases, precompile that RTL again with `set_goal_option addrules` *<rule-name>* command. Only then that built-in message would appear during the precompiled RTL usage.

Impact of the ignorerules Option While Using Pre-compiled Dump

While using precompiled RTL dump, if you disable a built-in rule by using the `set_option ignorerules {` *rule-names* `}` command, but that rule was enabled while generating that precompiled dump, the corresponding built-in messages for the rule are not reported during the precompiled dump usage if you specify the `set_option hdllibdu yes` command.

Mapping a File Extension with a Compilation Language

The set_option libhdlfile and set_option libhdlf specifications may contain files of different extensions. For example, consider the following set_option libhdlfiles specifications containing files of different extensions:

```
set_option libhdlfiles LL1 {f1.vhd f2.vhd f3.vhd}
set_option libhdlfiles LL2 {g1.vh93 g2.vh87 g3.vhd}
set_option libhdlfiles LL3 {h1.v h2.v h3.v}
set_option libhdlfiles LL4 {k1.sv k2.sv k3.sv k4.v k4.v2k}
```

By default, SpyGlass compiles these files in some standard language, such as VHDL or Verilog.

However, you may want to compile these files into some specific languages, such as VHDL87, VHDL93, or Verilog2001. For example, compile the .vhd93 extension files in the VHDL93 language and the .v2k extension files in the Verilog 2000 language.

You can specify a compilation language for a particular file extension in either of the following ways:

- Specify a default mapping between a file extension and the corresponding compilation language by using the LIBHDL_EXTMAP and LIBHDL_LANG_INFERENCE keys in .spyglass.setup.

  For details, see *Inferring Language from File Extension During Compilation*.

- Specify compilation options directly in the .f file of the set_option libhdlf specification.

  For details, see *Specifying Compilation Options in a Source File*.

SpyGlass infers a compilation language for a file extension in the above-specified order. That is, a default mapping specified by configuration keys has the first priority to determine a compilation language for a file extension. If you have not specified any mapping, compilation options in the .f file of the `set_option libhdlf` specification are considered.

Inferring Language from File Extension During Compilation

You can enable SpyGlass to infer a compilation language automatically for a specific file extension by providing a default mapping between the file extension and its compilation language.

To infer a compilation language automatically from a file extension, specify the following information in the .spyglass.setup file:

- Specify a mapping between file extensions to their corresponding languages.

  To specify this mapping, use the LIBHDL_EXTMAP configuration key in the following manner:

  LIBHDL_EXTMAP = <file-extension> <file-type>

  Following are the examples of using the LIBHDL_EXTMAP key to specify a mapping between different file extensions and their corresponding languages:

```
LIBHDL_EXTMAP = .v verilog
LIBHDL_EXTMAP = .v2K verilog2000
LIBHDL_EXTMAP = .sv systemverilog
LIBHDL_EXTMAP = .vh87 vhdl87
LIBHDL_EXTMAP = .vh93 vhdl93
LIBHDL_EXTMAP = .vhd vhdl93
```

- Enable the automatic inference feature of determining a language type from a file extension.

  To enable this feature, set the LIBHDL_LANG_INFERENCE configuration key to YES in .spyglass.setup, as shown in the following example:

LIBHDL_LANG_INFERENCE = YES

By default, this key is set to NO.

NOTE: *To specify the mapping in a project file, use the libhdl_extmap project file command. For more information on the libhdl_extmap command, see libhdl_extmap.*

Example

Say you want to compile the following source files in the same SpyGlass run:

L1.f	**L2.f**	**L3.f**
f1.vhd87	g1.vhd93	h1.v
f2.vhd87	g2.vhd93	h2.v
f3.vhd87	g3.vhd93	h3.v

In the above case, to enable SpyGlass to infer a compilation language automatically for the files specified in the above source files, set the LIBHDL_EXTMAP and LIBHDL_LANG_INFERENCE environment variables in .spyglass.setup, as shown below:

```
LIBHDL_EXTMAP = .v verilog2000
LIBHDL_EXTMAP = .vh87 vhdl87
LIBHDL_EXTMAP = .vh93 vhdl93
```

LIBHDL_LANG_INFERENCE = YES

Now, you can compile a set of source files in the same SpyGlass run by specifying the following commands in a project file:

```
set_option lib L1 PL1
set_option lib L2 PL2
set_option lib L3 PL3
set_option libhdlf L1 L1.f
set_option libhdlf L2 L2.f
set_option libhdlf L3 L3.f
```

SpyGlass then automatically infers a language of files from their file extensions during compilation process. Therefore, L1.f is compiled in the VHDL 87 language, L2.f is compiled in the VHDL 93 language, and L3.f is compiled in the Verilog language.

NOTE: *To perform compilation process correctly, you should specify file names in a source file (.f) in the order of their dependency.*

Overriding the Default Compilation Language

For some file extensions, you may want to specify a different compilation language other than the default language inferred from the default mapping.

To specify a different compilation language, perform the following actions:

1. Specify the -disable_libhdl_lang_inference command in the source file (.f) of the set_option libhdlf specification.
2. Specify an appropriate language switch, such as -disablev2k, -enableSV, -93 and -87 to specify a different compilation language.

   Alternatively, you can specify the command on command-line along with the commands for performing compilation.

Example

Consider the following libhdlfiles specifications:

```
set_option libhdlfiles L1 {f1.v2k f2.v2k f3.v2k}
set_option libhdlfiles LL2 {g1.vh87 g2.vh87 g3.vhd}
```

Also, consider the following default mapping specified in the .spyglass.setup file:

```
LIBHDL_EXTMAP = .v2K verilog2000
LIBHDL_EXTMAP = .vh87 vhdl87

LIBHDL_LANG_INFERENCE = YES
```

In this case, SpyGlass will compile the.v2k files in Verilog2000 language and the .vhd87 files in the VHDL87 language.

Now, if you want to override the default mapping for f1.v2k and f2.v2k files to specify the compilation language as SV instead of Verilog2000, perform the following steps:

1. Specify such .v2k files in a separate .f file (say f4.f), as shown below:

   ```
   f4.f
   ```

```
f1.v2k
f2.v2k
```

2. Specify the -disable_libhdl_lang_inference option in the f4.f file, as shown below:

```
f4.f

f1.v2k
f2.v2k
-disable_libhdl_lang_inference
```

3. Specify the -enableSV option in the f4.f file, as shown below:

```
f4.f

f1.v2k
f2.v2k
-disable_libhdl_lang_inference
-enableSV
```

   Alternatively, you can specify the -enableSV option on command-line while specifying commands for compilation.

Specifying Compilation Options in a Source File

Modify the source file by specifying the following commands that can affect the compilation process:

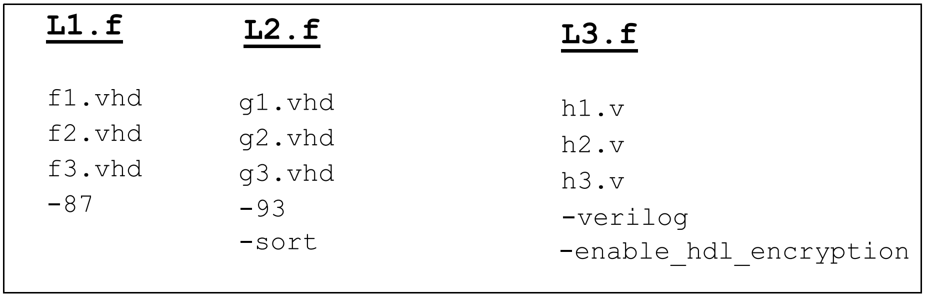
- Language standard directive options, such as `disablev2k`, `enableSV`, and `87`:
- Other options, such as:

-hdlin_translate_off_skip_text	-pragma	-top
-allow_module_override	-nodefparam	-sort
-allow_celldefine_as_top	+define	+resetall
-enable_hdl_encryption	+incdir	-param
-hdlin_synthesis_off_skip_text	-sfcu	-macro_synthesis_off
-relax_hdl_parsing		

After specifying the above commands in a source file, specify that source file by using the following command:

```
set_option libhdlf <logical-library-name> "<source-files>
```

For example, you can specify a language applicable for a set of files in the .f file by using appropriate command-line options, as shown below:



```
L1.f        L2.f              L3.f

f1.vhd      g1.vhd            h1.v
f2.vhd      g2.vhd            h2.v
f3.vhd      g3.vhd            h3.v
-87         -93               -verilog
            -sort             -enable_hdl_encryption
```

In the above example:

- The files specified in the L1.f file are considered as VHDL 87 source files.
- The files specified in the L2.f file are considered as VHDL 93 source files.
- The files specified in the L3.f file are considered as Verilog source files and are encrypted into a binary format.

You can then compile these files in the same SpyGlass run by specifying the following commands in a project file:

```
set_option lib L1 PL1
set_option lib L2 PL2
set_option lib L3 PL3
set_option libhdlf L1 L1.f
set_option libhdlf L2 L2.f
set_option libhdlf L3 L3.f
```

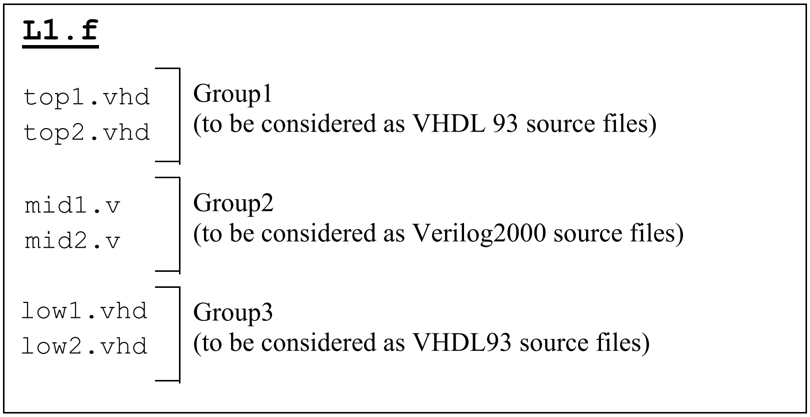
Specifying Files in the Order of Their Dependencies

Modules of different source file types may have dependencies among them. Therefore, you should define them in the order of their dependencies in the file specified by the `libhdlf` command.

For example, consider the following L1.f file:

```
L1.f
top1.vhd
top2.vhd
mid1.v
mid2.v
low1.vhd
low2.vhd
```

In the above example, files in L1.f are compiled by forming the following three groups internally to maintain dependencies and language standards to be used during compilation process:



L1.f

top1.vhd
top2.vhd

Group1
(to be considered as VHDL 93 source files)

mid1.v
mid2.v

Group2
(to be considered as Verilog2000 source files)

low1.vhd
low2.vhd

Group3
(to be considered as VHDL93 source files)

Compiling Verilog Files Containing SystemVerilog Keywords

It may happen that some Verilog files being compiled contain SystemVerilog keywords. During compilation of these files, if you specify the `set_option enableSV yes` command, SpyGlass reports a fatal violation.

To handle such cases, perform the compilation process by using any of the following approaches:

- *Compiling the Set of Verilog and SystemVerilog Files Separately*
- *Using File Extension Based Compilation Flow*

Compiling the Set of Verilog and SystemVerilog Files Separately

In this approach, perform the following steps:

1. Divide the file list (say verilog.v, verilog1.v, system.v, and system1.v) under the following sets:
   - ❒ SystemVerilog files (SystemVerilog mode list)

     Let this list be sv_file_list.f containing system.v, and system1.v.
   - ❒ Remaining Verilog files (Verilog mode list)

     Let this list be verilog_file_list.f containing verilog.v, and verilog1.v.
2. Compile the files of the SystemVerilog mode list by specifying the following commands in a project file (say project1.prj):

```
read_file -type sourcelist sv_file_list.f
set_option lib sver_lib phy_path2
set_option work sver_lib
set_option enable_precompile_vlog yes
set_option enableSV yes
```

3. Compile the files of the Verilog mode list by specifying the following commands in a project file (say project2.prj):

```
read_file -type sourcelist verilog_file_list.f
set_option lib ver_lib phy_path1
set_option work ver_lib
set_option enable_precompile_vlog yes
```

4. Specify the top-level module and specify the logical to physical mapping for libraries by specifying the following commands in a project file (say project3.prj):

```
set_option lib ver_lib phy_path1
set_option lib sver_lib phy_path2
set_option top top_module_name
```

After performing the above steps, SpyGlass compiles all the specified files.

Using File Extension Based Compilation Flow

In this approach, perform the following steps:

1. Change the extension of the SystemVerilog file containing SystemVerilog constructs to .sv, as shown in the following example:

   ```
   system.v  =>  system.sv
   system1.v  =>  system1.sv
   ```

2. Map the .sv extension with the SystemVerilog language by setting the following keys in the .spyglass.setup file:

   ```
   LIBHDL_EXTMAP = .sv systemverilog
   LIBHDL_LANG_INFERENCE = yes
   ```

   For details, see *Inferring Language from File Extension During Compilation*.

3. Compile all the source files in a single file list by specifying the following commands in a project file:

   ```
   set_option lib ver_lib phy_path1
   set_option libhdlf ver_lib verilog_file_list.f
   set_option top top_module_name
   ```

   Where file_list.f contains verilog.v, verilog1.v, system.sv, and system1.sv files.

After performing the above steps, SpyGlass compiles all the specified files in one go.

Working with Encrypted Design Files

IP vendors spend huge amounts of time, effort, and money to develop design files for IPs. This raises the need to secure these design files from infringement problems. One way to protect design files from such problems is to encrypt these files.

You can specify encrypted files for SpyGlass analysis in the same way as you specify un-encrypted Verilog/VHDL design files. For example, you can specify encrypted files through GUI, project file, or batch commands, such as `-v`/`-y`.

This section discusses the use model of IP encryption in SpyGlass and details of the encryption flow.

Introducing the Use Model for IP Encryption in SpyGlass

You can encrypt design files (IPs) by using the SpyGlass encryption engine `spyencrypt`. After encryption, pass the encrypted files for SpyGlass analysis. During analysis, SpyGlass internally decrypts such files.

The following figure shows the use model of IP encryption in SpyGlass:

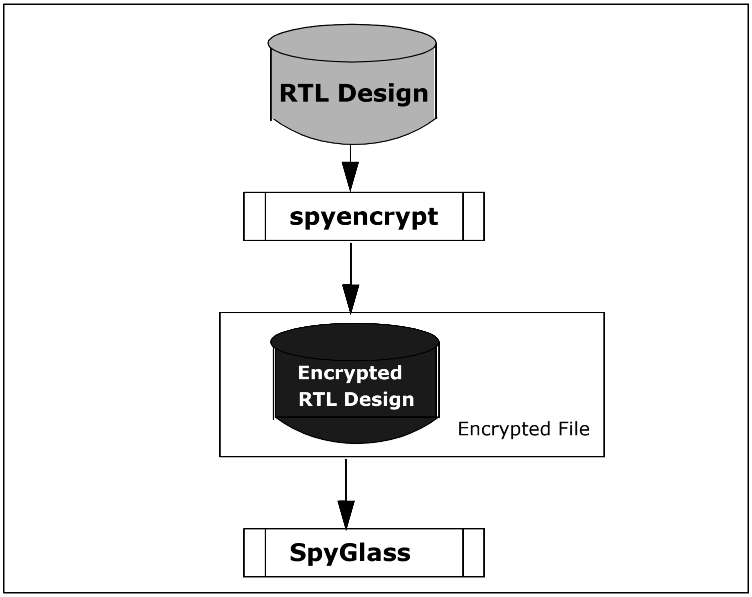


FIGURE 142. IP Encryption Use Model in SpyGlass

Encrypting IPs by Using the spyencrypt Utility

To encrypt design files, specify RTL files of IPs to the `spyencrypt` utility.

The syntax of the `spyencrypt` utility is as follows:

```
spyencrypt  <RTL-Files>
   -outdir <output-directory>
   [  -encrypt_ext "<extension-string>"  ]
   [  -no_encrypt_ext  ]
   [  -help  ]
```

Once you run the `spyencrypt` utility to encrypt design files, SpyGlass generates the spyencrypt_summary.rpt report containing the status of encryption. For details on this report, see *Viewing Encryption Summary in a Report*.

NOTE: *Cross-probing to RTL is not supported for encrypted design files.*

NOTE: *Schematic of design units specified through encrypted files is not visible.*

Arguments of the spyencrypt Utility

Details of the arguments of the `spyencrypt` utility are discussed below:

`<RTL-Files>`

This argument specifies a space-separated list of design files to be encrypted. For example, the following command encrypts the top.v and mid.vhdl files:

```
spyencrypt top.v mid.vhdl -outdir enc_dir
```

After running the above command, the top.v.sge and mid.vhdl.sge encrypted files are created in the ./enc_dir directory.

By default, SpyGlass appends the .sge extension to the name of an encrypted design file. To change this default extension, use the *-encrypt_ext "<extension-string>"* argument.

NOTE: *You can use wildcard characters to encrypt all files in a particular directory. For example, you can specify* `dir/*.v` *or* `dir/*.vhdl` *specification to encrypt all Verilog or VHDL files of the* `dir` *directory.*

NOTE: *Before specifying RTL files for encryption, update the `include and `uselib compiler directives appropriately to reflect the encrypted file names. This step is not required if you use the -no_encrypt_ext argument.*

`-outdir <output-directory>`

This argument specifies the directory in which encrypted files should be saved.

For example, consider the following command:

```
spyencrypt top.v mid.vhdl lib/bottom.v -outdir enc_dir
```

When you run the above command, SpyGlass generates the following encrypted files:

- enc_dir/top.v.sge

- enc_dir/mid.vhdl.sge
- enc_dir/bottom.v.sge

If you specify the name of a non-existent directory to the -outdir argument, spyencrypt creates that directory. However, if you specify a hierarchical path of a non-existent directory, spyencrypt creates that directory under that path only if that path exists.

For example, consider that the enc/outdir/ path exists and you specify enc/outdir/dir to the -outdir argument. In this case, spyencrypt creates the dir directory under enc/outdir/. However, if the enc/outdir/ path does not exist, spyencrypt does not create the dir directory and reports an error message.

-encrypt_ext "<extension-string>"

(Optional) This argument specifies an extension string to be appended to the names of the encrypted design files.

NOTE: *Do not include a period (.) while specifying an extension string.*

Consider the following command:

```
spyencrypt top.v mid.vhdl -outdir enc_dir -encrypt_ext "en"
```

When you run the above command, SpyGlass generates the following encrypted RTL files with the .en extension:

- enc_dir/top.v.en
- enc_dir/mid.vhdl.en

-no_encrypt_ext

(Optional) This argument disables appending of any extension to the name of encrypted design files.

Specify this argument if you do not want any extra extension to appear in the name of an encrypted design file.

For example, consider the following command:

```
spyencrypt top.v mid.vhdl -outdir enc_dir -no_encrypt_ext
```

When you run the above command, SpyGlass generates the following encrypted design files:

- enc_dir/top.v
- enc_dir/mid.vhdl

NOTE: *If you specify the* `-encrypt_ext` *and* `-no_encrypt_ext` *commands together, SpyGlass ignores the* `-encrypt_ext` *command. Therefore, names of the encrypted files do not contain the extension specified by the* `-encrypt_ext` *command.*

For example, consider the following command:

```
spyencrypt test.v test1.vhdl lib/vlib.v -outdir enc_dir
-encrypt_ext "ev" -no_encrypt_ext
```

The above command generates the following files:

- enc_dir/test.v
- enc_dir/test1.vhdl
- enc_dir/vlib.v

-help

(Optional) This argument lists the names of `spyencrypt` arguments and their usage.

Encrypting IPs Spread Across a Hierarchical Directory Structure

Design files to be encrypted may be present in a hierarchical directory structure.

For example, consider the following directory structure containing design files to be encrypted:

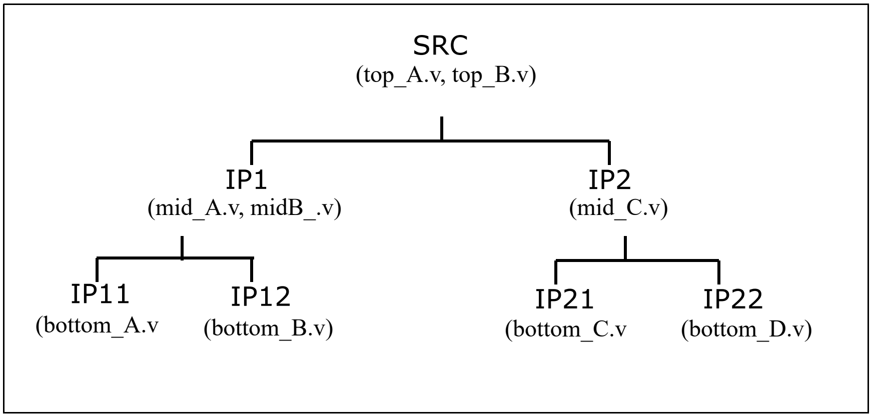


FIGURE 143. Directory Structure - Design Files to be Encrypted

To encrypt all design files present in the above directory structure, perform the following steps:

1. Move to the root directory (SRC).
2. Encrypt all files present under this directory.

   This is shown in the following example:

   ```
   spyencrypt top_A.v top_B.v -outdir SRC_encr
   ```

NOTE: *Ensure that the path of the directory created by the* `-outdir` *argument is such that a parallel directory structure is created similar to the existing hierarchical directory structure. This is shown in Figure 144.*

3. Move to the next directory in the hierarchy.
4. Repeat Step 2 until you encrypt all design files present in the entire hierarchy.

After performing the above steps, a directory structure (containing encrypted design files) is created parallel to the existing hierarchical directory structure, as shown in the following figure:

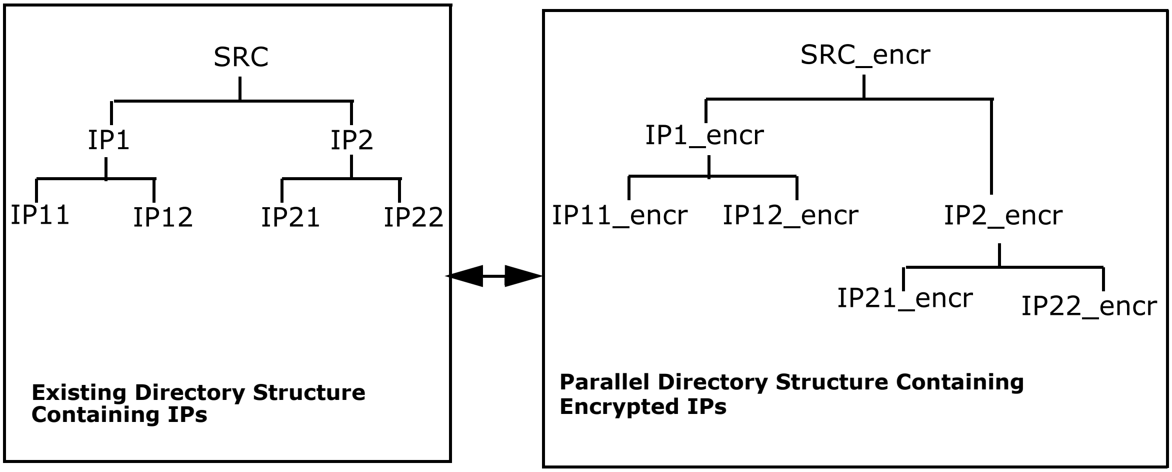

SRC
IP1
IP2
IP11
IP12
IP21
IP22
Existing Directory Structure Containing IPs
SRC_encr
IP1_encr
IP11_encr
IP12_encr
IP2_encr
IP21_encr
IP22_encr
Parallel Directory Structure Containing Encrypted IPs

FIGURE 144. Directory Structure - Encrypted Design Files

NOTE: *Please note the following points:*

- *If multiple design files with the same name are encrypted in the same directory, only the last file passed to the* `spyencrypt` *utility is encrypted.*
- *If a module present in the middle of a design hierarchy is encrypted by using spyencrypt, SpyGlass considers the entire hierarchy below that module as encrypted, even though modules in that hierarchy may not be explicitly encrypted.*

  For example, consider the design hierarchy `top.mid.bottom.leaf`*, where each module is defined in a separate RTL file, top.v, mid.v, bottom.v, and leaf.v.*

  In this case, if any module (say mid.v) is encrypted by using `spyencrypt`*, SpyGlass considers the entire hierarchy below that module as encrypted (in this case, bottom.v and leaf.v), even though bottom.v and leaf.v are not encrypted using* `spyencrypt`*.*

Viewing Encryption Summary in a Report

When you encrypt design files by using the `spyencrypt` utility, SpyGlass generates the spyencrypt_summary.rpt report containing the status of encryption.

Following is a sample of the spyencrypt_summary.rpt report:

```
-----------------------------------------------------------
Original File    Encrypted File  Status   Reason
----------------------------------------------------------
enc/top.v.sge  N.A               FAIL     Encrypted File
                                          can't be encrypted

src/mid.v.gz   N.A               FAIL     GZ File can't be
                                          encrypted

src/bottom.v   enc/bottom.v.sge  PASSED   N.A
```

NOTE: *Please note the following points:*

- *SpyGlass does not support nested/double encryption. The encryption status in the spyencrypt_summary report appears as FAILED for such cases with the following message:*

  ```
  GZ File can't be encrypted
  ```

- *SpyGlass does not encrypt a compressed file. The encryption status in the spyencrypt_summary report appears as FAILED for such cases with the following message:*

  ```
  Encrypted File can't be encrypted
  ```

Specifying Encrypted Files for SpyGlass Analysis

Specify encrypted files for SpyGlass analysis in any of the following ways:

- *Specifying Encrypted Files through GUI*
- *Specifying Encrypted Files through a Project File*

NOTE: *SpyGlass reports syntax errors if you pass encrypted files that are compressed.*

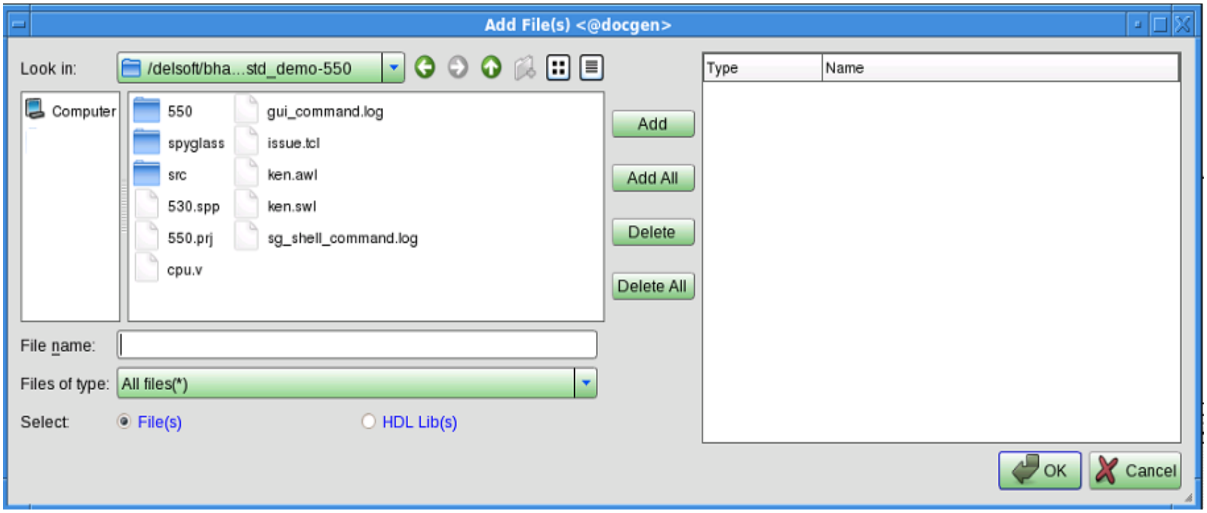
Specifying Encrypted Files through GUI

To specify encrypted files for SpyGlass analysis through GUI, perform the following steps:

1. Click the *Add Design Files* tab under the *Design Setup* tab.

2. Click the *Add File(s)* link.

   The *Add File(s)* dialog appears. The following figure shows this dialog:



Add File(s) <@docgen>

Look in: /delsoft/bha...std_demo-550

Computer

550
spyglass
src
530.spp
550.prj
cpu.v
gui_command.log
issue.tcl
ken.awl
ken.swl
sg_shell_command.log

File name:

Files of type: All files(*)

Select ● File(s) ○ HDL Lib(s)

FIGURE 145. Specifying Encrypted Files

3. In the above dialog, select the encrypted file name (in this case, test.v.sge).
4. Click the *Add* button.

   The selected file now appears in the right-most pane of the above dialog, as shown in the following figure:

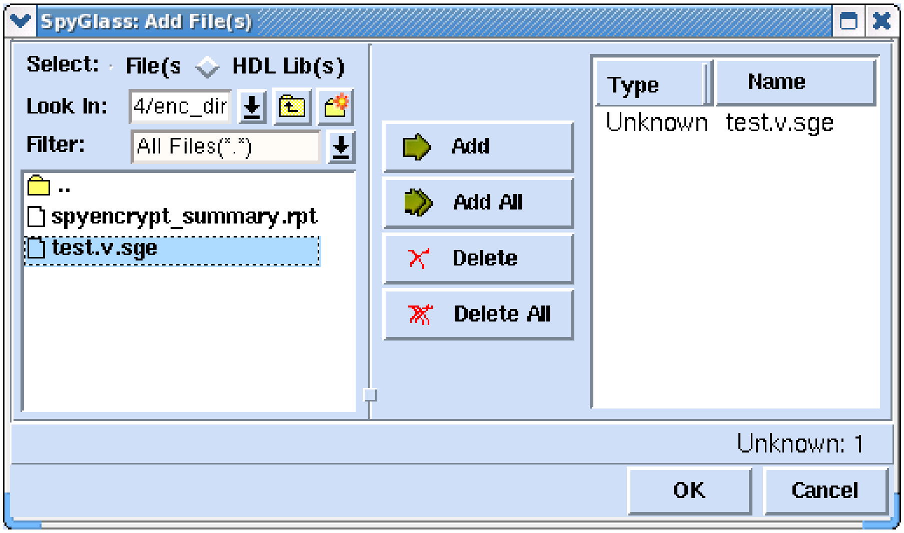
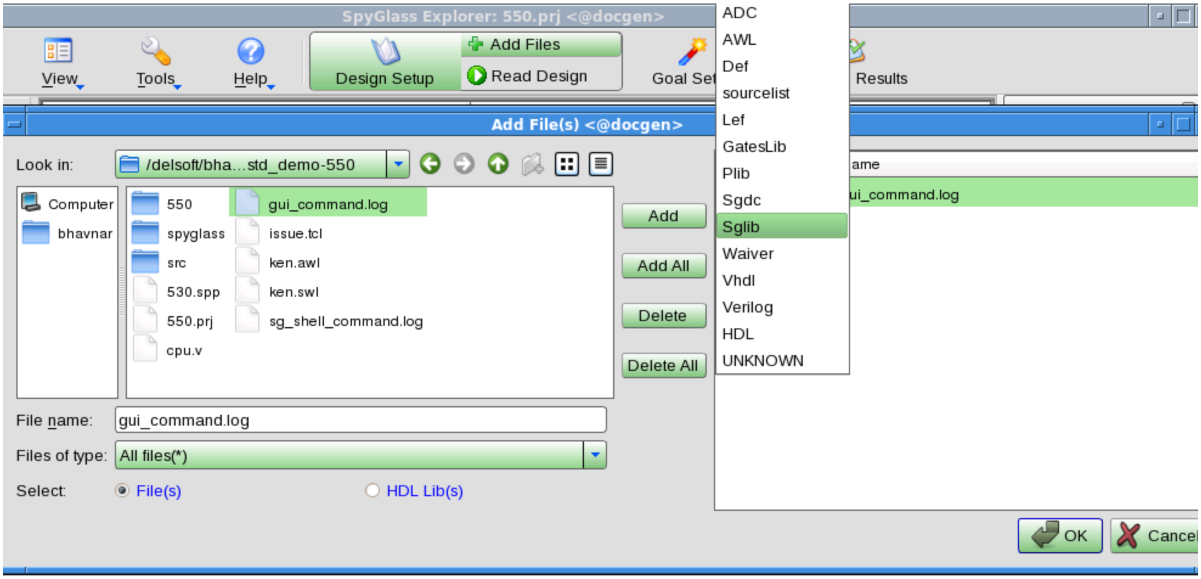


FIGURE 146. Specifying Encrypted Files - Add File(s)

5. In the above dialog, click on the *Unknown* text under the *Type* column. A drop-down list appears, as shown below:



Look in: /delsoft/bha...std_demo-550

File name: gui_command.log

Files of type: All files(*)

Select: ● File(s) ○ HDL Lib(s)

FIGURE 147. SpyGlass Add File(s)

6. Select an appropriate option (in this case *Verilog*) from the drop-down list.
7. Click the *OK* button to close the *Add File(s)* dialog.

After performing the above steps, the selected encrypted file appears under the *Add File(s)* tab.

Specifying Encrypted Files through a Project File

To specify encrypted IPs for SpyGlass analysis through a project file, perform the following steps:

1. Use the `read_file` command in a project file to specify an encrypted file. This is shown in the following example:

```
read_file -type verilog ./ATRENTA/top.v.sge
```

NOTE: *Change the* `incdir` *project file command to reflect the correct include path. In addition, if you change the default extension by using the* `-encrypt_ext` *command, make changes in the* `libext` *and* `v/y` *command specifications accordingly.*

2. Specify the project file for SpyGlass analysis, as shown in the following example:

   ```
   spyglass -project test.prj
   ```

Working with Mixed-Language Designs

To analyze lower-level blocks described in a different HDL language, you must precompile the lower-level blocks into a library.

Instantiating Verilog Modules in VHDL Architectures

A Verilog module can be instantiated inside VHDL architecture either as an entity instance or as a component instance.

Instantiating as Component Instance

In this case, you should first create a component declaration. In addition, you must ensure the following:

- For default binding, that is, when binding is not done through component configuration, component name, port names, and generic names should be same as the corresponding Verilog identifiers for module name, port names, and parameter names.
- Number of ports or generics and their bit-width in VHDL component declaration must be same as those of ports or parameters in Verilog module definition.

The following example instantiates a Verilog module in a VHDL design unit as a component instance by using default binding:

```
//test.v

module comp (a, b);
input a;
output b;
...
endmodule


--test.vhd

entity ent is
port (entIn : in std_logic;
entOut : out std_logic);
```

```
end ent;

architecture Behave of ent is
component comp
port (a : in std_logic; b : out std_logic );
end component ;
begin
Inst1 : comp port map ( a => entIn, b => entOut );
...
end Behave;
```

Configuration Specification Based Binding

For configuration specification-based binding, the component name, port names, and generic names can be same or different from the corresponding Verilog identifiers for module name, port names, and parameter names. However, the number of ports or generics and their bit-width in the VHDL component declaration must be the same as those of ports or parameters in the Verilog module definition.

Following is an example of instantiating a Verilog module in a VHDL design unit as a component instance by using configuration specification based binding:

```
//test.v

module comp (A1, B1);
input A1;
output B1;
...
endmodule


-- test.vhd

entity top is
port (in1 : in std_logic; out1 : out std_logic);
end top;
architecture arch_top of top is
component my_comp1
port (C1 : in std_logic; D1 : out std_logic);
end component;
for inst1 : my_comp1 use entity work.comp
```

```
port map (A1 => C1, B1 => D1);
begin
inst1 : my_comp1 port map (C1 => in1, D1 => out1);
...
end arch_top;
```

Instantiating as Entity Instance

In this case, no additional declaration is required. You can directly instantiate a Verilog module using the VHDL syntax for instantiation of an entity.

Following is an example of instantiating a Verilog module in a VHDL design unit as an entity instance:

```
//test.v

module comp (a,b);
input a;
output b;
...
endmodule
--test.vhd
entity ent is
port (entIn : in std_logic;
entOut : out std_logic);
end ent;
architecture Behave of ent is
begin|
Inst1 : entity comp port map
( a => entIn, b => entOut );
...

end Behave;
```

A configuration declaration can reference Verilog modules wherever an entity reference is intended. However, it must not extend beyond the module interface and the instantiations within Verilog module description will not be accessible to the configuration.

Following is an example of instantiating a Verilog module in a VHDL design unit as a configuration declaration:

```
//test.v

module comp (a,b);
input a;
output b;
...
endmodule

--test.vhd

entity ent is
port (entIn : in std_logic;
entOut : out std_logic);
end ent;
architecture Behave of ent is
component mod
port (a : in std_logic; b : out std_logic );
end component ;
begin
Inst1 : mod port map ( a => entIn, b => entOut );
...
end Behave;
configuration config of ent is
for Behave
for Inst1 : mod
use entity work.comp(<identifier>);
end for;
|end for;
end configuration;
```

NOTE: *<identifier> is tool-specific. For SpyGlass, <identifier> is Verilog or module.*

Searching Master of an Instance

In a given design, SpyGlass searches for the master of an instance in VHDL architecture as per the following order:

- VHDL source files
- precompiled VHDL libraries
- Verilog source files
- Verilog libraries specified using the set_option v or set_option y commands in the project file

- precompiled Verilog libraries
- SpyGlass compatible Synopsys Liberty™ file

Restrictions

Mixed-language semantics impose the following restrictions on the use of any of the above instantiations:

- Design unit being instantiated should be a Verilog module. A Verilog built-in gate or UDP cannot be instantiated.
- All ports in Verilog module should be named port. SpyGlass does not supports unnamed ports.

Instantiating VHDL Design Units In Verilog Modules

A VHDL design unit can be instantiated in Verilog modules just like any Verilog module instantiation. As Verilog does not have the concept of architecture or libraries, the escaped identifier is used to describe the instantiation from a specific library.

The following table describes the allowed format and their interpretations:

Format	**Description**
\myLibrary.myEntity(myArch)	Architecture myArch of entity myEntity from logical library myLibrary
\myEntity(myArch)	Architecture myArch of entity myEntity from logical library work
\myLibrary.myEntity	MRA Architecture of entity myEntity from logical library myLibrary
\myLibrary.myConfigDecl	Configuration declaration myConfigDecl from logical library myLibrary
myName	Either configuration declaration or entity myName from logical library work

Searching for Master of an Instance

In a given design, SpyGlass searches for the master of an instance in the Verilog module as per the following order:

- Verilog source files
- Verilog libraries specified using the set_option v and set option y commands in the project file
- precompiled Verilog libraries
- VHDL source files
- precompiled VHDL libraries
- SpyGlass compatible Synopsys Liberty™ files

Examples of Instantiating VHDL Design Units in Verilog Modules

Example 1

Instantiating Architecture myArch of entity myEntity from logical library myLibrary (\myLibrary.myEntity(myArch))

```
--test.vhd

entity ent is
port (entIn : in std_logic;
entOut : out std_logic );
end ent;
architecture Behave of ent is
begin
...
end Behave;


//test.v

module mod (a,b);
input a;
output b;
\mylib.ent(Behave) inst1(a,b);
...
endmodule
```

Example 2

Instantiating Configuration Declaration myConfigDecl from logical library myLibrary (\myLibrary.myConfigDecl)

```
--test.vhd
```

```
entity ent is
port (entIn : in std_logic;
entOut : out std_logic );
end ent;
architecture Behave of ent is
begin
...
end Behave;
configuration config of ent is
for Behave
end for;
end configuration;
```

```
//test.v
```

```
module mod (a,b);
input a;
output b;
\mylib.config inst1(a,b);
...
endmodule
```

Example 3

How to reference VHDL Records across language boundaries:

```
//test.v
```

```
module e1 (in1, in2, out1);
input in1, in2;
output out1;
...
endmodule
```

```
--test.vhd
```

```
entity top is
...
end top;
architecture top of top is
type X is record
f1 : bit;
f2 : bit;
f3 : bit;
end record;
signal sig : X ;
component e1
port (in1, in2 : bit; out1 : out bit);
end component;
begin

inst : e1 port map(
in1 => sig.f1,
in2 => sig.f2,
out1 => sig.f3);
...
end top;
```

Mapping Data Types

Instantiation of a design unit described in one HDL inside another design unit implemented in the other HDL requires certain adaptations and data type conversions at port and generic/parameter interface. For example, VHDL instantiation of a Verilog module can associate VHDL signals and values with Verilog ports and parameters. Similarly, Verilog instantiation of a VHDL design unit can associate Verilog nets and value with VHDL ports and generics.

Mapping between VHDL Generics and Verilog Parameters

An instance of VHDL design unit in a Verilog module can override default generic values through appropriate parameter mapping. Similarly, an instance of a Verilog module inside a VHDL design unit can override default parameter values through appropriate generic mapping.

Mixed-language support in SpyGlass supports *VHDL Port Mapping to Verilog Ports*.

Mixed-Language support in SpyGlass supports the following data-mapping:

VHDL Generic Type	Verilog Parameter Type
VHDL integer	Verilog integer
VHDL real	Verilog real
VHDL time	Verilog integer or real (multiplied with appropriate timescale directive)
VHDL string	Verilog string
VHDL enumeration	Verilog integer based on ‘VAL() attribute in VHDL

VHDL Port Mapping to Verilog Ports

Verilog ports are based on language-defined data type that supports both logic simulations at logic 0/1/X/Z level as well as signal-strength modeling for transistor circuit simulation.

For mixed-language support, SpyGlass only supports Verilog logic-level based on 0/1/X/Z logic and it is mapped to the following data types in VHDL:

- VHDL Generic Type Verilog Parameter Type
- VHDL integer Verilog integer
- VHDL real Verilog real
- VHDL time Verilog integer or real (multiplied with appropriate timescale directive)
- VHDL string Verilog string
- VHDL enumeration Verilog integer based on 'VAL() attribute in VHDL
- bit or std_logic
- bit_vector or std_logic_vector

Current Limitation with Mixed-language Designs in SpyGlass

- VHDL design units instantiated in a Verilog module cannot have unconnected terminals in the port mapping.
- Port mapping across language boundaries is case-sensitive.
- In some designs, there can be multiple reporting of the same SYNTH and Elaboration errors.
- Syntax errors are suppressed during synthesis of a Mixed-Language design with 'define macro declaration of the following type:

  ```
  'define macro(A,B,C) A|B|C
  ```

Working with DesignWare® Modules

SpyGlass can expand DesignWare® modules and generate logic for these modules during SpyGlass analysis.

Prerequisites for Enabling DesignWare Flow

If a design contains instances of DesignWare modules, perform the following actions before running SpyGlass analysis:

- Specify the DesignCompiler license to generate Verilog netlist for DesignWare modules.
- Specify the path for DesignCompiler installation. For details, see *Specifying Path of DesignCompiler Installation*.

NOTE: *SpyGlass uses your license and installation of the DesignWare tool set (DesignCompiler, DesignWare-Basic, and DesignWare-Foundation) to generate the GTECH mapped netlists. SpyGlass internally generates the required wrapper files, scripts, etc.*

Specifying Path of DesignCompiler Installation

Specify the path of the DesignCompiler installation in your work environment by setting the SPYGLASS_DC_PATH environment variable to a valid DesignCompiler installation, as shown in the following example:

```
setenv SPYGLASS_DC_PATH /net/DC2003/linux
setenv SPYGLASS_DC_PATH /net/DC2003
```

The SPYGLASS_DC_PATH key can be set to either of the above directory settings.

Alternatively, you can specify the above path in the .spyglass.setup file. In this file, you need to define the setup key, SPYGLASS_DC_PATH, and set its value to the DesignCompiler installation area path.

NOTE: *Please note the following points:*

- If you have set the SPYGLASS_DC_PATH environment variable and defined the SPYGLASS_DC_PATH setup key, then SpyGlass gives preference to the setting specified in the environment variable.

- The previous DC_PATH environment variable has been replaced by the SPYGLASS_DC_PATH environment variable.
- If you choose to set SPYGLASS_DC_PATH to some dc_shell script (and not the installation area) so that the DesignWare files (packages/dware) and library path (dw/) from dw01 to dw06 are not deducible from SPYGLASS_DC_PATH, set the SPYGLASS_DC_DWARE_FILES_PATH and SPYGLASS_DC_DW_FILES_PATH variables in the .spyglass.setup file, as shown below, so that SpyGlass gets the required files:

  ```
  SPYGLASS_DC_DWARE_FILES_PATH = <dc-installation>/packages/
  dware/
  ```

  ```
  SPYGLASS_DC_DW_FILES_PATH = <dc-installation>/dw/
  ```

NOTE: *You can not set the keywords SPYGLASS_DC_DWARE_FILES_PATH and SPYGLASS_DC_DW_FILES_PATH as environment variables.*

The license for DesignCompiler must be available prior to running SpyGlass.

Your license setups should also include your DesignCompiler licenses. To ensure that DesignCompiler is not run again during subsequent runs, SpyGlass stores the netlists that are generated once. During subsequent runs, SpyGlass checks whether netlists already exist for the instantiated DesignWare components before invoking DesignCompiler. If these netlists already exist (generated during earlier runs), DesignCompiler is not invoked for those DesignWare components. However, DesignCompiler is invoked for any instance of DesignWare component that does not have a netlist already visible to SpyGlass. This allows you to change the instantiation of a DesignWare component (if required) without any loss of analysis and without causing unnecessary run of DesignCompiler (unless actually required).

Enabling the DesignWare Flow

To enable the DesignWare flow, specify the following command in a project file:

```
set_option dw yes
```

This option results in the following:

- SpyGlass looks at the design description to identify instances of the DesignWare components and the parameter/generic values used with these instantiations.
- SpyGlass invokes Synopsys® DesignCompiler to generate netlists (in terms of GTECH cells) corresponding to these instances of DW components.
- SpyGlass uses this DesignCompiler-generated netlists (of DesignWare components) for analysis of the design. This causes SpyGlass to perform a more accurate analysis on the design.

If your design contains instances of DesignWare modules listed in *Table 12*, SpyGlass requires the DesignCompiler license to generate netlist for these modules.

Table 13 lists DesignWare modules for which the DesignCompiler license is not required.

Reusing Netlist of DesignWare Modules during SpyGlass Analysis

To use DesignWare modules in SpyGlass, specify the location of netlists generated by DesignCompiler by specifying the following command in the project file:

```
set_option lib SPY_DW_WORK <dir-name>
```

When you specify the above command, SpyGlass stores the generated netlists and picks them from the *<dir-name>* directory.

Changing the netlist location may be useful if you want to reuse the netlists generated on one design during subsequent runs on other designs. During subsequent runs of SpyGlass, if the desired netlist is found in the specified location, SpyGlass picks that netlist. Otherwise, DesignCompiler generates the netlist and stores it at the specified location for subsequent usage.

NOTE: *Refer to the SpyGlass Design Read-In Methodology for details on customizing this feature for your specific design/site requirements.*

Notes

- If you have not defined VHDL component declaration of the DesignWare components correctly, SpyGlass may report built-in errors, such as STX_VH_11, STX_VH_464, and WRN_384.

- In case of save-restore in DesignWare, if the you changes the lib SPY_DW_WORK mapping (set_option lib <logical-lib-name> <physical-path>) to some other area in the restore run, restore is unsuccessful.
- Verilog-95 standard is not compatible with DesignWare compilation.

List of DesignWare Modules Supported in SpyGlass

The following table lists DesignWare modules that require the DesignCompiler license for Verilog netlist generation:

TABLE 12 DesignWare Modules that Require Design Compiler License

DW02_cos	DW02_rem	DW02_sin
DW02_sincos	DW02_sqrt	DW03_cntr_gray
DW03_reg_s_pl	DW04_crc32	DW04_shad_reg
DW04_sync	DW_8b10b_dec	DW_8b10b_enc
DW_8b10b_unbal	DW_arb_2t	DW_arb_dp
DW_arb_fcfs	DW_arb_sp	DW_arbiter_2t
DW_arbiter_dp	DW_arbiter_fcfs	DW_arbiter_sp
DW_asymfifo_s1_df	DW_asymfifo_s1_sf	DW_asymfifo_s2_sf
DW_asymfifoctl_s1_df	DW_asymfifoctl_s1_sf	DW_asymfifoctl_s2_sf
DW_bc_1	DW_bc_2	DW_bc_3
DW_bc_4	DW_bc_5	DW_bc_7
DW_crc_p	DW_crc_s	DW_data_sync_1c
DW_data_sync_na	DW_data_sync	DW_div_seq
DW_div_sat	DW_dpll_sd	DW_ecc
DW_fifo_s1_df	DW_fifo_s1_sf	DW_fifo_s2_sf
DW_fifoctl_s1_df	DW_fifoctl_s1_sf	DW_fifoctl_s2_sf
DW_fifoctl_s2dr_sf	DW_fir_seq	DW_fir
DW_fp_add	DW_fp_addsub	DW_fp_cmp

TABLE 12 DesignWare Modules that Require Design Compiler License

DW_fp_div	DW_fp_flt2i	DW_fp_i2flt
DW_fp_mult	DW_fp_sub	DW_fp_sum3
DW_fp_sum4	DW_gray_sync	DW_iir_dc
DW_iir_sc	DW_inv_sqrt	DW_llfifocntl_s1_df
DW_mult_seq	DW_pipe_mgr	DW_piped_mac
DW_pulse_sync	DW_pulseack_sync	DW_ram_2r_w_a_dff
DW_ram_2r_w_a_lat	DW_ram_2r_w_s_dff	DW_ram_2r_w_s_lat
DW_ram_r_w_a_dff	DW_ram_r_w_a_lat	DW_ram_r_w_s_dff
DW_ram_r_w_s_lat	DW_ram_rw_a_dff	DW_ram_rw_a_lat
DW_ram_rw_s_dff	DW_ram_rw_s_lat	DW_reset_sync
DW_sla	DW_sqrt_seq	DW_sra
DW_stack	DW_stackctl	DW_stream_sync
DW_sync	DW_tap_uc	DW_tap
DW_wc_d1_s	DW_wc_s1_s	DW_fp_dp2
DW_fp_dp3	DW_pricod	DW_fifoctl_2c_df
DW_log2	DW_exp2	DW_fp_div_seq
DW_thermdec	DW_data_qsync_lh	DW_dct_2d
DW_fifo_2c_df	DW_asymfifoctl_2c_df	DW_decode_en
DW_fp_add_DG	DW_fp_addsub_DG	DW_fp_cmp_DG
DW_fp_dp4	DW_fp_sub_DG	DW_fp_mac_DG
DW_fp_mult_DG	DW_fp_exp	DW_fp_exp2
DW_fp_ifp_conv	DW_fp_invsqrt	DW_fp_ln
DW_fp_log2	DW_fp_mac	DW_fp_recip
DW_fp_sincos	DW_fp_sqrt	DW_fp_square
DW_FP_ALIGN	DW_arb_rr	DW_ifp_addsub
DW_ifp_fp_conv	DW_ifp_mult	DW_ln

TABLE 12 DesignWare Modules that Require Design Compiler License

DW_sincos	DW_sqrt_rem	DW_lp_piped_fp_mult
DW_lp_piped_fp_recip	DW_lp_piped_fp_sum3	DW_lp_piped_mult
DW_lp_piped_sqrt	DW_lp_piped_prod_sum	DW_lp_pipe_mgr
DW_lp_piped_div	DW_lp_piped_ecc	DW_lp_piped_fp_add
DW_lp_piped_fp_div	DW_lp_fifoctl_1c_df	DW_lp_fifo_1c_df
DW_asymdata_inbuf	DW_asymdata_outbuf	DW_data_qsync_hl
DW_ram_r_w_2c_dff		

The following table lists DesignWare modules that do not require the Design Compiler license, because Verilog netlist is available for such modules:

TABLE 13 DesignWare Modules not Requiring Design Compiler License

DW01_absval	DW01_add	DW01_addsub
DW01_ash	DW01_binenc	DW01_bsh
DW01_cmp2	DW01_cmp6	DW01_csa
DW01_dec	DW01_decode	DW01_incdec
DW01_mux_any	DW01_prienc	DW01_satrnd
DW01_sub	DW02_mac	DW02_multp
DW02_mult_2_stage	DW02_mult_3_stage	DW02_mult_4_stage
DW02_mult_5_stage	DW02_mult_6_stage	DW02_mult
DW02_prod_sum	DW02_sum	DW02_tree
DW03_bictr_dcnto	DW03_bictr_decode	DW03_bictr_scnto
DW03_lfsr_dcnto	DW03_lfsr_load	DW03_lfsr_scnto
DW03_lfsr_updn	DW03_pipe_reg	DW03_shftreg
DW03_updn_ctr	DW04_par_gen	DW_addsub_dx
DW_bin2gray	DW_cntr_gray	DW_div_pipe

TABLE 13 DesignWare Modules not Requiring Design Compiler License

DW_div	DW_gray2bin	DW_inc_gray
DW_lbsh	DW_lod	DW_lsd
DW_lzd	DW_minmax	DW_mult_dx
DW_mult_pipe	DW_norm_rnd	DW_norm
DW_prod_sum_pipe	DW_rash	DW_sqrt_pipe
DW_rbsh	DW_shifter	DW_square
DW_squarep	DW_sqrt	DW_pl_reg
DW_lza	DW02_prod_sum1	

Using DesignWare Functions

SpyGlass currently supports only the following DesignWare functions:

- DWF_bin2gray
- DWF_gray2bin
- DWF_inc_gray

If you invoke any of these functions in an RTL, SpyGlass handles them during synthesis.

Specifying Pragmas in HDL Code

You can specify certain pragmas in your code. SpyGlass supports different types of pragmas for Verilog and VHDL code.

Supported Pragmas for Verilog

SpyGlass supports translate_off/translate_on pragmas for Verilog. Within these pragmas, SpyGlass ignores the code for compilation, syntax checks, etc.

Supported Pragmas for VHDL

SpyGlass supports translate_off/translate_on and synthesis_off/ synthesis_on pragmas for VHDL.

For more information, see *Synopsys translate_off/on Pragmas*.

Working with Black Boxes

SpyGlass infers black boxes whenever it cannot find a model for a design unit. Often, these are inadvertent and should be resolved by supplying the models for them. The table below describes various sources in which black boxes are not expected:

Sources of unexpected black boxes	**Resolution**
Design HDL	Check that the HDL files are supplied correctly and are synthesizable. Refer to section "Basic Design-read" for details.
precompiled library	Check that the HDL library was compiled correctly. Refer to section "Precompiling HDL into a library" for details.
DesignWare Components	Enable the DesignWare compilation feature. Refer to section "Dealing with DesignWare Components" for details.
Technology library	Provide the .lib or .sglib technology file. Refer to the "Compiling technology libraries" section for details.

You should replace models for un-synthesizable design units with synthesizable models. The *ReportUnsynthesizedDU* reports the un-synthesizable design units.

Inferring Black Boxes

When SpyGlass analyzes a design containing ASIC cell instances or instances of modules not defined in the source files, SpyGlass checks for the corresponding cell or module definition in the associated Synopsys library (.lib file). Instances for which the corresponding cell or module definition is found, SpyGlass inserts the correct port interface. For all other instances, SpyGlass assumes them as black boxes with a port interface containing all input/inout ports. However, this assumption may result in problems such as feedback loops around the inout ports after synthesis.

To address this problem, SpyGlass provides the black box inference feature. SpyGlass use this feature to heuristically determine black box module wrappers (port names, port sizes, port directions, port order, and the module parameters) for each black box type by analyzing black box instances.

This feature improves SpyGlass performance for designs with instances of real black boxes (that is, modules whose definition is not available at that time). Ideally, designs should not have any black boxes.

For Verilog black box instances in mixed-language mode, SpyGlass infers similar information as in the Verilog-only mode. SpyGlass, however, skips processing of black box instances that appear in both Verilog and VHDL design units.

NOTE: *SpyGlass understands module interface definition in Synopsys Liberty^{TM} files (*.lib *files), and hence, the black box inference feature is required for real black boxes only.*

NOTE: *The black box inference feature is restricted to Verilog design flows only as VHDL designs strictly require component declarations before use and hence do not require such preparatory steps.*

Understanding the Black Box Inference Feature

The black box inference feature works at the following two levels:

- After RTL analysis

  SpyGlass heuristically determines the port information as follows:

Port Attribute	**Inferring Method**
Number of ports	From instance port map
Port names	For named port connections in the black box instance, the port names are the same as those specified in the design. For positional port mapping, SpyGlass uses an internally defined naming method.
Port directions	All ports are assumed to be of type input.
Port sizes	Same as the width of the widest (vector) signal connected to the port when checked across all instances of the same black box. Then, the right hand bit of the port width range is always 0, and the left hand bit is one less than the size of the widest connected vector.

When SpyGlass Verilog analyzer encounters the first instance of a black box, it creates a master module for it, with port/parameter interface matching the current instantiation. As it encounters more instances of the same black box, it enhances the port/parameter interface of the master module as required.

- After synthesis and flattening

  SpyGlass heuristically determines the port directions after flattening based on the connectivity of the net connected to the black box port:

Net connected to the port is	**Port Direction is inferred as**
A hanging net or is also connected to an inout port of a synthesizable module	inout
Set at least once and may or may not be read	input
Never set and is read at least once	output

Using the Black Box Inference Feature

To use the black box inference feature, specify the following command in the project file:

```
set_option inferblackbox yes
```

You must specify all available Verilog source files and the .lib files during design-read. SpyGlass processes only those modules for black box inference that are not defined in any of these files.

SpyGlass creates inferred black box wrapper modules in a file named sgBlackbox.v (created in the spyglass_sch directory in the current working directory) that has the port directions inferred at flattened netlist-level and uses it for rule-checking.

In this case, the RTL description-level rules may flag false messages or may not flag messages around the black box instances as all black box ports are assumed to be input type ports.

Checking the Inferred Information

If the port/parameter information inferred by this feature appears to be different from what you expected, review the sgBlackbox.v file or the sgBlackbox.v file (created in the spyglass_sch directory in the current working directory) that has the inferred black box wrapper module descriptions.

NOTE: *You must copy the* sgBlackbox.v *file from the* spyglass_sch *directory to another directory (to the current working directory, for example) as the* spyglass_sch *directory is overwritten after each SpyGlass run.*

Now check the sgBlackbox.v file for the following information:

- Number of black box wrappers modules

  If this number is very large, it indicates that a significant fraction of design is based on structural instances for which no information is supplied. Therefore, the result of SpyGlass analysis is potentially incomplete or inaccurate since a large number of such modules increases the macro-level uncertainty about interaction of these black box instances with other parts of the design. One possibility is that you have not supplied all the design files. In any case, you should ensure that all black box module wrappers are only those for which no information is actually available at the time of running SpyGlass.

- Black box wrapper module port interface

  For each (actual) black box wrapper module, you should inspect all port information — port names, port sizes, port directions, and order of ports. The inferred definition lists all instances based on which way the wrapper was inferred. Because all instances together may still not have certain information that you know otherwise, however, you should modify the inferred module wrapper for any of the port attributes. You can also add any ports that are never used in instances within that design.

- Black box wrapper module parameter interface

  For each (actual) black box wrapper module, you should also inspect parameter inferred because at least that many parameters have been used for parameter value override at the time of instantiation. However, almost all-existing SpyGlass functionality does not depend on the use of these black box instance parameter overrides. Hence, you can specifically look at the parameter interface only if the parameters actually affect the port size of some of the black box module ports.

NOTE: *If you find that certain port directions inferred for black boxes are not suitable for your design, use the following command in the project file to infer port directions:*

```
set_option inferblackbox_iterations <int-value>
```

Where *<int-value>* refers to the number of iterations (effort) spent by SpyGlass to converge on an estimate of port direction.

Using the Corrected Inferred Information

You can use the corrected inferred information for SpyGlass analysis run by running SpyGlass with all available Verilog source files, .lib files, the corrected sgBlackbox.v file and other required options.

Stopping Black Box Analysis

You can leave design units that do not have models as black boxes during design-read.

For advanced analysis, additional modeling information may be required that you can supply by using constraints. For example, a PLL will not have a synthesizable model, but you may need to model clock-paths through them depending on what is being analyzed.

To stop analysis forcefully when SpyGlass finds a black box, specify the following command in the project file:

```
set_option nobb yes
```

Handling Out of Memory Situations

You can overcome an out of memory failure by running SpyGlass on a machine configured with more physical RAM and swap space. By default, SpyGlass runs in 64-bit mode. If you are using 32-bit mode, you must switch to 64-bit mode so more memory is available.

If there are large synthesizable memory blocks in the design, you can greatly reduce the required amount of memory by specifying the following command in the project file:

```
set_option handlememory yes
```

The above command reduces the depth of memory and allows most types of analysis to continue unaffected.

There are cases where full memory model is required and specifying the above command will alter the analysis. For example, analyzing FIFO structure may require the full depth of the memory core used. The memory cores used in these types of structures are generally small and not affected by the command above. To control the minimum size for memories affected by the large memory-handling feature, specify the following command in the project file:

```
set_option mthresh <value>
```

The default value is 4096.

NOTE: *If SpyGlass runs out of memory during analysis, it will exit with the following error message:*

```
>% ERROR[100] Memory Allocation Failed. Exiting …
```

Setting options to reduce memory size affects some analysis results. For details, see *Memory Reduction Feature*.

Reporting Messages at Module Boundary

SpyGlass allows you to report flat-level rule messages up to the instance boundaries of specified module types (Verilog 'celldefine modules, Library modules, and user-specified modules).

Flat-level rule-checking in SpyGlass assumes only leaf-level instances in the flat netlist. These instances can be SpyGlass synthesis tool primitives/ macros instances, .lib instances, or black box instances.

However, you may at times want to treat specified module definition as a single unit for flat-level rule-checking, meaning that analysis should work as if there are only instances of these modules in the flattened netlist.

A typical case in which you would prefer this behavior is for Interface Logic Model (ILM) models of black boxes. Here, you provide the basic interface details of black box modules that are sufficient for rule-checking and want SpyGlass to use whatever it needs from inside the module definition (in terms of connectivity of input to output, etc.) but should not report any messages for nodes inside it.

NOTE: *This feature is not applicable for built-in messages. It is only applicable for messages of product rules that have been enhanced for this feature.*

SpyGlass refers to such module definitions as BBOX_MODEL type modules as these are effectively black boxes from the point-of-view of SpyGlass reporting and SpyGlass does not report anything inside these modules.

Identifying Modules

The current implementation understands the following type of modules as BBOX_MODEL type modules:

1. Black box ILM model

   SpyGlass identifies a design unit (Verilog/VHDL) as a BBOX_MODEL type module if its definition contains the following SpyGlass bbox_model pragma:

   ```
   // spyglass bbox_model (For Verilog)
   ```

   ```
   -- spyglass bbox_model (For VHDL)
   ```

   The BBOX_MODEL type module definition contains basic interface-level details with some logic around the interface so that the details are sufficient for SpyGlass rule-checking.

2. Verilog 'celldefine Modules

   The Verilog 'celldefine modules can be identified as possible BBOX_MODEL type modules as SpyGlass can uniquely recognize them based on the corresponding 'celldefine directive.

3. SpyGlass-compiled Gates Library Modules

   SpyGlass identifies all gates in a SpyGlass-compiled gate library as BBOX_MODEL type modules.

Enabling the Feature

The DEFAULT_BBOX_MODEL configuration key in the SpyGlass Configuration file (the .spyglass.setup file) allows you to define BBOX_MODEL types to be recognized for SpyGlass rule-checking.

NOTE: *For details about designing and using the SpyGlass Configuration file, see The Configuration File in SpyGlass.*

NOTE: *The override order for the DEFAULT_BBOX_MODEL configuration key values is $CWD (highest) > $HOME > $SPYGLASS_HOME (lowest).*

Impact of the Feature

When you enable the BBOX_MODEL type module recognition feature and provide the required valid details, the impact on the SpyGlass rule-message reporting is as follows:

1. SpyGlass ignores design units identified as BBOX_MODEL type modules for RTL description-level rules and hierarchical netlist-level rules.
2. If the rule-violation involves “internal” nodes of the BBOX_MODEL type modules, then the rule message reports at the boundary of the BBOX_MODEL type module instantiation.
3. The schematic windows in SpyGlass show the BBOX_MODEL type module instants as black boxes. You cannot traverse inside these modules.
4. The rule-violation highlighting is always at the boundary of the BBOX_MODEL type module instantiation when a rule message involving the “internal” nodes is viewed.

Controlling the RTL Synthesis Engine

As explained earlier, once SpyGlass completes its pre-processing, it analyzes your RTL design in one, two or three steps, depending on the rule checks you request.

1. SpyGlass checks standard style and SpyGlass lint solution rules and then logs messages in the Violation Database. If you request no other checks, SpyGlass Analysis ends at this point.
2. If you request rule checks that require inferred logic, SpyGlass accesses its internal RTL synthesis engine. The engine creates a design using generic gates. It then uses this design to detect inferred elements such as latches, flip-flops, and counters. Each element contains references back to your RTL HDL description, letting SpyGlass relate message reports directly to your source code. The design coming out of the synthesis engine in this second step is hierarchical.

   Definitions of the generic gates used by the SpyGlass synthesis engine are available in the <your-inst-dir>/SpyGlass-x.y.z/SPYGLASS_HOME/auxi/target_libs/generic/rtlc.prim.v and <your-inst-dir>/SpyGlass-x.y.z/SPYGLASS_HOME/auxi/target_libs/generic/rtlc.prim.vhdl for Verilog and VHDL cells respectively.

NOTE: *If SpyGlass detects HDL syntax messages in your RTL design, it will not synthesize your code and will not proceed to the second and third steps.*

3. Synthesis converts your high-level circuit description into a hierarchical netlist of generic gates.

   If you still want to perform rule-checking that are best performed on a flat netlist (such as synchronization logic, combinational loops, and reset rules), SpyGlass runs a flattener on the hierarchical netlist. Checks in this final step can include any form of netlist checking.

   During the flat netlist rule-checking, SpyGlass ignores empty top-level design units because it serves no purpose to perform netlist-level rule checking on empty design units.

Limiting Analysis of Memories

If you include memories (two-dimensional register arrays) in your Verilog/VHDL design, you can have SpyGlass analyze them for connectivity messages after the synthesis step. SpyGlass can generate a register and associated connections for each bit of memory it compiles.

As the size of memory arrays increases, however, the real memory and runtime requirements for the analysis increase dramatically. At some point, you are better off compiling small arrays into registers and treating larger arrays as black boxes. Use the following project file option to set an upper limit on the size of the memory arrays compiled into registers:

```
set_option mthresh <value>
```

The SpyGlass default size for compiling memories is 4096 (4K) bits. It is not recommended to set the threshold to a very large number because of hardware memory requirements and runtime degradation.

NOTE: *SpyGlass reports how many bits of memory it finds in each module as part of its standard runtime dialog. You can decide on a reasonable memory limit after running one pass without using this feature.*

We recommend that you define large memories in a separate module. This allows SpyGlass to continue checking your design in detail, since if it encounters a large memory it black boxes that module and has no impact on any other part of the design. This also makes it easier to replace the RTL definition of a memory with an explicit array if you are using one for a particular ASIC.

Preserving all instances and nets in a design

The SpyGlass RTL synthesis engine normally retains hanging nets (open-ended interconnections among logic gates) and hanging/ unconnected instances. If you want to remove these instances and nets in your design so that the result matches that of your main synthesis engine, you can do so by supplying the `set_option nopreserve yes` command in the project file. Preserving all instances and nets makes it easier for you to relate inferred logic back to your source code.

Interpreting Synthesis Pragmas

SpyGlass reads most synthesis pragmas that affect RTL description and some synthesis pragmas that effect the generation of netlist.

Built-in VHDL Synthesis Pragmas

SpyGlass interprets all Built-in VHDL synthesis pragmas except

SYN_INTEGER_TO_BIT_VECTOR and SYN_X_EQL pragmas.

Optimization-Related Synthesis Pragmas

SpyGlass ignores optimization-related synthesis pragmas.

Analysis-Related Synthesis Pragmas

SpyGlass reads and interprets the following analysis-related synthesis pragmas:

Synthesis Pragma	Applicable for...		Whether interpreted?
	Verilog	VHDL	
translate_off/translate_on	Yes	Yes	Yes
synthesis_off/synthesis_on	Yes	Yes	Yes
analysis_off/analysis_on	No	Yes	No
force_off/force_on	No	Yes	No
dc_script_begin/dc_script_end	Yes	Yes	No
full_case, parallel_case	Yes	NA	Yes
state_vector	Yes	No	No
enum	Yes	No	Yes
goal	Yes	No	No
map_to_module	Yes	No	No
return_port_name	Yes	No	No
resource	Yes	No	No

Interpreting Synthesis Pragmas

SpyGlass interprets different types of synthesis pragmas, as discussed in the following examples.

Synopsys translate_off/on Pragmas

Use this design-read option to ignore VHDL code within the pragma block, Synopsys `translate_off/translate_on`.

Setting this field is equivalent to specifying the following command in the Atrenta Console project file:

```
set_option hdlin_translate_off_skip_text yes
```

By default, this option is enabled. To disable this option, set the value of the *disable_hdlin_translate_off_skip_text* command to `yes`.

Interpreting Synopsys synthesis_off/on Pragmas

By default, SpyGlass ignores the VHDL design code block between Synopsys synthesis_off/synthesis_on pragmas for synthesis but not for analysis. Therefore, SpyGlass runs syntax-checking and RTL description-level rules on the design code block.

Now consider that you specify the following command in the project file:

set_option hdlin_synthesis_off_skip_text yes

When you specify the above command, SpyGlass considers the design code block lines as comment lines and does not perform syntax-checking and RTL description-level rule-checking.

The hdlin_synthesis_off_skip_text option has no effect on the interpretation of Synopsys translate_off/translate_on pragmas.

Managing the Design Hierarchy

Managing design hierarchy enables you to perform bottom-up debug by defining a lower hierarchy, analyzing it, moving to a higher point in the design hierarchy, and running analysis from that point down.

You can black box lower-level blocks (for example, it is being worked on by another member of the project, or it is a hard macro, or if the block has previously been analyzed).

You can manage your design hierarchy in the following ways:

- By specifying design files that should be analyzed individually.
- By specifying design files that should be analyzed as a complete hierarchy that contains a top module followed by various child modules.

Specifying a Top-level Design Unit

A top-level design unit is a module interpreted on a design hierarchy in such a way that all design units instantiated directly or indirectly under the specified top-level design unit is included in the scope of SpyGlass analysis.

All the remaining designs units in specified source file(s) that are not instantiated in top-level design unit are excluded from the scope of SpyGlass analysis.

NOTE: *The DetectTopDesignUnits rule reports top-level design units.*

NOTE: *For VHDL designs, SpyGlass can automatically determine the top of a hierarchy if you have specified the sort option in the set_option command.*

Advantage of Specifying a Top-Level Design Unit

Setting a top-level design unit improves CPU time and memory requirements to a significant extent due to the following reasons:

- You can restrict SpyGlass analysis to a design hierarchy starting from a top-level design unit.
- You can ignore design units that are not relevant for current analysis.

Setting a Top-Level Design Unit

To set a top-level design unit, use the following command in the project file:

```
set_option top <top>
```

Alternatively, specify the name of the top-level module in the *Top Level Design Unit* field under the *Set Read Options* tab.

NOTE: *If the stop option was used on a design unit with a hierarchy, any modules that are unused by other design units would be inferred as top design units.*

It is highly recommended that you set a top-level design unit during *Stage 1: Setting up the Design (Design Setup)* in all SpyGlass runs, except while precompiling HDL libraries.

Multiple Top-Level Design Units

When you run the design read process, SpyGlass may encounter multiple top-level design units due to the following reasons:

- Incomplete elaboration

  Check if there are some ELAB errors. SpyGlass cannot create a hierarchy tree correctly because of these ELAB errors. As a result, an underlying module becomes an independent top-level module.

- HDL libraries specified incorrectly

  In this case, SpyGlass treats unused design units as independent design units, that is, top-level design units.

SpyGlass allows you to specify multiple top-level design units for lint type analysis. However, a single top-level design unit is generally expected for more advanced analysis.

Language-Specific Behavior While Specifying a Top-Level Module

Specifying top-level module(s) affect the use-model and flow of SpyGlass at various stages of design reading, analysis, synthesis and rule-checking. Due to differences in semantics of Verilog and VHDL design descriptions, there are certain unavoidable differences in functionality of the top-level feature. Please be aware of such cases.

The following table describes language-specific behavior while specifying top-level modules:

SpyGlass Processing	**Verilog design**	**VHDL design**
Design Input	The whole design should not contain any syntax errors. Although SpyGlass may try to skip syntax error in design units outside the specified top-level design unit hierarchy, such behavior cannot be guaranteed and is discouraged.	
Design Analysis	SpyGlass reports syntax errors and warnings in a design hierarchy within the specified top design units only. Such semantics are compatible with use of the +top-module option in other Verilog tools.	SpyGlass performs analysis of all VHDL design units, and therefore, reports VHDL syntax errors for all design units. The reason for such behavior is that SpyGlass processes the top-level feature in VHDL design during elaboration stage and, therefore, requires analysis of full VHDL design.
Design Elaboration	-	SpyGlass completes VHDL the design elaboration starting at design unit specified with the Top-Level feature. This step determines the exact design hierarchy under the specified “top” design unit, and takes care of binding as per the VHDL language requirements.

SpyGlass Processing	Verilog design	VHDL design
Design Unit list of RTL rule-checking	SpyGlass computes this list based on Verilog Object Model created after use of the Top-Level feature at Verilog analysis time. All SpyGlass rules (excluding Verilog language syntax error) are checked on this list.	SpyGlass computes this list from VHDL elaboration of the specified design unit. SpyGlass performs all the rules checks on this list, excluding VHDL language syntax error and semantic error or warnings.
Design units for hierarchical and flat netlist rule checks	SpyGlass performs design synthesis for design units under the specified (Top-Level) design units only. Therefore, SpyGlass performs netlist rules checking on such design units only.	
Lexical Rules	SpyGlass performs lexical rules checking on input Verilog or VHDL design source. The list of RTL modules falling within the specified design hierarchy is passed to lexical rules. Hence, rule-check within design unit (or module) boundary is restricted to specified design units only. However, for HDL code not within any design unit, the rule-checking shall depend on the nature of lexical rule.	

NOTE: *Functionality of the Stop feature is closely linked with the Top-Level feature in guiding a design hierarchy to be checked. In contrast to the Top-Level feature, there are no syntax/semantic errors or warning messages (except for elaboration errors and AnalyzeBBox messages) reported on design units specified by the Stop feature.*

Stopping Design Units

You may need to stop some design units in the following cases:

- If some design units are currently placeholders for some information to be added later. For example, a pre-designed code or intellectual property is yet to be provided.
- If a particular design unit is still under development.

To prevent analysis of such design units, use the following command in the project file:

```
set_option stop <design-unit>
```

Alternatively, specify the name of such design units in the *Stop Design Unit(s)* field under the *Set Read Options* tab.

NOTE: *You can specify the* `stop` *command with the* `stopdir` *and the* `stopfile` *commands.*

Implications After Stopping Design Units

Stopping design unit(s) has the following implications:

- To ensure successful parsing of the whole design, the stopped design unit can not have any syntax error. Presence of a syntax error in a stopped design unit may interfere in identifying the boundary of such design unit itself, and therefore interfere in parsing of the remaining design units that may exist later in the design source file(s).
- No rule-checking occurs on stopped design units for rules that work at a design unit level, such as RTL, ELAB, LEXICAL, and VSDU.

  For rules, such as VSTOPDU or FLAT rules that work on a complete design, a stopped design unit is considered as a black box during traversal.

  SpyGlass does not report any built-in messages for stopped design units except for elaboration errors and AnalyzeBBox messages. SpyGlass reports elaboration errors to guide you about a design hierarchy that is being checked, whereas the *AnalyzeBBox* rule messages reports about design units that have been stopped and whether that matches your expectation.
- No rule-checking occurs for all the design units instantiated within a stopped design unit.

  However, if SpyGlass analysis is for the whole design, the underlying hierarchy below such stopped design units can be treated as independent top modules, and rule-checking can then be done for such modules. In either of these cases, the generic or parameter value expected to be passed from top design hierarchy cannot be passed to design hierarchy below the stopped design unit.

Checks Performed on Stopped Design Units

SpyGlass performs the following checks on design units marked as stopped:

- Lexical checks (Line length, use of tab, indents)
- Name checks (both unique and reserved names)
- Checks disallowing use of specified synthesis pragmas
- Checks on the use of sufficient numbers of parentheses in expressions
- Checks on the use of hard constants
- Checks that should not disable out of a loop
- Checks on the use of multi-line comments

Using the Top and Stop Features Together

Consider the design hierarchy, as shown in the following figure:

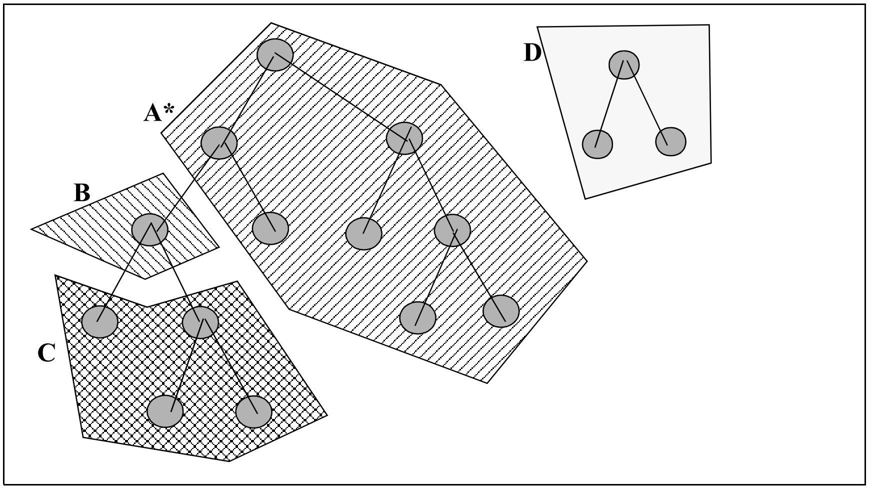


FIGURE 148. Design Hierarchy for Use of Top-Level and Stop Features

In above figure:

- A denotes the higher-level design hierarchy described by shaded area A*, in which the design unit A is the top-level design unit.
- B denotes the design unit to be excluded by specifying the following command in the SpyGlass project file:

  ```
  set_option stop <design_unit>
  ```

- C denotes the design units that were instantiated in the design unit B. Therefore, all these design units were a part of the design hierarchy under A if the design unit B was not stopped. However, as implementation inside B is hidden for SpyGlass analysis, rule-checking may or may not occur for design units under C depending on usage of other related options. See Table: *Using Top-Level and Stop Features Together* for examples that illustrate control of design hierarchy for rule-checking by use of various options.
- D denotes another design hierarchy independent from A, B, or C.

The following table describes use of above two options, when applied to the design hierarchy illustrated in Figure :

TABLE 14 Using Top-Level and Stop Features Together

Options used	**Design units rule-checked**
Default (no features used)	A, B, C, and D
`Stop B`	A, C, and D
`Top A and Stop B`	A
`Top A`	A, B, and C
`Top D`	D
`Top D and Stop B`	D (B is not in hierarchy of D and, hence, the Stop feature has no effect)

As seen from the above examples, whenever you stop a design from SpyGlass analysis, you must also specify a top-level design unit to declare the scope correctly and unambiguously.

SpyGlass does not report any parsing message except for elaboration errors on design units in C when you specify A as top and B as stopped.

Notes

Please note the following points regarding the use of any of the above features:

- The options for hierarchical inclusion/exclusion of design units are the top-level and stop features. You cannot use these options simultaneously with an equivalent design unit feature, which is for immediate analysis of the specified design units.
- You cannot completely control syntax errors in a design source by using these options. The design is expected to be free from syntax and elaboration errors. Judicious use of the Top-Level and Stop features can help avoid design units that have known elaboration errors.
- The argument value specified with these options should carefully follow the syntax for representing design unit names in Verilog and VHDL.

Ignoring Files and Design Units From SpyGlass Analysis

An ignored file or a design unit is skipped from SpyGlass analysis at the design read (or parsing) stage.

You might need to ignore a file or a design unit from SpyGlass analysis because of the following reasons:

- When multiple designers work on different blocks of the same design.

  In such a scenario, different design blocks are in different states of maturity. So you might want to run SpyGlass analysis on design blocks for which development is complete, and ignore the rest.
- When some design blocks have some known fatal issues.

  In such cases, you can ignore such design blocks from SpyGlass analysis and test other design blocks.
- If a design unit contains a genuine black box, you can ignore such design units so that SpyGlass reports *InfoAnalyzeBBox* message instead of *ErrorAnalyzeBBox* message.

Difference between Ignored and Stopped Design Units

There is a subtle difference between ignored design units and stopped design units (that is, design units specified by the *set_option stop <du-name>* command).

When a design unit is stopped, SpyGlass only ignores the logic (or functionality) of that stopped design unit from rule-checking. However, SpyGlass performs checking on the interface of that design unit. Therefore, if the interface has any issues, SpyGlass reports appropriate violation messages.

However, when a design unit is ignored, SpyGlass skips the body as well as the interface of that design unit from compilation. Therefore, no violations are reported for an ignored design unit.

Ignoring Files Containing Design Units

When you ignore a file from SpyGlass analysis, SpyGlass treats the design units instantiated in a design of that file as black boxes.

Ignoring Files through GUI

To ignore a file, right-click on that file appearing in the *HDL Files* section under the *Add Design Files* tab and select the *Ignore File* option from the shortcut menu.

The ignored file is indicated with the ℹ icon prefixed to the design file name.

You can also ignore a file after the design read stage. To do so, right-click on that file appearing in the *File View* page and select the *Ignore File* option from the shortcut menu.

Ignoring Files through a Project File

To ignore a file, use the following command in the project file:

```
set_option ignorefile <file-name>
```

The following example ignores the design file `myaddlfile.v`:

```
set_option ignorefile myaddlfile.v
```

NOTE: *You can specify a relative or absolute path of a file in the above command.*

Support for Wildcard and Regular Expressions

You can use wildcard and regular expressions while specifying file names in the above command. The following table shows some examples of using various expressions in this command:

Example	Description
`set_option ignorefile {a*}`	Ignores all files (in the current directory) whose names match the wildcard a* expression. For example, a1, aa1, and abc.
`set_option ignorefile {dir1/*}`	Ignores all files in the dir1 directory
`set_option ignorefile {dir?/*}`	Ignores all files in directories that match the wildcard expression dir?. For example, dir1, dir2, and dir3.

If a file name includes wildcard characters, such as `*` or `?`, enclose such names in single quotes, and ensure that the wildcard characters appearing in the file name are preceded by a backslash (\). For example, if the name of the file to be ignored is `'abc*d'`, specify the file name as `'abc\*d'` in the `ignorefile` command.

However, if you want to ignore to two files, such as `abc1d` and `abc2d`, specify them as `"abc*d"` in `ignorefile` command.

NOTE: *You can specify the ignorefile command with the ignoredu command.*

Ignoring Individual Design Units

You can ignore a specific VHDL design unit or Verilog module at the design read (parsing) stage.

Ignoring Design Units through GUI

To ignore a VHDL design unit or a Verilog module, perform the following steps:

1. Display the *Module View* page by selecting the under the *Analyze Results* tab.

2. Right-click on a design unit or a module in the *Module View* page and select the *Set Ignore Module* option from the shortcut menu.

After performing the above steps, the selected design unit or module is ignored and the i icon appears before the name of that design unit or module.

Ignoring Design Units through a Project File

To ignore a design unit from SpyGlass analysis, use the following command in the project file:

```
set_option ignoredu <design-unit-name>
```

You can specify a design unit in any of the following formats:

Format	Description
For VHDL	
<entity-name>	Ignores the specified entity and all its architectures in all logical libraries
<entity-name>.<arch-name>	Ignores the specified architecture of the specified entity in all logical libraries
<lib-name>.<entity-name>	Ignores the specified entity and all its architectures for the specified logical library
<lib-name>.<entity-name>.<arch.name>	Ignores the specified architecture of the specified entity in the specified logical library
<lib-name>	Ignores the specified logical library
ALL.<arch-name>	Ignores all architectures of the name <arch-name> in all logical libraries
For Verilog	
<module-udp-name>	Ignores the specified module or UDP
<lib-name>.<module-udp-name>	Ignores the specified module or UDP from the specified logical library

The following example ignores the Verilog module `block1`:

```
set_option ignore block1
```

The following example ignores the Verilog modules `block1` and `block2`:

```
set_option ignoredu {block1 block2}
```

Support for Wildcard and Regular Expressions

You can use wildcard and regular expressions while specifying design unit names in the above command. The following table shows some examples of using various expressions in this command:

Example	Description
`set_option ignoredu {lib1.*}`	Ignores all design units picked from the lib1 logical library
`set_option ignoredu {e.*}`	Ignores all architectures of the `e` entity
`set_option ignoredu {*.e}`	Ignores all entities named `e` from any library

It is mandatory that escape wildcard characters are present in an escaped name. This is important to ensure that any unexpected matches do not occur. For example, you must specify `\a123*` as `"\a123\*"`.

In addition, please note that the following examples have different meanings:

- `set_option ignoredu {a1*}`

  This example matches `a11`, `a12`, and `a13`.

- `set_option ignoredu {\a1\*}`

  This example matches `\a1*`.

Analyzing Selective Design Hierarchy

You can perform selective synthesis and rule-checking on your design by specifying:

- Design units on which rule-checking should be done.

  You can specify such design units in the *Check IP* field under the *Set Read Options* tab.

- Design hierarchy (level) for which synthesis and rule-checking should be done.

  You can specify the design hierarchy in the *Check DU* field under the *Set Read Options* tab.

Working with 'celldefine Modules

By default, SpyGlass processes Verilog modules enclosed in the 'celldefine and 'endcelldefine compiler directives as normal source modules if the 'celldefine modules are described in a source file and processes them as normal library modules if the 'celldefine modules are described in a library file.

SpyGlass expects that the instances of such 'celldefine modules are given instance names. If you do not specify instance names, SpyGlass reports a warning and automatically names such unnamed instances as _SpyInst_0, _SpyInst_1, and so on.

Performing Rule-Checking on 'celldefine Modules

Use the check_celldefine option of the set_option command to perform rule-checking on the 'celldefine modules.

The following is the syntax of using this option:

```
set_option check_celldefine yes
```

The following table describes the effect of the absence of the check_celldefine option for different type of rules:

Rule Type	**Without check_celldefine option**
HDL File Parsing Rules	Language syntax errors checking and reporting
HDL Semantics Rules	No checking or reporting.
RTL description Rules	No checking or reporting.
Hierarchical netlist-level Rules	No checking or reporting. No messages are reported on `celldefine modules. However, SpyGlass reads and uses functional model of these cells.
FLAT netlist-level Rules	In full-design FLAT netlist view, no messages are reported on `celldefine modules. However, the functional model of these cells are read and used by SpyGlass.
Lexical Text source Rules	No checking or reporting.

Performing Hierarchical Rule-Checking in 'celldefine Modules

By default, SpyGlass ignores the top that is inside a 'celldefine module for rule-checking. To enable rule-checking on 'celldefine module top's hierarchy, specify the following command in the project file:

```
set_option allow_celldefine_as_top yes
```

Working with Methodologies

A methodology is a collection of sub-methodologies or a collection of goals. Each sub-methodology may further contain sub-methodologies or a set of goals where each goal is a collection of rules.

The following figure illustrates the sample structure of a methodology:

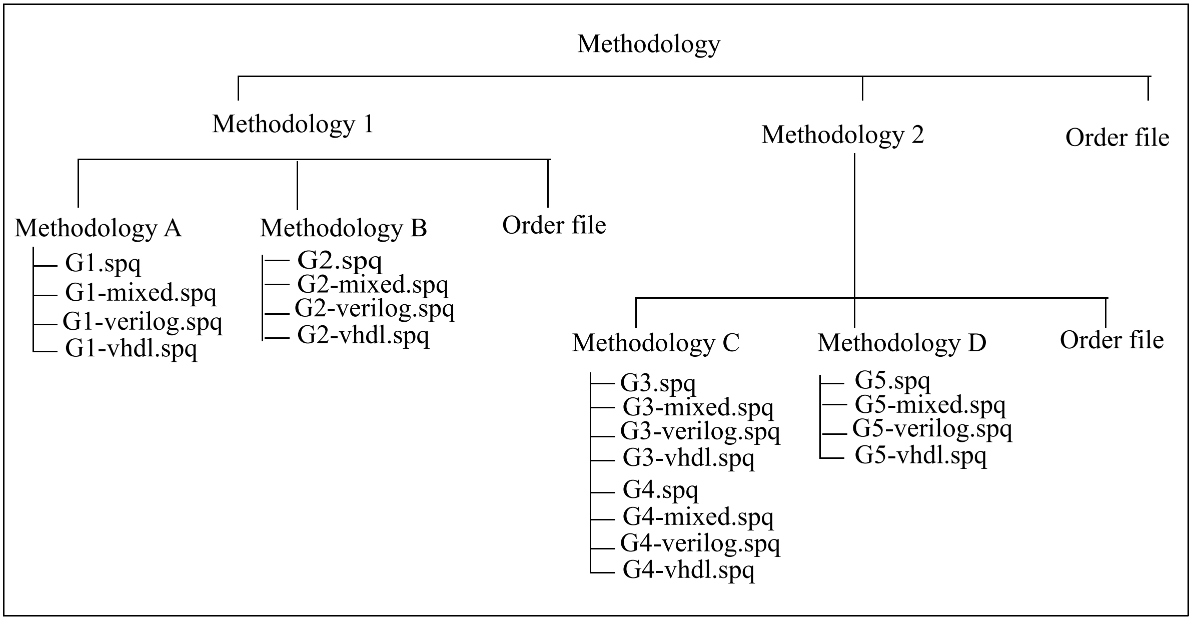


FIGURE 149. Sample Methodology Structure

NOTE: *Each methodology should contain an Order File that contains entries of goal file paths related to a methodology directory.*

This section explains the following topics:

- *Goal Files*
- *GuideWare Reference Methodology*
- *Specifying an Active Methodology*
- *Specifying a Current Methodology*
- *Configuring a Methodology*
- *Creating Custom Methodologies*
- *Selecting a Custom Methodology*
- *Comparing Methodologies*
- *Copying and Inheriting Methodologies*
- *Migrating Custom Goals*

- *Order File*
- *Map File*

Goal Files

Details of each goal are present in a goal file (.spq file).

You add a goal file in a methodology by importing that goal file in the methodology. For details, see *Importing Goals*.

Naming Convention of a Goal File

The name of a goal file has the following naming convention:

<goal-name>-*<language>*.spq

In the above naming convention, the *<name>* argument should have alphanumeric characters.

The *<language>* argument is optional. Use this argument to create goals files for different languages, such as Verilog, VHDL, and mixed. For example, you can create goal files with the names info_data-mixed.spq, info_data-verilog.spq, and info_data-vhdl.spq. Therefore, depending upon the language in your design, the tool automatically picks an appropriate goal file. For details, see *Selection of Goal Files based on Language Mode*.

NOTE: *It is not necessary to create goal files for each language.*

Details Present in a Goal File

In each goal file, you can specify details, such as goal name, language applicable, short help of the goal, detailed description of the goal, rules of the goal, and parameter settings for that goal.

The following is a sample goal file, info_data:

```
=template++++++
info_data mixed //goal name and language
*
Informational data //short help
*
This goal is used to report informational data related to a
design. //detailed description
```

```
=cut+++++++++++

//-------------------------------------------------
// Policy Registration
//-------------------------------------------------

-policies=Audits,area,clock-reset,erc,lint

//-------------------------------------------------
// General Setup commands
//-------------------------------------------------

-mixed //Allow mixed language

//-------------------------------------------------
// Policy Specific Parameter Setting
//-------------------------------------------------

-enable_handshake=yes
-enable_fifo=strict
-distributed_fifo=yes

//-------------------------------------------------
// Rule Registration
//-------------------------------------------------

-rules Audit2ID
-rules Audit2Stats
-rules Clock_info03
-rules Clock_info05
-rules W438
-rules LogicDepth
-ignorerule CMD_define_severity02

//-------------------------------------------------
// End of Rule Registration
//-------------------------------------------------
```

Selection of Goal Files based on Language Mode

A goal can have multiple goal files. For example, the goal `myGoal` can have the following files:

- myGoal.spq
- myGoal-mixed.spq
- myGoal-verilog.spq
- myGoal-vhdl.spq

When you run a goal, SpyGlass picks one goal file based on whether you have specified a language mode or not.

Selection of a Goal File when the Language Mode is none or mixed

In such cases, the following preference is used (starting from high priority) to pick a goal file:

1. <goal-name>.spq
2. <goal-name>-mixed.spq

Selection of a Goal File when the Language Mode is Verilog or VHDL

In such cases, the following preference is used (starting from high priority) to pick a goal file:

1. <goal-name>-<verilog | vhdl>.spq
1. <goal-name>.spq
2. <goal-name>-mixed.spq

GuideWare Reference Methodology

GuideWare reference methodology is the default methodology used by SpyGlass.

This methodology provides guidance to chip designers to address various design issues by running a set of goals that are fine-tuned for high-quality results and low noise. These goals are organized in a specific manner in three fields of use that GuideWare supports for an SoC design development flow.

Structure of the GuideWare Reference Methodology

The following figure illustrates the structure of the GuideWare methodology:

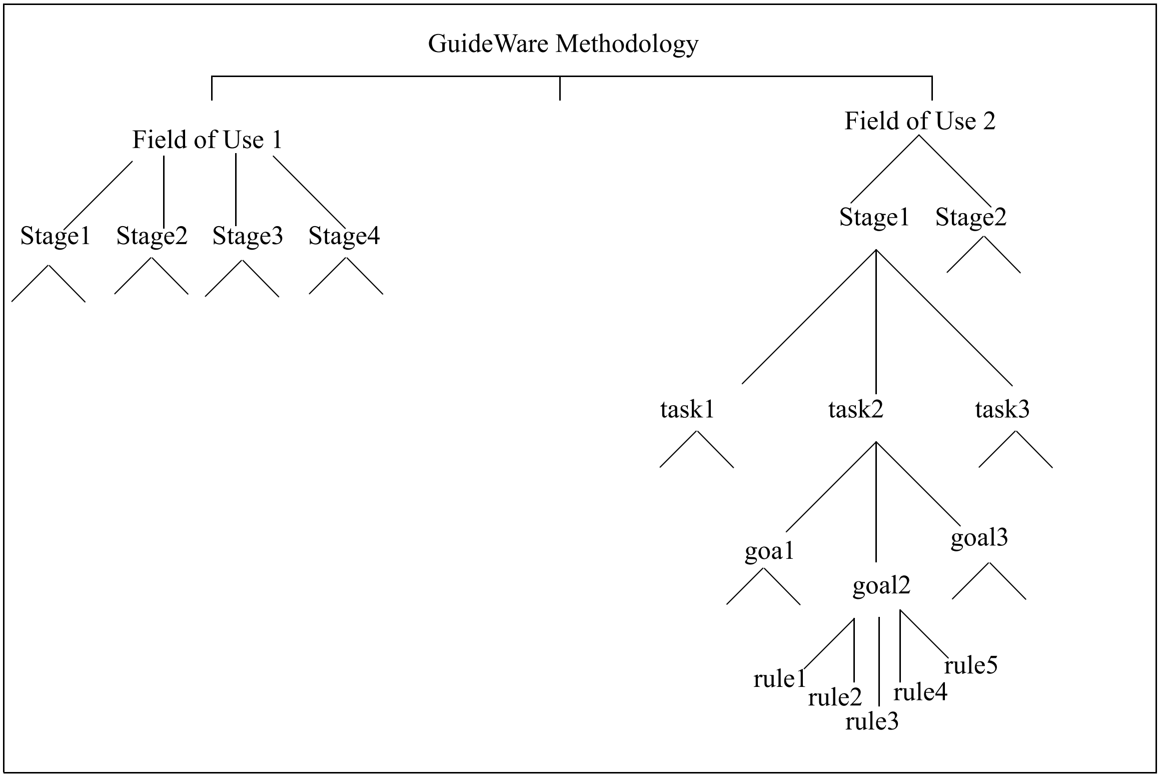
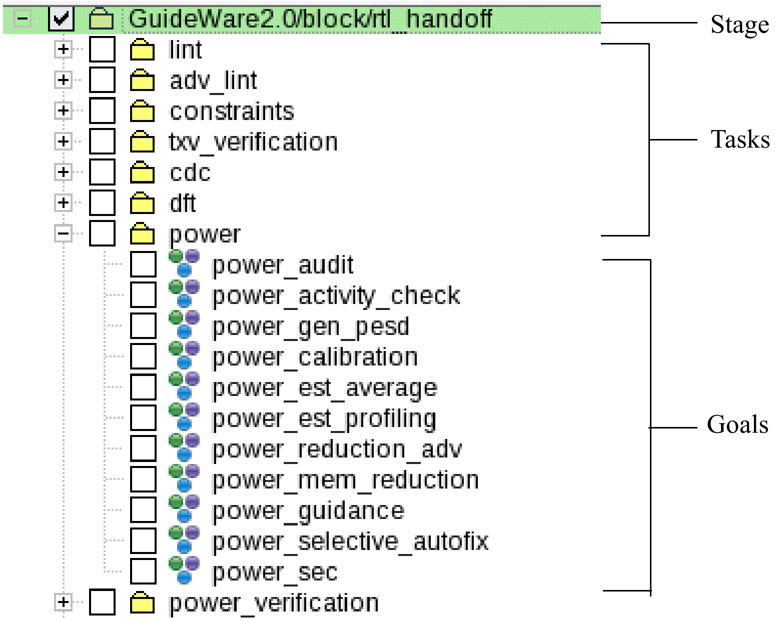


FIGURE 150. GuideWare Methodology Structure

The following table contains details of various components of the above structure:

Component	Description
GuideWare Reference Methodology	Contains a set of methodologies that are aligned with the chip development process. These methodologies provide guidance to designers to address design issues throughout the SoC development flow. The guidance is provided in the form of goals that are fine-tuned for high quality results and low noise.
Field of Use (methodology)	Refers to a phase in an SoC flow. **NOTE:** *Refer to SpyGlass GuideWare User Guide for details on various fields of use and available goals.*
Stage (milestone)	Refers to a sub-methodology, known as a design stage in a particular field of use.
Task	Refers to a sub-methodology that contains a set of goals.
Goal	Refers to a collection of rules.
Rule	Refers to a check to detect a specific type of design issue.

For example, the following figure illustrates the alignment of goals in the `New_RTL` field of use:



```
⊟ ☑ GuideWare2.0/block/rtl_handoff
    ⊞ □ lint
    ⊞ □ adv_lint
    ⊞ □ constraints
    ⊞ □ txv_verification
    ⊞ □ cdc
    ⊞ □ dft
    ⊟ □ power
          □ power_audit
          □ power_activity_check
          □ power_gen_pesd
          □ power_calibration
          □ power_est_average
          □ power_est_profiling
          □ power_reduction_adv
          □ power_mem_reduction
          □ power_guidance
          □ power_selective_autofix
          □ power_sec
    ⊞ □ power_verification
```

FIGURE 151. Goal Alignment in New_RTL Field of Use

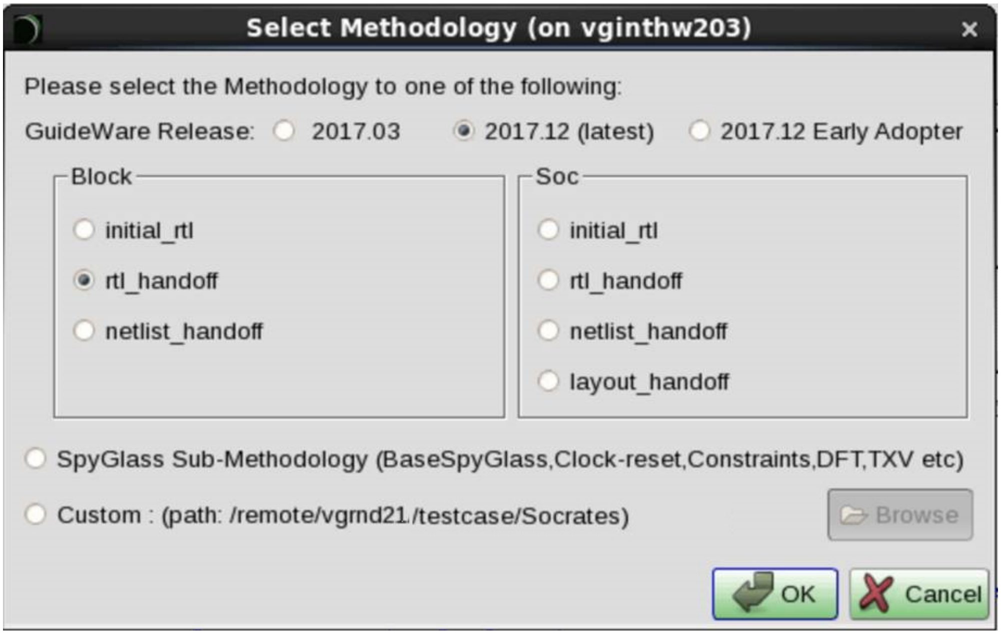
Specifying an Active Methodology

An active methodology refers to a default methodology that gets loaded when you start SpyGlass.

To specify an active methodology, perform the following steps:

1. Click the *Select Methodology* link under the *Goal Setup & Run* tab.

   The *Select Methodology* dialog appears, as shown in the following figure:



Select Methodology (on vginthw203)

Please select the Methodology to one of the following:

GuideWare Release: ○ 2017.03 ● 2017.12 (latest) ○ 2017.12 Early Adopter

Block

○ initial_rtl

● rtl_handoff

○ netlist_handoff

Soc

○ initial_rtl

○ rtl_handoff

○ netlist_handoff

○ layout_handoff

○ SpyGlass Sub-Methodology (BaseSpyGlass,Clock-reset,Constraints,DFT,TXV etc)

○ Custom : (path: /remote/vgrnd21/testcase/Socrates)

FIGURE 152. Select Methodology

In the above dialog, you can select any option described in the following table:

Option	Description
GuideWare Release	Select one of the methodology versions: ● 2017.03 ● 2017.12 (latest) ● 2017.12 Early Adopter
Block	Select one of the following options based on the design stage for a block/IP: ● initial_rtl ● rtl_handoff ● netlist_handoff
Soc	Select one of the following options based on the design stage for a block/IP: ● initial_rtl ● rtl_handoff ● netlist_handoff ● layout_handoff
SpyGlass Sub-Methodology	Select this option to load SpyGlass goals, installed at: <your-inst-directory>/ SPYGLASS_HOME/Methodology directory
Custom	Select this option to specify a directory containing your own custom methodologies/ goals as an absolute path or a path relative to the current directory. After selecting this option, click the *Browse* button to browse to the required directory.

2. Click the *OK* button in the *Select Methodology* dialog.

After performing the above steps:

- The selected methodology becomes the active methodology.
- The following command is generated in the project file:

```
set_option active_methodology <methodology-name>
```

  The Next time you load the same project file, SpyGlass loads the methodology specified by the `active_methodology` option.
- The active methodology is loaded under the *Select Goal* tab.

- The active methodology gets loaded whenever you open the *Methodology Configuration System* (MCS) window.

Using the active_methodology Option

The active_methodology option tracks the last methodology that you selected before closing a project, and restores it back when the same project is loaded again.

If the active_methodology option is missing in the project file, SpyGlass considers the methodology for which any setup is present (as specified by current_methodology scope). However, if multiple methodology setups are present in the project file, SpyGlass considers the last specified methodology as the active methodology.

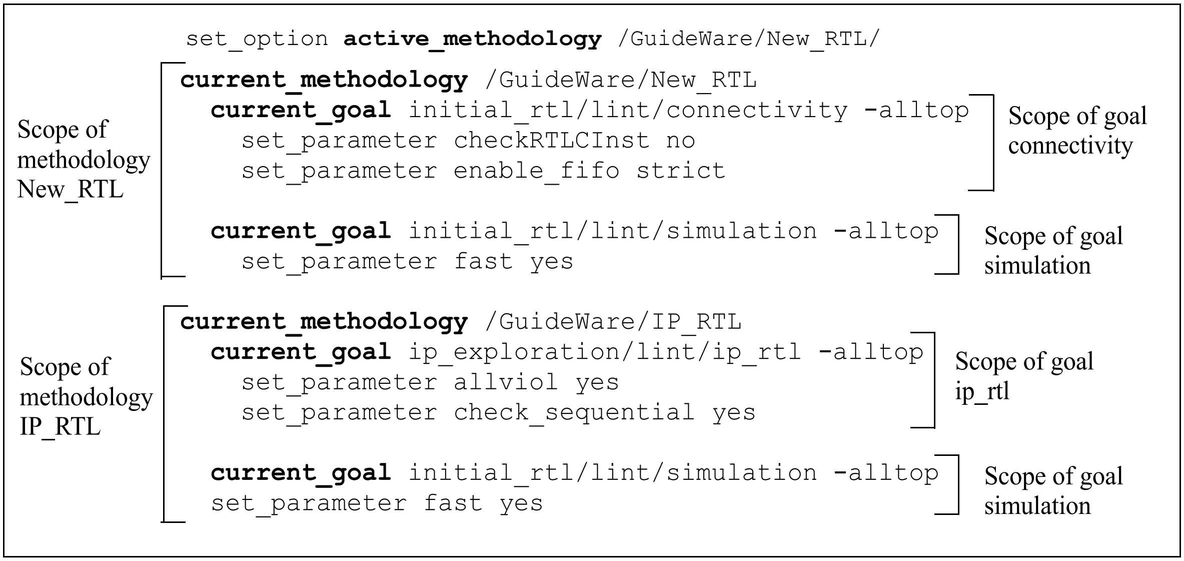
It is possible that both the active_methodology option and methodology setups are missing in the project file. In such cases, SpyGlass considers the default methodology as specified in the SpyGlass configuration file.

Specifying a Current Methodology

A current methodology defines a scope of each methodology selected in a particular session.

Defining the scope of each methodology allows goal setups for multiple methodologies to co-exist in a single project file without any name conflicts.

You can define a scope for each methodology by using the `current_methodology` command in a project file, as shown in the following example:



```
set_option active_methodology /GuideWare/New_RTL/
current_methodology /GuideWare/New_RTL
  current_goal initial_rtl/lint/connectivity -alltop
    set_parameter checkRTLCInst no
    set_parameter enable_fifo strict

  current_goal initial_rtl/lint/simulation -alltop
    set_parameter fast yes

current_methodology /GuideWare/IP_RTL
  current_goal ip_exploration/lint/ip_rtl -alltop
    set_parameter allviol yes
    set_parameter check_sequential yes

  current_goal initial_rtl/lint/simulation -alltop
  set_parameter fast yes
```

Scope of methodology New_RTL
Scope of goal connectivity
Scope of goal simulation
Scope of methodology IP_RTL
Scope of goal ip_rtl
Scope of goal simulation

FIGURE 153. Define Methodology Scope

For more information on the Tcl-based usage of the *current_methodology* command, refer to the *current_methodology* section of the *SpyGlass Tcl Shell Interface User Guide*.

Each `current_methodology` specification contains different commands corresponding to goal settings made within the scope of that methodology.

Therefore, within the scope of the selected methodology, you may run various goals and perform different goal settings, such as updating goal options and parameters. After saving the changes, you may select another methodology, run another set of goals, and specify goal settings within the scope of that methodology.

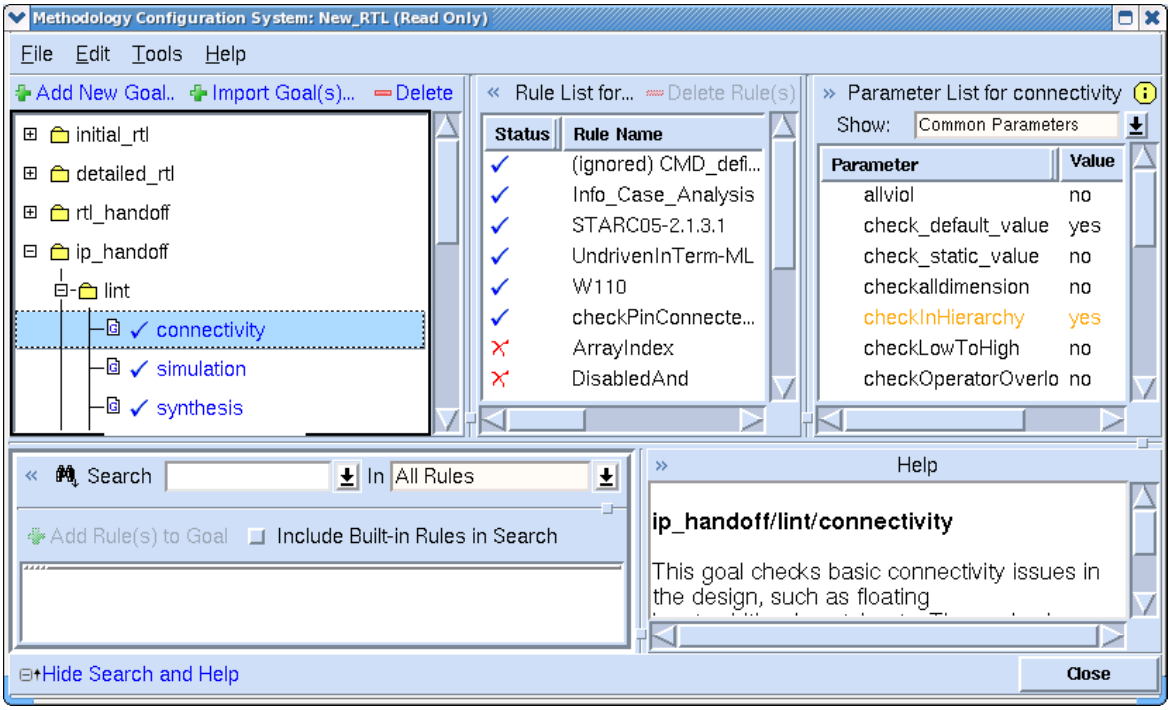
If no current_methodology is present for a current_goal specification, SpyGlass considers the current_methodology specification present in the configuration file.

Configuring a Methodology

To configure an existing methodology, open the *Methodology Configuration System* window by performing any of the following actions:

- Click the *Click here* link in the *Select Methodology* dialog.
- Select the *Tools-> Methodology Configuration System* menu option.

The following figure shows the *Methodology Configuration System* window:



Status	Rule Name
✓	(ignored) CMD_defi...
✓	Info_Case_Analysis
✓	STARC05-2.1.3.1
✓	UndrivenInTerm-ML
✓	W110
✓	checkPinConnecte...
✗	ArrayIndex
✗	DisabledAnd

Parameter	Value
allviol	no
check_default_value	yes
check_static_value	no
checkalldimension	no
checkInHierarchy	yes
checkLowToHigh	no
checkOperatorOverlo	no

ip_handoff/lint/connectivity

This goal checks basic connectivity issues in the design, such as floating

FIGURE 154. Methodology Configuration System

NOTE: *By default, the MCS window works in the mixed language mode. If you try to open the MCS window when the language mode is other than* `mixed`*, SpyGlass displays a Warning dialog. In this dialog, you can change the language mode by selecting the required language from the drop-down list.*

The *Methodology Configuration Window* is divided into various sections, as described in the following table:

Section	**Description**
Menu bar section	Provides various options that enable you to perform different functions, such as adding/saving methodologies, adding sub-methodologies, adding/ importing goals, and comparing methodologies. For details, see*The Methodology Configuration System Menu Bar*.
Toolbar section	Add/import/delete goals, select/deselect rules, etc.
Goals section	Lists sub-methodologies/goals that are available within a methodology.
Rules section	Lists the rules related to the selected goal in a spreadsheet format. The columns display the status of the rule (enabled/disabled), rule name, rule group, the product to which the rule belongs and the severity of the rule. You can show/hide the columns based on your requirement. To do so, right-click on the *Rules List* section, and select the *Configure Columns* shortcut menu option.
Parameter section	Lists parameters associated with the selected goal
Search section	Enables you to search for rules in the rule list of all products or in the current methodology.
Help section	Displays the help for a sub-methodology, goal, rule, or parameter, based on the selection

NOTE: *The MCS window enables you to modify only the currently loaded methodology. Therefore, if you select a different methodology from the Select Methodology dialog, the goals displayed in the MCS window are not affected.*

Creating and Modifying a Methodology

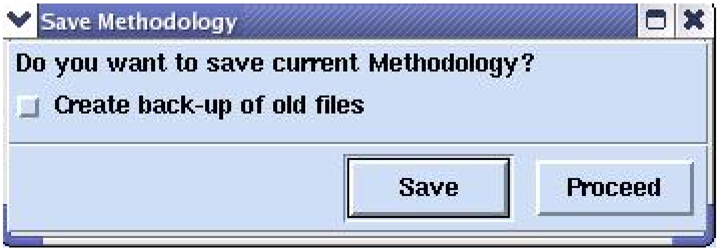
You can create or modify a methodology and specify details, such as goals, rules, and parameters for that methodology.

Creating a Methodology

To create a methodology, perform the following steps:

1. Select the *New Methodology* option from the *File* menu in the *Methodology Configuration System* window. Alternatively, use the <*Ctrl*> + <*N*> key combination from the keyboard.

   The *Save Methodology* dialog appears to enable you to save the currently loaded methodology as shown in the following figure:



Save Methodology

Do you want to save current Methodology?

□ Create back-up of old files

FIGURE 155. Save Methodology

2. In the above dialog, you can perform the following actions:
   - ❒ Click the *Save* button to save the current methodology.
   - ❒ Select the *Create back-up of old files* option if you want to create a backup of files of the currently loaded methodology. The backup files are saved in the `<Methodology-dir>`/Methodology_Backup/ directory.
   - ❒ Click the *Proceed* button if you do not want to save the currently loaded methodology.

   After performing the required actions in the *Save Methodology* dialog, the *New Methodology* dialog appears, as shown in the following figure:

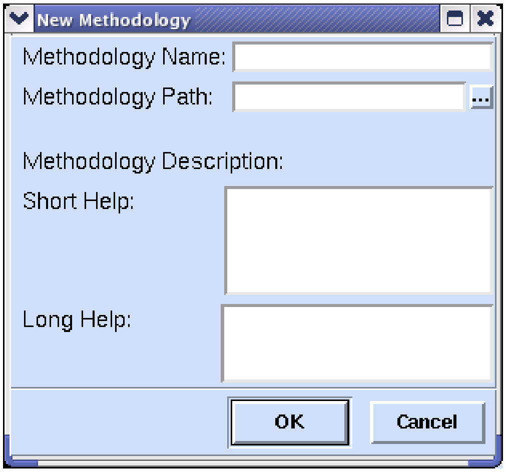


FIGURE 156.

3. Specify the required details in the appropriate fields of the *New Methodology* dialog.

   The following table describes the various fields of this dialog:

Field	Description
Methodology Name	Specifies the name of the new methodology
Methodology Path	Specifies the path where the new methodology should be saved.
Short Help	Specifies the short description of the new methodology being created. This description is displayed next to the methodology name in the *Goal Selection* section
Long Help	Specifies a detailed description of the new methodology being created. This description is saved in the order file.

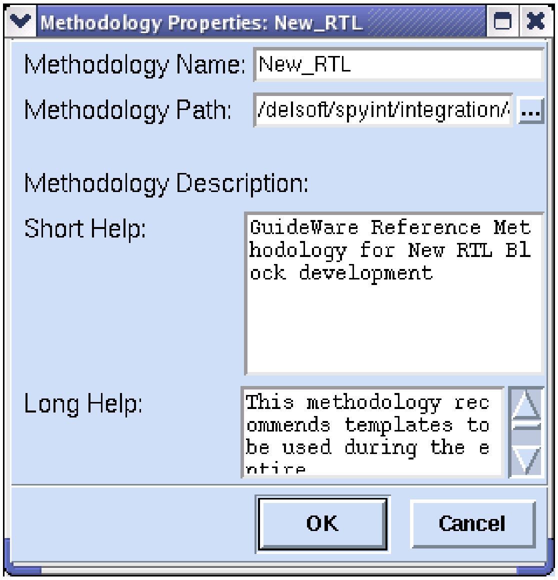
4. Click the *OK* button.

After performing the above steps, the newly created methodology appears in the *MCS* window.

This methodology does not contain any goal unless specified. You can either create new goals or import existing goals for that methodology. For details, refer to *Creating Goals* and *Importing Goals*.

Modifying a Methodology

To modify an existing methodology, select the *Methodology Properties* option from the *File* menu. This displays *Methodology Properties* dialog, as shown in the following figure:



Methodology Name: New_RTL

Methodology Path: /delsoft/spyint/integration/

Methodology Description:

Short Help: GuideWare Reference Methodology for New RTL Block development

Long Help: This methodology recommends templates to be used during the entire

FIGURE 157. Methodology Properties

In the above dialog, update the required details, such as methodology name, path, and description, in the appropriate fields, and click the *OK* button to save changes.

If you change the name of the methodology, the order file and goals move to a new directory of the specified methodology name and the same methodology directory.

If you change the directory where the methodology is located, SpyGlass creates a copy of the existing methodology in the new directory.

You can also customize a methodology by modifying goals contained in that methodology. For example, you can activate/deactivate, add, or delete a goal based on your requirements (see *Modifying Goals* for more details).

NOTE: *If you make any changes in a methodology in the MCS window, an asterisk (*) is* *appended to the methodology name in the title bar of the MCS window indicating that the selected methodology contains unsaved changes.*

Creating and Modifying a Sub-methodology

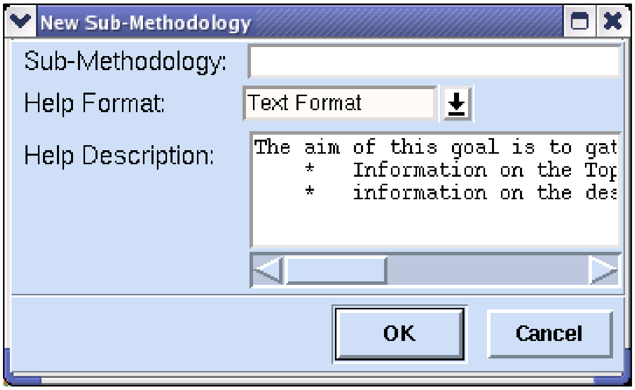
You can add a sub-methodology under an already existing methodology and later add goals to that sub-methodology.

Creating a Sub-Methodology

To add a sub-methodology, perform the following steps:

1. Right-click on the methodology for which you want to add a sub-methodology, and select the *Add Sub-Methodology* option from the shortcut menu.

   The *New Sub-Methodology* dialog appears, as shown in the following figure:



Help Format: Text Format

```
The aim of this goal is to gat
    *   Information on the Top
    *   information on the des
```

FIGURE 158. New Sub Methodology - Text Format

2. Specify the name of the new sub-methodology in the *Sub-Methodology* text box.
3. Select a help format (*Text Format* or *HTML Format*) from the *Help Format* drop-down list.
4. If you selected the *Text Format* option, type the help for the sub-methodology in the *Help Description* text field.

However, if you selected the *HTML Format* option, the *New Sub-Methodology* dialog changes, as shown in the following figure:

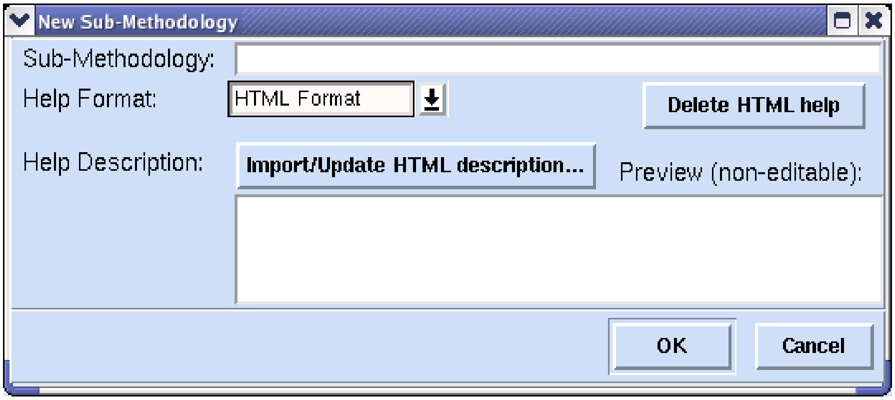


FIGURE 159. New Sub Methodology - HTML Format

In the above dialog, perform the following steps:

a. Click the *Import/Update HTML Description* button to add an HTML file that contains the help for the sub-methodology.

The *Open File* dialog appears.

b. In the *Open File* dialog, select the required HTML file.

c. Click the *Open* button in the *Open File* dialog.

This closes the *Open File* dialog, and contents of the selected HTML file appear in the *Help Description* text field.

NOTE: *If an HTML file containing images is present at C:\HTML\help.htm, save the images used in that file at C:\HTML\help_files\. The HTML file should refer to these files with the relative path (for example, <img src = "help_files/img1.jpg">).*

5. Click the *OK* button in the *Sub-Methodology* dialog.

After performing the above steps, the new sub-methodology appears in the tree structure of the *Goals* section in the *Methodology Configuration System* window and the goal selection window (see *Selecting a Goal*).

In addition, when you select that sub-methodology, help of that sub-methodology appears in the *Help* window.

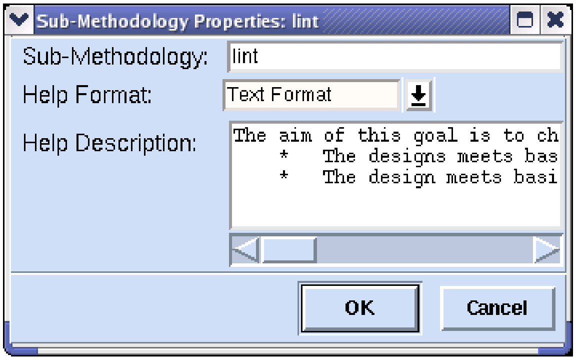
NOTE: *To delete a sub-methodology, right-click on the methodology and select the Delete Sub-Methodology option from the shortcut menu.*

Modifying a Sub-Methodology

To modify a sub-methodology, perform the following steps:

1. Right-click on the sub-methodology, and select the *Sub-Methodology Properties* option from the shortcut menu.

   This displays the *Sub-Methodology Properties* dialog, as shown in the following figure:



Sub-Methodology: lint

Help Format: Text Format

Help Description:

```
The aim of this goal is to ch
    *   The designs meets bas
    *   The design meets basi
```

FIGURE 160. Sub-Methodology Properties

2. Specify the required details in the appropriate fields of the above dialog.
3. Click the *OK* button,

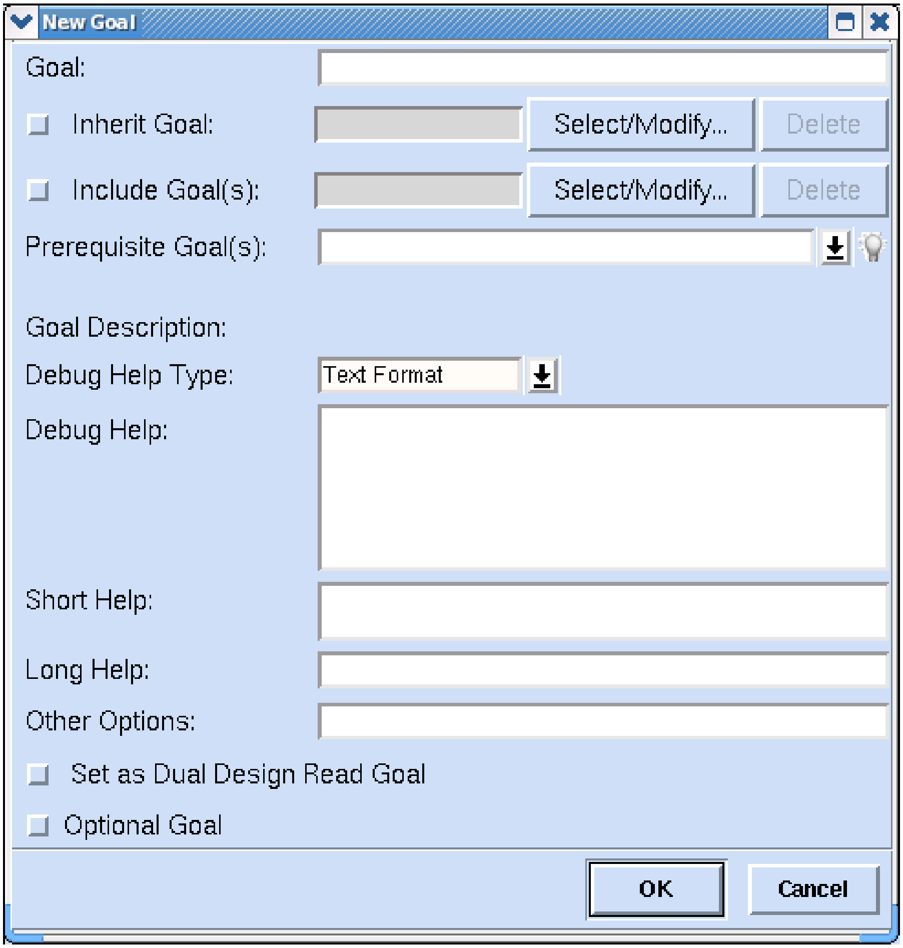
After performing the above steps, the specified changes appear in the *MCS* window. For example, if you have changed the help of a sub-methodology, the old help description is replaced with the new help description in the *Help* window.

Creating Goals

To create a goal for a methodology (or a sub-methodology), perform the following steps:

1. Open the *New Goal* dialog. For details, see *Displaying the New Goals Dialog*.

The following figure shows the *New Goal* dialog:



Debug Help Type: Text Format

FIGURE 161. New Goal

2. Specify the required details for the new goal in the appropriate fields of the *New Goal* dialog. For details, see *Specifying Details in the New Goal Dialog*.
3. Click the *OK* button.

After performing the above steps, the new goal appears under the selected methodology in the tree structure in the *Goals* section of the *Methodology Configuration System* window and the goal selection window.

After adding a goal for a methodology, you can add rules for that goal. See *Adding Rules in a Goal* for details.

Displaying the New Goals Dialog

To display the *New Goals* dialog, perform any of the following actions:

- Right-click on the methodology, and select the *Add New Goal* option from the shortcut menu.
- Select the *Add New Goal* option from the *Edit* menu.
- Click the *Add New Goal* option on the MCS window toolbar.
- Use the *<Ctrl>* + *<G>* key combination on the keyboard.

Specifying Details in the New Goal Dialog

The *New Goal* dialog contains various fields, as described in the following table:

Field	Description
Goal	(Mandatory) Specifies the name of the goal **NOTE:** The name of each goal should be unique.
Inherit Goal	Specifies a goal to be inherited in the new goal being created. For details on inherited goal, see *Including and Inheriting GuideWare Goals*. To inherit a goal, select the *Select/Modify* button. The *Inherit Goal* dialog appears in which you can select the required goal. Select or deselect the corresponding check box to enable or disable the inherited goal. To delete the inherited goal from the goal being created, click the *Delete* button.

Field	**Description**
Include Goal(s)	Specifies a goal to be included in the new goal being created. For details on included goal, see *Including and Inheriting GuideWare Goals*. To include a goal, select the *Select/Modify* button. The *Include Goal(s)* dialog appears in which you can select the required goals. Select or deselect the corresponding checkbox to enable or disable the included goals. To delete the included goals from the goal being created, click the *Delete* button.
Prerequisite Goals	Specifies the name of the prerequisite goal(s) for the currently selected goal. You can either enter goal names in this field, or select the goals that you want to consider as prerequisite goals for the new goal being created. To select goals, click the ⤓ button adjacent to the *Prerequisite Goals* field, and select the required goal(s) from this list. The prerequisite goals appear in the *Prereq Goals* column of the goal selection window (see *Selecting a Goal*).
Debug Help Type	Specifies the format (text or HTML) in which the goal debug help should be visible.
Debug Help	Specifies the debug information that helps you debug issues reported by this goal. This help is visible during the *Analyze Results* stage when you select the *Goal Debug Help* option in the *Help* section of the *Results* pane.
Short Help	Specifies the short description of the goal.
Long Help	Specifies the detailed description of the goal. The long help is visible in the *Help* window under the *Methodology Configuration System* and the *Select Goal* tab when modified methodology is loaded.
Other Options	Specifies additional command-line options.

In addition, this dialog contains the following options:

Option	Description
Set as Dual Design Read Goal	Select this option to make the goal as DDR-specific goal.
Optional Goal	Select this option to make the goal as optional.

Importing Goals

To import goals in a methodology, perform the following steps:

1. Open the *Import Goal(s)* dialog by performing any of the following actions in the *MCS* window:
   - ❒ Right-click on a methodology, and select the *Import Goal(s)* option from the shortcut menu.
   - ❒ Select the *Import Goal(s)* option from the *Edit* menu.
   - ❒ Click the *Import Goal(s)* link.

   The following figure shows the *Import Goal(s)* dialog:

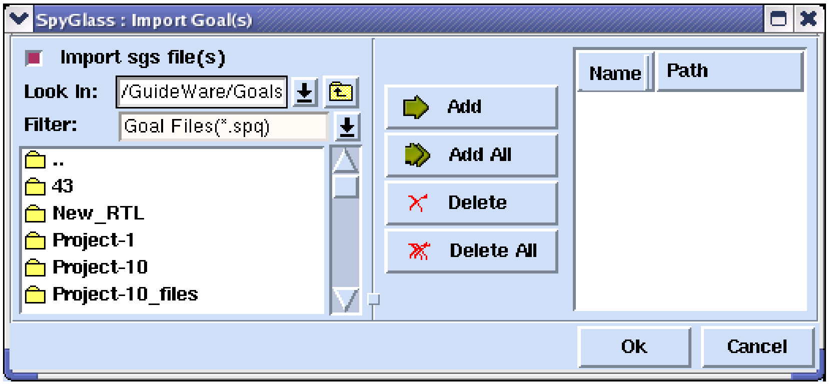


FIGURE 162. Import Goal(s)

2. In the *Look In* text field of the above dialog, specify the path of the directory where goals are present.
3. Select a goal or directory containing the required goals.

You can also select multiple goals and directories.

4. Click the *Add* button to add the selected goal or directory to the methodology.

   To add all goals present in the specified directory, click the *Add All* button. You can also import .sgs files along with the goal(s) by selecting the *Import sgs file(s)* option.

5. Click the *OK* button.

After performing the above steps, the specified goals appear in the selected methodology.

Deleting Goals

To delete a goal from a methodology, right-click on that goal and select the *Delete* option from the shortcut menu.

Copying Goals

You can copy a goal and paste it anywhere in the current methodology.

To copy a goal, right-click on that goal and select the *Copy Goal* option from the shortcut menu.

To paste this goal, right-click at the desired location in the current methodology and select the *Paste Goal* option from the shortcut menu.

Modifying Goals

You can modify a goal of a methodology by:

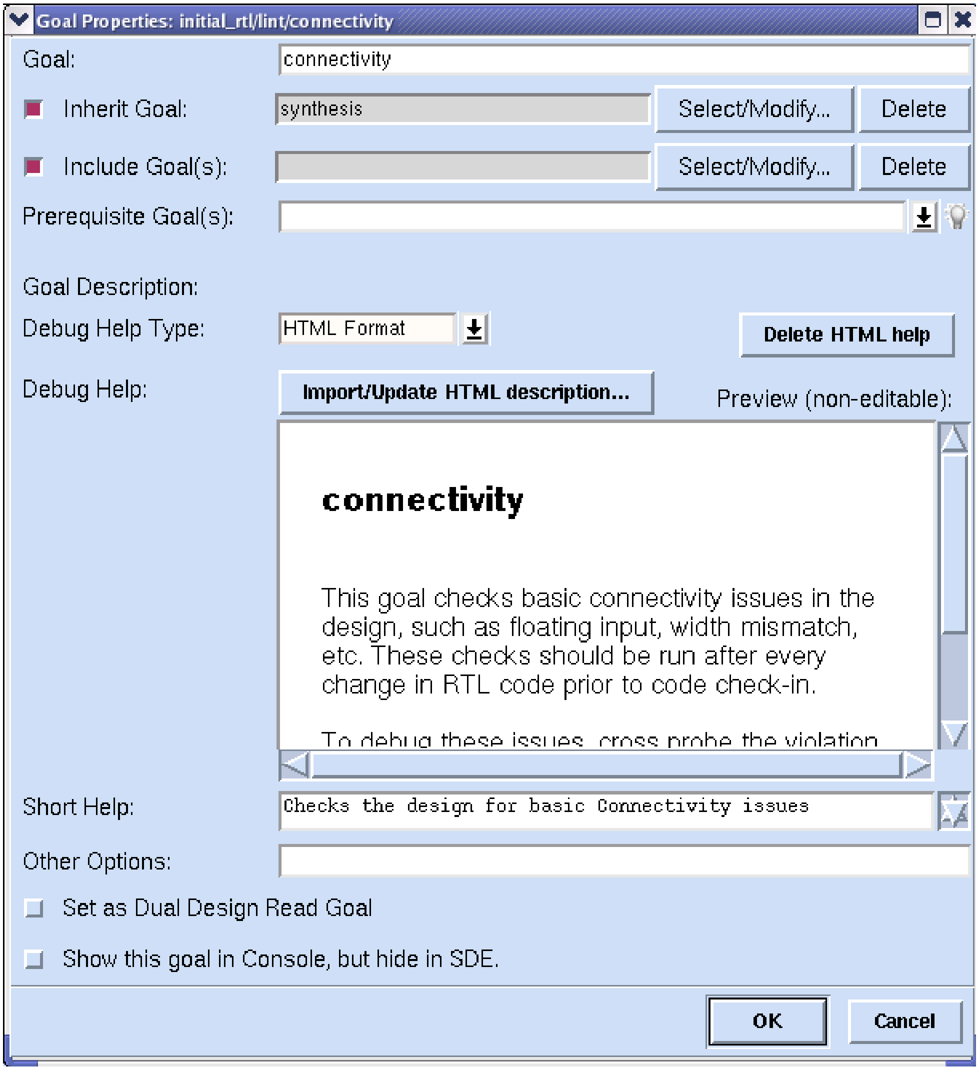
- *Modifying Goal Properties*
- *Enabling/Disabling a Goal*
- *Updating Rules of a Goal*
- *Adding Rules in a Goal*
- *Modifying Parameters of a Goal*

Modifying Goal Properties

To modify goal properties, perform the following steps:

1. Right-click on a goal name in the *MCS* window, and select the *Goal Properties* option from the shortcut menu.

   This displays the *Goal Properties* dialog, as shown in the following figure:



Goal Properties: initial_rtl/lint/connectivity

Goal: connectivity

■ Inherit Goal: synthesis

■ Include Goal(s):

Prerequisite Goal(s):

Goal Description:

Debug Help Type: HTML Format

Debug Help:

Preview (non-editable):

connectivity

This goal checks basic connectivity issues in the design, such as floating input, width mismatch, etc. These checks should be run after every change in RTL code prior to code check-in.

To debug these issues, cross probe the violation

Short Help: Checks the design for basic Connectivity issues

Other Options:

□ Set as Dual Design Read Goal

□ Show this goal in Console, but hide in SDE.

FIGURE 163. Modify Goal Properties

2. In the above dialog, specify the required details in appropriate fields.
3. Click the *OK* button.

After performing the above steps, the specified changes appear in the selected goal. For example, when you modify the description of a goal in the *Long Help* field, the modified help replaces the old help of that goal in the *Help* window.

Enabling/Disabling a Goal

An enabled goal appears in the *Goals* section of the *MCS* window. The name of such goals is preceded by the ✓ symbol.

If you do not want a goal to be part of your analysis run, disable that goal by clicking the ✓ symbol adjacent to that goal. The ✗ symbol appears preceding that goal name indicating that the goal is disabled. Such goals do not appear in the list of goals available in the goal selection window.

To enable a disabled goal, click the ✗ symbol adjacent to the goal name.

Updating Rules of a Goal

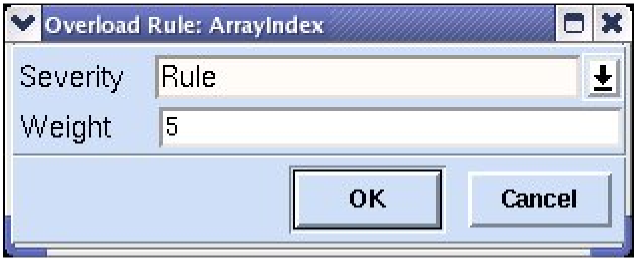
When you click on a goal in the *Goals* section of the *MCS* window, SpyGlass displays rules related to that goal in the *Rules List section* of the *MCS* window. From this rule list, you can select the required rule(s) and perform the required actions such as *Overloading a Rule*, *Enabling/Disabling a Rule*, *Deleting a Rule*, or *Ignoring a Rule*.

Overloading a Rule

To overload a rule displayed in the *Goals* section, perform the following steps:

1. Right-click on that rule and select the *Overload Rule* option from the shortcut menu.

This displays the *Overload Rule* dialog, as shown in the following figure:



Overload Rule: ArrayIndex

Severity: Rule

Weight: 5

FIGURE 164. Overload Rule

2. In the above dialog, specify the details, such as severity and weight in appropriate fields.
3. Click the *OK* button to save the changes.

After performing the above steps, the changes appear in goal files at the time of saving the methodology.

By default, the *Rules List* section displays rules that are recommended for an enabled goal. The rest of the rules are disabled. However, you can enable such rules based on your requirements.

Enabling/Disabling a Rule

You can enable or disable a rule in the same manner as you enable or disable a goal in the *Goals* section of the *MCS* window.

Alternatively, right-click on the rule and select the *Disable Rule* (if the rule is enabled) or *Enable Rule* (if the rule is disabled) option from the shortcut menu.

Deleting a Rule

To remove a rule from a goal, right-click on the rule and select the *Delete Rule* option from the shortcut menu. Alternatively, select the rule and click the *Delete Rule(s)* option in the *MCS* toolbar.

Ignoring a Rule

If you do not want a rule to run for a particular goal, you can ignore that rule.

To ignore a rule, right-click on the rule name, and select the *Ignore Rule(s)* option from the shortcut menu. You can also select multiple rules to be ignored. This menu option is disabled for a rule that is already ignored.

An ignored rule appears as `(ignored)`*`<rule-name>`* in the rule list of the selected goal.

If you want to run a rule that is ignored, right-click on the rule name and select the *Add Rule(s)* option from the shortcut menu. You can also select multiple rules. This menu option does not appear for a rule that is already added either by using the `-rule` or `-addrule` option in a goal file.

Adding Rules in a Goal

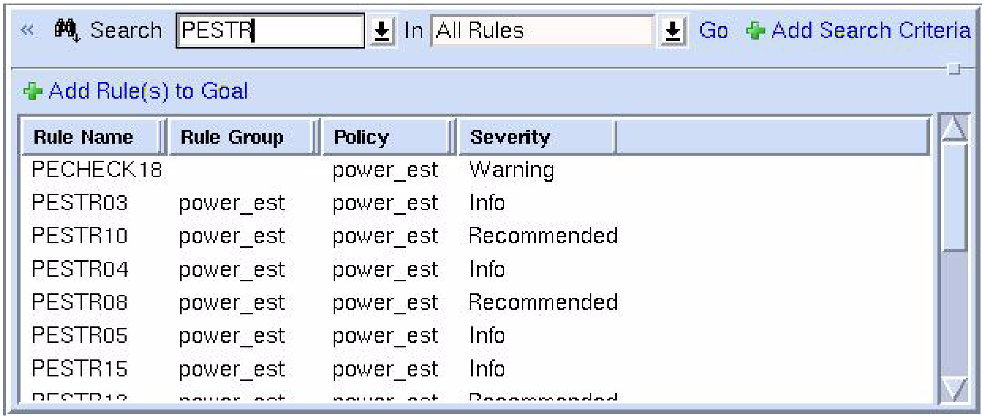
To add rules in a goal, perform the following steps:

1. Search the required rules that you want to add in a goal in the *Search* section of the *MCS* window. For details on searching rules, see *Searching Rules*.
2. Select the required rules to be added in a goal from the filtered rule list obtained after the search operation. You can select multiple rules by pressing the <*Ctrl*> key and clicking the required rules.
3. Right-click on the selected rules, and select the *Add Rule(s) to Goal* option from the shortcut menu. Alternatively, select the *Add Rule(s) to Goal* option in the *Search* section.

Once the above steps have been performed, SpyGlass displays the selected rules for that goal in the *Rule List* section of the *MCS* window.

Searching Rules

You can search rules in the *Search* section of the *MCS* window, as shown in the following figure:



Search: PESTR In: All Rules

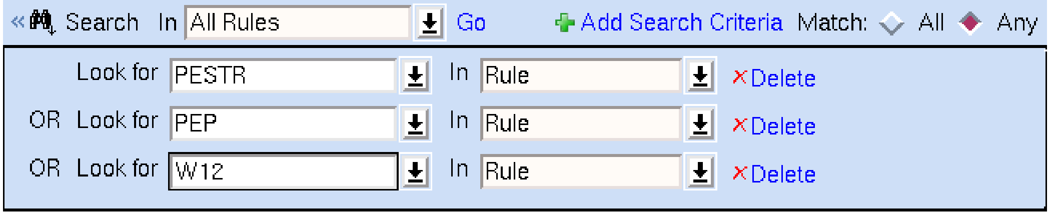
Rule Name	Rule Group	Policy	Severity
PECHECK18		power_est	Warning
PESTR03	power_est	power_est	Info
PESTR10	power_est	power_est	Recommended
PESTR04	power_est	power_est	Info
PESTR08	power_est	power_est	Recommended
PESTR05	power_est	power_est	Info
PESTR15	power_est	power_est	Info
PESTR13	power_est	power_est	Recommended

FIGURE 165. Search Rules

In the above section, specify the search text in the *Search* textbox and click the *Go* link. Once you click the *Go* link, SpyGlass displays all the rules matching the specified search criteria in a spreadsheet format.

By default, SpyGlass searches all SpyGlass rules. To confine your search among rules of the selected methodology only, select the *Current Methodology* option from the *In* drop-down list.

You can specify multiple search criteria by clicking the *Add Search Criteria* link multiple times. Each time you click this link, SpyGlass adds additional fields, as shown in the following figure:



In: All Rules

Look for: PESTR In: Rule

OR Look for: PEP In: Rule

OR Look for: W12 In: Rule

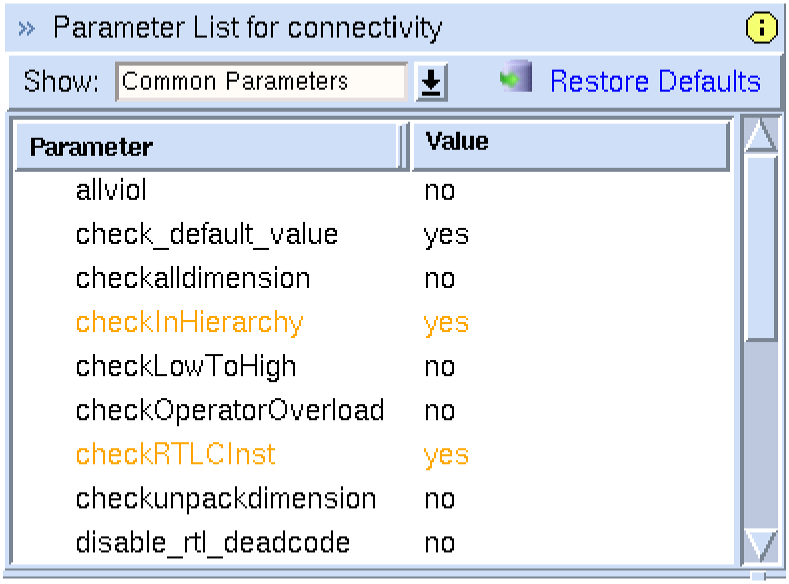
FIGURE 166. Search Fields

You can click the *Delete* link corresponding to the search criteria that you want to remove.

Modifying Parameters of a Goal

When you select a goal, parameters related to that goal appear in the *Parameter(s)* section of the *MCS* window.

The following figure shows the *Parameter(s)* section:



Parameter	Value
allviol	no
check_default_value	yes
checkalldimension	no
checkInHierarchy	yes
checkLowToHigh	no
checkOperatorOverload	no
checkRTLCInst	yes
checkunpackdimension	no
disable_rtl_deadcode	no

FIGURE 167. Parameters List

For each parameter, the corresponding value appears in the *Value* column. You can modify this value as per your requirement. If you want to assign all parameters their respective default values, click the *Restore Defaults* link.

NOTE: *Some goals do not use default parameter values. For details on such goals, see Goals That Do Not Use Default Parameter Value.*

The *Parameter(s)* section also contains the *Show* drop-down menu. The following table describes various options of this menu:

Option name	Purpose
Common Parameters	Displays commonly used parameters of the selected goal. This option is selected by default when you open the *MCS* window.
Other Parameters	Displays the parameters that are not commonly used for the selected goal
All Parameters	Displays all the parameters (common and un-common) of the selected goal
Rule Parameters	Displays the parameters of the selected rule

Dragging and Dropping Sub-Methodologies and Goals

You can drag and drop sub-methodologies and goals in the required hierarchy in the *MCS* window.

For example, consider the following *MCS* window:

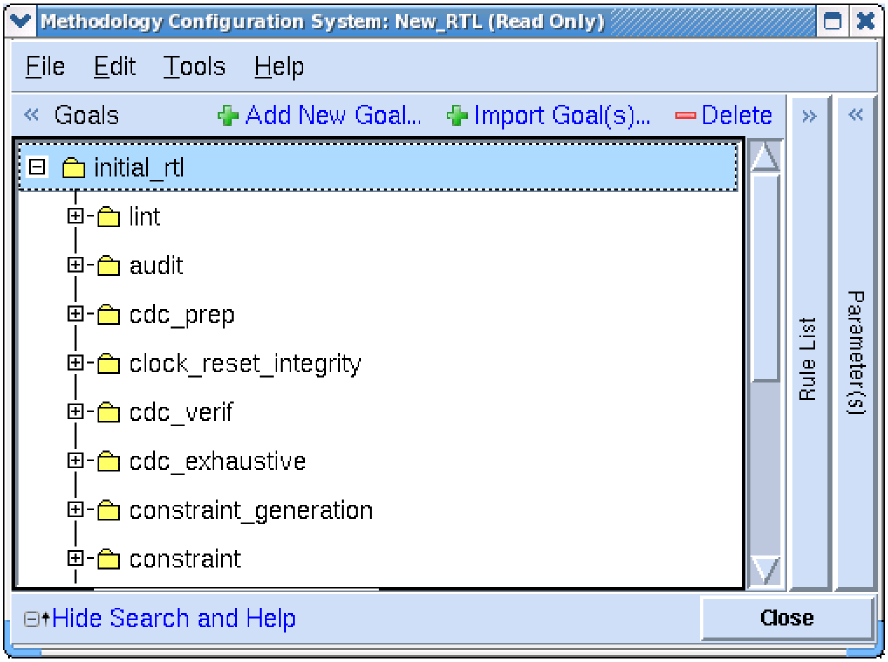


FIGURE 168. MCS Window

In the above window, if you want to move the *audit* sub-methodology inside the *lint* sub-methodology, perform the following steps:

1. Select the *audit* sub-methodology.
2. Drag the *audit* sub-methodology to the *lint* sub-methodology.

   When you release the mouse button, the following menu appears:

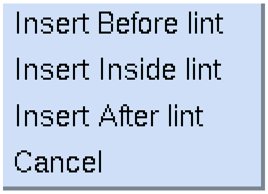


FIGURE 169. Drag and Drop Methodologies - Right-Click Menu

3. From the above menu, select the *Insert Inside lint* option.

   The *audit* sub-methodology now appears under the *lint* sub-methodology, as shown in the following figure:

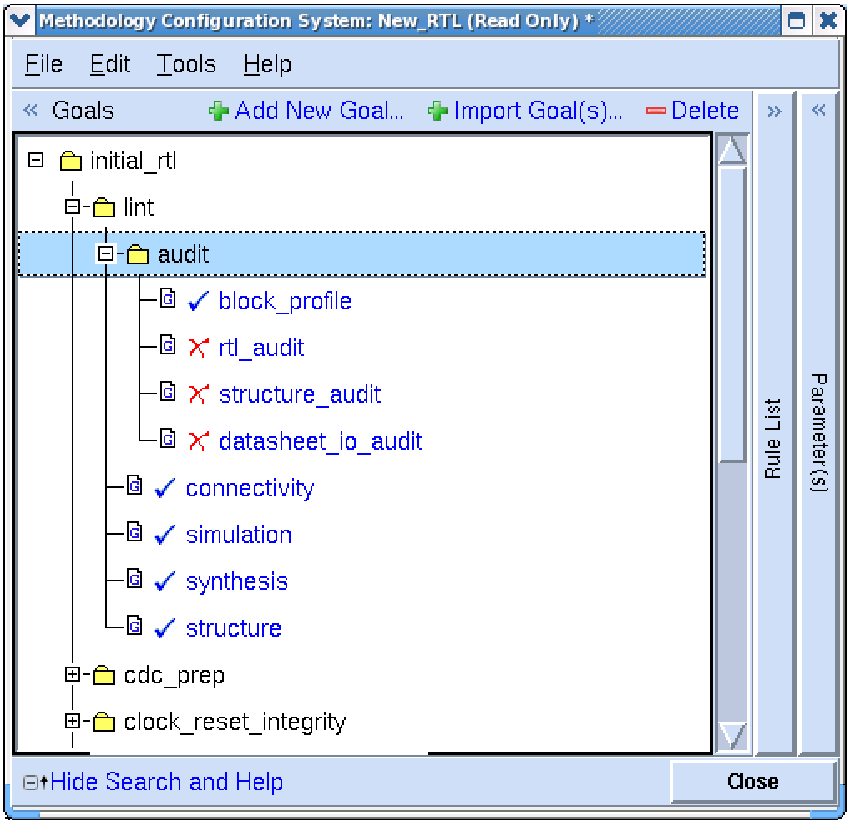


FIGURE 170. MCS Window - Methodology/Goal List

NOTE: *The sub-methodology being moved appears as the first folder under the tree of the destination sub-methodology.*

Similarly, you can drag and drop goals across different sub-methodologies or within the same sub-methodology.

For example, to move the *synthesis* goal from the *lint* sub-methodology to the *clock_reset_integrity* sub-methodology, perform the following steps:

1. Select the *synthesis* goal.
2. Drag the *synthesis* goal to the *clock_reset_integrity* sub-methodology.

   When you release the mouse button, the following menu appears:

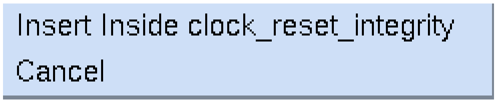


FIGURE 171. Drag and Drop Goals - Right-Click Menu

3. From the above menu, select the *Insert Inside clock_reset_integrity* option.

   The *synthesis* goal now appears as the first goal under the *clock_reset_integrity* sub-methodology, as shown in the following figure:

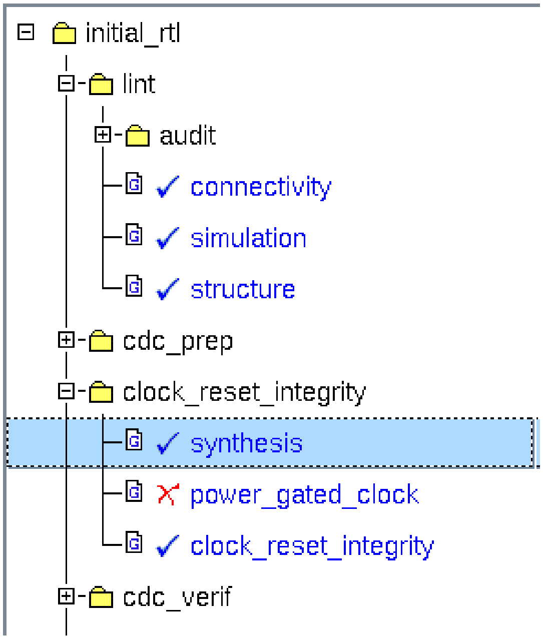


FIGURE 172. MCS Window - Goal List

You can also move goals at specific positions under a sub-methodology. For example, if you want to move the *simulation* goal of the *lint* sub-methodology after the *power_gated_clock* goal under the *clock_reset_integrity* sub-methodology, perform the following steps:

1. Expand the *clock_reset_integrity* sub-methodology.
2. Select the *simulation* goal.

3. Drag the *simulation* goal on the *power_gated_clock* goal.

   When you release the mouse button, the following menu appears:

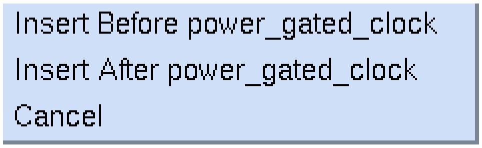

Insert Before power_gated_clock
Insert After power_gated_clock
Cancel

FIGURE 173. Right-Click Menu

4. From the above menu, select the *Insert After power_gated_clock* option.

After performing the above steps, the *simulation* goal appears after the *power_gated_clock* goal under the *clock_reset_integrity* sub-methodology.

Creating Custom Methodologies

Apart from using existing GuideWare methodologies, you can create your own custom methodologies that contain customized goals within goal files (.spq).

Customizing Goals

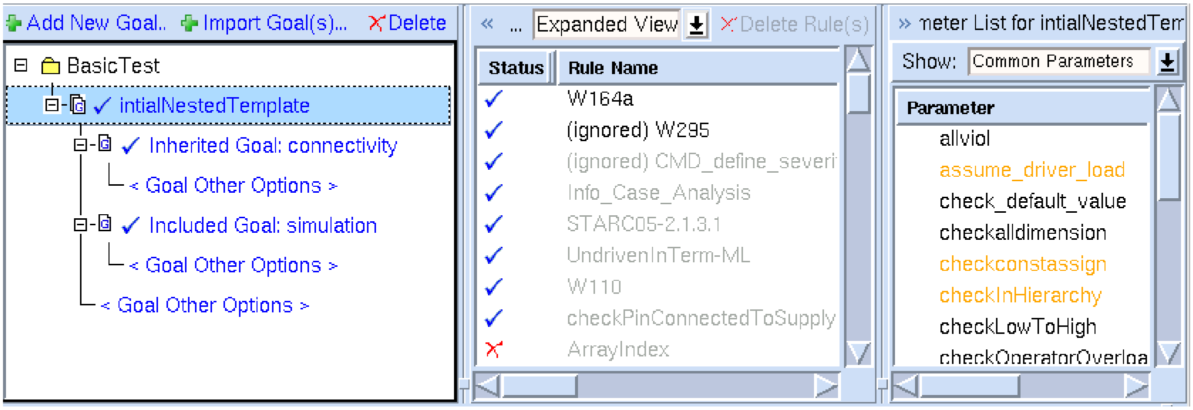
You can customize a goal in the following ways:

- By adding and/or removing rule(s) from a goal.
- By updating parameter value of rules.
- By defining your own rule severity by using the `define_severity` option.
- By deriving existing GuideWare goals in the goal file.

Including and Inheriting GuideWare Goals

You can derive existing GuideWare goals in another goal by including or inheriting a goal within another goal. You can include and/or inherit a goal in a goal file or by using the *MCS* window.

The included and inherited goals appear as separate nodes below the parent goal in the *MCS* window. The following figure shows a goal tree containing an included and inherited goal:



- BasicTest
  - ✓ intialNestedTemplate
    - ✓ Inherited Goal: connectivity
      - < Goal Other Options >
    - ✓ Included Goal: simulation
      - < Goal Other Options >
    - < Goal Other Options >

Status	Rule Name
✓	W164a
✓	(ignored) W295
✓	(ignored) CMD_define_severi
✓	Info_Case_Analysis
✓	STARC05-2.1.3.1
✓	UndrivenInTerm-ML
✓	W110
✓	checkPinConnectedToSupply
✗	ArrayIndex

Parameter
allviol
assume_driver_load
check_default_value
checkalldimension
checkconstassign
checkInHierarchy
checkLowToHigh
checkOperatorOverloa

FIGURE 174. Goal Tree

In the above figure, the `intialNestedTemplate` goal inherits the `connectivity` goal and includes the `simulation` goal.

Including/Inheriting Goals in a Goal File

Within a goal file, you can derive existing GuideWare goals in the following ways:

- *By including existing goal file in the parent goal file*

  Including a goal means copying all the options specified in included goal to the parent goal without inheriting its help (from.help files) or setup (from.sgs file).

  To include a goal in your parent goal, specify the following command in the goal file:

  ```
  -include_goal <goal-path>
  ```

- *By inheriting existing goal file in the parent goal file*

  Inheriting a goal means copying all the options specified in the inherited goal to the parent goal as well as inheriting its help (from.help files) and setup (from.sgs file). If the parent goal has its own help and setup created, SpyGlass ignores the help/setup from the inherited goal.

  To inherit a goal in your parent goal, specify the following command in the goal file:

```
-inherit_goal <goal-path>
```

The *<goal-path>* argument refers to the path of the goal file to be included or inherited in the parent goal. You can refer to `$SPYGLASS_HOME` to specify this path.

If you specify a relative path, SpyGlass resolves that path with respect to the current working directory. You can also refer to any other environment variable, if defined, while specifying the goal path. If that environment variable is not found, SpyGlass reports the appropriate error message.

SpyGlass reads the included or inherited goals in the same order as they are specified in parent goals.

Using Environment Variables in Included/Inherited Paths

Setting absolute paths for included or inherited goals works fine but has certain limitations, as discussed below:

- A methodology can be used by another user only if the network path location, as seen by that user, is the same as what you have.
- Moving a methodology to another directory location renders it unusable until you correct the absolute paths.

For these reasons, it is recommended that you use environment variables while specifying paths. The following examples explain this in detail.

Example 1

Consider a corporate-level methodology `M1` and you want to create a local customized methodology `M2` by including/inheriting one or more goals from `M1` and share with other users.

In this case, it is recommended that you use a standard environment variable in paths specified in the `M2` methodology. For example, following is the sample line in one of the `M2` methodology goal inheriting the `a/b/c/goal1` goal from corporate-level methodology `M1`:

```
#Inside M2 methodology, goal 'x/y/z/goal11'
-inherit_goal $MY_CORPORATE_METHODOLOGY_DIR/a/b/c/
goal1
```

To use the `M2` methodology, set the

MY_CORPORATE_METHODOLOGY_DIR environment variable to point to the M1 directory before invoking SpyGlass, as shown below:

```
setenv MY_CORPORATE_METHODOLOGY_DIR <M1 path>
```

Example 2

Consider that you have a methodology M1 and within this methodology, you want to create additional goals by referring one or more goals of the same methodology.

In this case, it is recommended that you use SpyGlass internally defined variable METHODOLOGY_HOME in a hierarchical goal. For example, following is the sample line in one of the new goal referring another goal from the same methodology:

```
-inherit_goal $METHODOLOGY_HOME/a/b/c/goal1
```

The METHODOLOGY_HOME variable is set by SpyGlass automatically when a methodology loaded. Do not define this variable. If it is set, SpyGlass ignores it.

Please note that SpyGlass also provides the SPYGLASS_HOME environment variable. Using this variable in inherited/included goal paths implies that the goals are picked from the actual release that is run. As a goal setup is subject to change across releases, it may lead to different sets of messages and possibly other issues. If you want to maintain the goal setup, you can define your own environment variable (as explained in *Example 1*) to point to a specific release location or a copy of a methodology from a specific release.

NOTE: *Do not edit a methodology created with environment variables using the MCS window as paths expand to absolute paths while saving. This issue will be fixed in a future release.*

Specifications Provided in the Included/Inherited Goal

SpyGlass performs different actions based on the specifications provided in the included/inherited goal file.

The following are the various specifications that you can provide in an included/inherited file:

- *-rule/-addrule/-ignorerule(s) Specification*

- *Parameter Specification*
- *define_severity Specification*
- *overloadrule Specification*

-rule/-addrule/-ignorerule(s) Specification

SpyGlass ignores the `-ignorerule` specification for a particular rule in the included/inherited goal file if you specify the `-rule/-addrule` specification for the same rule in the parent, included, and/or inherited goal file. In addition, SpyGlass reports an appropriate warning message in such cases.

Consider a parent goal file that contains the following specifications:

```
-policies=lint
...
-include_goal included-mixed.spq
-addrule W18
...
```

In addition, consider the `included-mixed.spq` goal file (given in the parent goal) that contains the following specifications:

```
-policies=clock-reset,lint
-addrule W391
-ignorerule W18
-ignorerule W391
....
```

Now when the parent goal runs, SpyGlass ignores the W18 and W391 rules and reports the following warning for these rules:

```
WARNING [342] Rule/Group 'W18' specified at File: parent-
mixed.spq, Line: 6 has been ignored due to the following -
ignorerule(s) specifications -

-ignorerule W18(File: included-mixed.spq, Line: 6)
```

Parameter Specification

If you specify a rule parameter more than once in an included/inherited goal or the parent goal, SpyGlass considers the last parameter specification.

Consider a parent goal file that contains the following specifications:

```
...
-use_inferred_clocks=no
-include_goal included-mixed.spq
...
```

In addition, consider the `included-mixed.spq` included goal file (given in parent goal) that contains the following specification:

```
...
-use_inferred_clocks=yes
...
```

In this example, SpyGlass considers the `-use_inferred_clocks=yes` specification and reports the following warning:

```
WARNING [341] Parameter 'use_inferred_clocks' specified
multiple times at following locations -

-use_inferred_clocks=no (File: parent-mixed.spq, Line: 22)

-use_inferred_clocks=yes (File: included-mixed.spq,Line: 6)

All specifications except the last would be ignored.
```

define_severity Specification

If you specify the `define_severity` specification for a rule in the parent, included, and/or inherited goal more than once, SpyGlass decides the `define_severity` specification to be considered in the following manner:

- The first `define_severity` specification, if present in the parent goal, is considered.
- Otherwise, the first `define_severity` specification present in the included/inherited goals is considered.

Consider a parent goal file that contains the following specifications:

```
...
-include_goal included-mixed.spq
...
-define_severity Audits+Audit3run+Warning
...
```

In addition, consider the `included-mixed.spq` included goal file (given in the parent goal) that contains the following specifications:

```
...
-define_severity Audits+Audit3run+ERROR
...
```

In this example, SpyGlass reports the following warnings:

```
WARNING [345]    Severity for 'Audit3run' defined multiple
times for policy 'Audits' in included/inherited goal and parent
goal ( parent-mixed.spq ) at following places -
ERROR (File: included-mixed.spq,Line: 5)
Warning (File: parent-mixed.spq,Line: 50)
First definition in parent goal (severity class 'Warning')
would be honored and rest would be ignored.
```

Consider another example in which no `define_severity` specification is present in the parent goal file, but the following `define_severity` specifications are present in the included goal file:

```
...
-define_severity Audits+Audit_Info+Warning
...
-define_severity Audits+Audit_Info+INFO
...
```

In this example, SpyGlass reports the following warning:

```
WARNING [346]    Severity for 'Audit_Info' defined multiple
times for policy 'Audits' in included/inherited goal (inside
parent goal parent-mixed.spq ) at following places -
Warning (File: included-mixed.spq,Line: 3)
INFO (File: included-mixed.spq,Line: 6)
All except the very first specification would be ignored.
```

overloadrule Specification

If multiple `overloadrule` specifications are present for a particular rule in the goal file, SpyGlass overrides and merges the specified overload specifications with subsequent specifications.

If you specify different severity labels in these specifications, SpyGlass considers the last severity label and reports a warning message.

Consider a parent goal file that contains the following specifications:

```
...
-overloadrules=Ac_sanity05+severity=Warning
-overloadrules=W226+severity=Info+verilog
...
```

Also, consider the `included-mixed.spq` included goal file (given in the parent goal) that contains the following specifications:

```
...
-overloadrules=Ac_sanity05+severity=Error+verilog+vhdl
-overloadrules=W226+severity=Error+vhdl
...
```

The *W226* rule of the SpyGlass *lint* solution is registered in both Verilog and VHDL. Therefore, in the above example, SpyGlass applies severity label for both Verilog and VHDL versions of the W226 rule and does not report any warning. SpyGlass, however, reports the following warning for the Ac_sanity05 rule:

```
WARNING [347]    Multiple severity overload specifications
found for rule 'Ac_sanity05' (registered in language
'Verilog+VHDL') in included/inherited goal and parent goal
(parent-mixed.spq) at following places -

Warning (Language: Undefined) (File: parent-mixed.spq, Line:4)

Error (Language: Verilog+VHDL) (File: included-mixed.spq,
Line:7)

Only last severity class would be used.
```

Checks Performed on the Goal File

SpyGlass performs various checks on the goal file, and reports errors in the following cases:

- If the language of the inherited or included goal is not same as the current language mode.
- If the `include_goal` and/or `inherit_goal` command is encountered within an already included or inherited goal.
- If the parent goal inherits more than one goal. In such cases, SpyGlass considers the first inherited goal, and ignores the rest of the inherited goals.

- If recursive/duplicate include/inherit goal specifications are present in the same parent goal.

NOTE: *To suppress warning messages reported on* `include_goal/inherit_goal` *goal specification inside parent goal, specify the* `-suppress_nested_template_msgs` *option in the parent goal file.*

Including/Inheriting Goals in the MCS Window

To include or inherit a goal in the *MCS* window, perform the following steps:

1. Right-click on a goal.
2. Select the *Inherit Goal* or *Include Goal(s)* option from the shortcut menu.

   The *Inherit Goal* or *Include Goal(s)* dialog appears depending upon the selection.
3. Select the goal to be inherited. If you have selected the *Include Goal(s)* option, you can select multiple goals.
4. Click the *Add* button.
5. Click the *OK* button.

The inherited or included goal appears under the selected goal in the *MCS* window.

The following figure shows an example of the *synthesis* goal inherited by the *connectivity* goal:

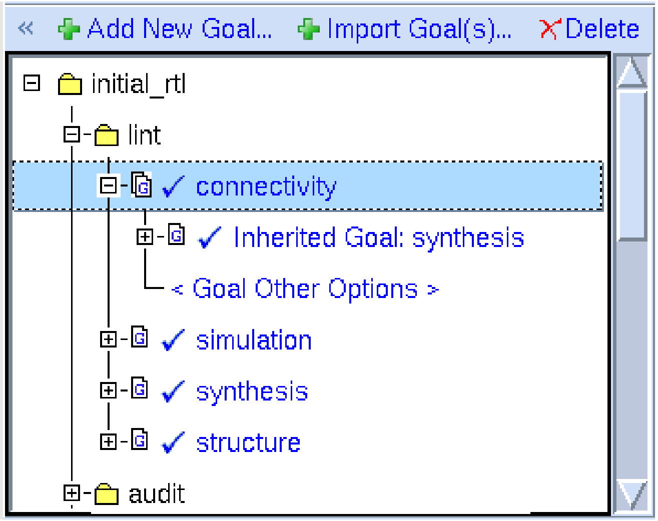
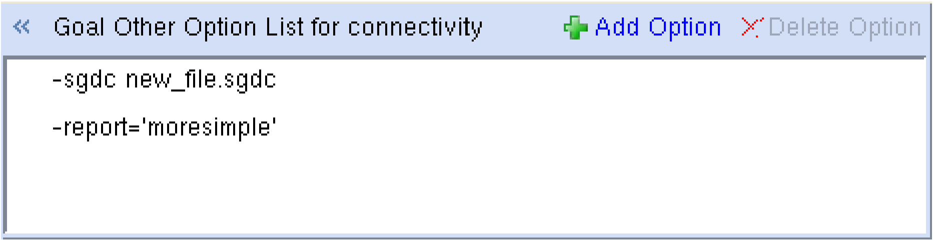


FIGURE 175. Goal Inheritance

Viewing and Adding Options for an Included or Inherited Goal

Within the hierarchy of each included or inherited goal, a separate node, *Goal Other Options*, is present. Select this node to view various options set for a goal. Such options do not appear in the *Rule List* or *Parameter List* for that goal.

When you select the *Goal Other Options* node, the *Goal Other Options List* section appears in place of the *Rule List* section to display various options set for a goal. The following figure shows a list of options for a goal:



```
-sgdc new_file.sgdc
-report='moresimple'
```

FIGURE 176. Goal Other Option List for Connectivity

In the above page, you can add more options by clicking *Add Option*.

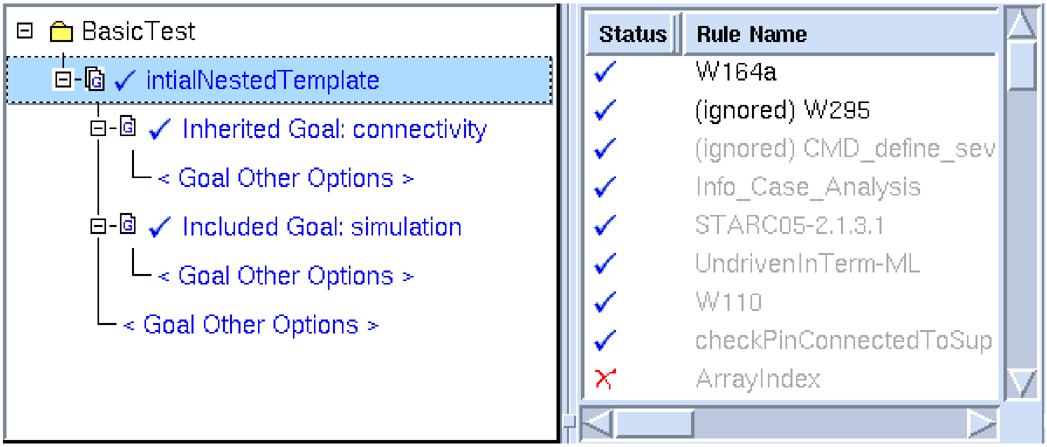
Viewing Rules and Parameters of Included/Inherited Goals

Click on an included or inherited goal to load its rules and parameters into respective rules and parameters section of the *MCS* window.

When you click on a parent goal that contains included and inherited goals, the rule list appearing in the rules section is a consolidated list of all rules of the included, inherited, and parent goal. You can view this list in any of the following formats:

- *Expanded View*: This is a flat list of all rules of a nested goal and its included and inherited goal. Select the *Expanded View* option from the drop-down list for this view.

  The following figure shows an expanded view of rules:



BasicTest
✓ intialNestedTemplate
✓ Inherited Goal: connectivity
< Goal Other Options >
✓ Included Goal: simulation
< Goal Other Options >
< Goal Other Options >

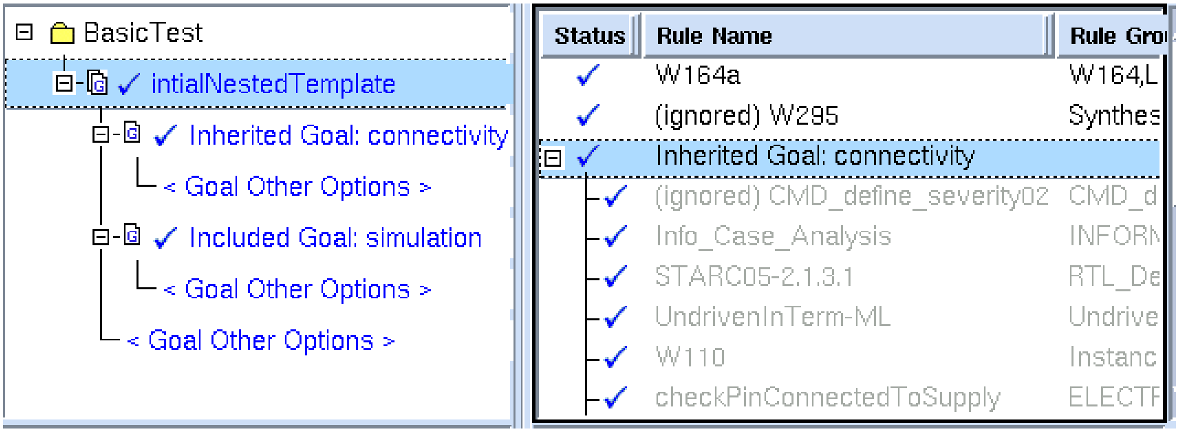
Status	Rule Name
✓	W164a
✓	(ignored) W295
✓	(ignored) CMD_define_sev
✓	Info_Case_Analysis
✓	STARC05-2.1.3.1
✓	UndrivenInTerm-ML
✓	W110
✓	checkPinConnectedToSup
✗	ArrayIndex

FIGURE 177. Expanded View

- *Hierarchical View*: This is a hierarchical list that shows rules for included and inherited goals under separate nodes. Rules of the parent node appear at the root-level.

  Select the *Hierarchical View* option from the drop-down list for this view.

  The following figure shows a hierarchical view of rules:



Status	Rule Name	Rule Grou
✓	W164a	W164,L
✓	(ignored) W295	Synthes
✓	Inherited Goal: connectivity	
✓	(ignored) CMD_define_severity02	CMD_d
✓	Info_Case_Analysis	INFORN
✓	STARC05-2.1.3.1	RTL_De
✓	UndrivenInTerm-ML	Undrive
✓	W110	Instanc
✓	checkPinConnectedToSupply	ELECTF

FIGURE 178. Hierarchical View

Enabling/Disabling Rules of a Parent Goal

To disable a rule of a parent goal, click the ✓ symbol preceding the rule name. This disables that rule and the ✗ symbol appears preceding the rule name.

Similarly, to enable a rule of a parent goal, click the ✗ symbol preceding the rule name. This enables that rule and the ✓ symbol appears preceding the rule name.

You cannot enable/disable rules of an included or inherited goal as these goals cannot be modified from a parent goal.

Selecting a Custom Methodology

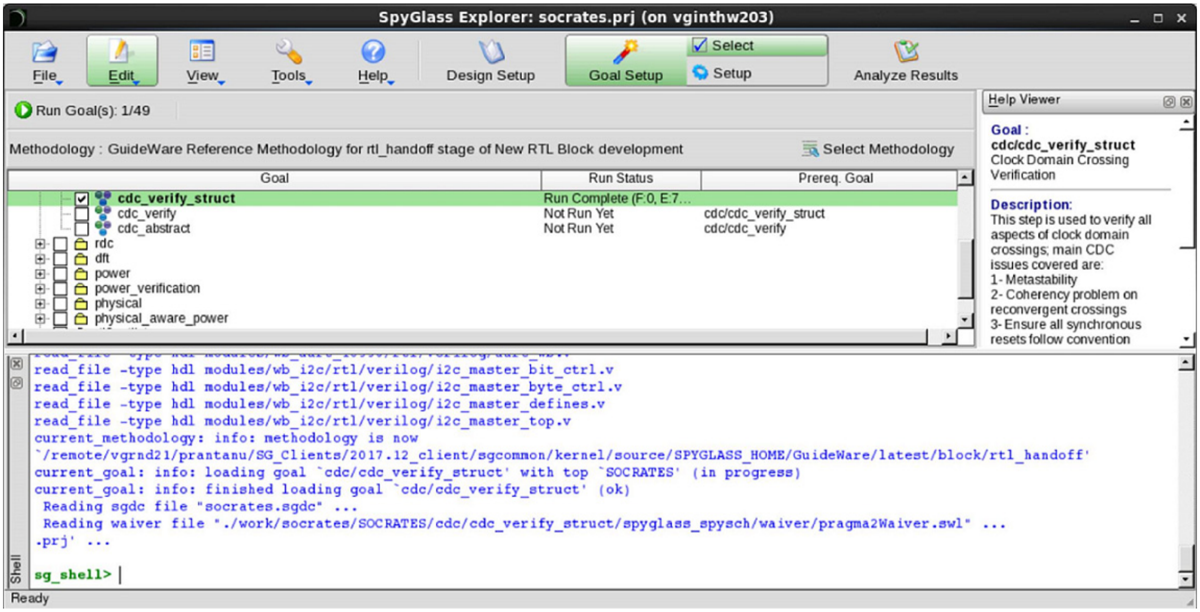
To use goals of a custom methodology, perform the following steps:

1. Click the *Select Methodology* link under the *Select Goal* tab.

   The *Select Methodology* dialog appears.
2. In the *Select Methodology* dialog, select the *Custom* option.
3. Select the *Browse* button to select the required custom methodology.

   The *Select Directory* dialog appears.
4. In the *Select Directory* dialog, select the required directory of your custom methodology.
5. Click the *OK* button to close the *Select Directory* dialog.
6. Click the *OK* button to close the *Select Methodology* dialog.

After performing the above steps, SpyGlass displays goals of the specified custom methodology under the *Select Goal* tab, as shown in the following figure:



Methodology : GuideWare Reference Methodology for rtl_handoff stage of New RTL Block development

Goal	Run Status	Prereq. Goal
☑ **cdc_verify_struct**	Run Complete (F:0, E:7…	
□ cdc_verify	Not Run Yet	cdc/cdc_verify_struct
□ cdc_abstract	Not Run Yet	cdc/cdc_verify
□ rdc		
□ dft		
□ power		
□ power_verification		
□ physical		
□ physical_aware_power		

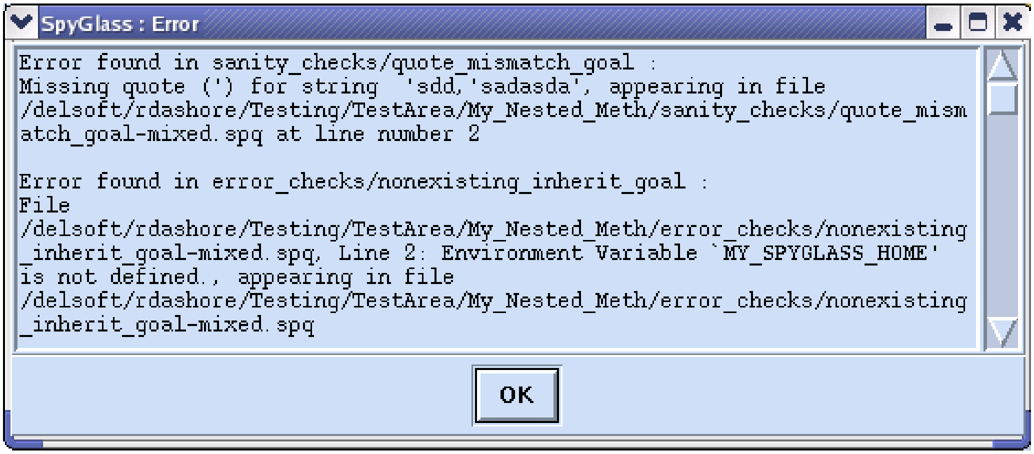
Goal :
cdc/cdc_verify_struct
Clock Domain Crossing Verification

Description:
This step is used to verify all aspects of clock domain crossings; main CDC issues covered are:
1- Metastability
2- Coherency problem on reconvergent crossings
3- Ensure all synchronous resets follow convention

```
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_bit_ctrl.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_byte_ctrl.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_defines.v
read_file -type hdl modules/wb_i2c/rtl/verilog/i2c_master_top.v
current_methodology: info: methodology is now
`/remote/vgrnd21/prantanu/SG_Clients/2017.12_client/sgcommon/kernel/source/SPYGLASS_HOME/GuideWare/latest/block/rtl_handoff'
current_goal: info: loading goal `cdc/cdc_verify_struct' with top `SOCRATES' (in progress)
current_goal: info: finished loading goal `cdc/cdc_verify_struct' (ok)
 Reading sgdc file "socrates.sgdc" ...
 Reading waiver file "./work/socrates/SOCRATES/cdc/cdc_verify_struct/spyglass_spysch/waiver/pragma2Waiver.swl" ...
.prj' ...

sg_shell>
```

FIGURE 179. Goal List for Custom Methodology

In the above figure, goals highlighted in red indicate some error(s) in such goals. When you open the *Methodology Configuration System* window, SpyGlass first displays the error details of all such goals in the *Error* dialog, as shown in the following figure:



```
Error found in sanity_checks/quote_mismatch_goal :
Missing quote (') for string  'sdd,'sadasda', appearing in file
/delsoft/rdashore/Testing/TestArea/My_Nested_Meth/sanity_checks/quote_mism
atch_goal-mixed.spq at line number 2

Error found in error_checks/nonexisting_inherit_goal :
File
/delsoft/rdashore/Testing/TestArea/My_Nested_Meth/error_checks/nonexisting
_inherit_goal-mixed.spq, Line 2: Environment Variable `MY_SPYGLASS_HOME'
is not defined., appearing in file
/delsoft/rdashore/Testing/TestArea/My_Nested_Meth/error_checks/nonexisting
_inherit_goal-mixed.spq
```

FIGURE 180. Error Dialog for Goals

After viewing the error details in the above dialog, click the *OK* button to display the *Methodology Configuration System* window.

In the *Methodology Configuration System* window, when you select a goal containing error(s), an *Error* dialog appears displaying the error details of only the selected goal.

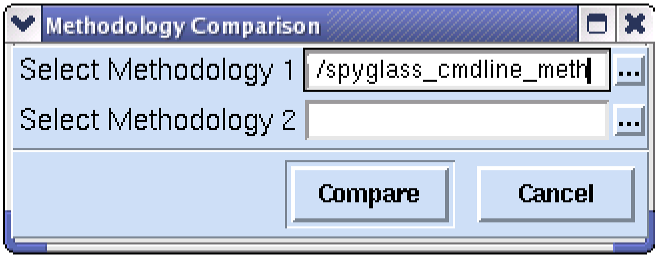
Comparing Methodologies

You can compare two methodologies in the *MCS* window to view differences between them.

To compare methodologies, perform the following steps:

1. Select the *Tools -> Compare -> Methodologies* menu option in the *MCS* window.

   The *Methodology Comparison* dialog appears, as shown in the following figure:



Methodology Comparison

Select Methodology 1 /spyglass_cmdline_meth

Select Methodology 2

FIGURE 181. Specify Methodologies for Comparison

2. In the above dialog, specify the required methodologies to be compared in the *Select Methodology1* and *Select Methodology2* fields.

   By default, the main methodology of the current session appears in the *Select Methodology1* field.

3. Click the *Compare* button.

   The *Methodology Comparison* dialog appears showing differences between the two specified methodologies.

The following figure shows an example of methodology comparison in the *Methodology Comparison* dialog:

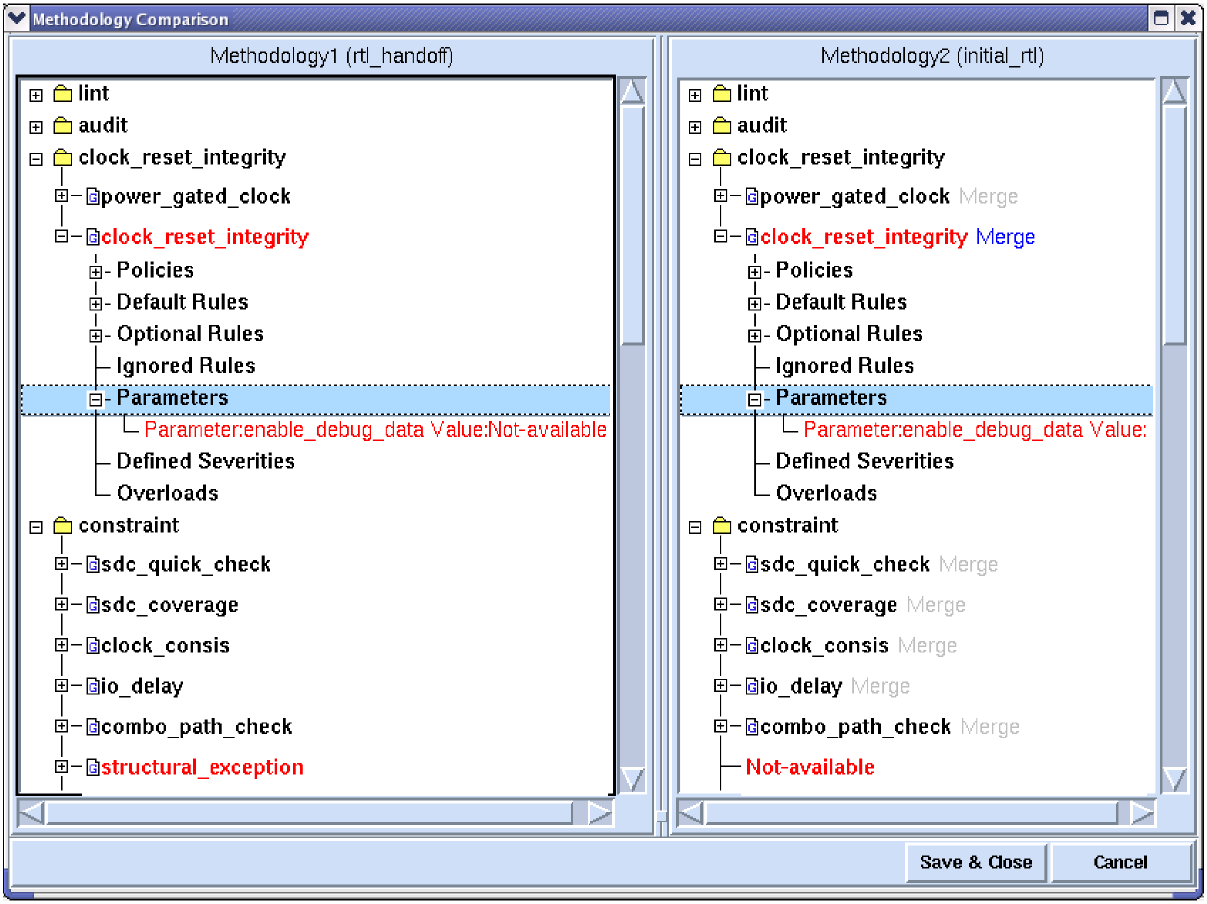


FIGURE 182. Methodology Comparison Results

Red entries in this dialog indicate a difference. For example, in the above dialog, there is a difference between the *clock_reset_integrity* goals of the two methodologies. You can expand these goals to view the difference.

If some information present in one methodology is missing in another methodology, the *Not available* text appears in the latter methodology. For example, in the above dialog, the *structural_exception* goal under the *constraint* hierarchy is present in *Methodology 1* but is missing in the constraint hierarchy of *Methodology 2*. Therefore, the text *Not available* text appears in *Methodology 2*.

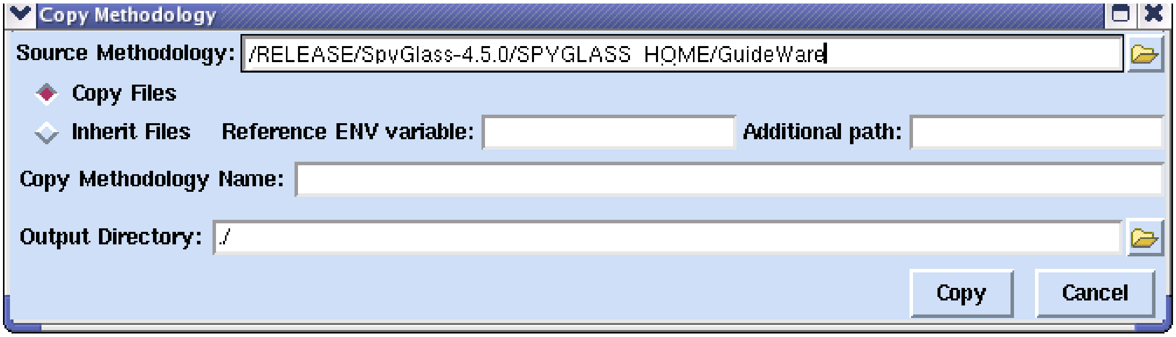
Merging the Differences

Click the *Merge* link in the right-most section to merge the corresponding data in the main methodology of the current session.

For example, in the above figure, click *Merge* adjacent to the *clock_reset_integrity* goal in the right-most section to overwrite the corresponding settings from the right-most methodology in the left-most methodology.

Copying and Inheriting Methodologies

To copy or inherit a methodology, select the *Tools -> Copy Methodology* menu option in the *MCS* window. This displays the *Copy Methodology* dialog, as shown in the following figure:



Source Methodology: /RELEASE/SpyGlass-4.5.0/SPYGLASS_HOME/GuideWare

● Copy Files

○ Inherit Files

Copy Methodology Name:

Output Directory: ./

FIGURE 183. Copy Methodology

In the above dialog, select the *Copy Files* or *Inherit Files* option to copy or inherit a methodology, respectively. For details, see *Copying a Methodology* and *Inheriting a Methodology*.

Copying a Methodology

Copying a methodology creates an exact copy of the specified methodology in the specified output directory. All the copied goal files contain details of rules and parameters.

To copy a methodology by using the *Copy Methodology* dialog, perform the following steps:

1. Specify a methodology to be copied in the *Source Methodology* text box.

   Alternatively, click 📂 to browse to the methodology to be copied.
2. Select the *Copy Files* option.
3. Specify the name of the methodology that should contain the copied files in the *Copy Methodology Name* text box.

4. Specify the directory in which the methodology should be copied in the *Output Directory* text box.
5. Click the *Copy* button.

After performing the above steps, all files of the specified methodology are copied in the specified output directory.

Inheriting a Methodology

Inheriting a methodology creates an exact structure of the specified methodology in the specified output directory.

However, unlike copying a methodology, goal files in this case do not contain details of rules and parameters. Instead, the goal files only contain the `-inherit_goal` command, as shown in the following example:

```
-inherit_goal $SPYGLASS_HOME/GuideWare/New_RTL/initial_rtl/
lint/structure-verilog.spq
```

To inherit a methodology by using the *Copy Methodology* dialog, perform the following steps:

1. Specify a methodology to be inherited in the *Source Methodology* text box.

   Alternatively, click 📂 to browse to the methodology to be inherited.
2. Select the *Inherit Files* option.
3. Specify a reference environment variable in the *Reference ENV Variable* text box.

   For details, see *Specifying a Reference Environment Variable*.
4. Specify an additional path after the reference environment variable path in the *Additional Path* text box.

   For details, see *Specifying an Additional Path*.
5. Specify the name of the methodology that should contain the inherited files in the *Copy Methodology Name* text box.
6. Specify the directory in which the methodology should be inherited in the *Output Directory* text box.
7. Click the *Copy* button.

After performing the above steps, all files of the specified methodology are inherited in the specified output directory.

Specifying a Reference Environment Variable

A reference environment variable is a variable that is used to set a reference point after which the path of the specified methodology exists.

For example, you may specify the reference environment variable as `SPYGLASS_HOME` that is internally set to the following path:

```
RELEASE/SpyGlass-<version>/SPYGLASS_HOME/
```

When the tool encounters this reference environment variable, it automatically expands it to the path this variable is internally set. For example, consider the following `inherit_goal` specification of an inherited goal file:

```
-inherit_goal $SPYGLASS_HOME//ip_audit/lint/ip_netlist-
mixed.spq
```

In the above example, the `$SPYGLASS_HOME` reference environment variable internally expands to its complete path, as shown below:

```
-inherit_goal RELEASE/SpyGlass-<version>/SPYGLASS_HOME//
ip_audit/lint/ip_netlist-mixed.spq
```

Specifying an Additional Path

An additional path is a path that exists in continuation to the path set by a reference environment variable.

SpyGlass appends this path to the path set by a reference environment variable. For example, after specifying the `$SPYGLASS_HOME` reference environment variable, if you specify the additional path as `ABC`, the tool appends this path to `$SPYGLASS_HOME` reference environment variable, as shown below:

```
$SPYGLASS_HOME/ABC/
```

Now consider the following `inherit_goal` specification of an inherited goal file:

```
-inherit_goal $SPYGLASS_HOME/ABC//ip_audit/lint/ip_netlist-
mixed.spq
```

In the above case, the highlighted path internally expands in the following manner:

```
-inherit_goal RELEASE/SpyGlass-<version>/SPYGLASS_HOME/
ABC//ip_audit/lint/ip_netlist-mixed.spq
```

Migrating Custom Goals

A methodology, such as GuideWare is designed to use goals during different stages of RTL development. You can modify these goals to create your own custom goals depending upon your requirement.

However, to ensure a structured flow for design analysis, it is recommended to migrate your custom goals along with goals of existing methodologies.

Migrating goals creates a new methodology that contains rules from custom goals as well as goals from existing methodologies specified by the user.

Migrating custom goals requires you to perform the following tasks:

1. *Comparing Goals*
2. *Migrating Goals*

Comparing Goals

You can compare custom goals with an existing methodology, such as GuideWare goals and analyze differences between these goals.

For example, you can compare rules that are common in custom goals and goals in an existing methodology. Similarly, you can compare rules that are present in custom goals but missing in goals of the specified methodology.

To compare goals, perform the following steps:

1. Select the *Tools -> Compare -> Goals with Methodology* menu option.

   The *Compare custom goals* dialog appears, as shown in the following figure:

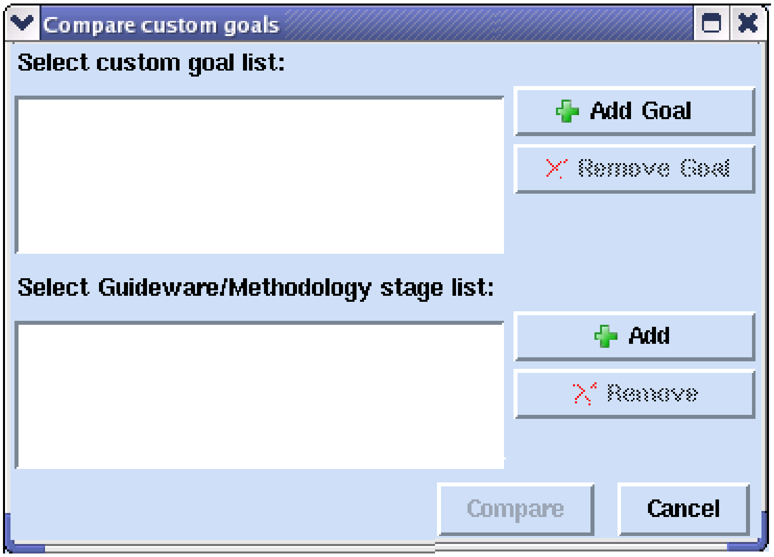


FIGURE 184. Compare Custom Goals

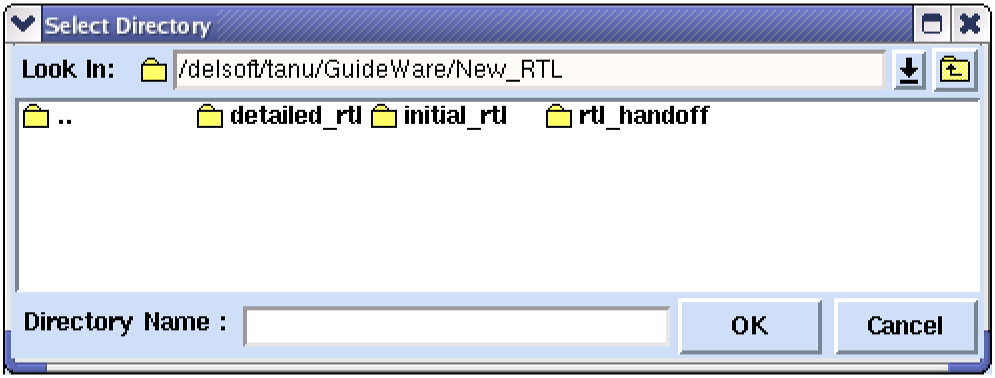
2. In the above dialog, click the *Add Goal* button.

   The *Select File(s)* dialog appears, from which you can select the required custom goals.

3. Click the *Add* button.

   The *Select Directory* dialog appears, from which you can select directories for standard GuideWare goals.

   For example, the following figure shows the *Select Directory* dialog, from which you can select the required GuideWare stages:



Select Directory

Look In: /delsoft/tanu/GuideWare/New_RTL

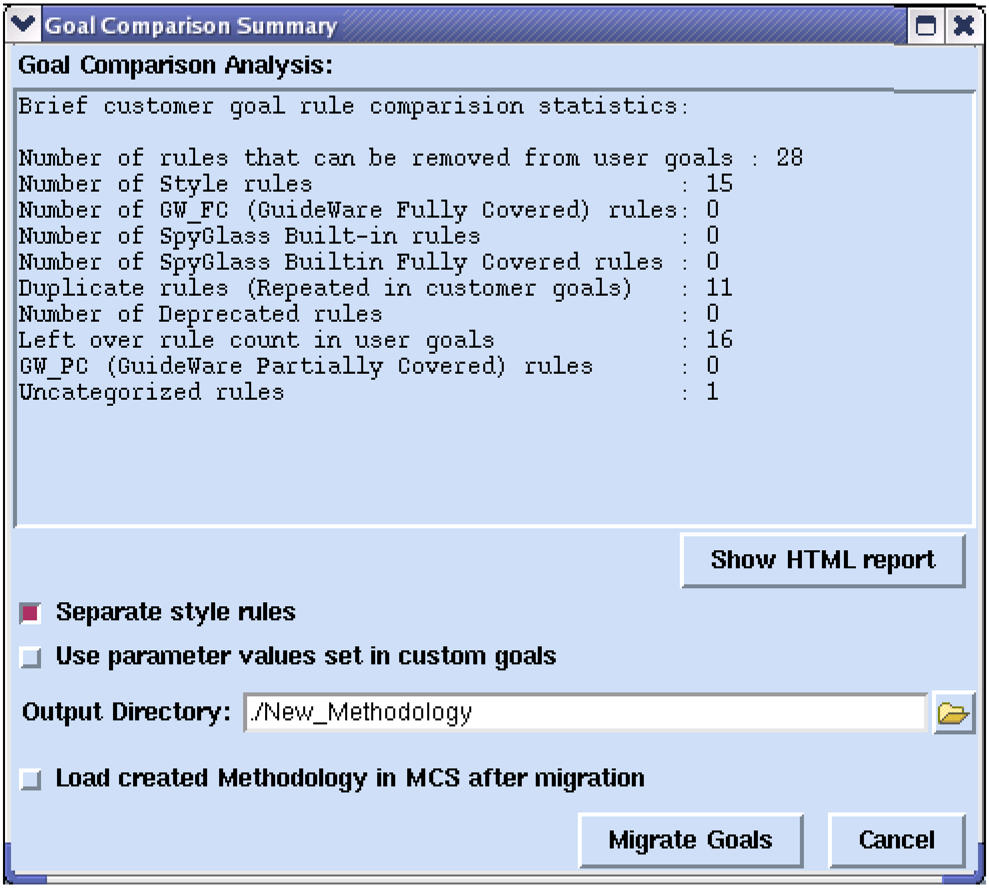
.. detailed_rtl initial_rtl rtl_handoff

Directory Name :

FIGURE 185. Select Directory

4. Click the *Compare* button.

   The *Goal Comparison Summary* dialog appears, as shown in the following figure:



Goal Comparison Analysis:

```
Brief customer goal rule comparision statistics:

Number of rules that can be removed from user goals : 28
Number of Style rules                              : 15
Number of GW_FC (GuideWare Fully Covered) rules: 0
Number of SpyGlass Built-in rules                  : 0
Number of SpyGlass Builtin Fully Covered rules : 0
Duplicate rules (Repeated in customer goals)      : 11
Number of Deprecated rules                         : 0
Left over rule count in user goals                 : 16
GW_PC (GuideWare Partially Covered) rules          : 0
Uncategorized rules                                : 1
```

☒ Separate style rules

☐ Use parameter values set in custom goals

Output Directory: ./New_Methodology

☐ Load created Methodology in MCS after migration

FIGURE 186. Goal Comparison Summary

The above dialog displays a brief comparison summary between rules of the specified custom goals and goals of the specified methodology.

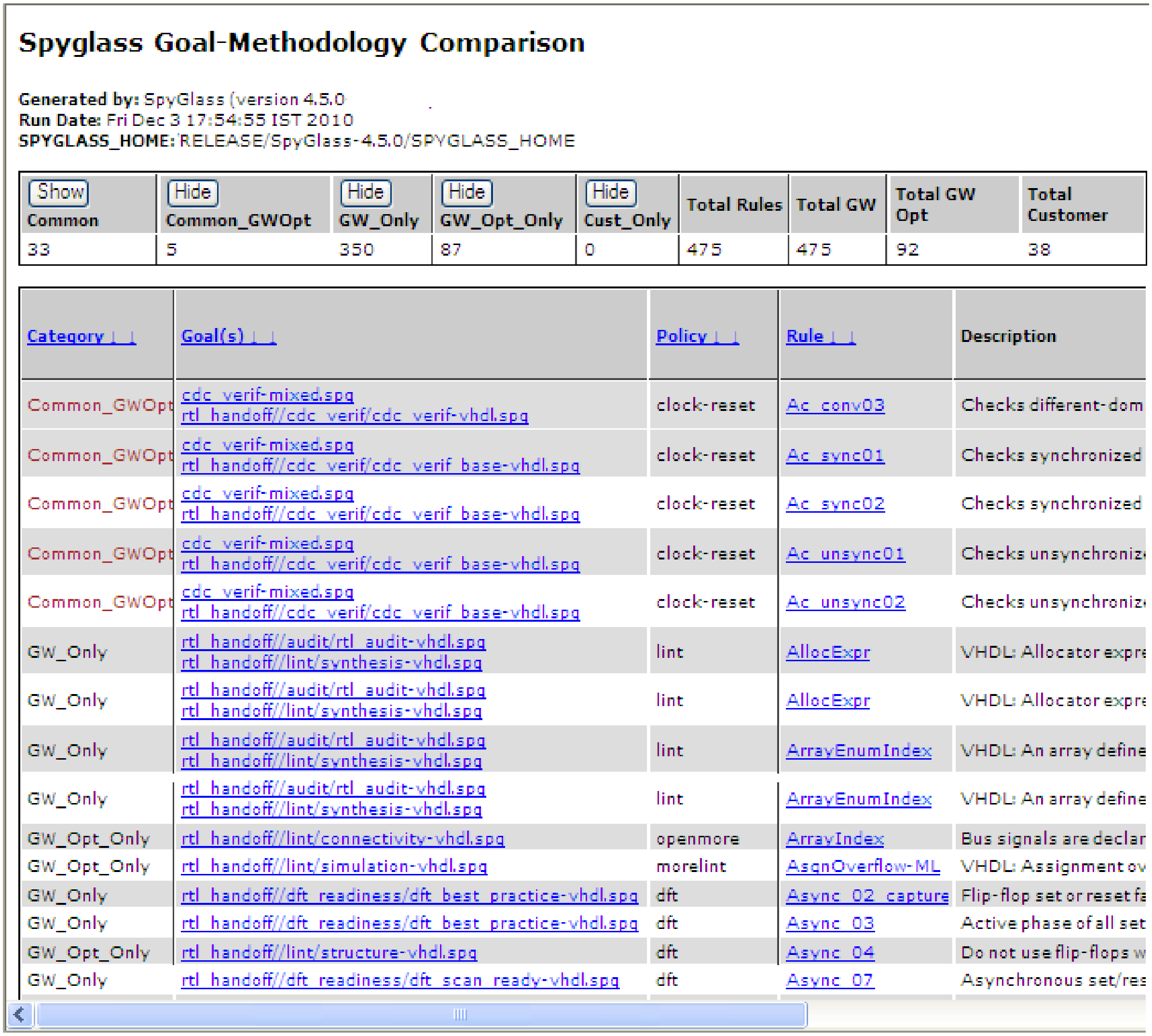
These rules are described in the following categories:

Category	**Description**
Common	Number of common rules between custom goals and GuideWare. These are GuideWare non-optional (mandatory) rules.
Common_GWOpt	Number of common rules that are specified as 'optional' in GuideWare
GW_Only	Number of GuideWare rules included in the migration result output flow. They do not appear in custom goals.
GW_Opt_Only	Number of GuideWare 'optional' rules included in the migration result output flow. They do not appear in custom goals. (Note: These rules are not mandatory but can be considered by user if interested).
Cust_Only	Number of custom included only rules (not part of GuideWare)
Total Rules	Total rule count found in GuideWare and custom goals
Total GW	Total GuideWare non-optional rules
Total GW Opt	Total GuideWare optional rules
Total Customer	Total rules in custom goals

You can view a detailed comparison summary of rules in a separate browser by clicking the *Show HTML report* button. For details, see *Viewing the HTML Report for Comparison*.

Viewing the HTML Report for Comparison

The following figure shows a sample HTML report displaying a detailed comparison summary:



Spyglass Goal-Methodology Comparison

Generated by: SpyGlass (version 4.5.0
Run Date: Fri Dec 3 17:54:55 IST 2010
SPYGLASS_HOME: RELEASE/SpyGlass-4.5.0/SPYGLASS_HOME

Show Common	Hide Common_GWOpt	Hide GW_Only	Hide GW_Opt_Only	Hide Cust_Only	Total Rules	Total GW	Total GW Opt	Total Customer
33	5	350	87	0	475	475	92	38

Category ↓ ↓	Goal(s) ↓ ↓	Policy ↓ ↓	Rule ↓ ↓	Description
Common_GWOpt	cdc_verif-mixed.spg rtl_handoff//cdc_verif/cdc_verif-vhdl.spg	clock-reset	Ac_conv03	Checks different-dom
Common_GWOpt	cdc_verif-mixed.spg rtl_handoff//cdc_verif/cdc_verif_base-vhdl.spg	clock-reset	Ac_sync01	Checks synchronized
Common_GWOpt	cdc_verif-mixed.spg rtl_handoff//cdc_verif/cdc_verif_base-vhdl.spg	clock-reset	Ac_sync02	Checks synchronized
Common_GWOpt	cdc_verif-mixed.spg rtl_handoff//cdc_verif/cdc_verif_base-vhdl.spg	clock-reset	Ac_unsync01	Checks unsynchroniz
Common_GWOpt	cdc_verif-mixed.spg rtl_handoff//cdc_verif/cdc_verif_base-vhdl.spg	clock-reset	Ac_unsync02	Checks unsynchroniz
GW_Only	rtl_handoff//audit/rtl_audit-vhdl.spg rtl_handoff//lint/synthesis-vhdl.spg	lint	AllocExpr	VHDL: Allocator expre
GW_Only	rtl_handoff//audit/rtl_audit-vhdl.spg rtl_handoff//lint/synthesis-vhdl.spg	lint	AllocExpr	VHDL: Allocator expre
GW_Only	rtl_handoff//audit/rtl_audit-vhdl.spg rtl_handoff//lint/synthesis-vhdl.spg	lint	ArrayEnumIndex	VHDL: An array define
GW_Only	rtl_handoff//audit/rtl_audit-vhdl.spg rtl_handoff//lint/synthesis-vhdl.spg	lint	ArrayEnumIndex	VHDL: An array define
GW_Opt_Only	rtl_handoff//lint/connectivity-vhdl.spg	openmore	ArrayIndex	Bus signals are declar
GW_Opt_Only	rtl_handoff//lint/simulation-vhdl.spg	morelint	AsgnOverflow-ML	VHDL: Assignment ov
GW_Only	rtl_handoff//dft_readiness/dft_best_practice-vhdl.spg	dft	Async_02_capture	Flip-flop set or reset fa
GW_Only	rtl_handoff//dft_readiness/dft_best_practice-vhdl.spg	dft	Async_03	Active phase of all set
GW_Opt_Only	rtl_handoff//lint/structure-vhdl.spg	dft	Async_04	Do not use flip-flops w
GW_Only	rtl_handoff//dft_readiness/dft_scan_ready-vhdl.spg	dft	Async_07	Asynchronous set/res

FIGURE 187. Detailed Goal Comparison Summary

Migrating Goals

Migrating goals is a process in which user-specified custom goals are merged with the specified methodology. After migration, SpyGlass creates a new methodology that contains rules from custom goals as well as rules from the specified methodology. You can specify the name for this methodology in the *Output Directory* field of the *Goal Comparison Summary* dialog.

For example, you may want to migrate some custom goals with `initial_rtl` and `rtl_handoff` stages of the *GuideWare/New_RTL* methodology. In this case, custom goals are compared with the goals of the `initial_rtl` and `rtl_handoff` stages and a new methodology is created that contains rules from the `initial_rtl` and `rtl_handoff` stages as well as rules that were specified in the custom goals but were not present in any of these stages.

Before migrating goals, you can:

- Select the *Separate style rules* option to separate coding style-specific rules from custom goals to a single `style_checks` goal.
- Select the *Use Parameter Values Set In Custom Goals* option to overwrite parameter values of the specified methodology with parameter values set in custom goals.
- Select the *Load created methodology in MCS after migration* option to load the newly created methodology containing migration results in the *MCS* window.

After selecting the required options, click the *Migrate Goals* button in the *Goal Comparison Summary* dialog. This creates a new methodology of the specified name that contains a combined set of rules present in custom goals and specified methodologies.

Order File

An order file contains path of goal files relative to a methodology directory. This path is used to specify the order in which goals are arranged in a methodology.

Each methodology contains one order file that defines the order of all its goals.

For example, consider the following sample structure:

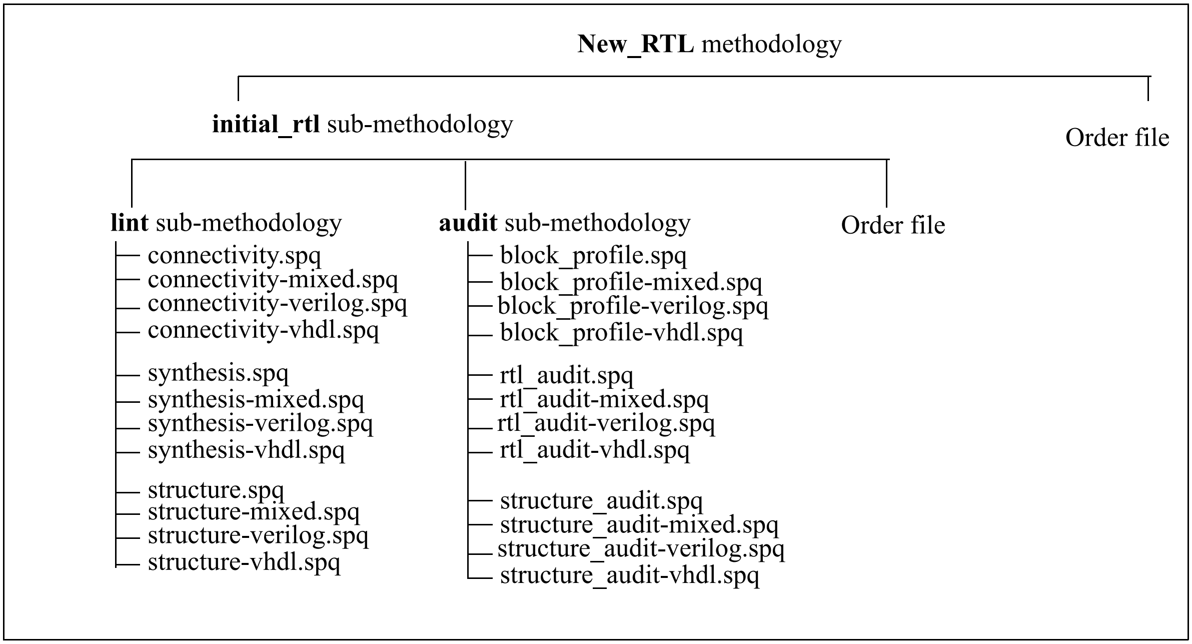


FIGURE 188. Sample Directory Structure

For the above example, the order file under the `New_RTL` methodology should contain the following entries:

```
initial_rtl
initial_rtl/lint/connectivity*
initial_rtl/lint/synthesis*
initial_rtl/lint/structure*
```

```
initial_rtl/audit/block_profile*
initial_rtl/audit/rtl_audit*
initial_rtl/audit/structure_audit*
```

Similarly, the order file under the `initial_rtl` methodology should contain the following entries:

```
lint/connectivity*
lint/synthesis*
lint/structure*
audit/block_profile*
audit/rtl_audit*
audit/structure_audit*
```

Each line in an order file specifies one goal-path entry.

NOTE: *The goal name should not contain `–<language>`.spq in an order file. It should contain an asterisk (*) after the name.*

Viewing Order of Goals Defined in an Order File

You can view the order of goals defined in the order file in either of the following ways:

- In GUI, display the *Methodology Configuration Window* or select the *Select Goal* tab of the *Goal Setup & Run* stage.
- In batch, specify the `-showgoals` option.

NOTE: *If the order file of a methodology does not contain entry for a particular goal of that methodology, SpyGlass does not display such goal in GUI or batch.*

Format of an Order File

An order file is divided into the following two sections:

- *Commented Section*
- *Goal Description and Attributes Area*

Commented Section

This section provides the description of the methodology. This section is present between `=methodology` and `=cut`, and is present at the top of the order file.

You can view the description provided by this section under the *Select Goal* tab or in the *Methodology Configuration System* window.

NOTE: *Each order file contains only one comment.*

The commented section contains the following details in the specified order:

- The first line starts with the `=methodology` string to indicate the beginning of a comment.
- Next line specifies the methodology name for which the order file is present.
- (Optional) Next line specifies the name of the parent methodology, if present, for the current methodology in the following format:

  ```
  OLDMETH: <methodology-name>
  ```
- Next line contains `*` to indicate the beginning of the short help of the methodology.
- Next line specifies a one-liner short help of the methodology.
- (Optional) Next line specifies the short help of the parent methodology, if present, for the current methodology in the following format:

  ```
  OLDDESC: <short-help>
  ```
- Next line contains `*` to indicate the end of the short help of the methodology.
- Next line contains the beginning of the long help of the methodology.

  If the current methodology has a parent methodology, the long help of the parent methodology is in the following format:

  ```
  OLDDESC: <long-help>
  ```
- Last line contains `=cut` to indicate the end of commented area.

A sample order file is given in the *Sample Order File* section.

Goal Description and Attributes Area

This section contains name and relative-path of each goal of the methodology and attributes of each goal.

The order in which goals appear in SpyGlass GUI (*Select Goal* tab and *Methodology Configuration System* window) is based on the order of goals specified in this section.

Following are the details of this section:

- The name of the methodology directory is displayed first, followed by the path of all goals under that methodology, as shown in the following example:

  ```
  initial_rtl
  initial_rtl/lint/connectivity*
  initial_rtl/lint/simulation*
  initial_rtl/lint/synthesis*
  initial_rtl/lint/structure*
  ```

- Each goal name is appended with `*` or `%`, to indicate whether they are .spq or .spc files, respectively.
- The `PREREQ:` tag specifies the path of prerequisite goal(s) for a particular goal.

  This tag is present in the same line where the goal path is present. This tag is followed by the path of prerequisite goal, as shown in the following example:

  ```
  rtl_handoff/constraint_generation/gen_sdc* PREREQ:
  rtl_handoff/constraint/sdc_quick_check
  ```

  In case of multiple prerequisite goals, specify a comma-separated list of paths of prerequisite goals.

  The prerequisite goal path(s) must be relative to the methodology directory.
- The `!HIDE` tag appearing before the goal name indicates that the corresponding goal will be hidden in SpyGlass window. Such goals are, however, visible in the *Methodology Configuration System* window.
- The `DDR_GOAL` tag is used for DDR specific goals. Setups of such goals are mandatory.

Map File

Map file is used to trace back the reference of the new goal, which is present in a methodology, to the original goal, which is present in another methodology.

You can specify a map file along with an order file in a methodology to mark the mappings between the goals in different methodologies.

For example, following describes mapping between GuideWare 1.0 and GuideWare 2.0 goals:

```
initial_rtl/lint/connectivity,initial_rtl/lint/
simulation,initial_rtl/lint/synthesis,initial_rtl/lint/
structure::lint/lint_rt
```

In the above example, the `lint_rt` goal in GuideWare 2.0 represents following four goals in GuideWare 1.0:

- connectivity
- simulation
- synthesis
- structure

Working with SpyGlass Design Constraints

SpyGlass Design Constraints (SGDC) are used to:

- Provide additional design information that is not apparent in the RTL description.
- Restrict SpyGlass analysis to a set of objects.

Consider a scenario in which you want to specify the names of clock nets to be checked. While SpyGlass can infer clocks in the design, you may want to restrict the analysis to only a handful of clocks or specify other clocks that could not be inferred. In this case, you can specify the required clock information by using the appropriate constraint.

NOTE: *The design constraint files can have any extension. However, it is recommended to use the* .sgdc *extension to facilitate better recognition and handling.*

NOTE: *The previous method of supplying design constraints using embedded design pragmas is still supported for backward compatibility. However, it is strongly recommended that you use the design constraints file method that is superior. If both the design constraints file and embedded design pragmas are specified, SpyGlass uses the design constraints file only and ignores the embedded design pragmas. Similarly, if you have not specified a design constraints file but have embedded design pragmas in the source code, SpyGlass reads these pragmas and creates a design constraints file,* pragma2constraints.sgdc, *located in the goal result directory under spyglass_spysch. For example, <project_name>/<top_name>/<full_goal_name_and_path/spyglass_spysch/constraints/*

pragma2constraints.sgdc.

Specifying SGDC Files to SpyGlass

Specify SGDC files in either of the following ways:

- By using the `read_file -type sgdc` *`<SGDC-file-name>`* command in a project file
- By using the *Add Files* option under the *Add Design Files* tab in SpyGlass GUI

Creating an SGDC File

You can write different constraint specifications in an SGDC file. An SGDC file can be a text file of any extension. However, it is recommended to save such files with the .sgdc extension.

For details on any constraint, refer to the SpyGlass Consolidated Constraints Application Note.

Adding Comments in an SGDC File

To add comments in an SGDC file, use # or // before the comment, as shown in the following example:

```
# comment1
```

```
//comment2
```

The # comment identifier must be the first character in a code line or must have only whitespace before it. All the text after the comment identifier till the end of the line is considered as a comment.

The // comment identifier can be specified anywhere in a code line. All the text after the comment identifier till the end of the line is considered as a comment.

Defining a Scope for Constraints

A scope defines the design unit in which the specified constraints are applicable.

To define a scope, use the `current_design` *`<design-unit>`* command before writing SGDC commands, where *`<design-unit>`* can be any of the following:

For Verilog:	<module-name>	
For VHDL:	<entity-name>	<entity-name>.<archname>
	<configuration-name>	<libname>.<configuration-name>

The following example defines scopes for different constraint specifications:

```
current_design B1
  clock -testclock -name tclk1 -value rtz
  testmode -name tm1 -value 1
```

Scope for design unit B1

```
current_design B2
  clock -testclock -name tclk2 -value rto
  testmode -name tm2 -value 0
```

Scope for design unit B2

```
current_design B3
  clock -testclock -name tclk3 -value rto
```

Scope for design unit B3

For more information on the Tcl-based usage of the *current_design* command, refer to the *current_design* section of the *SpyGlass Tcl Shell Interface User Guide*.

If you specify constraints without defining a scope for them, such constraints are applicable to the entire design. In the following example, the `waive` constraint is applicable for the whole design because no `current_design` command is specified before the `waive` constraint:

```
waive -file *. -rules ALL
```

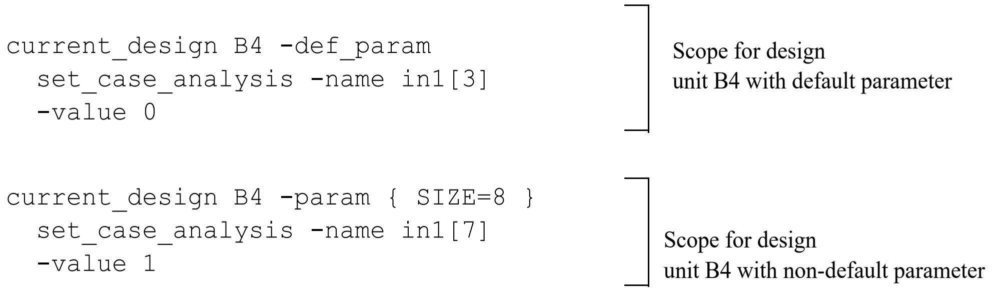
NOTE: *SpyGlass checks the design unit name in case-insensitive manner. Therefore, if your Verilog design has two modules named* `FOO` *and* `foo`*, specifying a* `current_design` *keyword line with* `FOO` *or any of its case variants as its argument will result in the same set of constraints on both* `FOO` *and* `foo` *modules.*

Please note that some products, such as SpyGlass DFT solution, work only on flattened netlists. Therefore, the `current_design` command must specify only top-level design units for these products. However, if there are multiple top-level design units in a design, specify the `current_design` command for each top-level design unit; and all the constraints related to that top-level design unit must follow the corresponding `current_design` line.

For a parametrized design unit, the `-def_param` switch of the current design is used to define scope specific to its default parameter. The `param` parameter is used for user-specified values. The `-param` switch of the current design accepts list of non-default parameters in the following format:

```
<param>=<value>
```

For example consider a design unit having instances of parametrized design unit `B4`, one instantiated with default parameter value and other with overridden parameter value `'8'`. Following specification defines the scope for default and non-default parametrized design unit `B4`:



```
current_design B4 -def_param
  set_case_analysis -name in1[3]
  -value 0
```

Scope for design unit B4 with default parameter

```
current_design B4 -param { SIZE=8 }
  set_case_analysis -name in1[7]
  -value 1
```

Scope for design unit B4 with non-default parameter

SGDC Convention for Packed Arrays

Packed array elements are referred in an RTL using the dot separator ("."). However, while referring to such elements in an SGDC file, use square brackets ([]).

Consider the following example:

```
typedef struct packed {
rx_g  rx;
ctrl_g  ctrl;
lo_g  lo;
} AD;
module test(input c1, c2, d1, d2, input AD AD_IN, output
[1:0] o1, o2, o3);
flop f1 (AD_IN.lo , c2, o3);
endmodule
```

In the above example, the dot separator is used for AD_IN.lo. However, in an SGDC file, this element is specified as `AD_IN[lo]`, as shown below:

```
input -name AD_IN[lo] -clock c2
```

Specifying Multiple current_design Specifications for a Design Unit

You can specify multiple `current_design` specifications for a particular design unit, as shown in the following example:

```
current_design B1
  clock -testclock -name tclk1 -value rtz
  testmode -name tm1 -value 1

current_design B2
  clock -testclock -name tclk2 -value rto
  testmode -name tm2 -value 0

current_design B1
  clock -testclock -name tclk3 -value rto
```

In the above example, there are two `current_design` lines for design unit B1 that specify two clocks (tclk1 and tclk3) and one testmode (tm1).

Consider the following example:

```
current_design B3
  testmode -name tm1 -value 1

current_design B3 -def_param
  clock -testclock -name tclk1 -value rto
  testmode -name tm2 -value 0

current_design B3 -param { SIZE=8 }
  clock -testclock -name tclk2 -value rto
```

In the above example, the current_design specification without -def_param and -param switches is applicable for all the design versions of design unit B3. When B3 is instantiated with default parameter values, clock (tclk1) and testmodes (tm1, tm2) are visible. Similarly, when B3 is instantiated with parameter SIZE=8, a clock (tclk2) and a testmode (tm1) are visible.

Specifying Configuration Name with current_design Command

If you specify a configuration name with the current_design command, ensure that the specified configuration is not present in multiple precompiled libraries.

If a configuration name is present in multiple precompiled libraries, avoid using the configuration name. Instead, use *<entity-name>*.*<architecture-name>* for which the intended configuration is defined.

For multiple architecture VHDL designs, use the *<entity-name>*.*<arch-name>* format to specify the current_design command only while analyzing the design with the hdllibdu project file command.

Specifying Multiple Values for a Constraint Argument

If a constraint argument accepts multiple values, specify values in either of the following ways:

- Specify a list of names in a single constraint specification, as shown in the following example:

  ```
  voltagedomain -isosig top.isig1 top.isig2
  ```

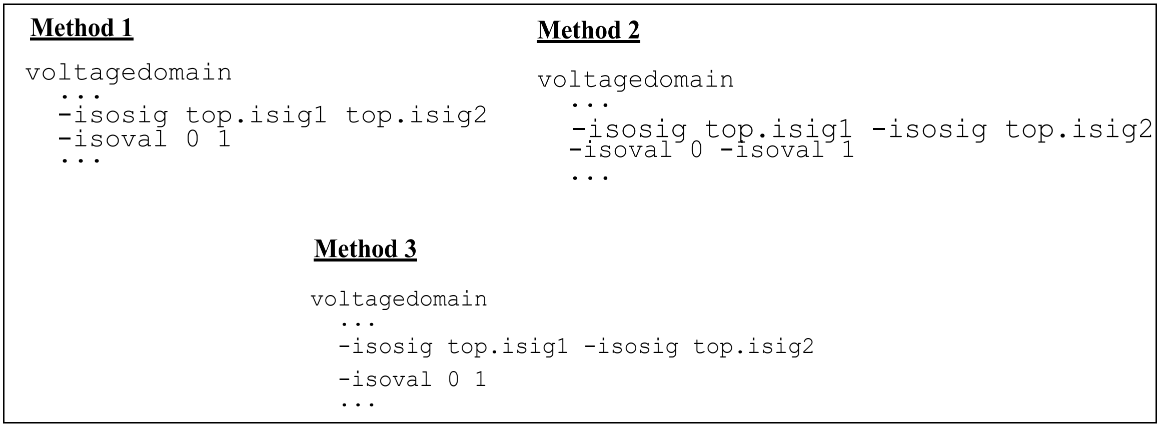
- Specify each value in a separate constraint specification, as shown in the following example:

  ```
  voltagedomain -isosig top.isig1
  voltagedomain -isosig top.isig2
  ```

Handling Interdependencies between Different Arguments

When two arguments of the same constraint have interdependency, you must specify the exact matching number of values with each argument. For example, the `-isosig` argument and the `-isoval` arguments of the `voltagedomain` constraint are interdependent.

You can handle these interdependencies in any of the following ways:



Method 1

```
voltagedomain
   ...
   -isosig top.isig1 top.isig2
   -isoval 0 1
   ...
```

Method 2

```
voltagedomain
   ...
   -isosig top.isig1 -isosig top.isig2
   -isoval 0 -isoval 1
   ...
```

Method 3

```
voltagedomain
   ...
   -isosig top.isig1 -isosig top.isig2
   -isoval 0 1
   ...
```

The style used in one argument can be different than the style used in the other interdependent argument.

NOTE: *The purpose and function of each design constraint keyword is product-specific and is described in the product rules reference document of the respective product where the design constraint can be used. For example, the SpyGlass CDC solution uses the* `clock` *and* `reset` *design constraint keywords (besides many other design constraint keywords), and the SpyGlass CDC Rules Reference describes how these design constraints are used for the product. In addition, a product can have its own product-specific design constraint keywords.*

NOTE: *Application of a design constraint keyword may be different in different products. For example, the* `-domain` *argument of the* `clock` *design constraint keyword is important when used with the SpyGlass CDC solution but is ignored when used with SpyGlass DFT solution. Similarly, the* `-testclock` *argument is important in the SpyGlass DFT solution but is ignored by the SpyGlass CDC solution.*

Including an SGDC File in Another SGDC File

Use the `include` directive to include an SGDC file in another SGDC. The `include` directive is used in the following format:

```
include <file-name>
```

In the following example, the constraint_include.sgdc file is included in the constraint.sgdc:

```
// Contents of constraint.sgdc

include constraint_include.sgdc

current_design and_new
test_mode -name in1 -value 1
```

```
// Contents of constraint_include.sgdc

   current_design test
   test_mode -name test.w1 -value 1
```

In the above case, when you specify the constraint.sgdc file during SpyGlass analysis, SpyGlass expands the contents of this file to the following:

```
current_design test
  test_mode -name test.w1 -value "1"

current_design and_new
  test_mode -name in1 -value "1"
```

Specifying Signal Names

Certain constraint arguments accept names of signals, such as clock signals and low-power signals.

Based on the design hierarchy in which such signals are present or the type of signals, such as scalar or vector signals, you must specify signal names in a correct format so that SpyGlass can identify them correctly.

Specifying Signal Names based on Signal Types

Signal name specification varies based on signal type, such as a scalar signal (for example, `clk1`), a bit-select of a vector signal (for example, `CLK[2]`), or a part-select of a vector signal (for example, `CLK2[0:2]`).

Note the following points:

- You can directly specify a multi-dimensional array bit-select and part-select with SpyGlass design constraints. SpyGlass performs some sanity checking after synthesis.
- You can also specify array of instances (Verilog) in escaped format with SpyGlass design constraints. For example, consider the following specification:

  ```
  M1 U1[0:2](a,b);
  ```

  For the above specification, you can specify instances as '`\U1[0] `', '`\U1[1] `', or '`\U1[2] `' (quotes not required).

  However, range specification is not supported. Therefore, '`\U1[0:2] `' is not supported.

Specifying Signal Names based on Design Hierarchy

Signal name specification varies based on design hierarchy, as described below:

- Simple signal

For example, signal specification `clk1` means that this signal is in the design unit identified by the `current_design` specification.

- Module signal name

  For example, signal specification `top.CK1,` where the prefix specified before the period (`.`) hierarchy separator is same as the name of the design unit in the `current_design` specification. In this case, the description is equivalent to simple name specification as above.

- Hierarchical signal name

  For example, the signal specification `top.U1.U2.CK1,` where multiple values specified with the dot (`.`) hierarchy separator identifies the design hierarchy within the `current_design` specification. This detailed specification may begin with either the name of the design unit in the `current_design` specification or the instance name within the design unit in the `current_design` specification.

NOTE: *It is not required to specify the top-level design unit name (which is specified with* `current_design`*) in a hierarchical name. Thus, both* `top.U1.U2.CK1` *and* `U1.U2.CK1` *are acceptable (and are the same) under* `current_design top`*.*

In all of the above cases, SpyGlass first searches the reported signal as `PORT` signal, and then as `NET` signal.

NOTE: *You can specify escaped names by enclosing them in double quotes as in* `“\myvlogsig1”`*,* `“\myvhdlsig#11\”`*. You only need to escape the double quote character in an escaped name as in* `“\myvlogsig\”23\””`*,* `“myvhdlsig\”5\””`*.*

NOTE: *You can also use Synopsys-style escaped names by specifying the following command in SpyGlass project file:*

```
set_option support_sdc_style_escaped_name yes
```

By default, SpyGlass supports the dot (`.`) character (main; always supported) and the forward slash (`/`) character (additional; set in the default SpyGlass Configuration file) as the hierarchy separator. Use the command named `set_hsep` to specify your own additional hierarchy character. Thus, you can use any Synopsys-style hierarchy separator in SpyGlass Design Constraints files.

The following example specifies the @ character as the additional hierarchy character:

```
...
set_hsep @
current_design top
  clock -name top@clk1 ...
...
```

Defining and Using Variables

Variables are used to store values that can be used as argument values of constraints.

Once you define a variable and assign a value to it, you can use that variable name as the value of a constraint argument. SpyGlass internally expands that variable name to its value for that argument.

Defining Variables

To define a variable in an SGDC file, use the following command:

```
setvar <variable-name> <variable-value>
```

For example, the following command defines the variable `myvar1` and assigns the value `clk` to this variable:

```
setvar myvar1 clk
```

Using Variables

You can use a variable in any of the following formats:

- `$`*`<variable-name>`*
- `${`*`<variable-name>`*`}`.

For example, in the following `clock` constraint specification, the variable `myvar1` is used as the value of the `-name` argument:

```
clock -name $myvar1
```

For the above command, SpyGlass internally assigns the value `clk` to the `-name` argument of the `clock` constraint.

NOTE: *Non-variable strings that start with $ should be escaped with a backslash to avoid confusion.*

Note the following points:

- You can define only one variable per line.
- A variable definition can span over multiple lines using the backslash continuation character.
- There is no = or := between the variable name and its value, to keep it consistent with Tcl format.
- Variable names must start with a letter and can contain letters, numerals, and underscore characters.
- Variable names are case-sensitive. Thus, `xyz` and `XYZ` are different variables.
- The variable value can be any string consisting of one or more words. You must enclose multi-word values within double quotes.
- Double quotes used in variable names are a part of the variable name itself.
- A variable remains visible within the scope of its containing SpyGlass constraints file. Thus, it is also visible in the included SpyGlass constraints files, if any.
- A variable becomes visible immediately from the next line after its definition and remains visible till the end of the file.
- You can define a variable multiple times in a file. In such cases, every definition overrides the previous definition and the current definition is applicable for subsequent commands.
- You can refer a variable in its definition as well. This allows you to redefine the variable with additional values in the same SpyGlass constraints file.

  For example, following are the allowed definitions:

  ```
  ...
  setvar var1 b/c
  ...
  setvar var1 $var1/d
  ```

```
...
```

Starting with the first definition, the value of variable `var1` is `b/c` till it is redefined again. Then, the value of the variable `var1` becomes `b/c/d`.

- A variable definition can refer other variables that are already defined as in the following example:

```
...
setvar var1 a/b/c
...
setvar var2 $var1/d
...
```

- You can also use the operating system-level environment variables in the SGDC files. In such cases, the name of a local variable should not be the same as that of an existing operating system-level environment variable.

Handling Duplicate Constraint Specifications

If you specify multiple specifications for a constraint that can be applied only once on a design object, the following actions occur:

- SpyGlass considers only the last specification of that constraint.
- SpyGlass reports the *SGDCWRN_115* warning and ignores the rest of the specifications of that constraint.

Consider the following example:

```
current_design top

set_case_analysis -name in -value 0
set_case_analysis -name in -value 1
```

For the above example, SpyGlass considers only the last `set_case_analysis` constraint specification and ignores the first constraint specification that sets the value of the `in` pin to `0`.

Handling Nets Declared in a Sequential Block

Consider the following example:

```
module TOP (in, temp, clk1, clk2);
  input in, temp, clk1, clk2;
  BASIC U_BASIC (in, temp, clk1, clk2);
endmodule

module BASIC (in, temp, clk1, clk2);
  input in, temp, clk1, clk2;
  reg outb, outc, outd;
  generate
  begin:GENBLOCK
  always @ (posedge clk2)
  begin:BLOCK1
    reg abc;
    abc <= outc;
    outc <= abc & temp;
    end
    end
  endgenerate
endmodule
```

In the above example, if you want to use abc, specify the following notation in the SGDC file:

```
current_design TOP

clock -name clk1 -domain clk1
clock -name clk2 -domain clk2

cdc_false_path -from TOP.U_BASIC.in -to
"TOP.U_BASIC.\GENBLOCK.BLOCK1.abc "
```

Conditionally Specifying SGDC Constraints

To use the same SGDC file for different functional and testing analysis modes, compile different commands from the same SGDC file based on different conditions. These conditions are in the form of expressions made by using SGDC variables and a given set of common logical operators.

Use the `if-else` statement to implement conditional compilation of SGDC commands. Following is the syntax of the `if-else` statement:

```
if {<condition>} [then] {
  sgdc commands)
  ...
  ...
} elseif {<condition>} [then] {
  (sgdc commands)
  ...
  ...
} else {
  (sgdc commands)
  ...
  ...
}
```

The following operators are supported in the conditional expression:

`&&`, `||`, `!`, `==`, `!=`, `>`, `>=`, `<`, `<=`, `in`, `ni`

The following is the order of precedence (from highest to lowest) of these operators:

`!`, `>=`, `<=`, `==`, `!=`, `in`, `ni`, `&&`, `||`

Following are some examples:

```
setvar a 2
if {$a == 1} {
  constraint xyz
} elseif {$a == 2} {
  constraint abc
}
setvar a abcd
```

```
if {$a == "abcd"} {
  constraint xyz
} else {
  constraint abc
}
```

Using the SG_OPERATING_MODE Variable

SpyGlass enables you to set the value of the special variable `SG_OPERATING_MODE` from the command-line as well, through the `set_option operating_mode` *<value>* command in the project file.

Please note that other SGDC variables can be set from within an SGDC file (using setvar) or from the environment only.

The following is the order of precedence (from highest to lowest) for resolving multiple definitions of the `SG_OPERATING_MODE` variable:

1. `SG_OPERATING_MODE` variable set through the `operating_mode` option
2. `SG_OPERATING_MODE` variable set through a local variable inside an SGDC file
3. `SG_OPERATING_MODE` variable set through an environment variable

Note the following points:

- If the condition given in the `if` statement is an invalid expression then neither the `then` part nor the `else` part is interpreted.
- After providing the condition following the `if` keyword, you may or may not give the `then` keyword. However, you must begin the `then` block with { on the same line.
- Always enclose `then` and `else` blocks within curly brackets ({ }).
- Always give the `else` keyword before the beginning of an `else` block.
- The list operators (`in` and `ni`) can be used only when the RHS is a list variable.

  SpyGlass automatically constructs a list variables when their values are given as a space-separated list. In this case, specify the entire list in quotes to the setvar command, as shown in the following example:

```
setvar b "x y z x1 x2"
if {"y" in $b} then {
  constraint abc
} else {
  constraint xyz
}
```

Example of Using the SG_OPERATING_MODE Variable

This section provides some examples of using the SG_OPERATING_MODE variable.

Example 1

This example demonstrates the precedence of SG_OPERATING_MODE setting done inside an SGDC file over its environment variable value.

In the first if-elseif block of this example, the opmode environment variable setting is used. In the second if-elseif block, the sysmode local setting, as done by the setvar command, is applicable.

On shell

```
setenv SG_OPERATING_MODE opmode
```

Commands Specified in the test.sgdc File

```
current_design dummy
if {$SG_OPERATING_MODE == "sysmode"} {
  clock -name a -value rto
} elseif {$SG_OPERATING_MODE == "opmode"} {
  clock -name b -value rtz
}

#local setting inside the SGDC file, it has precedence over
# environment variable setting
```

```
setvar SG_OPERATING_MODE sysmode

if {$SG_OPERATING_MODE == "sysmode"} {
  clock -name c -value rto
} elseif {$SG_OPERATING_MODE =="opmode"} {
  clock -name d -value rtz
}
```

Commands Populated Inside SGDC Object Model (OM)

The following is populated inside SGDC OM:

```
current_design dummy
    clock -name b -value rtz
    clock -name c -value rto
```

Example 2

This example is the same as *Example 1* above, except that instead of setting the environment variable, the `operating_mode` option is set inside the project file.

In this example, the first and second `if-elseif` blocks use the `opmode` value of the `operating_mode` option in the project file. The local setting made through `setvar` in the SGDC file is ignored because it has lower precedence over the `operating_mode` option setting.

Command Specified in a Project File

```
set_option operating_mode opmode
```

Commands Specified In the test.sgdc File

```
current_design dummy
if {$SG_OPERATING_MODE == "sysmode"} {
  clock -name a -value rto
} elseif {$SG_OPERATING_MODE == "opmode"} {
  clock -name b -value rtz
}

#local setting inside the SGDC file, it has lower precedence
# over "operating_mode" option setting in the project file
setvar SG_OPERATING_MODE sysmode


if {$SG_OPERATING_MODE == "sysmode"} {
  clock -name c -value rto
} elseif {$SG_OPERATING_MODE == "opmode"} {
  clock -name d -value rtz
}
```

Commands Populated Inside SGDC Object Model (OM)

The following is populated inside SGDC OM:

```
current_design dummy
    clock -name b -value rtz
    clock -name d -value rtz
```

Example 3

Consider two scenarios `S1` and `S2` for the `lint` SoC goal, and a single SGDC file, soc_lint.sgdc, capturing constraints for these two scenarios.

NOTE: *For details on scenarios, see Working with Scenarios.*

In this example, you can use the `operating_mode` option to configure contents of the given SGDC file as per the scenario requirement.

In this case, the setup for these scenarios can be as follows:

```
read_file -type sgdc soc_lint.sgdc
current_methodology $SPYGLASS_HOME/GuideWare/SoC
current_goal soc_rtl/lint/soc_rtl -top top -scenario S1
set_goal_option operating_mode S1
current_goal soc_rtl/lint/soc_rtl -top top -scenario S2
set_goal_option operating_mode S2
```

Where soc_lint.sgdc have constraints defined based on the operating_mode value, as shown below:

```
current_design top
if {$SG_OPERATING_MODE == "S1"} {    #setup for S1 scenario
    set_case_analysis -name top.N1 -value 1
    clock -name clk1 -value rtz
} elseif {$SG_OPERATING_MODE == "S2"} {   #setup S2 scenario
    set_case_analysis -name top.N1 -value 0
    clock -name clk2 -value rtz
}
```

Example 4

In this example, different scenarios are defined for a goal of the SpyGlass Power family. In this case, the SGDC file has activity information defined as per power estimation modes, such as pessimistic, standby, and nominal.

```
# Activity info for Power estimation (condition modal
# analysis - pessimistic and nominal)
# goal scenarios defined in the wb_subsystem.prj file define
# the SG_OPERATION_MODEs
# the SG_OPERATING_MODE is set in the .prj file using the
# set_goal_option operating_mode {<mode_value>}

if { $SG_OPERATING_MODE == "PESSIMISTIC_POWER" } {
    # pessimistic activity - result in higher average power
    activity -instname "wb_subsystem" -activity 1.10 -prob
    0.80 -all_primary_input -all_register_output

} elseif { $SG_OPERATING_MODE == "STANDBY_POWER" } {
    # standby power activity - result in higher average power
    activity -instname "wb_subsystem" -activity 0.05 -prob
    0.5 -all_primary_input -all_register_output

} elseif { $SG_OPERATING_MODE == "NOMINAL_POWER" } {
    # nominal activity - result in more nominal average power
    activity -instname "wb_subsystem" -activity 0.56 -prob
```

```
        0.45 -all_primary_input -all_register_output

    } else {
        # default case
        activity -instname "wb_subsystem" -activity 0.56 -prob
        0.45 -all_primary_input -all_register_output
    }
```

Processing of SGDC Files

SpyGlass processes SGDC files by:

- *Parsing SGDC Files*
- *Performing Syntax Checking in SGDC Files*

Parsing SGDC Files

During an analysis run, SpyGlass first parses SGDC files before processing source files, etc.

If any problem is present in the SGDC files, SpyGlass reports appropriate messages. Based on the severity of these messages, SpyGlass performs appropriate actions as discussed below:

- Aborts further processing if any syntax errors are reported.
- Continues with source file processing if only warning or informational messages are reported.

Design constraint file parsing messages are named as SGDCSTX_*<number>*, SGDCWRN_*<number>*, and SGDCINFO_*<number>* for error, warning, and informational messages, respectively.

Performing Syntax Checking in SGDC Files

SpyGlass provides SGDC file-checking rules, such as SGDC_*<command_name><number>*, that check the semantics of SGDC command specifications.

For example, the SGDC_clock01, SGDC_clock02, and SGDC_clock03 rules check the semantics of values specified for the -name, -value, and -freq arguments, respectively, of the clock constraint.

Processing SpyGlass Design and Waiver Pragmas

If you have not specified an SGDC file, but the source code contains embedded SpyGlass design and/or waiver pragmas, SpyGlass reads these pragmas and creates either pragma2constraint.sgdc SGDC file and/or pragma2waiver.swl waiver file containing equivalent constraints or waivers.

Consider the following sample RTL file with design & waiver pragmas:

```
module test1(in1, in2, in3, out);
//spyglass testmode in2 1
input in1, in2, in3;
output out;
wire clk, and_out;
reg out;
//spyglass disable_block STARC05-3.1.4.2
and A1(and_out, 1'b0, in1);
assign clk = in3 ? and_out : in2;
//spyglass enable_block STARC05-3.1.4.2

always @(posedge clk)
out <= in1;
endmodule
```

For the above design, the pragma2constraints.sgdc file generated contains the following lines:

```
//Path: <project_name>/<top_name>/<full_goal_name>/
//spyglass_spysch/constraints/pragma2constraint.sgdc
current_design test1
    test_mode -name test1.in2 -value 1
```

For the above design, the pragma2waiver.swl file generated contains the following lines:

```
//Path:<project_name>/<top_name>/<full_goal_name>/
//spyglass_spysch/waivers/pragma2waiver.swl
##Waive commands corresponding to HDL pragmas
    waive -file_lineblock "test.v" 7 10 -rule "STARC05-
```

```
3.1.4.2" -comment"RTL_PRAGMA: Waiver pragma in HDL source"
```

While creating an SGDC or waiver pragmas equivalent file, SpyGlass reports appropriate messages if there are problems in SpyGlass pragma specifications.

The embedded pragma parsing messages are also named as `SGDCSTX_`*`<number>`*, `SGDCWRN_`*`<number>`*, and `SGDCINFO_`*`<number>`* for error, warning, and informational messages, respectively. For details about these messages, refer to the *SpyGlass Built-in Rules Reference Guide*.

The generated SGDC file is parsed in a similar way as described in *Processing of SGDC Files* and appropriate messages are reported in case of any issues.

Recognizing Clocks

Different Atrenta standard products process clock information based on their specific rule-checking requirements. Refer to the respective product rules reference document for details.

The following table summarizes how different Atrenta standard products process clock information:

Task	SpyGlass lint Solution	SpyGlass STARC Solution	SpyGlass CDC Solution	SpyGlass DFT Solution
Clocks Used For Analysis	Automatic Detection	User-specified (for only two rules)	User-specified clocks and their domains	User-specified
Identification of Clocks[a]	Stops at combinational gates	Stops at combinational gates	Goes beyond combinational gates	Stops at combinational gates
Specification of Clocks	Not possible	Allowed (for only two rules)	Allowed, including internal nodes as clocks	Allowed, but only external pins/ports can be specified
Clock Domain	Same as clock source	Same as clock source	User-specified domain for each clock	Same as clock source
Simple Divider	Divided clock treated as a different domain from the Master clock	Divided clock treated as a different domain from the Master clock	Divided clock treated as a different but related domain from the Master clock	Divided clock treated as a different source and thus domain from the Master clock
Design Constraints	Not supported	Not supported	Supported from 3.2.0	Always supported

[a] **Clocks are identified by traversing backwards from each flip-flop's clock pin. The identified clocks might be used for all clock-related rules (as in SpyGlass lint Solution), or may just be informative for the users (as in SpyGlass CDC Solution).**

Converting SDC Attributes into SGDC Commands

Various design attributes, such as clock definitions and input-output constraints defined in an SDC file, are typically used for timing analysis of designs. However, these attributes are also vital for other engines, such as SpyGlass DFT solution and SpyGlass CDC solution. SpyGlass requires you to specify these attributes as SGDC files.

To specify such attributes as SGDC files, you can use the SDC-to-SGDC feature that automatically translates SDC format design attributes into corresponding SGDC commands.

SpyGlass translates the following SDC commands to SGDC commands:

- create_clock
- create_generated_clock

  This command is not commented out in the generated SGDC file only if the following conditions hold true:

  - If generated clocks have domains different than the domains of their source clocks
  - If generated clocks are specified on black boxes
- set_case_analysis
- set_clock_group
- set_clock_sense
- set_disable_timing
- set_false_path
- set_input_delay
- set_mode
- set_multicycle_path
- set_output_delay

Enabling the SDC-to-SGDC Translation Feature

To enable this feature, set the value of the *Enable SDC-to-SGDC translation* field in the *Set Read Options* tab to *Yes*.

Alternatively, specify the following command in the project file:

```
set_option sdc2sgdc yes
```

You must specify an SDC file name (containing SDC commands to be translated) in an SGDC file by using the `sdc_data` constraint, as shown in the following example:

```
current_design <design-name>
  sdc_data –file <sdc-file-name>
```

You can specify the SGDC file containing the above command in the design read stage.

NOTE: *You can also specify a compressed SDC file generated by using the gzip utility.*

Changing the Default Hierarchy Separator of the SDC2SGDC Constraints

Use the `use_hier_sep_slash` parameter to change the default hierarchy separator of the SGDC constraints.

By default, the hierarchy separator used in SGDC constraint's name field is `'.'`

Example

Consider the following `create_clock` command in SDC.

```
create_clock -name "CLK" -add -period 10.0
-waveform {0.0 5.0} [get_pins buf_cts/clkout]
```

The above command is converted into following SGDC clock using the default hierarchy separator, `'.'`, as shown below:

```
clock  -name "test_top.buf_cts.clkout"  -domain CLK
-edge { 0.000000 5.000000}  -period 12 -tag CLK
```

When the value of the `use_hier_sep_slash` parameter is set to `yes`, the same clock is converted to the following SGDC clock:

```
clock  -name "test_top/buf_cts/clkout"  -domain CLK
-edge { 0.000000 5.000000}  -period 12 -tag CLK
```

Specifying the Mode of Domain Inference

You can use the `sdc_domain_mode` parameter to specify the mode of domain inference. By default, the domain inference mode is `sta_compliant`. In this mode, the clock domains are extracted as per the following guidelines:

- A path verified by STA implies that the source and destination have the same domain and therefore the clocks in such a path will be assigned the same domain. SpyGlass CDC therefore does not verify such paths.
- A path not verified by STA implies that the source and destination have different domains and therefore the clocks in such a path will be assigned different domains. SpyGlass CDC therefore verifies such paths.

NOTE: *If none of the* `set_clock_group`*,* `set_clock_uncertainty`*,* `set_false_path` *constraints is specified for a clock pair, they are considered synchronous.*

The following table lists the additional details about the `sdc_domain_mode` parameter:

Used by	sdc2sgdc flow
Options	sta_compliant, pessimistic, strict, async, strict_sta, sta_scg
Default value	sta_compliant
Example	
Console/Tcl-based usage	`set_parameter sdc_domain_mode strict`
Usage in goal/source files	`-sdc_domain_mode=strict`

When the mode is set to `strict`, you must provide all clock relationships in a single `set_clock_groups` command. SpyGlass reports a FATAL error if domain is not inferred.

When the mode is set to `pessimistic`, the behavior is similar to that of sta_compliant with the exception that if none of the `set_clock_group`, `set_clock_uncertainty`, or `set_false_path` constraint is specified for a clock pair, they are considered asynchronous.

NOTE: *The values, strict and pessimistic, will be deprecated in a future release.*

When the mode is set to `async`, then no domain inference from sdc constraints is done and clocks are considered asynchronous to each other.

When the mode is set to `strict_sta` or `sta_scg`, only the user-specified set_clock_group command would be considered and all the other commands related to domain computation are ignored. Also, a spreadsheet (.csv) file showing clock relationship is generated.

NOTE: *The value, strict_sta, for the sdc_domain_mode parameter is deprecated and will be removed in a future release.*

Inferring cdc_false_path for Clocks in Different Domains

Use the *sdc_generate_cfp* option to infer *cdc_false_path* in case we have different domains that were assigned to the clocks but no asynchronous relationship was specified between these clocks.

Used by	SDC2SGDCPARSE
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_option sdc_generate_cfp yes`
Usage in goal/source files	`-sdc_generate_cfp=yes`

For example, consider the following SDC declaration:

```
create_clock -name Clk1 -period 10.00 in1
create_clock -name Clk2 -period 10.00 in2
create_clock -name Clk3 -period 10.00 in3
create_clock -name Clk4 -period 10.00 in4

set_clock_group -asynchronous -group { Clk1 } -group { Clk3 }
```

Following corresponding SGDC commands are generated when you set the value of the *sdc_generate_cfp* option to yes:

```
cdc_false_path -from Clk1 -to Clk2
cdc_false_path -from Clk1 -to Clk4
cdc_false_path -from Clk2 -to Clk1
cdc_false_path -from Clk2 -to Clk3
cdc_false_path -from Clk3 -to Clk2
cdc_false_path -from Clk3 -to Clk4
cdc_false_path -from Clk4 -to Clk1
cdc_false_path -from Clk4 -to Clk3
```

Capturing Domain Inferring Results

The following rules capture the domain inferring results:

- *Domain_Missing01*: Reports clocks for which no domain relationship could be inferred from SDC commands. The domain assigned to these clocks depends on the mode specified in the `sdc_domain_mode` parameter. The severity of this rule is Error by default. In case of "`strict`" mode, it reports a FATAL violation.
- *Domain_Conflict01*: Reports conflicts found during domain inference. The severity of this rule is Error by default. In case of "`strict`" mode, it reports a FATAL violation.
- *Domain_Matrix01*: Generates spreadsheet to show clock relationships and the inferred domain depending upon the mode specified in the `sdc_domain_mode` parameter. It is an informational rule.

NOTE: *If you generate the **cdc_false_path** constraint through the **sdc_generate_cfp** command, SpyGlass CDC ignores the asynchronous crossings for the paths specified in the **cdc_false_path** constraints.*

Handling of Generated Clocks

Use the `enable_generated_clocks` or `sdc_generated_clocks` parameter to dump the generated clocks having same domain in an uncommented form in the sgdc file.

If the enable_generated_clocks parameter is specified, then generated clocks are dumped in the sgdc file in the form of the generated_clock constraint.

If the sdc_generated_clocks parameter is specified, then generated clocks are dumped in the sgdc file in the form of the clock constraint.

By default, the generated clocks having the same domain, if not given on black box, are dumped in a commented form in the sgdc file.

Set the value of the enable_generated_clocks or sdc_generated_clocks parameter to yes to dump all the generated clocks in an uncommented form in the sgdc file.

Used by	sdc2sgdc flow
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter enable_generated_clocks yes`
Usage in goal/source files	`-enable_generated_clocks =yes`

Used by	sdc2sgdc flow
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter sdc_generated_clocks yes`
Usage in goal/source files	`-sdc_generated_clocks=yes`

Handling False Paths

The *false_path* constraint is generated in the sgdc file when the following commands are specified:

- **set_false_path**

  Consider the following SDC commands:

  ```
  set_false_path -from [get_pins f1/q] -through [get_pins
  A2/out] -to [ get_pins f2/in]
  set_false_path -from [get_pins f1/q] -through [get_pins
  A2/out] -to [ get_pins f2/in]
  ```

  The following false_path constraints are generated corresponding to the above SDC commands:

  ```
  false_path -from f1/q -to f2/in -through A2/out -type sfp
  false_path -from f1/q -to f2/in -through A2/out -type sfp
  ```

- **set_clock_group**

  Consider the following SDC commands:

  ```
  set_clock_group -logically_exclusive
  set_clock_group - physically_exclusive
  set_clock_group - asynchronous
  ```

  The following false_path constraints are generated corresponding to the above SDC commands:

  ```
  false_path -scg_logically_exclusive
  false_path -scg_physically_exclusive
  false_path -scg_asynchronous
  ```

Handling Multi-cycle Paths

The sg_multicycle_path constraint is generated in the sgdc file when the set_multicycle_path command is given.

Consider the following SDC commands:

```
set_multicycle_path  -from [get_pins f1/q] -through [get_pins
A2/out] -to [ get_pins f2/in] 2
```

The following sg_multicycle_path constraint is generated corresponding to the above SDC commands:

```
sg_multicycle_path -from f1/q -to f2/in -through A2/out
-path_multiplier 2
```

Handling Mutually Exclusive Clocks

The cdc_false_path constraint is generated in the sgdc file when the -logically_exclusive option is given with the set_clock_groups command.

Consider the following SDC commands:

```
create_clock -name Clk1 -period 10.00 in1
create_clock -name Clk2 -period 15.00 in2
set_clock_group -logically_exclusive
-group{ Clk1 } -group { Clk2 }
```

The following cdc_false_path constraints are generated corresponding to the above SDC commands:

```
cdc_false_path -from Clk1 -to Clk2
cdc_false_path -from Clk2 -to Clk1
```

Handling Directional Clocks

The cdc_false_path constraint is generated in the sgdc file when the set_clock_uncertainity command is used but the clock is inferred as asynchronous.

Consider the following SDC commands:

```
create_clock -name Clk1 -period 10.00 in1
create_clock -name Clk2 -period 15.00 in2
set_clock_groups -asynchronous
-group { Clk1 } -group { Clk2 }
set_clock_uncertainty -from Clk1 -to Clk2
```

The following cdc_false_path constraint is generated corresponding to the above SDC commands:

```
cdc_false_path -from Clk1 -to Clk2
```

Translating set_clock_sense command

The sdc *set_clock_sense* SDC command is converted to the *clock_sense* SGDC command during the sdc2sgdc flow.

Currently, the *set_clock_sense* command is translated only when you specify the -*stop_propagation* option.

For example, consider the following SDC command:

```
set_clock_sense -stop_propagation -clocks Clk2  [get_pins
orinst1/Z ]
```

Following is the converted SGDC command:

```
clock_sense -pins "top.orinst1.Z" -tag  Clk2
```

Translating set_disable_timing command

The *set_disable_timing* SDC command is converted to the *disable_timing* SGDC command when:

- Values for the -from and -to fields of the command is specified
- Object list in the command is a lib cell module or lib cell instance

For example, consider the following SDC command:

```
set_disable_timing -from A -to Z [ get_lib_cells lsi_10k/OR2
]
```

Following is the converted SGDC constraint:

```
- disable_timing -name OR2 -from A -to Z
```

Translating set_mode command

The *set_mode* SDC command in translated into the *set_lib_timing_mode* SGDC command, when the type of *set_mode* is cell.

For example, consider the following SDC command:

```
set_mode -type cell mode1 U1
```

Following is the converted SGDC constraint:

```
set_lib_timing_mode -modes mode1 -instances U1
```

Saving the Generated SGDC Commands in a File

By default, the generated SGDC commands are saved in the `sdc2sgdc_`*`<mode>`*`.sgdc.`*`<processID>`* file under the *`<project>`*`/`*`<top>`*`/DesignRead/spyglass_reports/sdc2sgdc` directory.

To save the generated SGDC commands in a different file, specify the file name in the *Specify the file to save output of SDC-to-SGDC translation* field in the *Set Read Options* tab.

Alternatively, use the following command in the project file:

```
set_option sdc2sgdcfile <file-name>
```

If the specified file already exists, SpyGlass overwrites the existing file.

Specifying the Mode of an SDC File

You can specify the mode of an SDC file that you want to translate into SGDC in the *Specify the mode of the SDC file to be translated to SGDC* field in the *Set Read Options* tab.

Alternatively, use the following command in the project file:

```
set_option sdc2sgdc_mode <mode-name>
```

Consider the following example in which you specify two different modes in an SGDC file:

```
sdc_data -file one.sdc -mode A
sdc_data -file two.sdc -mode B
```

If you only want to translate `one.sdc` file into SGDC, specify the mode as `A`.

Understanding Different Flows for Using This Feature

You can use this feature in either of the following ways:

- *Generating SGDC Commands as a Part of Goal Run*
- *Generating SGDC Commands as a Part of Design Read*

Generating SGDC Commands as a Part of Goal Run

Use this flow if you want to generate SGDC commands on-the-fly as part of a goal run.

For example, you can generate SGDC commands on-the-fly as part of running SpyGlass CDC solution analysis with limited information, such as missing reset information and any other tool-specific constraints.

Consider an example in which you want to generate SGDC on the fly while running the clock-reset/verif_base/cdc_verif_base goal. In this case, specify the following commands in the project file:

```
read_file -type verilog test.v
read_file -type sgdc sample.sgdc


set_option top test
current_methodology $SPYGLASS_HOME/GuideWare/New_RTL
current_goal initial_rtl/cdc_prelim/cdc_verif_base


set_goal_option sdc2sgdcfile test.sgdc
set_goal_option sdc2sgdc yes
```

Generating SGDC Commands as a Part of Design Read

Use this flow if you want to generate SGDC during design read process.

The following is an example of a project file used to implement this flow:

```
read_file -type verilog test.v
read_file -type sgdc sample.sgdc


set_option top test
set_option designread_enable_synthesis
set_option sdc2sgdcfile test.sgdc
set_option sdc2sgdc yes
```

Support for Virtual Clocks in sdc2sgdc Flow

Usage of virtual clocks affects the following parameters:

- *create_clock*
- *set_input_delay*
- *set_output_delay*

create_clock

The `clock` constraint dumped in the SGDC file has the `-name` option whose value is the name of the object serving as a clock source.

However, in case of virtual clock where the source object is empty, SpyGlass populates this field with:

- A real clock found in a design that matches the virtual clock.
- Actual name of the clock (that is, the field specified with the `-name` option in the SDC file) if a real clock-mapping to virtual clock is not found.

set_input_delay

If `set_input_delay` has a virtual clock as its clock source, SpyGlass stores it in an un-commented form.

If this virtual clock is mapped to some real clock, the `input` constraint uses the corresponding real clock. Otherwise, it refers to the virtual clock name only.

set_output_delay

If set_output_delay has a virtual clock as its clock source, SpyGlass updates the SGDC file in following manner:

- If this virtual clock is mapped to some real clock, the output constraint uses the corresponding real clock and it is stored in an uncommented form in the SGDC file.
- Otherwise, virtual clock name itself is used and the output constraint is stored in a commented form in the SGDC file.

Virtual to Real Clock Mapping

The virtual to real clock mapping occurs in either of the following ways:

- By traversal and matching characteristics and then the name
- By name-matching alone

You need to decide the manner by using the mapVirtualClkByName and either of the mapSuffixList or mapPrefixList options.

Limitations

The SDC-to-SGDC functionality has the following limitations:

- If you have specified more than one -corner option for a single mode, SpyGlass translates SDC files corresponding to only the first -corner option.

  For example, consider the following case:

  ```
  sdc_data -file file1.sdc -mode func -corner Best
  sdc_data -file file2.sdc -mode func -corner Worst
  sdc_data -file file3.sdc -mode func -corner Best
  ```

  Here, more than one -corner option is specified for the same mode func. Therefore, the SDC files (file1.sdc and file3.sdc) corresponding to the first -corner option (Best) are translated by the set_option sdc2sgdc yes command.

- If multiple clocks are defined on the same source object in an SDC file by using the -add option, SpyGlass reports an error message. However, if you do not specify the -add option, translation occurs only for the last clock definition and a Warning message appears.

  You can change the severity from Error to Fatal by using the overloadrules option of the set_option command in the project file. The following is an example of using the overloadrules option:

  ```
  set_option overloadrules SDC2SGDC_STX01+severity=FATAL
  ```

- If you specify multiple set_input_delay parameters on the same object for different clocks by using the -add_delay option, translation for delays happen in the order as defined in the SDC file.
- If you define the clock and set_case_analysis commands on the same object in the SDC file, the SpyGlass DFT solution reports a FATAL violation to indicate the conflict.
- All commands specified on the port/pin objects are translated on the connected net in the SGDC file.

Importing Block-Level SGDC Commands to Chip-Level

While integrating various design blocks at chip-level, SpyGlass provides a capability for migrating SGDC files of blocks to the chip-level for performing chip-level analysis.

To migrate from block-level SGDC files to chip-level, perform the following tasks:

1. Create a migration file that contains `-import` command(s) for importing the specified block-level SGDC file(s) to the chip-level.

   For details, see *Creating a Migration File*.

2. Generate hierarchical SGDC files from block-level SGDC files that you can use for subsequent chip analysis.

   For details, see *Generating Hierarchical SGDC File*.

3. Validate the generated hierarchical SGDC file(s).

   For details, see *Validating the Generated Hierarchical SGDC File*.

Creating a Migration File

A migration file is an SGDC file that contains `-import` command specifications for importing block-level SGDC files to the chip-level.

For each block-level SGDC file to be imported at the chip-level, use the following `import` command:

```
current_design <module-name>
sgdc -import <block-name> <block-level-SGDC-file>
```

The details of the above specification are as follows:

- The above specification imports the specified block-level SGDC file with respect to the specified module, *<module-name>*.
- SpyGlass applies the block-level SGDC file to design units matching any of the above specifications.
- The *<block-name>* argument can be specified in any of the following formats:

module	entity	entity.architecture

NOTE: *You should choose the same specification used in the* `current_design` *command in the block-level SGDC file.*

- You can provide the above specification multiple times for different blocks in the same chip-level SGDC file.
- You can specify an absolute or a relative path for the block-level SGDC file. If you specify a relative path, ensure that it is accessible from the current run directory.

Constraints Migrated From Block-Level to Chip-Level

While migrating block-level SGDC files to chip-level, the following constraints are migrated:

activity	always_on_buffer	always_on_cell
always_on_pin	aon_buffered_signals	antenna_cell
assertion_signal	cell_hookup	cell_pin_info
cell_tie_class	cdc_false_path	clock
domain_outputs	domain_inputs	domain_signal
ignore_crossing	input_isocell	levelshifter
multivt_lib	non_pd_inputcells	pg_pins_naming
pin_voltage	power_down	power_down_sequence
power_state	power_switch	ram_instance
ram_switch	retention_cell	retention_instance
special_cell	supply	switchoff_wrapper_instance
qualifier	voltage_domain	

NOTE: *Block-level* `cdc_false_path` *constraint is migrated to chip-level only if you specify clocks in the arguments of the block-level* `cdc_false_path` *constraint.*

Generating Hierarchical SGDC File

This task generates hierarchical SGDC file(s) that contain block-level SGDC commands with respect to the chip-level. You can use these files during subsequent chip analysis. For details, see *Generated Hierarchical SGDC File(s)*.

Generated Hierarchical SGDC File(s)

SpyGlass stores the generated hierarchical SGDC files in the gen_hiersgdc/spyglass_reports/imported_sgdc directory. The name of each generated file is in the following format:

<module-name>_<block-name>_<block-level-SGDC-file>

The generated output file contains two sections, as discussed below:

- The first section displays successfully imported commands.
- The second section contains commands that need user input or review.

  Generally, port names specified in the block command require user input.

Top-Level SGDC File

SpyGlass also generates a top-level SGDC file that contains:

- All the migrated block-level SGDC files.
- Migrated clock commands that are common in two or more block-level SGDC output files.

  These commands are in the commented form in the corresponding block-level SGDC output files. The name of the output file is *<module-name>*. sgdc. In subsequent chip-level analysis, you should specify the generated top-level SGDC file instead of migrated block-level SGDC files.

NOTE: *SpyGlass ignores any other SGDC command in the chip-level SGDC file in this generation step. However, the tool does not ignore the* `set_case_analysis` *and* `assume_path` *SGDC commands, if specified at the chip-level, and uses them during migration of block-level SGDC commands.*

Validating the Generated Hierarchical SGDC File

To validate the generated SGDC file, you do not have to specify the migrated SGDC files generated in the *Generating Hierarchical SGDC File* step.

Currently, SpyGlass performs the following checks in the validation mode:

- Clock validation: Checks whether the top-level clocks reach the block clock ports
- Clock domain validation: Checks whether the top-level clocks connected to the block clock ports comply with the block-level domain specifications

In case of any discrepancy present in the above mentioned checks, SpyGlass reports appropriate violations. You then need to perform the following steps:

1. Correct the chip-level specifications for clock.
2. Re-run the validation task with the new specifications.
3. If no violation is reported, use the final SGDC file in the subsequent chip-level analysis.

NOTE: *The* `sgdc -import` *command should be used only in the generation mode, validation mode, and SpyGlass CDC abstraction flow.*

Implementing Scoping in SGDC Commands

SpyGlass allows a scoping mechanism in SGDC commands by default. The scoping mechanism is implemented by using the `::` operator.

Consider the following example:

```
current_design <du-name>
my_command -name M::i1.i2.net -value 0
```

In the above example, `M::` specifies the scoping mechanism, which means to find all instances of module `M` in:

- All instances of design unit, *`<du-name>`*, if that design unit is not a top-level design unit.
- Design unit, *`<du-name>`*, if it is a top-level design unit.

Then, the value `0` is applied on net `i1.i2.net` in all these instances.

In the above example, scoping within `M::` notation is known as *local scoping*, and scoping in the non top `current_design` is called *global scoping*.

Note the following points:

- `M` can be in `m`, `e`, and `e.a` format, where `m`, `e`, and `a` refer to module, entity, and architecture name, respectively.
- Path followed by `::` should be relative to `M` and should not have `M` preceded to it.
- The resultant value after translation should be a valid value for the concerned field. For example, `M::port` specification will change to hierarchical terminal. A fatal violation is reported if the concerned field does not take hierarchical terminal as a valid value.
- Scoping specifications is not supported in `-noenv` constraints. A fatal violation is reported for such specifications.

Scoping When Design is at Top-Level

Consider the following command:

```
current_design <du-name>
my_command -name M::i1.i2.net -value 0
```

If the design is a top-level design unit, SpyGlass searches all instances of M in that design unit and replaces them with M::i1.i2.net specification. For example, consider a case in which there are two instances of M, namely top.mi1 and top.I1.mi2. Then, M::i1.i2.net is replaced by top.mi1.i1.i2.net and top.I1.mi2.i1.i2.net.

However, if any of the resultant design objects do not exist in the design, SpyGlass reports a fatal violation. This fatal violation is reported after synthesis because these checks run on NOM.

Depending upon the type of the -name field, the following two cases may arise:

- If the -name field is a key/scalar field

  In this case, the command is split into two commands, as given below:

  ```
  current_design top
  my_command -name top.mi1.i1.i2.net -value 0
  my_command -name top.I1.mi2.i1.i2.net -value 0
  ```

  Here, if any one of the two paths does not exist, the corresponding SGDC command is deleted. In this case, if the port, M::in1, of M is referred, it is converted to hierarchical terminal i.e., top.mi1.in1 and top.I1.mi2.in1.

- If the -name field is a list type of field

  In this case, the two paths are added to the same command, as shown below:

  ```
  my_command -name top.mi1.i1.i2.net top.I1.mi2.i1.i2.net -
  value 0
  ```

  However, there will also be an option to have multiple commands created, one for each instance in case of list type field as well.

Wildcard Support at Top-Level

Consider the wildcard specification, as shown in the following example:

```
my_command -name "M*::i1.net" -value 0
```

In the above example, `M*` is first matched with all modules in the hierarchy under *<du-name>*. Consider that it matches with two module names, `M1` (Verilog module) and `m2` (VHDL module). Then, `-name "M*::i1.net"` specification will get changed to `-name "M1::i1.net m2::i1.net"` specification. Also consider that design has objects matching to "M1::i1.net but not to "`m2::i1.net`". In this case, SpyGlass doesn't report a violation because there is at least one matching design object (`M1::i1.net`). Fatal violation is reported only if no matching design object exist in the design. Also note that this fatal violation is reported after synthesis because these checks run on NOM. Further processing continues based on the cases discussed above (depending upon whether the `-name` is key/scalar field or list type of field).

Consider another wildcard specification, as given below:

```
my_command -name "M::*.net" -value 0
```

In the above specification, consider that there are two instances of `M`, namely `top.mi1` and `top.I1.mi2`. Then, the wildcard specification will first expand to `-name "top.mi1.*.net top.I1.mi2.*.net"`. Now, the following cases may arise:

- If `-name` is registered with `--wildcard` or `--wildcard_inline_expand`, then '*' will match to one level of hierarchy. For example, it will match to `top.mi1.L3I1.net` and `top.I1.mi2.L4I1.net` and NOT to `top.mi1.L3I1.L4I1.net`.
- If `-name` is registered with `--wildcard_support` or `--wildcard_support_full`, the wildcard expression is left as it is for the product to handle it.

NOTE: *The options,* `--wildcard_support` *and* `--wildcard_support_full` *mean that '*' is expected to match multiple levels of hierarchy (i.e., it should also match* `top.mi1.L3I1.L4I1.net`*.) Kernel does not provide this support and, therefore, such specifications are currently handled by the products themselves.*

Conflict Resolution at Top-Level

If a value generated due to scoping conflicts with an explicit value specified by the user, the value generated by scoping is deleted. This provides you the flexibility to override one or more specifications generated through scoping. Consider the following example:

```
current_design top
set_case_analysis M::in -value 0
set_case_analysis top.mi1.in -value 1
```

Here, consider that the module, M, is instantiated ten times in top (i.e., top.mi1, top.mi2, …, top.mi10). In this case, the set_case_analysis constraint will not allow duplicate specifications in the -name field. Therefore, one of the ten generated commands (one for top.mi1.in) is deleted. However, if two values, each generated by scoping, are duplicates then SpyGlass flags a fatal violation.

However, SpyGlass will report the following two specifications as duplicate specifications:

```
current_design top
set_case_analysis -name M::in -value 0
set_case_analysis -name M::in -value 1
```

Scoping When Design is at the Block-Level

Scoping specifications at block-level undergo through hierarchical translations. The difference between scoping specifications at top-level and scoping specifications at block-level is that only those instances of scoped module are considered that are instantiated in the instances of the block module. For example:

```
current_design block
set_case_analysis -name M::in -value 0
```

In the above example, consider that the block is instantiated twice in top, (i.e., top.bi1 and top.bi2). Also consider that module, M, is instantiated thrice in the block (i.e., block.mi1, block.mi2, and block.mi3). Then, the above specification will generate six commands (2*3) at the top-level, as shown below:

```
current_design top
set_case_analysis -name top.bi1.mi1.in -value 0
set_case_analysis -name top.bi1.mi2.in -value 0
set_case_analysis -name top.bi1.mi3.in -value 0
set_case_analysis -name top.bi2.mi1.in -value 0
set_case_analysis -name top.bi2.mi2.in -value 0
set_case_analysis -name top.bi2.mi3.in -value 0
```

However, if M is instantiated in the top module (that is, outside the block), that instance will not be considered. Further, if any of the design objects in these generated commands do not exist, SpyGlass flags a fatal violation.

Wildcard Support at Block-Level

Consider the wildcard specification, as given below:

```
current_design block
set_case_analysis -name M*::in -value 0
```

Here, M* is first replaced by its respective matches from within the module, block. These scoping specifications then undergo through hierarchical translations.

Consider that it matches with "top.b.m1::in", "top.b.m2::in" (where, block is instantiated as 'top.b' in top and module, M is instantiated twice in the block, that is, block.m1 and block.m2). Also, consider that only "top.b.m1::in" is present in the design. SpyGlass doesn't report a violation because there is at least one matching design object (top.b.m1::in). Fatal violation is reported only if no matching design object exist in the design. Also note that this fatal violation is reported after synthesis because these checks run on NOM.

Conflict Resolution at Block-Level

Conflict resolution at block-level is implemented depending upon the following cases:

- Consider the following example:

```
current_design block
set_case_analysis -name M::in -value 0
set_case_analysis -name block.mi1.in -value 1
```

Here, the second command will take precedence over the command generated through scoping.

- Consider the following example:

```
current_design block
set_case_analysis -name M::in -value 0
set_case_analysis -name block.mi1.in -value 1
current_design top
set_case_analysis -name top.bi1.mi1.in -value 0
```

Here, the top-level specification takes precedence, and no error is reported for duplicate specification.

- Consider the following example:

```
current_design block
set_case_analysis -name M::in -value 0
set_case_analysis -name block.mi1.in -value 1
current_design top
set_case_analysis -name block::mi1.in -value 1
set_case_analysis -name top.bi1.mi1.in -value 0
```

Here, all the block-level commands are ignored. For example, `mi1` of `M` inside block is ignored.

Handling SystemVerilog Objects in SGDC

SpyGlass handles different SystemVerilog objects in different ways.

Handling SystemVerilog Interface Port/Terminal

Consider the following example:

```
interface intf (output z_intf, input a_intf);
endinterface

module topper(output z,input a);
  intf i1(z,a);
  top T1(i1);
endmodule


module top(intf inst);
  mid M2(inst.z_intf,inst.a_intf);
endmodule
```

In the above example, if the `z_intf` or `a_intf` port of the `top` module is to be referred in SGDC, it should be as follows:

```
current_design topper
test_mode -name "topper.T1.inst_z_intf" -value 1
test_mode -name "topper.T1.inst_a_intf" -value 1
```

Here, the interface port is named as
"*<interface-instance-name>_<interface-port-name>*"

Handling SystemVerilog Interface Containing a Modport

Consider the following example, in which a SystemVerilog interface referred in SGDC contains a modport:

```
interface intf;
  wire z_intf,a_intf;
  modport M1 (output z_intf, input a_intf);
  modport M2 (output a_intf, input z_intf);
endinterface

module top;
  intf i1();
  intf i2();
  top_low T1(i1.M1,i2.M2);
endmodule

module top_low(intf.M1 inst1, intf.M2 inst2);
  wire inst1_z_intf, inst2_a_intf;
  assign inst1_z_intf = inst2_a_intf;
  mid M1(inst1.z_intf,inst1.a_intf);
  mid M2(inst2.a_intf,inst2.z_intf);
endmodule

module mid(output z, input a);
  assign z = ~a;
endmodule
```

In the above example, if module `top_low`, ports `z_intf` (of modport `M1`) and `a_intf` (of modport `M2`) are to be referred in SGDC, it should be as follows:

```
current_design top
test_mode -name "top.T1.inst1_z_intf" -value 1
test_mode -name "top.T1.inst2_a_intf" -value 0
```

Here, the modport interface port is named as "*<interface-instance-name>_<interface-port-name>*".

Handling SV Structure or Union

For an SV structure or union, or for cases in which a port or a net is declared with these complex types, you should specify SGDC constraints as shown in the following example:

```
typedef struct packed {
  logic [2:0] opcode;
  logic [1:0] rtid;
  } hc_cmdlane_t;

module top(input hc_cmdlane_t in1,in2,input [1:0] in3
[1:0], input clk1, clk2, output out1, out2);
  FD2 a_fd2_1(in2.opcode[0], clk1, , t);
  //FD2 a_fd2_2(in1.opcode[1], clk1, , out1);
  FD2 a_fd2_3(in2.opcode[2], clk1, , out1);
  FD2 a_fd2_5(in2.rtid[0], clk1, , out1);
  FD2 a_fd2_6(in2.rtid[1], clk1, , out1);
  FD2 a_fde_6(in3[0][0], clk1, , out1);
  FD2 a_fde_7(in3[0][1], clk1, , out1);
  FD2 a_fde_8(in3[1][0], clk1, , out1);
  FD2 a_fde_9(in3[1][1], clk1, , out1);
endmodule
```

```
current_design top
abstract_port -module top -ports q%\in2.rtid [1:0]% -clock
 clk1 abstract_port -module top -ports q%\in2.rtid % -clock
 clk1 abstract_port -module top -ports q%\in2.opcode [0]%
 -clock clk1
```

In the above example, a struct port or a net is named as per the following naming convention:

"\<struct-instance-name>.<struct_port_name> "

Handling for-generate Constructs

Consider the following example:

```
module test(output z,input a);
  parameter p1 = 3, p2 = 4;
  generate
    genvar c,i;
    for(c =0;c <2;c++)
    begin
      wire w1;
      for(i = 0;i<2;i++)
      begin
        mid m1(z,a,w1);
      end
    end
  endgenerate
endmodule
module mid(output z, input a,input w1);
  assign z = ~a;
endmodule
```

In the above example, specify SGDC constraints for the `m1` instance and `w1` wire inside `for-generate`, as given below:

```
current_design test
test_mode -name "\genblk1[0].genblk1[0].m1 .a"
-value 1

test_mode -name "\genblk1[0].genblk1[1].m1 .a"
-value 1

test_mode -name "\genblk1[1].genblk1[0].m1 .a"
-value 1

test_mode -name "\genblk1[1].genblk1[1].m1 .a"
-value 1

test_mode -name "\genblk1[0].w1 " -value 1
```

You can specify any SGDC constraint by using naming conventions for objects, as shown in the above example for the `testmode` constraint.

Consider another example in which the `set_case_analysis` constraint is used on the SystemVerilog design containing named `for-generate` block:

```
module test(clk,enable,in1,out1);
  input clk,enable;
  input [3:0]in1;
  output[3:0]out1;
  generate genvar i;
    for (i=0; i<4; i=i+1) begin:extend
      mod1
      ins(.CLK(clk),.E(enable),.IN1(in1[i]),.OUT1(out1[i]));
    end
  endgenerate
endmodule

module mod1(CLK,E,IN1,OUT1);
  input CLK,E;
  input IN1;
  output OUT1;
  reg OUT1;
  always @(posedge CLK)
    if(E) OUT1 <= IN1;
endmodule
```

For the above example, the generated netlist contains instances with the following names:

```
mod1   \extend[0].ins  (.CLK(clk), .E(enable), .IN1(in1[0]),
.OUT1(out1[0]));

mod1   \extend[1].ins  (.CLK(clk), .E(enable), .IN1(in1[1]),
.OUT1(out1[1]));

mod1   \extend[2].ins  (.CLK(clk), .E(enable), .IN1(in1[2]),
.OUT1(out1[2]));

mod1   \extend[3].ins  (.CLK(clk), .E(enable), .IN1(in1[3]),
.OUT1(out1[3]));
```

You can refer the above names in the `set_case_analysis` constraint (or any other SGDC constraint), as shown below:

```
set_case_analysis -name "test.\extend[1].ins .E" -value 0
```

Basically, you can refer an object (say *<obj>*) as part of the `generate` block by using the following convention:

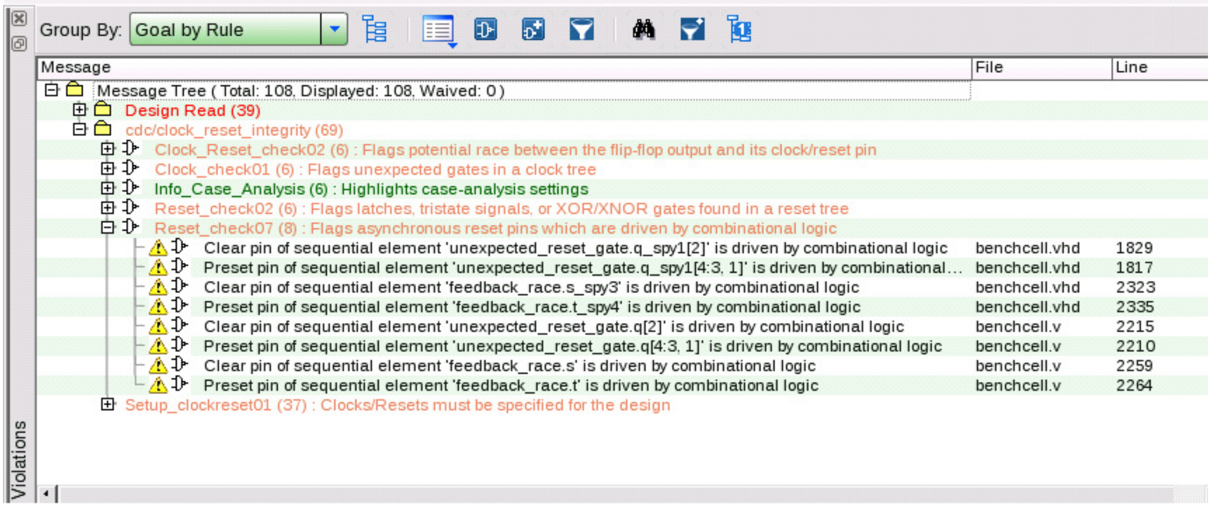
```
"\<generate_block1_label>[block1_index].<generate_block2_lab
el>[block2_index]...<generate_blockN_label>[blockN_index].
<obj> "
```

If the `blockX` is unnamed, *<generate_blockX_label>* is considered as `genblk`*<block_number>*. You can refer to the schematic for the complete name given to the unnamed block including *<block_number>*.

Further, for any of the SystemVerilog scenarios mentioned above or otherwise, you can always check design object names appearing in the schematic and refer those names in the SGDC constraints.

Working with SpyGlass Messages

SpyGlass displays all violation messages of the currently loaded goal in the *Results* pane, as shown in the following figure:



Group By: Goal by Rule

Message	File	Line
Message Tree (Total: 108, Displayed: 108, Waived: 0)		
Design Read (39)		
cdc/clock_reset_integrity (69)		
Clock_Reset_check02 (6) : Flags potential race between the flip-flop output and its clock/reset pin		
Clock_check01 (6) : Flags unexpected gates in a clock tree		
Info_Case_Analysis (6) : Highlights case-analysis settings		
Reset_check02 (6) : Flags latches, tristate signals, or XOR/XNOR gates found in a reset tree		
Reset_check07 (8) : Flags asynchronous reset pins which are driven by combinational logic		
Clear pin of sequential element 'unexpected_reset_gate.q_spy1[2]' is driven by combinational logic	benchcell.vhd	1829
Preset pin of sequential element 'unexpected_reset_gate.q_spy1[4:3, 1]' is driven by combinational...	benchcell.vhd	1817
Clear pin of sequential element 'feedback_race.s_spy3' is driven by combinational logic	benchcell.vhd	2323
Preset pin of sequential element 'feedback_race.t_spy4' is driven by combinational logic	benchcell.vhd	2335
Clear pin of sequential element 'unexpected_reset_gate.q[2]' is driven by combinational logic	benchcell.v	2215
Preset pin of sequential element 'unexpected_reset_gate.q[4:3, 1]' is driven by combinational logic	benchcell.v	2210
Clear pin of sequential element 'feedback_race.s' is driven by combinational logic	benchcell.v	2259
Preset pin of sequential element 'feedback_race.t' is driven by combinational logic	benchcell.v	2264
Setup_clockreset01 (37) : Clocks/Resets must be specified for the design		

FIGURE 189. Violation Messages for Currently Loaded Goals

To view the violation messages of another goal, load that goal by selecting it from the drop-down list in the Analyze Results tab, as shown in the following figure:

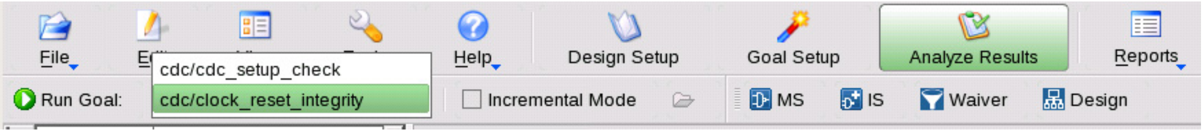


FIGURE 190. Run Goal

When you double-click on a violation message, the following actions occur:

- A source file containing the corresponding issue appears in a separate tab in the *Source* section. Then name of this tab is the same as the name of the source file.
- The violating line appears in a different color in the code present in that source file.
- The corresponding violating portion is highlighted in the schematic.

Next time, when you select a different message, all existing selections and probes are removed.

Working with Multiple Messages

The first selected message is known as the *main message*. All subsequently selected messages are considered as *auxiliary messages*.

After selecting a main message, select auxiliary messages by double-clicking them, keeping the <*Ctrl*> key pressed.

Effects of Selected Messages in the Schematic

When you double-click on a message and open the *Incremental Schematic* window, the schematic shows the portions of the design resulting in a violation.

When you select auxiliary messages, the schematic changes based on the type of auxiliary message selected (static or non-static).

Selecting Static Auxiliary Messages

Such messages load design attributes on the already loaded objects in the schematic, as shown in the following figure:

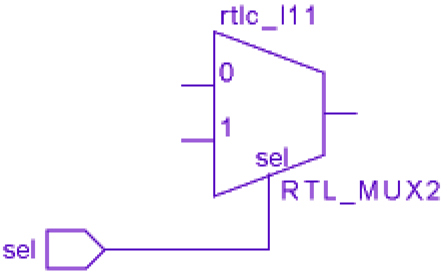
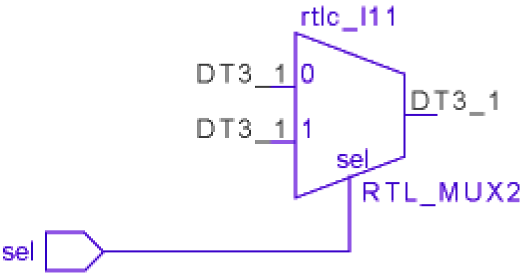
Portions of a design loaded by double-clicking a main message.	
Attributes loaded on the already loaded design objects when a static auxiliary message is double-clicked.	 rtlc_I11 DT3_1 0 DT3_1 1 DT3_1 sel RTL_MUX2 sel

FIGURE 191. Static Auxiliary Messages

Selecting Non-Static Auxiliary Messages

Such messages load the violating portions of a design over the already loaded design in the schematic, as shown in the following figure:

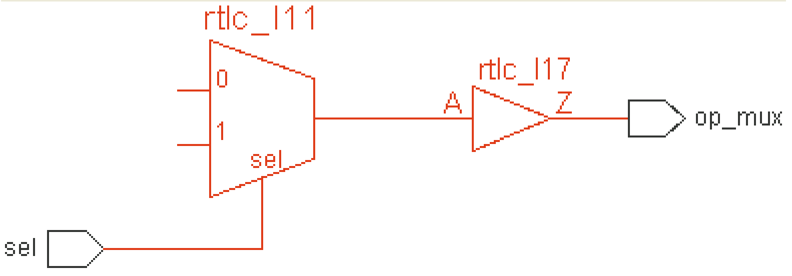
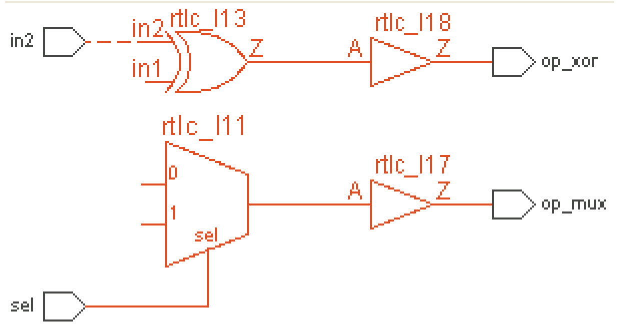
Portions of a design loaded by double-clicking a main message.	 rtlc_I11 0 1 sel rtlc_I17 A Z op_mux sel
Violating design portions reported by an auxiliary message loaded over the design already loaded design.	 in2 in2 rtlc_I13 in1 Z rtlc_I18 A Z op_xor rtlc_I11 0 1 sel rtlc_I17 A Z op_mux sel

FIGURE 192. Non-Static Auxiliary Messages

Selecting Auxiliary Messages without Selecting a Main Message

To select one or more auxiliary messages without first selecting the main message, double-click the messages with the *<Ctrl>* key pressed.

You can select up to 32 auxiliary messages. If you select more than 32 messages, SpyGlass displays the *Warning* dialog that prompts you to deselect some of the selected messages.

Messages Affecting Multiple Source Lines/Files

If an issue reported by the selected message is related with multiple lines in a source code and these lines are in multiple source files, SpyGlass displays each source file in a separate tab in the *Source* section.

Each tab name in the *Source* section indicates the name of the source file.

Multiple Lines Affected in the Same Source File

In this case, the tab name displays the source file name and the number of affected lines in the source file. For example, test.v (3).

Initially, the first affected line appears. Use the *<Shift>+<N>* (next line) and *<Shift>+<P>* (previous line) key combinations to move among the affected source lines.

Multiple Lines Affected in Different Source Files

In this case, one tab appears for each affected source file. The tab name displays the source file name and the number of affected lines in the source file.

Initially, the first affected line appears. In this case, you can do the following:

- Use the *<Shift>+<N>* and *<Shift>+<P>* key combinations to move among the affected source lines in the same source file.
- Use the *F6* and *F7* keys to move between the tabs.

Multiple Messages Selected

If you have selected a main message that highlights only a single source line, no new tab appears. However, if you select an auxiliary message (<Ctrl>+double-click), a new tab is displayed for the main message and another tab is displayed for the auxiliary message in the *Source* section even if both messages are in the same source file. Use the *F6* and *F7* keys to move between the tabs.

If you select multiple messages, a separate tab appears for each auxiliary message in the *Source* section. You can view the source file of the selected message (out of multiple selected messages) in the *Source* section tab using the *Jump To Focus* button in the *Legend* window. You can also go to the related auxiliary Source Window tab for that message by pressing <Ctrl>+<G>.

Limiting the Number of Messages Generated

You may want to limit messages generated during SpyGlass analysis for the following reasons:

- A rule may report a large number of messages of the same type because of which the total count of reported messages is huge.

  In such cases, you can limit the number of reported messages saved in the violation database for each rule. For details, see *Limiting the Number of Messages Reported for a Rule*.

- A particular rule may not indicate a serious problem.

  In such cases, you can waive that message so that it does not appear in the list of reported messages in the *Results* pane. For details, see *Waiving Messages*.

The following types of messages are not added in the message count of the SpyGlass results summary report:

- Messages that exceed the specified limit for one or more rules
- Waived messages

SpyGlass indicates the number of such messages by reporting the following message:

```
Suppressed 20 messages (5 waived)
```

In the above example, 20 messages are suppressed due to waiver or rule over limit settings. Out of these suppressed messages, 5 messages were suppressed due to waiver.

Limiting the Number of Messages Reported for a Rule

To limit the total number of messages for a rule, perform any of the following actions:

- Specify an upper limit of messages per rule by using the following command in the project file:

  ```
  set_option lvpr <value>
  ```

- Specify the maximum number of messages to be reported per rule in the *Maximum Messages Per Rule* field under the *Design Read Options* tab.

Limiting the number of messages for a rule is useful when errors are difficult to identify because one or more rules may produce multiple messages of the same type.

Waiving Messages

If a particular message does not indicate a serious problem, you can waive that message.

Waiving messages suppresses the display of messages based on your requirements at different stages of design analysis. Such messages are removed from the reported message list.

You can waive a message in any of the following ways:

- Through the *Waiver Editor* window.

  For details, see *Using the Waiver Editor Window*.
- Through the *Results* pane.

  For details, see *Using the Results Pane to Waive Messages*.
- Through a project file.

  For details, see *Waiving Messages through a Project File*.
- Through the `waive` constraint in a *Waiver File* (.awl file).

  For details, see *Waiving Messages by Using the waive Constraint*.
- Through SpyGlass pragmas in source code.

  For details, see *Waiving Messages by Using SpyGlass Pragmas*.

Using the `waive` constraint is the preferred method because this approach does not affect the source files. The waivers are written in a separate file and can be used with modified source files as long as the modifications do not invalidate the design constraints. However, you should use embedded SpyGlass waiver pragmas if you need to waive messages at any level below the design unit level in the source file.

Waiver File

A waiver file (.awl file) is used to waive messages reported after SpyGlass runs.

It is an SGDC-format file that contains specifications of the `waive` constraint that is used to waive specific types of messages. For information on the this constraint, see *Waiving Messages by Using the waive Constraint*.

You can specify waiver files to SpyGlass through GUI, project file, or batch. This is described in the next sections.

Creating a Waiver File

You can create a waiver file in any of the following ways:

- By using an editor program

  Create a file in an editor, write `waive` constraint specifications in that file, and save that file as a waiver file (.awl file).

- By using the *Waiver Editor* window

  Specify details in the appropriate fields in the *Waiver Editor* window. Based on the details specified, SpyGlass generates corresponding `waive` constraints in a waiver file.

  You can use this window to create or modify waiver files.

  For details on this window, see *The Waiver Editor Window*.

Creating Goal-Based Waiver

Goal-based waivers enables you to waive goal-specific messages. You can create different waiver files for different goals as per the requirement. You can also extend the goal-specific waivers to support scenarios.

You can control the scope of waiver files by individually selecting the waiver files and make them applicable globally or specific to a goal.

For example, consider the following commands:

```
new_project test
read_file -type waiver project.swl
current_goal G1 -alltop
```

```
read_file -type waiver goal.swl
```

For .awl files, user should specify type as awl, as shown below:

```
read_file -type awl <file_name>.awl
```

In the above example, the waiver file project.swl has the project scope, Therefore, it is applicable to both project and the G1 goal. While the file goal.swl is applicable to only the G1 goal

Setting Default Waiver File

Use set_option command to create default waiver file at the project level.

Use set_goal_option command to create default waiver file at the goal level.

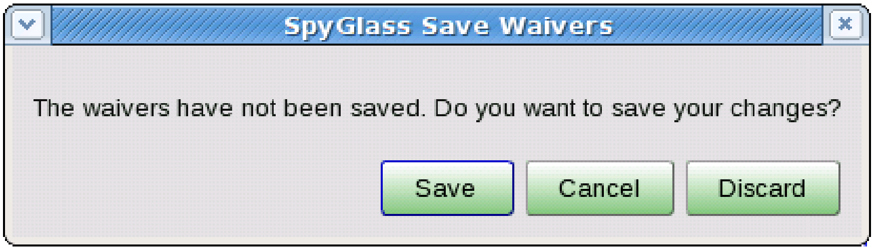
Consider the following example:

```
new_project new
set_option default_waiver_file project.awl
waive -rule r1

current_goal G1 -alltop

waive -rule r2
set_goal_option default_waiver_file goal.awl
waive -rule r3


save_project

---------------------------------
$ cat  project.awl
waive -rule {  {r1}  }
waive -rule {  {r2}  }


$ cat goal.awl
waive -rule {  {r3}  }
```

In the above example, default waive file - project.swl is created at project - level and default waiver file - goal.swl is created at the goal level.

Handling Unsaved Changes in Waiver Files

After editing a waiver file in the *Waiver Editor* window, if you close this window without saving the changes and run goals again, the following dialog appears:



SpyGlass Save Waivers

The waivers have not been saved. Do you want to save your changes?

FIGURE 193. Unsaved Files

SpyGlass determines the unsaved status of a waiver file in either of the following ways:

- By comparing contents of the current Object Model (OM) with the latest file copy on the disk
- By checking if the waiver file has changed after loading in the *Waiver Editor* window. SpyGlass checks this by referring to the timestamp of the waiver file along with the unsaved status (denoted by the + sign) appearing adjacent to that waiver file in the *Waiver Editor* window.

In this case:

- Click the *Reload* button to reload the *Waiver File* from the disk in the *Waiver Editor* window and continue SpyGlass analysis.
- Click the *Continue* button to:
  - ❒ Apply waivers present in the *Waiver File* on the disk (applicable in batch mode).
  - ❒ Apply waivers present in the *Waiver Editor* window (applicable in GUI mode).
- Click the *Save* button to save the *Waiver File* to the disk and continue SpyGlass analysis.

Including a Waiver File in Another Waiver File

Use the include directive to include a waiver file in another waiver file. The include directive is used in the following format:

```
include <file-name>
```

In the following example, the waiver_include.swl file is included in the waiver.swl:

```
// Contents of waiver.swl                  // Contents of
waiver_include.swl
include waiver_include.swl                waive -rule "XYZ"
waive -rule "ABC"
```

In the above case, when you specify the waiver.swl file during SpyGlass analysis, SpyGlass expands the contents of this file to the following:

```
waive -rule "XYZ"
waive -rule "ABC"
```

Effects of Waiving Messages

Waiving message(s) affect the following:

- SpyGlass results summary that is generated at the end of a SpyGlass analysis run

  Count of waived messages is not added in message counts in the SpyGlass results summary. Instead, the following message appears to indicate the number of such messages:

  ```
  Suppressed 20 messages (5 waived)
  ```

  In the above example, 20 messages are suppressed due to message waiver or rule message over limit settings. Out of these suppressed messages, 5 messages were suppressed due to message waiver.

- Violation database

  SpyGlass modifies the rule severity of waived messages to `waiver[original-severity]` in the violation database. For example, consider the following violation message:

  ```
  W127@@@@Warning@@rules_w127_1.v@@31@@1@@5@@Delay value
  ```

```
should not contain X or Z
```

If you waive the above message, and the issue reported by the corresponding rule message is present in a design, the following message is written in the violation database:

```
W127@@@@waiver[Warning]@@rules_w127_1.v@@31@@1@@5@@Delay
value should not contain X or Z
```

SpyGlass shows the severity of the waived message as `waiver[original-severity]`.

- Waiver report

  The *Waiver* report lists all waived messages. You can view this report from the *Reports* menu option.

Auto-Migration of Waivers

When rule messages change between SpyGlass releases, waiver files of previous release may become incompatible for use in the current release. To ensure compatibility, SpyGlass automatically upgrades the old message to the new rule message in the same run.

To avoid migration of waivers, use the `set_option disable_auto_migrate_waiver` command. If this option is provided, then the waiver messages are not migrated to the current release. You can use this option, if the waivers have been already migrated using -gen_compat_waiver flow, and are up-to-date with respect to the current release.

NOTE: *You can also migrate the waivers separately to new version using option -gen_compat_waiver.*

Waiving Messages through GUI

In GUI, you can waive messages based on the following mechanisms:

- **Specifying Message-Based Filters**: You can specify message-based filters in one of the following ways:
  - ❒ By invoking the *Waiver Editor* window.

    See *Using the Waiver Editor Window*.

- ❐ By using right-click options in the *Results* pane.

  See *Using the Results Pane to Waive Messages*.

- ■ **Specifying Object-Based Filters**: Object-based waivers allow you to filter corresponding messages based on the design objects.

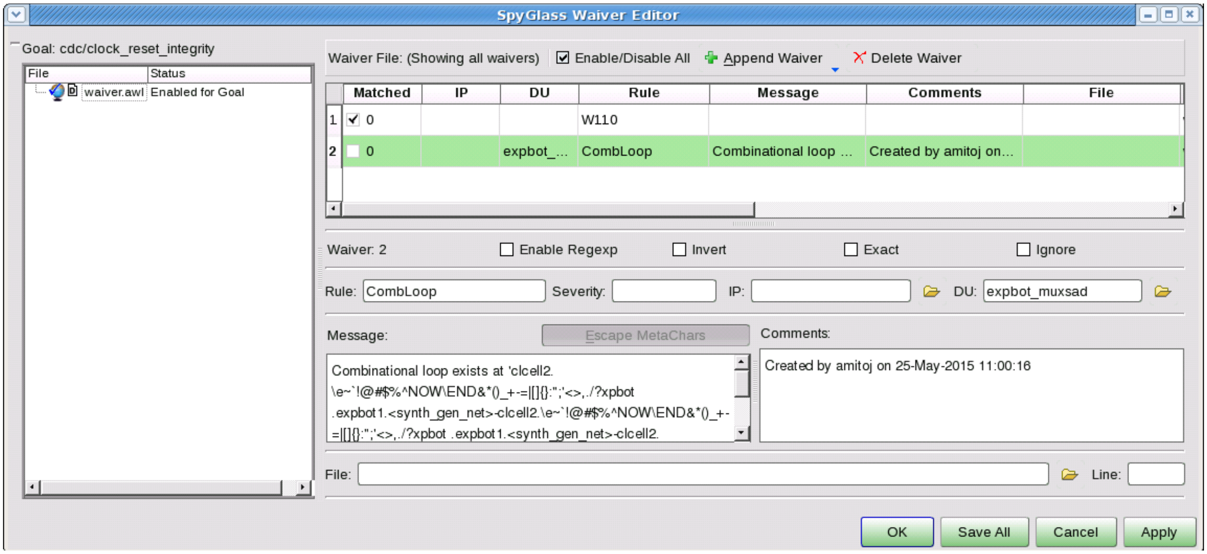
  You can apply object-based waivers using one of the following ways:

  - ❐ *Using the Message Tree/Waiver Tree to Waive Messages*
  - ❐ *Using the Schematic Window to Waive Messages*

Using the Waiver Editor Window

The *Waiver Editor* window enables you to specify waiver expressions to waive different types of violation messages for the currently loaded goal. These waiver expressions are in the form of `waive` constraints that are saved in a *Waiver File* (.awl). These waiver files appear in the left-most section of the Waiver Editor window.

The following figure shows the *Waiver Editor* window:



File	Status
waiver.awl	Enabled for Goal

Waiver File: (Showing all waivers) ☑ Enable/Disable All

	Matched	IP	DU	Rule	Message	Comments	File
1	☑ 0			W110			
2	□ 0		expbot_...	CombLoop	Combinational loop ...	Created by amitoj on...	

Waiver: 2 □ Enable Regexp □ Invert □ Exact □ Ignore

Rule: CombLoop Severity: IP: DU: expbot_muxsad

Message:

```
Combinational loop exists at 'clcell2.
\e~`!@#$%^NOW\END&*()_+-=|[]{}:";'<>,./?xpbot
.expbot1.<synth_gen_net>-clcell2.\e~`!@#$%^NOW\END&*()_+-
=|[]{}:";'<>,./?xpbot .expbot1.<synth_gen_net>-clcell2.
```

Comments: Created by amitoj on 25-May-2015 11:00:16

File: Line:

FIGURE 194. Waiver Editor Window

In the above window, you can add new or existing waiver files for the currently loaded goal.

To open the above window, perform any of the following actions:

- Select the *Tools -> Waiver Editor* menu option.
- Select the *Waiver* option (or icon) from the *Results* pane.
- Right-click on the rule header in the Msg Tree page, and select the **Waive All Messages of Select Rule(s)** option from the shortcut menu.

For details on the above window, see *The Waiver Editor Window.*

Using the Results Pane to Waive Messages

To waive violations through the *Results* pane, right-click on a message or a node displaying a rule title and select the appropriate options from the shortcut menu.

You can also select multiple messages by selecting the required messages with the <*Shift*> key pressed.

When you select an option from the shortcut menu, the following actions occur in the tool:

- The *Waivers Editor* window appears, in which new rows are added for the selected messages. These rows contain waiver expressions in the form of `waive` constraints that are saved in a *Waiver File*.
- Waived messages appear in the *Waiver Tree* pane in the *Violations* pane. For details on this page, see *Waiver Tree*.

Waiving Selected Messages

Right-click on a message, and select the *Waive Selected Messages* option from the shortcut menu.

NOTE: *This option appears only if the Enable advanced waiver creation preference option is not set in the Preferences dialog.*

Waiving Specific Type of Messages

Right-click on a message, and select the *Waive* option from the shortcut menu. A sub-menu appears displaying the following options:

Option in the Sub-Menu	Description
Selected Message(s)	Select this option to waive all the selected messages.
This Exact Message	Select this option to waive the first selected message.
All Messages in This File	Select this option to waive all messages of the file corresponding to the selected message.
All Messages Of This Module	Select this option to waive all messages of the module corresponding to the selected message.
All Messages with This Severity	Select this option to waive all messages of the severity of the selected message.
Custom	Generates waiver command using fields set through **Set custom waiver options** and displays the **Waiver Editor** window.
Set custom waiver options	Enables you to set the fields which should be added, if available, while generating waiver through **Custom waiver** menu item.
Set custom regexp sub-fields	Enables you to select the fields to be used for regex matching.

NOTE: *This option appears only if the Enable advanced waiver creation preference option is set in the Preferences dialog.*

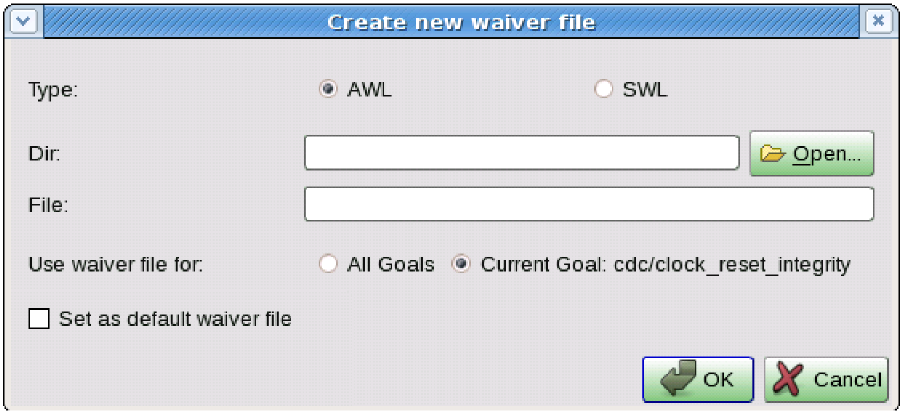
Waiving All Messages of a Rule

Right-click on the node displaying the rule title (that is the node under which messages of a particular rule are reported), and select the *Waive All Messages Of Selected Rule(s)* option from the shortcut menu.

Setting a Default Waiver File

Right-click on a message and select the *Select Default Waiver File* option from the shortcut menu. A sub-menu appears that lists all the current waiver files. Select the required *Waiver File* from this list.

However, if you want to create a new *Waiver File* that you want to set as the default waiver file, select the *Create New Waiver File* option from the sub-menu. The *Create new waiver file* dialog, as shown in the following figure:



Create new waiver file

Type: ● AWL ○ SWL

Dir:

File:

Use waiver file for: ○ All Goals ● Current Goal: cdc/clock_reset_integrity

□ Set as default waiver file

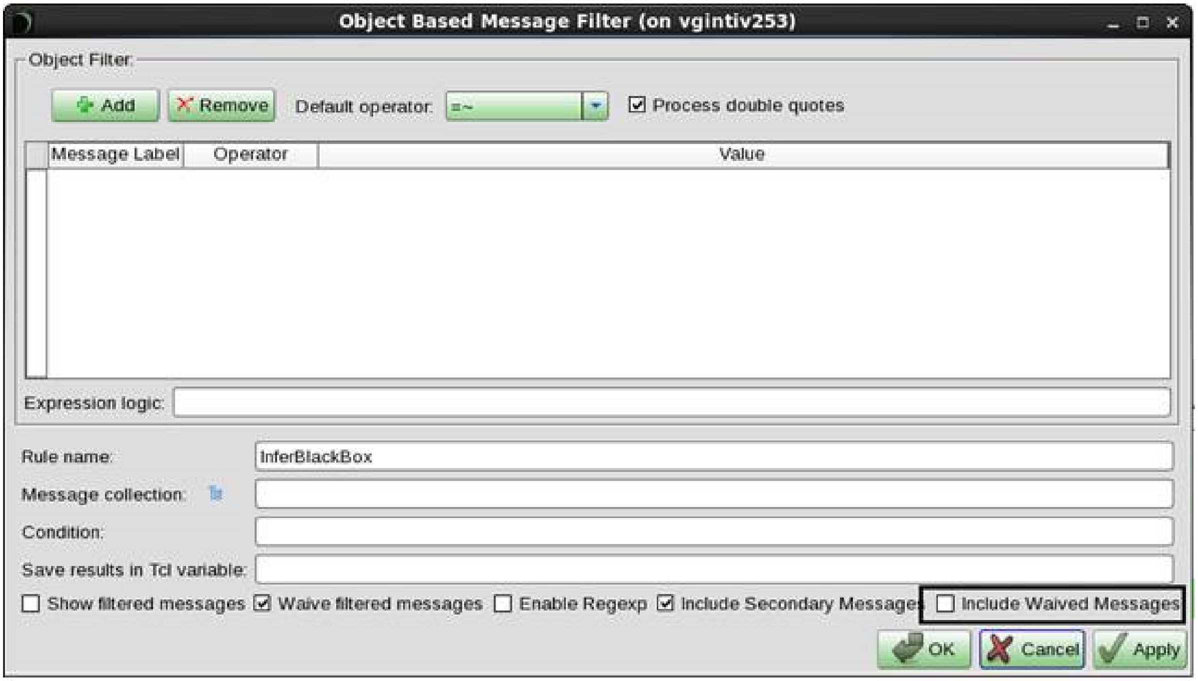
FIGURE 195. Create New Waiver File

In the above dialog, specify the name of the file and select the *Set as default waiver file* option.

Using the Message Tree/Waiver Tree to Waive Messages

You can filter and waive messages for a message tree/waiver tree using the Object Based Message Filter window. To do so, click the Object-Based Message Search icon ().

The following Object-Based Message Filter window is displayed (*Figure 196*):



Default operator: =~

☑ Process double quotes

Rule name: InferBlackBox

Message collection:

Condition:

Save results in Tcl variable:

□ Show filtered messages ☑ Waive filtered messages □ Enable Regexp ☑ Include Secondary Messages □ Include Waived Messages

FIGURE 196. Object-Based Message Filter

To add waivers, perform the following steps from the Object Based Message Filter window:

1. Click Add.

   A row is added to the Object Filters pane, which is used to generate object filter expression.

2. Specify the options described in *Table 15*.

TABLE 15 Object Based Message Filter Options

Option	Description
Operator	The default value is "=~". If you change the operator, the new operator is used for new additions to the object filter table. This operator is consistent across Spyglass sessions.
Message Label	Specifies the selected message label. You can chose from the available message labels. The message labels list is created based on the current violation set.
Operator	Specifies the logical operator. By default, operator is `&&` for all the message labels.
Value	Specifies the value for the message label.
Expression logic	Is generated automatically as you fill up the rows. However, you can edit this field.
Rule Name	Specifies the rule name for which you want to waive the messages. Specify either the *Rule Name* or the *Message Collection* argument. It is not allowed to specify both the arguments simultaneously.
Message Collection	Specifies the message collection name, which you generate in the sg_shell. You can specify both variable names and Tcl commands as inputs, which will generate a message collection. For example: ● $var ● [filter_messages -of_rule [get_rules LPSVM04A]] Specify either the *Rule Name* or the *Message Collection* argument. It is not allowed to specify both the arguments simultaneously.
Condition	Specifies the regular expression that is specified in the following format: `<procedure_name> arguments` **Note**: If you specify this option, you need not specify message label, operator, and expression logic.

Option	Description
Save results in Tcl variable	Allows you to store the waived or filtered message collection in a Tcl variable for retrieving later.
Show filtered messages	Select this option to show filtered messages in the Message Tree window.
Waive filtered messages	Select this option to waive filtered messages and display them in the Waiver Tree.
Enable Regexp	Select this option to enable regular expression match in other fields.
Include Secondary Messages	Select this option to consider secondary messages for the message filtering or waiving.
Include Waived Messages	Select this option to include waived messages for message filtering. This option is off by default.

3. Click **OK** to view the messages in the Message Tree.
4. Click **Apply** to apply the waiver.

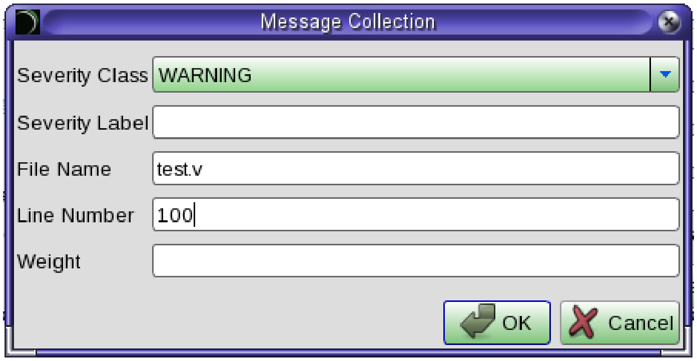
   The **Filtered Messages** window is displayed, which would contain the waived/filtered messages.
5. Click **Cancel** to cancel the waiver information.

If the Object Based Message Filter dialog box is already open, double-clicking on a violation message in Message Tree automatically updates the Object Based Message Filter dialog box with the available message labels and values attached to the message.

Filtering Messages Based on Message Attributes

To filter the messages based on the message attribute, select the ( ) button next to the Message Collection label (*Figure 196*).

This displays the Message Collection dialog box as shown in *Figure 197*:



Message Collection

Severity Class WARNING

Severity Label

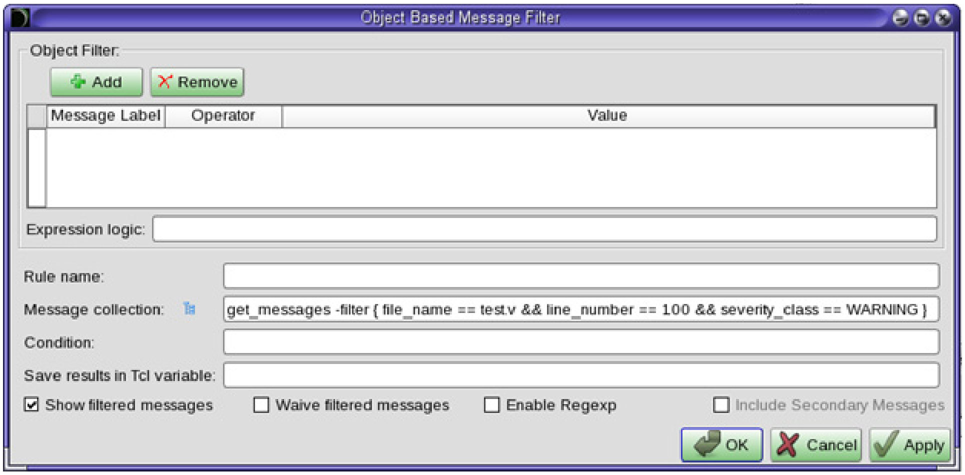
File Name test.v

Line Number 100

Weight

FIGURE 197. Message Attribute

Specify the details as required and press OK. This adds the appropriate *get_messages* command to the Message collection field in the Object Based Message Filter dialog box as shown in *Figure 198*:



Object Based Message Filter

Object Filter:

Message Label	Operator	Value

Expression logic:

Rule name:

Message collection: get_messages -filter { file_name == test.v && line_number == 100 && severity_class == WARNING }

Condition:

Save results in Tcl variable:

☑ Show filtered messages ☐ Waive filtered messages ☐ Enable Regexp ☐ Include Secondary Messages

FIGURE 198. Message Attributes Added

Using the Schematic Window to Waive Messages

You can specify waivers for a design object in the Modular Schematic/ Incremental Schematic window.

To waive messages for the selected label in the Modular Schematic/ Incremental Schematic, perform the following steps:

1. Right-click on a design object in the Modular/Incremental Schematic.
2. Click Waive/Show related messages from the right-click menu.

3. The message label collection is displayed as shown in *Figure 199*:

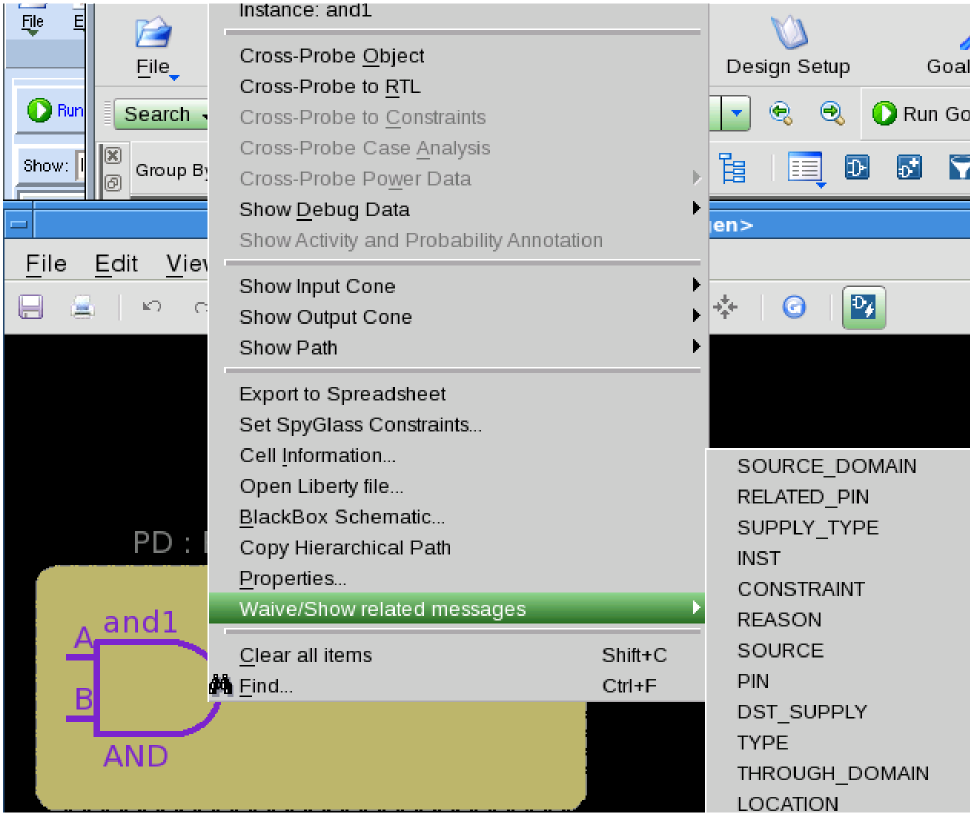
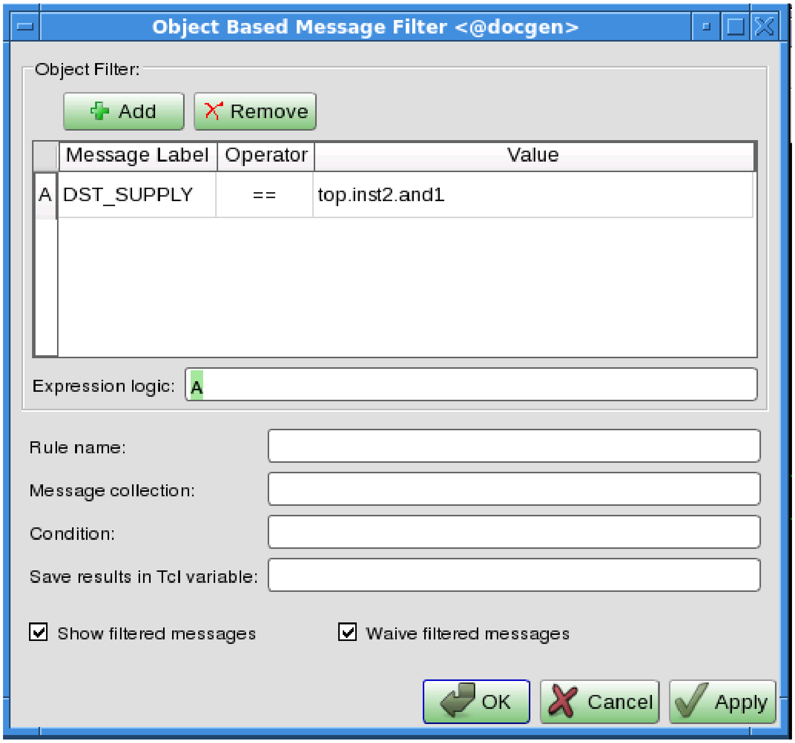


FIGURE 199. Waive/Show Related Messages Menu

4. Select a message label from the available labels.
5. The Object-Based Message Filter window is displayed with the selected label-information pre-filled.

6. Specify the information specified in *Table 15* to specify waiver information and waive the messages accordingly.



Object Based Message Filter <@docgen>

Object Filter:

	Message Label	Operator	Value
A	DST_SUPPLY	==	top.inst2.and1

Expression logic: A

Rule name:

Message collection:

Condition:

Save results in Tcl variable:

☑ Show filtered messages ☑ Waive filtered messages

FIGURE 200. Object-Based Message Filter

7. Perform the steps 2 to 5 described in the *Waiver Tree* section to fill in the waiver

Waiving Messages through a Project File

Use the following command in a project file to specify a *Waiver File* to waive messages:

```
read_file -type waiver <waiver-file-name>
```

Waiving Messages by Using the waive Constraint

The `waive` constraint enables you to waive messages by various categories, such as by source files, by design units, by rules, etc.

You can specify this constraint in a file that is of the same format as an SGDC file. For details on creating and using an SGDC file, see the *Working with SpyGlass Design Constraints* chapter. Then, you can supply the file containing `waive` constraint specifications using the `-waiver` command in a waiver file.

Syntax of the waive Constraint

The following is the syntax for specifying the `waive` constraint:

```
waive [ -ignore ] [ -regexp ] [ -disable ]
   [ -file <file-list> ]
   [ -file_line <file-line> ]
   [ -file_lineblock <file-sline-eline> ]
   [ -du <du-list> | <logical-lib-name> ]
   [ -ip <ip-list> | <logical-lib-name> ]
   [ -rule | -rules <rule-list> | <keyword> ]
   [ -except <rule-list> | <keyword> ]
   [ -msg <message> ]
   [ -severity <label> ]
   [ -weight <weight> ]
   [ -weight_range <weight-value> <weight-value> ]
   [ -import <block_name> <block-waive-file> ]
   [ -comment <comment> ]

<keyword> ::=
   ALL | ALL_INFO | ALL_WRN | ALL_ELAB
 | ALL_SYNTHERR | ALL_SYNTHWRN
```

For more information on the Tcl-based usage of the *waive* command, refer to the *waive* section of the *SpyGlass Tcl Shell Interface User Guide*.

Argument Details of the waive Constraint

The following table contains the details of various arguments of the `waive` constraint:

Argument	Description
-file_lineblock	Use this argument to waive messages for a block of lines in a source file. <file-sline-eline> is a space-separated tuple of source file name, start line number, and end line number in the following format: <file-name> <line1> <line2> This means that a message reported in the file <file-name> between the line numbers <line1> and <line2> is considered for the waive constraint. It is required that <line2> is greater than or equal to <line1>. **Note**: Use multiple -file_line/-file_lineblock arguments, each with one argument. This method is not recommended if the source code is expected to change. In such cases, use either pragma-based waivers (see *Waiving Messages by Using SpyGlass Pragmas*) or other waiver arguments described later in this table.
-file_line	Use this argument to waive rule messages for a particular line of a source file. <file-line> is a space-separated pair of source file name and line number in the following format: <file-name> <line-num> **Note**: Use multiple -file_line/-file_lineblock arguments, each with one argument. This method is not recommended if the source code is expected to change. In such cases, use either pragma-based waivers (see *Waiving Messages by Using SpyGlass Pragmas*) or other waiver arguments described later in this table.
-file	Use this argument to waive all messages for the specified files. You can specify a space-separated list of source file names (<file-list>) in this argument.

Argument	Description
-du and -ip	Use the -du argument to waive the rule messages for the specified design units or all design units in the specified library. This argument is particularly useful for RTL coding style checks where the reported message is clearly localized within a design unit. Use the -ip argument to waive rule messages for the specified design units (IP blocks), including the ones that are below its hierarchy or all design units in the specified IP library. <du-list> refers to a space-separated list of logical library name <logical-lib-name> of a precompiled Verilog/VHDL library or design unit names, such as: ● Module names <module-name> for Verilog ● Entity names in the format <entity-name> for entity and all its architectures ● <entity-name>.<arch-name> for the entity and the specified architecture ● Package names <pkg-name> ● Configuration names <config-name> (for VHDL) **NOTE:** By default, only the waived message count is reported in the IP/Legacy Waiver Report section of the Waiver report when the -ip argument of the waive constraint has been specified. Use the -report_ip_waiver option to have the actual waived messages also printed. Note the following points: ● You are required to specify the -du or -ip arguments if no other argument of the waive constraint is specified. ● If you want SpyGlass to consider the schematic highlight information of a violation to waive violations on design units, use the following command: set_option use_du_sch_hier yes ● If a module is instantiated in multiple IPs but you do not provide the waive -ip specification for each of these IPs, SpyGlass does not waive violations on such module instances when you specify the following command: set_option use_du_sch_hier yes By default, SpyGlass waives violations on only those module instances that are present in the IPs specified by the waive -ip specification.
-rule/-rules	Use these arguments to waive messages of the specified rules, rule groups, or products or by rule type keywords. <rule-list> refers to a space-separated list of rule names, rule group names, or product mnemonics. This argument is case-sensitive.

Argument	Description
-except	Use this argument to exclude the specified rules, rule groups, or products or by rule type keywords from the scope of the waive constraint. <rule-list> refers to a space-separated list of rule names, rule group names, or product mnemonics. **NOTE:** If you specify the same rule to the -rule/-rules and -except arguments, preference is given to the -except argument.
-msg	Use this argument specify a message to be waived. The `waivers_translate_generate_names` option should be enabled while using a non-escaped `generate block` name or an `instance array` name in the -msg field.
-severity	Use this argument to waive messages of the specified severity class or severity label. <label> refers to the actual severity-label or a SpyGlass severity class. If a rule is overloaded (customized), overloaded values are considered by this argument.
-weight	Use this argument to waive the messages of the rules with the specified weight. <weight> refers to the actual rule weight value.
-weight_range	Use this argument to waive the messages of the rules with the weight within the specified range (both range values inclusive). <weight-value> refers to a positive integer number.
-comment	Use this argument to add waive constraint comment as a single line text string enclosed in double quotes. This comment appears in the Waiver report and the sign_off report. <comment> refers to a valid string.
-import	Use this argument to enable importing the waiver file (.swl) specified at the block-level to be used at the chip-level. For more details, see *Support for Hierarchical Waivers*. <block_name> refers to the name of the block in the top-level chip, and <block-waive-file> refers to the name of the waiver file applied to the specified block <block_name>.

Argument	Description
-ignore	This argument causes SpyGlass to list only the waived message count in the Adjustments Waiver Report section of the Waiver report and not the actual waived message(s). Use the -report_adjustment_waiver option to override the -ignore argument so that the actual waived messages are also printed.
-regexp	Use this argument to allow use of regular expressions in many arguments. For more details, see *Using Regular Expressions and Wildcard Characters*.
-disable	Use this argument to disable the waive constraint.

Details of the waive Constraint

The details of the `waive` constraint are described below:

- You should specify the `current_design` keyword with the `waive` constraint.
- Files specified by using the *-file*, *-file_line*, or *-file_lineblock* arguments are searched by using both the specified file base-name and the specified path.
- You must use at least one of the arguments from one of the following argument groups:

  ```
  Group 1: -file/-file_line/-file_lineblock, -du, -ip
  Group 2: -rule/-rules, -msg, -severity, -except
  ```

- When you use more than one argument from `Group 1`, a message is waived if any one of the argument conditions is met. When you supply more than one argument from `Group 2`, a message is waived only if all argument conditions are met. If you supply arguments from both `Group 1` and `Group 2`, a message is waived only if any one of the `Group 1` argument conditions is met **and** all `Group 2` argument conditions are met.
- The *-file* argument is ignored if the file specified by this argument is also specified in the *-file_line* or *-file_lineblock* argument.
- If you specify a design unit name by using the `-du` argument, the scope of the `waive` constraint is the specified design unit and does not include the design units instantiated in the specified design unit.

- If you specify a design unit name by using the `-ip` argument, the scope of the `waive` constraint is the specified design unit and its complete hierarchy.
- If you specify the *-except* argument but do not specify the *-rule/-rules* argument, it is assumed that the *-rule/-rules* argument has been specified with the `ALL` keyword.
- Specify the SpyGlass severity classes as uppercase names and the severity labels as mixed-case or lowercase names with the *-severity* argument.
- To waive SpyGlass built-in error, warning, and info messages, use the following keywords:

Use	**To waive**
ALL_INFO	All the analyzer (language) info messages
ALL_WRN	All the analyzer (language) warning messages
ALL_ELAB	All elaboration messages
ALL_SYNTHERR	All synthesis error messages
ALL_SYNTHWRN	All synthesis warning messages
ALL	All of the above plus all rule messages

NOTE: *You can waive all types of built-in rules except the built-in STX error rules because these rules are mandatory checks.*

NOTE: *If you specify the* `ALL` *keyword, then all (built-in and rule) messages will be waived for files and/or design units for which it is specified.*

NOTE: *You cannot waive product rules of severity class FATAL.*

Please note that all keywords are case-sensitive. You can also provide a combination of these keywords to waive messages of more than one type.

- For `waive` constraint, all the occurrences of multiple consecutive spaces (spaces or tabs) between message words are reduced to just one space. Therefore, do not adopt such messaging. In addition, SpyGlass does not waive messages that extend to two or more lines.
- While using the `waive` constraint to waive messages, you must enclose the exact message in double quotes, `q/.../`, or `m/.../` depending

on whether you want the string to be interpreted as a wildcard, literally, or as a regular expression respectively.

- To get the exact message string for the `-msg` argument of the `waive` constraint, run SpyGlass Analysis that will generate that message. Then, open the Violation Database file in any ASCII text editor and copy the exact message string. Specify the whole message string, including leading and trailing spaces, in `q/<message string>/` where / is the start-end delimiter and should not be present in the message string. If / is part of the message string, then it is suggested to use some other delimiter as explained in the *Handling Special Names* section.
- You can also use double quotes to specify the exact message string for the `-msg` argument of the waive constraint. However, following three characters have a special meaning inside double quotes:
  - ❒ \ (escape character, used in escaped name)
  - ❒ $ (used for variable expansion)
  - ❒ " (double quote character)

  If any of the above characters appear in your message string, either use q/<message string>/ or escape these characters to treat them as literal characters inside the double quotes. In rest of the cases, it is equivalent whether we put the exact message string in q/.../ or double quotes.
- The additional difference between the usage of q/…/ and double quotes is the handling of wildcard characters. Anything specified inside q/../ is treated literally, including any wildcard characters such as, *, and ?. If you want to specify a wildcard pattern for your message string in the -msg argument of the waive constraint, then use double quotes to specify it.
- The q/…/ specification is also used when -regexp option is used in the waive constraint. To turn off regular expression matching in fields of the waive constraints, enclose the field values in the q/…/ specification. The field values are then treated as a literal string. For details, see *Selective Use of Regular Expressions* section.
- The `waive` constraint is not applied, if any of the source files, HDL files, SDC files, and library files, have syntax errors during parsing.
- If the variable part of the message changes for a SpyGlass version, the waiver applied on that message won't be applicable for the next SpyGlass version.

NOTE: *The* `waive` *constraint with the* `-du` *argument does not work on design units in VHDL libraries.*

Examples of Using the waive Constraint

Here are some examples of using the `waive` constraint:

- The following command waives the messages of the W146 and W336 rules for the test.vhd file:

  ```
  waive -file test.vhd -rules W146 W336
  ```

- The following command waives all messages for the test.v file:

  ```
  waive -file test.v -rules ALL
  ```

- The following command waives all analyzer (language) warning messages for the test.vhd file:

  ```
  waive -file test.vhd -rules ALL_WRN
  ```

- The following command waives the rule messages of W154 and W146 for the module named `upper`:

  ```
  waive -du upper -rules W154 W146
  ```

- The following command waives all messages for the architectures named `rtl1` and `rtl2` of entity `flop`:

  ```
  waive -du flop.rtl1 flop.rtl2 -rules ALL
  ```

- The following command waives all synthesis warning messages for the module named `upper`:

  ```
  waive -du upper -rules ALL_SYNTHWRN
  ```

- The following command waives the specified message for the test1.v and test2.v files:

  ```
  waive -file test1.v test2.v -msg “Blocking \
  assignment used inside a sequential block”
  ```

- The following waive constraint directive waives the specified message for the design unit named `upper`:

  ```
  waive -du upper -msg “Explicit named association \
  is recommended in instance references”
  ```

- The following waive constraint directive waives messages of all rules with severity label `Warning` for the test.vhd file:

```
waive -file test.vhd -severity Warning
```

- The following waive constraint directive waives messages of all rules with severity label `Info` for the architecture named `RTL` for entity named `a123`:

```
waive -du a123.rtl -severity Info
```

- The following waive constraint directive waives messages of all rules with severity label `Warning` for the test.v file and design unit named `upper`:

```
waive -file test.v -du upper -severity Warning
```

Using Regular Expressions and Wildcard Characters

The `waive` constraint supports both regular expressions and wildcard characters.

These regular expressions are similar to the C-type regular expressions. A C-type regular expression is a pattern that you specify in the pattern matching tools, such as *Lex* and *Flex*. These patterns are identical to the pattern specified in the UNIX commands, such as `egrep`.

A regular expression or a wildcard character can occur in the values of all arguments of the `waive` constraint except rules names specified with the `-rule`/`-rules` argument and `-except` argument, severity labels specified with the `-severity` argument, the line numbers specified in the `-file_line` and the `-file_lineblock` arguments, and the strings specified with the `-comment` argument.

Understanding Regular Expressions

A regular expression is a way of writing code based on a well-defined pattern that is widely used in the software industry.

NOTE: *You can use regular expressions only for the waive constraint.*

Using regular expressions in the `waive` constraint is suitable for tasks, such as waiving similar types of messages.

To use regular expressions in the `waive` constraint, specify the *-regexp* argument of this constraint. If you do not specify this argument, any value within quotes is expanded in the wildcard mode.

NOTE: *For information on the wildcard mode, see Wildcard Mode.*

Processing Regular Expressions

SpyGlass uses the `regcomp` and `regexec` commands (C/C++) of your operating system to process regular expressions. By using these commands, SpyGlass first compiles a regular expression as an extended regular expression. If no match occurs, it compiles the regular expression as a basic regular expression. Refer to the `regexec` man page for details of regular expression support. You can also refer to the `regrep` man page for more details.

Regular expression support is compliant to the support offered by Perl and Tcl.

Character Class in Regular Expressions

In regular expressions, a character class is a set of characters that meets certain criteria. It is defined by using square brackets, as shown in the following example.

```
[A-Z0-9]
```

In the above example, the character class represents all uppercase letters and numbers from 0 to 9.

Specifying the -rule/-except Argument

SpyGlass does not support regular expressions with the *-rule/-rules* and *-except* arguments of the `waive` constraint because regular expressions are not required for these arguments. You can specify a list of rule names as well as rule group names with the `-rule/-except` argument. If you want to specify a collection of rules with similar names, just specify the name of the rule group to which these rules belong.

Understanding Wildcard Characters

Asterisk (`*`) and question mark (`?`) are the supported wildcard characters, where `*` matches any string and `?` matches any one character.

Use wildcard characters more often than regular expressions because wildcard characters are easy to use and operate.

Using Regular Expressions or Wildcard Characters in the -msg Argument of the waive Constraint

While using regular expression or wildcard characters in the *-msg* argument, use a complete message string with regular expressions or wildcard characters for the part that is changing across messages.

Consider an example in which you want to waive the following messages:

```
Incompatible width for port 'srout'(width 9 in module 'sr') on
instance 'd8'(terminal width 1), [Hierarchy:srtop]
```

```
Incompatible width for port 'srout'(width 9 in module 'sr') on
instance 'd16_1'(terminal width 1), [Hierarchy:srtop]
```

```
Incompatible width for port 'srout'(width 9 in module 'sr') on
instance 'd16_2'(terminal width 1), [Hierarchy:srtop]
```

You can waive the above messages by specifying the following command:

```
waive -msg "Incompatible width for port 'srout'(width 9 in
module 'sr') on instance '.*'" -regexp
```

However, the recommended way is to specify a complete message string with regular expressions and wildcard characters in the *-msg* argument, as shown in the following command:

```
waive -msg "Incompatible width for port 'srout'(width 9 in
module 'sr') on instance '.*'(terminal width 1),
\[Hierarchy:srtop\]" -regexp
```

Using Special Characters in Regular Expressions

Regular expressions consist of eleven basic symbols, called meta characters, which have a defined meaning and purpose, as described in the following table.

TABLE 16 Meta characters used in regular expressions

Meta character	Symbol	Description
Minus sign	-	Specifies a range of characters when used inside a character class (enclosed in square brackets, such as [a-z])
Asterisk	*	Matches zero or more occurrences of the literal character or meta character it follows.

TABLE 16 Meta characters used in regular expressions

Question mark	?	Matches zero or one occurrence of the literal character or meta character it follows.
Square brackets	[ ]	Indicates a character class.
Period	.	Represents one instance of a class of characters that includes all characters.
Caret	^	Forces the matched text to begin at the first character in the text being searched. Also negates a character or character class when used after an opening bracket in a character class.
Dollar sign	$	Forces the expression to match text through the very last character of the text being searched.
OR bar	|	Acts as a Boolean OR, which allows the combination of two expressions or alternatives in a single expression.
Backslash	\	Escapes out certain meta characters so that they are treated as literal values.
Round brackets	()	Specifies open parenthesis, that is, `(`, and close parenthesis, that is, `)` , which are used to group (or bind) parts of search expression together.

NOTE: *When inside a character class (square brackets), you do not need to escape out the *, ?, and $ characters.*

Using Literal Characters in Regular Expressions

In addition to the special meta characters, regular expressions contain literal characters, which are regular characters, such as letters, numbers, and symbols. These characters are not interpreted as special characters.

If you want a literal value of a special character, precede it with a backslash (\). For example, to use a dollar sign as a literal character, specify `\$`. You can also enclose the symbol in brackets to indicate that it should be treated as a literal. For example, `[$]`.

SpyGlass interprets meta characters and literal characters differently.

When it encounters a meta character in a waive constraint, SpyGlass assumes the meta character to be a syntax component of the regular expression and evaluates the regular expression accordingly. When it encounters a backslash, SpyGlass treats the subsequent character as text rather than a syntactic component.

The following table shows how to use literal characters in regular expressions.

Characters	Description	Example
A-Z a-z	Alpha characters ABCDEFGHIJKLMNOPQRSTUVWXYZ Alpha characters abcdefghijklmnopqrstuvwxyz	
0-9	Numeric characters 0123456789	
-	To use a literal minus sign within a regular expression, the symbol must be escaped with a backslash character or contained inside brackets alone. If a literal minus sign occurs in a character class with other characters, it must be escaped with a backslash.	\- or [-] or [A-Z\-]
*	To use a literal asterisk within a regular expression, the asterisk symbol must be escaped with a backslash character or contained inside brackets.	* or [*]
?	To use a literal question mark within a regular expression, the question mark symbol must be escaped with a backslash character or contained inside brackets.	\? or [?]
[ ]	To use a literal left square bracket or right square bracket within a regular expression, the square bracket symbol must be escaped with a backslash character.	\[and \]
^	To use a literal caret symbol within a regular expression, the caret symbol must be escaped with a backslash character or contained inside brackets.	\^ or [^]
$	To use a literal dollar sign within a regular expression, the dollar sign symbol must be escaped with a backslash character or contained inside brackets.	\$ or [$]

<table>
<tr><td>|</td><td>To use a literal vertical bar symbol within a regular expression, the vertical bar symbol must be escaped with a backslash character or contained inside brackets.</td><td>\| or [|]</td></tr>
<tr><td>.</td><td>To use a literal period punctuation mark within a regular expression, the period punctuation mark must be escaped with a backslash character or contained inside brackets.</td><td>\. or [.]</td></tr>
<tr><td>\</td><td>To use a literal backslash symbol within a regular expression, the backslash symbol must be escaped with a backslash character or contained inside brackets.</td><td>\\ or [\]</td></tr>
<tr><td>~`!@#%&_
={};:' ",<></td><td>The remaining symbol characters are treated as literal characters and do not need to be escaped within regular expressions.</td><td></td></tr>
<tr><td>(space)</td><td>In regular expressions, the space character is ignored except when it appears inside a character class or is preceded by a backslash.</td><td>[ ] or \</td></tr>
<tr><td>{}</td><td>To use a literal left or right curly bracket within a regular expression, the curly bracket symbol must be contained inside brackets.</td><td>[{] [}]</td></tr>
<tr><td>()</td><td>To use a literal left round bracket or right round bracket within a regular expression, the round bracket symbol must be escaped or contained inside square brackets.</td><td>\(and \) or [(] and [)]</td></tr>
</table>

Selective Use of Regular Expressions

When you specify the *-regexp* argument with the `waive` constraint, SpyGlass processes values specified with all applicable arguments as regular expressions.

You can also specify values of a selected set of arguments that should be processed as regular expressions. To specify such values, use the `m/.../` (process as regular expression) format and `q/.../` (process as literal string) format. Then, you do not need to specify the `-regexp` argument.

Conversely, you may want to process the values of some arguments as normal values even when the `-regexp` argument is specified. In such cases, use the `q/.../` (process as literal string) format for values of such arguments.

The following table summarizes the effect of using the `m/.../` and `q/.../` formats with or without the `-regexp` argument:

-regexp option	m/.../ specified	q/.../ specified	both m/.../ and q/.../ not specified
Specified	Redundant (process all applicable as regular expression)	Process as literal string	Process all applicable as regular expression
Not specified	Process as regular expression	Process as literal string	Wildcard

Example 1 - Using the m/.../ Format

Consider the following command:

```
waive -file m/test/ -severity Info
```

The above command waives all messages of the `Info` severity in all files whose names contain string `test` (test.v, test.vhd, mytest.v, etc.).

Please note that value `test` does not have any regular expression character. Since it is matched using the substring matching method (`m/.../`), all file names containing `test` are matched.

Example 2 - Using the q/.../ Format

Consider the following command:

```
waive -regexp -file q/test.v/ -msg ".*[ ' \"]clk[ ' \"].*"
```

The above command waives messages that match the regular expression specified by the *-msg* argument for the test.v file only. This match occurs only if test.v is a valid file name. If the file name is invalid, all file names ending with test.v are matched.

Note that if you had not used the `q/.../` format in the value of the *-file* argument, the scope would have been all files whose names contain the string `test`*`<char>`*`v` where *`<char>`* is any one character. For example, it would have matched test.v, test.vhdl, test@v, test#vhdl, and so on.

Example 3 - Specifying Absolute or Relative File Names

Now, you may want to refer to a file using its absolute or relative path name. In this case, the value of the *-file* argument will contain the slash (/) character and you will not be able to use the slash (/) character as a delimiter. In this case, use another supported delimiter, as shown in the following example:

```
waive -file m@^\.\./src/.*test@ -severity Info
```

The above specification waives all messages of the `Info` severity in all files whose names contain the string `test` (test.v, test.vhd, mytest.v, my.test, etc.) that are located in the ../src directory with respect to the current working directory.

You must escape the dot characters in the path name. This is because the dot character is also a regular expression character.

However, note the following points:

- If you are only using wildcard characters, do not specify the *-regexp* argument because it will unnecessarily search for a match in all the mentioned arguments as a part of the waiver.
- If you want to apply a regular expression on a specific argument, such as *-msg*), use `m/.../` instead of the *-regexp* argument (due to the same reason mentioned in above point).

Handling Special Names

You cannot use the `m/.../` and `q/.../` formats with values that contain a forward slash (/) because it is used as a delimiter by these formats.

In this case, use one of the following delimiters:

!	@	%	^	&	*	;	~	?	<	>	+	=	|

For example, use the `m@...@` format or `q>...>` format provided the same delimiter is used as the starting delimiter and the ending delimiter (that is, `m<...>` is not allowed.) and the delimiter is not present in the value being enclosed (that is, q@name@top@ is not allowed). The m format and q format can use different delimiters.

Examples of Using Regular Expressions in the waive Constraint

Here are some examples:

- *Regular Expressions with -file and -du arguments*
- *Regular Expressions with the -severity argument*
- *Regular Expressions with the -msg argument*

Regular Expressions with -file and -du arguments

Here are some examples:

- Specify the following command to waive messages for all files or all design units:

  ```
  waive -regexp -file ".*" ...
  ```

  or

  ```
  waive -regexp -du ".*" ...
  ```

- Specify the following command to waive messages in all files whose names start with lib:

  ```
  waive -regexp -file "^lib.*" ...
  ```

- Specify the following command to waive messages in all design units whose names end with vwe:

  ```
  waive -regexp -du ".*vwe$" ...
  ```

Do not use absolute paths while specifying file names because there may be problems in porting the *Waiver File* when project files are moved.

NOTE: *Verilog is case-sensitive while VHDL is case-insensitive. Therefore, be careful while specifying design unit names.*

Regular Expressions with the -severity argument

You cannot use regular expressions with the *-severity* argument of the waive constraint. This is to ensure that no unintended message is waived as it often happens that very similar severity labels are registered in different products.

However, you can use regular expressions with other arguments and also specify the *-severity* argument. The most trivial case of using this argument is to waive all non-error messages. For example, the following specification waives all Warning and Info type messages in all files:

```
waive -regexp -file “.*” -severity Info
waive -regexp -file “.*” -severity Warning
```

Use the *-severity* argument with one of the *-file* arguments or the *-du and -ip* argument to waive non-essential messages, especially in the first run of a design through SpyGlass.

Regular Expressions with the -msg argument

Here are some examples:

- The following command waives all messages containing the `clk` string:

  ```
  waive -regexp -file “.*” -msg “.*[ ' \”]clk[ ' \”].*”
  ```

  The above command waives all messages containing the `clk` string in any of the following ways

  - `clk` with leading and trailing spaces
  - `'clk'`
  - `“clk”`

- The following command waives all messages for the `test` design unit that contain any combination of the `clk1` and `clk2` clocks:

  ```
  waive -regexp -file ".*" -msg ".*test\.clk1.*test\.clk2.*"
  waive -regexp -file ".*" -msg ".*test\.clk2.*test\.clk1.*"
  ```

  The above command waives all messages containing `test.clk1` and `test.clk2` in any order including the following message:

  ```
  Unsynchronized crossing: destination flop test.q1, clocked by
  test.clk2, source flop test.d1, clocked by test.clk1
  ```

If you specify the whole message in the `-msg` argument (and actual file/du names), you do not need to specify the `-regexp` argument. Just supply all actual values and place the entire message in the `q/.../` format. If you are placing a message in quotes, SpyGlass does not consider wildcard characters (* and ?), double quote character, dollar, and escape characters as literals. In such case, you must escape them.

Support for Hierarchical Waivers

SpyGlass provides the capability to chip-level designers to use all the waivers specified by a block-level designer on the block, during chip-level analysis. To support this feature, you can use the `waive -import` command, which enables you to import waivers specified in the block-level design into the chip-level design.

You may specify waivers to individual blocks separately in the top-level chip. The general syntax of the `waive` constraint for importing a waiver file for the specified block is as follows:

```
waive -import { {<block_name>}  {<block-waive-file1>} }
waive -import { {<block_name>}  {<block-waive-file2>} }
```

The *<block_name>* can be module name or entity name. It is not recommended to precede/append library name or architecture name, module or entity name.

SpyGlass applies the block waiver file to design units matching any of the above specifications.

You can specify the path of the waiver file *<block-waive-file>* as a relative or absolute path. If the path specification is relative, it should be accessible from the current run directory.

Consider the following example, where `B1` and `B2` are two blocks inside the top-level chip, and B1.swl and B2.swl are the waiver files applied to these two blocks, respectively:

waive –import B1 B1.swl
waive –import B2 B2.swl

You can specify multiple waiver files for a given block by specifying multiple `waive -import` constraints. You may also specify the same waiver file to two different blocks. In such a case, the block waivers are applied independently to the respective blocks.

NOTE: *The commands specified in the waiver file to be imported are applicable only to the hierarchy of the module specified with the* `waive -import` *constraint.*

The generated output file contains two sections. Section I displays successfully migrated `waive` commands. Section II is generated only if there are non/incompletely migrated commands. This section displays non/incompletely migrated commands, with inline reason of the migration failure.

If you specify two files for the same block, two new waiver files are generated corresponding to each (specified file). If you specify one waiver file for two different blocks, SpyGlass generates two files (one file for each block).

You can view the generated file to see if all waiver constraint commands have been migrated as intended. If not, you can modify this waiver file as per the requirements and use it in subsequent runs.

To specify block-level waiver file in the SWL format (block.swl), perform the following steps:

1. Read the top file using the following command:

   ```
   read_file -type waiver top.swl
   ```

2. Inside the top.swl file, specify the following waiver command:

   ```
   waive -import block block.swl
   ```

Alternatively, to specify the block-level waiver file in the AWL format (block.awl), perform the following steps:

1. Read the top file using the following command:

   ```
   read_file -type awl top.awl
   ```

2. Specify the following command on the interactive sg_shell

   ```
   waive -import {block block.awl}
   ```

NOTE: *A block-level waiver file specified in the AWL format does not work when specified in a top-level SWL file.*

Additional Information

Please note the following about the `waive -import` constraint:

- SpyGlass also supports nested imports of waiver files, that is, one `import` command can be specified inside another `import` command, as shown below:

  ```
  top.swl:   waive -import b1 b1.swl
  ```

```
b1.swl:    waive -import b2 b2.swl
b2.swl:    waive -file test.v
```

NOTE: *All the waiver files to be imported (b1.swl and b2.swl, in the above example) should be accessible from the current working directory.*

- The `-disable` argument is also supported with the `waive -import` constraint, as shown below:

  ```
  waive -import b1 b1.swl -disable
  ```

  The above specification will disable the `waive -import` command.

NOTE: *Only the* `-disable` *and* `-comment` *arguments are supported with* `waive -import` *constraint. No other argument is supported with* `waive -import` *constraint.*

- File names in the imported `waive -file/file_line/file_lineblock` commands are converted to file names matching under the hierarchy of the block being imported. This is to ensure that migration occurs for, and according to, the block being imported.
- If the `-ip/-du` fields are regular expressions in the `waive` command to be imported, then the regular expressions are converted to names matching under the block hierarchy only. This is to ensure that the regular expressions do not match any name outside the block hierarchy.
- It may happen that a block-level waiver file is written in an older version of SpyGlass release, and the top-level designer importing this block-level waiver file is working in a later version of SpyGlass release in which few rule messages have been changed with respect to the previous release. In this case, SpyGlass automatically upgrades the old message in the block-level waiver files to the new rule message.

  However, this does not work if the old message in the `waive` command is a substring of its complete message of that release.

Waiving Messages by Using SpyGlass Pragmas

To waive rule messages using the SpyGlass Waiver pragmas, embed the SpyGlass Waiver pragma directives at appropriate places in your design source code.

Then, the specified rules or rules of the specified rule groups are still checked for the source code block related to the SpyGlass Waiver pragmas and the corresponding rule messages are written to the Violation Database. However, these rule messages are not reported in the SpyGlass Message Reports.

Waiving Rule Messages for a Block of Code

To waive messages of one or more rules for a block of source code, use the `disable_block` and `enable_block` pragmas as follows:

for Verilog:

```
...
//spyglass disable_block <rule-list> | ALL | $VAR
...
//spyglass enable_block <rule-list> | ALL | $VAR
...
```

(See *Verilog Example of Using Waiver Pragmas for a Block of Code*)

for VHDL:

```
...
--spyglass disable_block <rule-list> | ALL | $VAR
...
--spyglass enable_block <rule-list> | ALL | $VAR
...
```

(See *VHDL Example of Using Waiver Pragmas for a Block of Code*)

Where *`<rule-list>`* is a space-separated list of rule names or rule group names for which the messages should be waived. Using the `ALL` keyword waives all messages of all rules for the block of source code.

Verilog Example of Using Waiver Pragmas for a Block of Code

Consider the following example, containing the usage of the `disable_block` and `enable_block` pragmas:

```
module test10 (q,clk, d,reset);
  input clk,d,reset;
  output q;
  reg q;
  reg [3:0] a;
  reg [7:0] b;
  reg [2:0]data;
  //spyglass disable_block W362
  always @(posedge clk)
  begin
    if (data <= (a[1] + b[2]) ) //violation will be
                                //waived off
      a <=17;
    else
      //spyglass enable_block W362
      b = 8'hbb;
  end
endmodule
```

VHDL Example of Using Waiver Pragmas for a Block of Code

Consider the following example, containing the usage of the `disable_block` and `enable_block` pragmas:

```
library IEEE;
use ieee.std_logic_1164.all;
entity top is
end top;
architecture rtl of top is
signal s1 :  bit_vector( 2 downto 0);
signal s2 :  bit_vector( 3 downto 0);
signal s3 : boolean;
signal t1, t2 : integer;
begin
  --spyglass disable_block W116
  process
  begin
    case s1 <= s2 is -- violation will be waived off
    when TRUE => s3 <= (s2 = s1); -- violation will be
                                  -- waived off
    when others => null;
    end case;
  end process;
  --spyglass enable_block W116
  t1 <= t2 when s1 >= s2; -- violation will not be
                          -- waived off
end rtl;
```

Waiving Rule Messages for a Single Line of Code

To waive messages of one or more rules for a single line of source code, use the `disable` pragma as follows:

for Verilog:

```
<design-line> //spyglass disable <rule-list> | ALL
```

(See *Verilog Example of Using Waiver Pragmas for Single Line of Code*)

for VHDL:

```
<design-line> --spyglass disable <rule-list> | ALL
```

(See *VHDL Example of Using Waiver Pragmas for Single Line of Code*)

Where *`<rule-list>`* is a space-separated list of rule names or rule group names for which the messages are to be waived for the single line of source code. Using the `ALL` keyword waives all messages of all rules for the single line of source code.

Specifying SpyGlass Waiver Pragmas

Please note the following while using the SpyGlass Waiver pragmas:

- All keywords (`spyglass`, `disable`, `disable_block`, `enable_block`, and `ALL`) are case-sensitive.
- Only spaces are allowed between keywords. You cannot use other white space characters, such as the tab character.
- There may or may not be any spaces between `//` (in case of Verilog) or `--`(in case of VHDL) and the keyword `spyglass`.
- You can also insert comments in the pragma line as follows:

  for Verilog:

  ```
  //waiver pragma --comment
  ```

  ```
  Example:
  ```

  ```
  if (data <= (a[1] + b[2]) ) //spyglass disable W362 --This
  is comment
  ```

  for VHDL:

```
--waiver pragma //comment
```

```
Example:
```

```
case s1 <= s2 is --spyglass disable W116 //This is a
comment
```

- You can also write multiple pragmas with comments on a single line as follows:

  ```
  //waiver pragma1 --comment //waiver pragma --comment
  ```

  SpyGlass considers the above specification as two different pragmas. However, if the starting directive is a comment or a non-waiver pragma, SpyGlass treats the whole line as a comment. For example, the following line in the design file will not result in any valid waiver pragma:

  ```
  //non-waiver pragma //spyglass disable rulename
  ```

- If there is no corresponding `enable_block` pragma for a `disable_block` pragma, then the scope of the `disable_block` pragma extends till the end of the source file in which it is specified.
- The scope of SpyGlass Waiver pragmas is limited to the source file in which they are specified. Writing a pragma in one source file and including this source file in another source file will not imply that the pragma is effective in the second file.

  Consider the following example:

  ```
  // test.v
  module test (input in1, output out1);
  //spyglass disable_block ALL
  `include "lib.v"
  complex_INPUT_WIDTH_struct_t i_data;
  …
  //spyglass enable_block ALL
  …
  endmodule
  ```

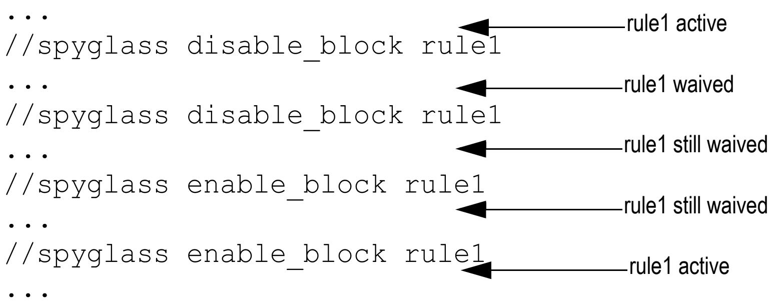
  In the above example, SpyGlass does not report any violation on the test.v file between the `disable_block` and `enable_block` pragmas. Though lib.v is included in test.v after the `disable_block` pragma, the waiver pragma will not be applicable to the included file. It will only be applicable to the design file in which it is specified.

NOTE: *The SpyGlass Waiver pragmas do not work on design units in VHDL libraries.*

Nested SpyGlass Waiver Pragmas

The `disable_block` and `enable_block` pragmas can be nested. However, the scope of the pragmas depends on the way they have been specified.

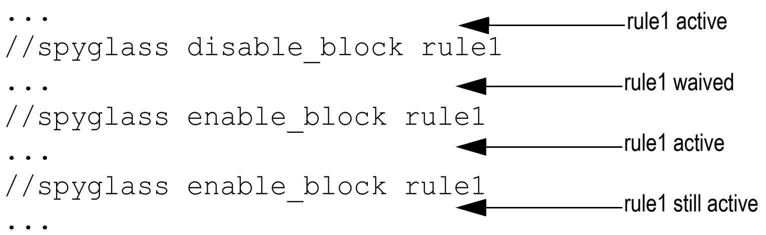
For example, consider the following specification:



```
...
//spyglass disable_block rule1
...
//spyglass disable_block rule1
...
//spyglass enable_block rule1
...
//spyglass enable_block rule1
...
```

Thus, in case of complete pairs of nested waiver pragmas of the same rule(s), the scope is the source code block between the *outermost* `disable_block` and `enable_block` pragma pair.

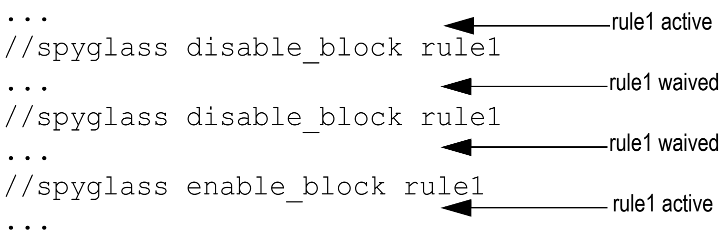
Now consider the following specification:



```
...
//spyglass disable_block rule1
...
//spyglass enable_block rule1
...
//spyglass enable_block rule1
...
```

Thus, in case of incomplete pairs of nested waiver pragmas of the same rule(s) with missing `disable_block` pragmas, the scope is the source code block between the *innermost* `disable_block` and `enable_block` pragma pair.

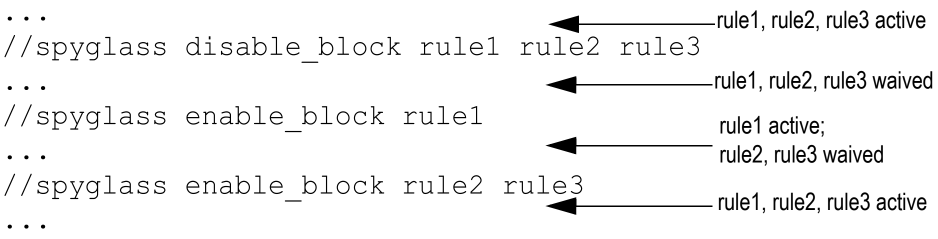
In case of incomplete pairs of nested waiver pragmas of the same rule(s) with missing `enable_block` pragmas, the scope is the source code block between the *outermost* `disable_block` and `enable_block` pragma pair as in the following example specification:



```
...
//spyglass disable_block rule1
...
//spyglass disable_block rule1
...
//spyglass enable_block rule1
...
```

Switching Off Waiver for Selective Rules of a Group

You can also selectively waive or activate a rule from a set of rules as shown in the following example specification:



```
...
//spyglass disable_block rule1 rule2 rule3
...
//spyglass enable_block rule1
...
//spyglass enable_block rule2 rule3
...
```

rule1, rule2, rule3 active
rule1, rule2, rule3 waived
rule1 active;
rule2, rule3 waived
rule1, rule2, rule3 active

Similarly, if you have waived for a rule group, you can selectively activate the rules in the rule group in the same manner.

NOTE: *You cannot selectively activate a rule for a source code block that you have waived by using the* `ALL` *keyword.*

Verilog Example of Using Waiver Pragmas for Single Line of Code

The following example contains the usage of waiver pragma for single line of code:

```
module test10 (q,clk, d,reset);
input clk,d,reset;
```

```
output q;
reg q;
reg [3:0] a;
reg [7:0] b;
reg [2:0]data;

always @(posedge clk)
begin
if (data <= (a[1] + b[2]) ) //spyglass disable W362
   a <=17;
else
   b = 8'hbb;
end
endmodule
```

VHDL Example of Using Waiver Pragmas for Single Line of Code

The following example contains the usage of waiver pragma for single line of code:

```
library IEEE;
use ieee.std_logic_1164.all;


entity top is
end top;

architecture rtl of top is
signal s1 :  bit_vector( 2 downto 0);
signal s2 :  bit_vector( 3 downto 0);
signal s3 :  boolean;
begin
process
  begin
    case s1 <= s2 is --spyglass disable W116
    when TRUE => s3 <= (s2 = s1);
    when others => null;
    end case;
end process;
end rtl;
```

Ignoring the SpyGlass Waiver Pragmas

By default, pragma-based waivers (inside design source files) are always effective. You can disable them by specifying the following command in the SpyGlass project file:

```
set_option ignorewaivers yes
```

Waiving Messages in Waiver/SGDC Files

You can waive messages in the *Waiver File*/SGDC file by embedding SpyGlass waiver pragma directives at appropriate places.

Then, the specified rules or rules of the specified rule group, which are waived, are not reported in the SpyGlass message reports and the corresponding rule messages are not written to the violation database.

Use the disable_block and enable_block pragmas to disable and enable rules, as shown in the following example:

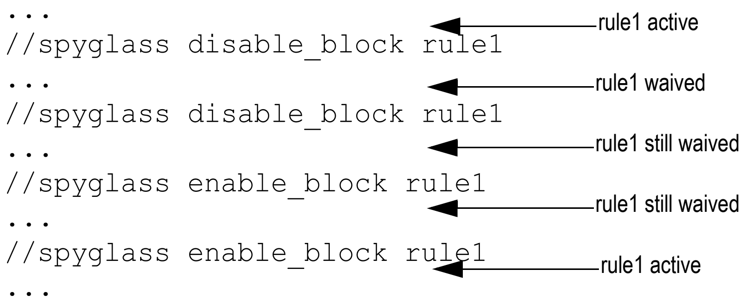
```
//spyglass disable_block R1
...
...<cmd>
...
//spyglass enable_block R1
```

In the above example, SpyGlass disables rule-checking for the R1 rule in the lines after //spyglass disable_block R1. However, SpyGlass resumes rule-checking for the R1 rule in the lines after //spyglass enable_block R1 is specified. Here, *<cmd>* will continue to work, if specified correctly, irrespective of the pragmas.

You can also use # instead of // while specifying the disable_block and enable_block pragmas. For example:

```
#spyglass disable_block R1
...
...
#spyglass enable_block R1
```

The disable_block and enable_block pragmas can be nested. However, the scope of the pragmas depends upon the way they have been specified. For example:



```
...
//spyglass disable_block rule1
...
//spyglass disable_block rule1
...
//spyglass enable_block rule1
...
//spyglass enable_block rule1
...
```

You can also specify a comma-separated list of rule names or the name of the rule group, as shown in the following examples:

- `//spyglass disable_block R1,R2`

  Where, R1 and R2 are the rules to be waived

- `//spyglass disable_block R1,G1`

  Where, G1 is the rule group name. In this case, all the rules belonging to this group are waived.

You can use the ALL keyword to waive messages of all the rules. After specifying ALL keyword in the disable_block pragma, you cannot explicitly enable a particular rule by specifying that rule name in the enable_block pragma. For example:

```
//spyglass disable_block ALL
...
//spyglass enable_block R1
...
```

Here, rule-checking for R1 remains off. In this case, you need to use //spyglass enable_block ALL to enable rule-checking of all the rules.

Existing Waiver Support in SpyGlass

Currently, SpyGlass supports spyok and verilint (for SpyGlass lint solution) waivers. Support of spyok waivers will continue in SpyGlass for backward compatibility.

Tagging Messages

Adding a tag on a violation message enables you to keep track of that message, which you may want to fix later or which you already fixed/ verified.

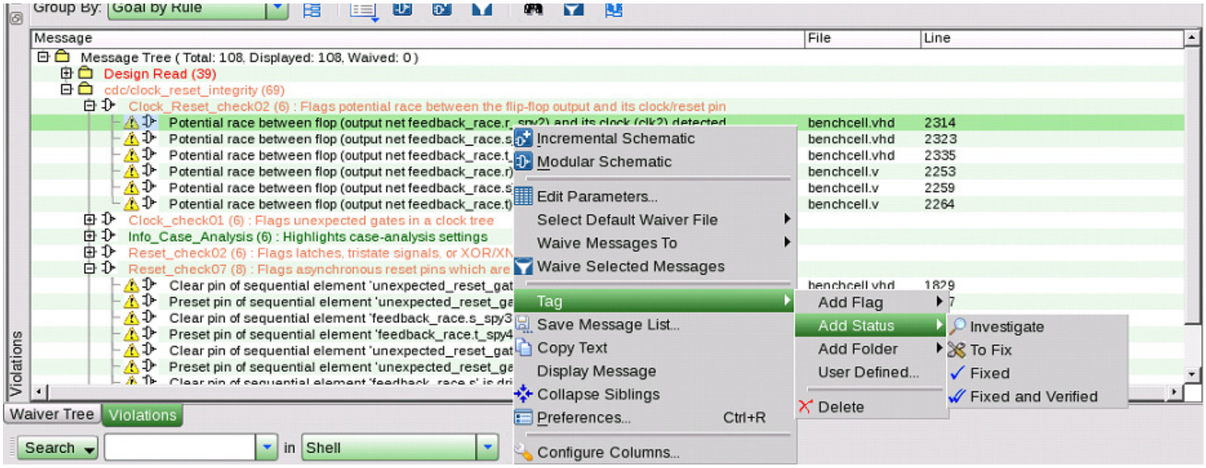
Based on your requirement, you can tag messages either with certain predefined identifiers, such as *Investigate*, *Fixed*, *ToFix*, and *VerifiedFixed,* or with your own custom tags.

Adding a Tag

To add a tag to a message, perform the following steps:

1. Select the message.
2. Select the *Add Tag* option in the right-most bar in the *Results* pane. Alternatively, right-click on the message and select the *Tag -> Add Custom* option from the shortcut menu.

   The *Add Message Tag* dialog appears, as shown below:



Group By: Goal by Rule

Message	File	Line
Message Tree (Total: 108, Displayed: 108, Waived: 0)		
Design Read (39)		
cdc/clock_reset_integrity (69)		
Clock_Reset_check02 (6) : Flags potential race between the flip-flop output and its clock/reset pin		
Potential race between flop (output net feedback_race.r_spy2) and its clock (clk2) detected	benchcell.vhd	2314
Potential race between flop (output net feedback_race.s	benchcell.vhd	2323
Potential race between flop (output net feedback_race.t_	benchcell.vhd	2335
Potential race between flop (output net feedback_race.r)	benchcell.v	2253
Potential race between flop (output net feedback_race.s	benchcell.v	2259
Potential race between flop (output net feedback_race.t)	benchcell.v	2264
Clock_check01 (6) : Flags unexpected gates in a clock tree		
Info_Case_Analysis (6) : Highlights case-analysis settings		
Reset_check02 (6) : Flags latches, tristate signals, or XOR/XN		
Reset_check07 (8) : Flags asynchronous reset pins which are		
Clear pin of sequential element 'unexpected_reset_gat	benchcell.vhd	1829
Preset pin of sequential element 'unexpected_reset_ga		7
Clear pin of sequential element 'feedback_race.s_spy3		
Preset pin of sequential element 'feedback_race.t_spy4		
Clear pin of sequential element 'unexpected_reset_gat		
Preset pin of sequential element 'unexpected_reset_ga		
Clear pin of sequential element 'feedback_race.s' is dri		

FIGURE 201. Add Message Tag

3. In the above dialog, specify your own tag in the textbox.

   Alternatively, you can select any predefined tag from the adjacent drop-down list.

4. Click the *Ok* button.

After performing the above steps, the specified tag appears (indicated by a tag icon) for the message. In addition, that tag also appears in the *Tag > Add User-Defined* shortcut menu option list.

You can also add a tag from a set of visual tags (identified by corresponding graphical icons) available through the *Tag > Add Flag* shortcut menu option. When you apply any of these visual tags to a message, the corresponding graphical icon is prefixed to the message.

Deleting a Tag

To delete a tag from a message, perform the following steps:

1. Select the message appearing in the *Results* pane.
2. Select the *Delete Tag* option in the right-most bar in the *Results* pane.

   Alternatively, right-click the message and select the *Tag -> Delete* option from the shortcut menu.

After performing the above steps, the tag applied to the message is deleted.

NOTE: *The Tag -> Delete shortcut menu option is enabled only if you have added a tag to a rule message.*

Handling SpyGlass Built-In Messages

Based on the severity of the built-in messages, you can fix the reported violation accordingly.

Handling Syntax Error Messages

If you encounter a syntax error message after design analysis, you must fix that error before SpyGlass can process the design any further. Most rule checks do not run if the design contains syntax errors.

Handling Language Warning Messages

If you encounter a language warning message, you should check that the potential problem indicated is expected and not a concern. SpyGlass will continue processing if language warnings are reported, although the nature of the analysis may be affected. For example, if you see a warning that the size of an expression does not match the size of the object to which it is assigned, you may decide that this is known but not important. On the other hand, you may realize that due to this problem, a value may be truncated or extended where no such modification was expected.

NOTE: *You can limit the number of WRN messages logged in the Violation Database as described in Waiving Messages.*

Handling Synthesis Warning Messages

A synthesis warning message appears in any of the following cases:

- The specified construct is not synthesizable and therefore, SpyGlass cannot synthesize the design unit containing that construct.
- In some cases, the construct is ignored during synthesis.

These messages are useful if you want to check a design for synthesizability, but it is also important to understand that SpyGlass runs most complex connectivity and functionality checks on a design by (automatically) synthesizing the design internally.

Design units that cannot be synthesized are skipped in this process and, therefore, are ignored in analysis of those connectivity and functionality rules.

SpyGlass reports the design units that have been skipped for this reason, but you should be aware that analysis of those design units will necessarily be incomplete.

Handling Synthesis Error Messages

A synthesis error message indicates the following:

- SpyGlass could not synthesize a design unit because of an un-synthesizable construct.
- SpyGlass replaces such design units by a black box in the resulting netlist.

The presence of such design units affects the flattening stage as well and the resulting rule-checking is inaccurate.

Handling Internal Messages

If a rule reports an internal message, SpyGlass displays the name of such rules as *SpyGlassInternalFatal*, *SpyGlassInternalError*, or *SpyGlassInternalWarning*, depending upon the rule severity.

Report such messages to Atrenta Support.

The Configuration File in SpyGlass

SpyGlass has a Configuration File feature using which you can specify configuration settings like default startup mode (SpyGlass GUI or batch), default product to be run, default language setting, and default report format.

The setup information in a SpyGlass Configuration file is read by SpyGlass transparently and automatically. Therefore, it differs from a command file that must be always specified by using the `read_file sourcelist` *`<file-name>`* project file command.

SpyGlass Configuration file defines working defaults that are lower priority and can be overridden by settings in a command file or directly through command-line.

The SpyGlass Configuration File is an ASCII text file named .spyglass.setup that can be located in four different locations so that you set four levels of configuration settings:

1. A configuration file specified using the *configfile* command.
2. A SpyGlass Configuration File located in the Current Working Directory.
3. A SpyGlass Configuration File located in your Home directory ($HOME)
4. A SpyGlass Configuration File specified by setting environment variable SPYGLASS_CUSTOMER_CONFIG_FILE

5. A SpyGlass Configuration File located in SpyGlass Installation Directory (<installation-dir>/SPYGLASS_HOME)

You can have a separate SpyGlass Configuration File in each of the five locations or a combination of these locations.

The following lists the order of priority of the configuration settings:

1. The configuration settings in a configuration file specified using the *configfile* command
2. The configuration settings in a SpyGlass Configuration File located in the Current Working Directory
3. The configuration settings in a SpyGlass Configuration File located in the User’s Home directory
4. The configuration settings in the SPYGLASS_CUSTOMER_CONFIG_FILE environment variable
5. The configuration settings in a SpyGlass Configuration File located in the SpyGlass installation Directory.

NOTE: *SpyGlass reads all the specified configuration files in the order specified above. The key-value pair first read by SpyGlass are honored and all other specifications for the same key-value pair in any other lower-priority file is ignored.*

Structure of SpyGlass Configuration File

The SpyGlass Configuration File (the .spyglass.setup file) contains the different configuration settings in following format:

```
[ DEFAULT_STARTUP_MODE = gui | batch ]
[ USE_32_BIT_EXECUTABLE_ONLY = no | yes ]

[ DEFAULT_LANGUAGE_MODE = VHDL | Verilog | mixed | def | none ]
[ DEFAULT_TEMPLATE = <goal-name> | none ]
[ DEFAULT_TEMPLATE_DIRECTORY = GUIDEWARE_NEW_RTL |
                               GUIDEWARE_IP_RTL |
                               GUIDEWARE_IP_NETLIST |
                               GUIDEWARE_SOC
[ DEFAULT_POLICY_FOR_SPYEXPLAIN = <product-list> | none | all ]
[ DEFAULT_REPORT_FORMAT = moresimple | none | <report-name> ]
[ DEFAULT_REPORT_FORMAT_FOR_SLC = <report-name> ]
[ DEFAULT_PRAGMA = default | none | <pragma-name-list> ]
[ DEFAULT_BBOX_MODEL = BBOX_ILM | BBOX_CELLDEFINE |
                       BBOX_LIBCELL | BBOX_ENCRYPTED_LIB
                       <any combination> | NONE ]
[ AUTOENABLE_RULEGROUP_PARAMETER_CONTROL = no | yes ]
[ AUTOENABLE_MEMORY_HANDLING = no | yes ]
[ AUTOENABLE_HUGE_SCHEMATIC_DISPLAY = no | yes ]

[ VHDL_LIB_MAP = SYNOPSYS
           $SPYGLASS_HOME/vhdl_libs/$SPYGLASS_PLATFORM/SYNOPSYS |
  { VHDL_LIB_MAP = <logical-lib-name> <physical-path> }]
[ AUTOENABLE_VHDL_SORT = no | yes ]
[ DEFAULT_VHDL_SORT_METHOD = <method> ]

[ AUTOENABLE_INFERBLACKBOX = no | yes | yes_netlist | yes_rtl ]
[ AUTOENABLE_PRECOMPILED_VLOG = no | yes ]

[{ COMMAND_OPTION_FILENAME = <file-name> }]
[{ COMMAND_FILE_ARGS = <arg-list> }]
[ SGDC_INCLUDE_FILE_PATH = <dir-name> ]
[ OVERLOAD = <named-overload-list> ]
[ DEFAULT_SLF_CONFIG_FILE = <file-name> ]
[ AUTOENABLE_BUILTIN_CHECKS_FOR_POLICY = <product-list> ]
[ DEFAULT_METHODOLOGY = $SPYGLASS_HOME/GuideWare/New_RTL ]
[ METHODOLOGY_SEARCH_PATH = <space-separated-list-of-directories> ]
[ UI_WAIVER_DEFAULT_REGEXP_EXCLUDE_FIELDS = <comma-separated-list-of-
options> ]
DC_DWARE_FILES_PATH = <file_path>
DC_DW_FILES_PATH = <file_path>

[ ABSTRACT_FILE_NAME_STYLE = <compat | short> ]
```

Please note the following points:

- The default value of each configuration setting is shown underlined. The default value is applicable if the corresponding configuration setting is not found in the SpyGlass Configuration File(s).
- You should specify commands in this file only in classic batch format. This file does not support Tcl format.
- The SpyGlass Configuration file can have comments of format (VHDL-like comments), format (Verilog-like comments), or # format (SpyGlass common format).
- The SPYGLASS_HOME environment variable is supported in a Configuration File and is appropriately set to the SpyGlass Home directory when SpyGlass is run. All user-defined environment variables are supported in a Configuration File just like in a command file.
- The Configuration File support replaces the currently available .spyrc feature from SpyGlass version 3.3.0 onwards. Thus, it is recommended that you move to the SpyGlass Configuration File method as the .spyrc feature may be removed in a future release.
- Following configuration keys are not supported when the configuration file is specified in the project file:
  - ❒ DEFAULT_STARTUP_MODE
  - ❒ USE_32_BIT_EXECUTABLE_ONLY
  - ❒ DEFAULT_LANGUAGE_MODE
  - ❒ DEFAULT_POLICY
  - ❒ DEFAULT_POLICY_FOR_SPYEXPLAIN
  - ❒ DEFAULT_REPORT_FORMAT_FOR_SLC

- ❒ DEFAULT_TEMPLATE
- ❒ DEFAULT_TEMPLATE_DIRECTORY
- ❒ LICENSE_EXPIRY_NOTIFICATION_DAYS
- ❒ LICENSE_QUEUING_INTERVALS_IN_SECS
- ❒ DEFAULT_METHODOLOGY
- ❒ DEFAULT_EXE_TYPE_ON_64BIT
- ❒ METHODOLOGY_SEARCH_PATH
- ❒ AUTO_UI_LICENSE_TIMEOUT
- ❒ UI_LICENSE_QUEUING_WHEN_FEATURE_EXISTS

General Configuration Settings

The following keys are used for general configuration settings:

DEFAULT_STARTUP_MODE	*DEFAULT_EXE_TYPE_ON_64BIT*
USE_32_BIT_EXECUTABLE_ONLY	

DEFAULT_STARTUP_MODE

The DEFAULT_STARTUP_MODE key sets the SpyGlass startup mode as the SpyGlass mode or the batch mode.

You can change the startup mode to batch mode or SpyGlass mode by changing the value of the DEFAULT_STARTUP_MODE key as follows:

- gui: Sets SpyGlass as the startup mode
- batch: Sets batch as the startup mode

NOTE: *By default, the startup mode is the SpyGlass mode (key value gui).*

The DEFAULT_STARTUP_MODE key can be overridden by the -gui or -batch command-line options for values batch and GUI respectively.

DEFAULT_EXE_TYPE_ON_64BIT

The DEFAULT_EXE_TYPE_ON_64BIT key sets the default executable binaries (64-bit or 32-bit) to be executed for SpyGlass on 64-bit architectures.

By default, 64-bit SpyGlass binaries are executed on 64-bit architectures. To specify 32-bit binaries to be executed on 64-bit architectures, set the value of the DEFAULT_EXE_TYPE_ON_64BIT key to 32.

The DEFAULT_EXE_TYPE_ON_64BIT key can be overridden by the *-32bit* or *-64bit* command-line options for values 64 and 32 respectively.

USE_32_BIT_EXECUTABLE_ONLY

The USE_32_BIT_EXECUTABLE_ONLY key has been deprecated as this key was used only for 64-bit HP platform, which is no longer supported.

Product and Rules Configuration Settings

The following keys are used to configure settings for products and rules:

DEFAULT_LANGUAGE_MODE	*DEFAULT_TEMPLATE*
DEFAULT_TEMPLATE_DIRECTORY	*DEFAULT_POLICY*
DEFAULT_POLICY_FOR_SPYEXPLAIN	*DEFAULT_REPORT_FORMAT*
DEFAULT_REPORT_FORMAT_FOR_SLC	*DEFAULT_PRAGMA*
DEFAULT_BBOX_MODEL	*AUTOENABLE_RULEGROUP_PARAMETER_CONTROL*
AUTOENABLE_MEMORY_HANDLING	*AUTOENABLE_HUGE_SCHEMATIC_DISPLAY*
AUTOENABLE_BUILTIN_CHECKS_FOR_POLICY	*ABSTRACT_FILE_NAME_STYLE*

DEFAULT_LANGUAGE_MODE

Use this key to set a language for the HDL sources for SpyGlass analysis if no language is specified.

If this setting is not available, you must specify the language at command-line or a command file.

The DEFAULT_LANGUAGE_MODE key can be overridden by any of the specified language (vhdl, verilog, mixed, or def).

DEFAULT_TEMPLATE

Use this key to set a goal to run.

By default, SpyGlass does not pick any goal.

You can set the value to a desired goal in the *<methodology-name>/<goal-name>* format. For example, the following setting sets the default goal to be the Coverage goal of the SpyGlass DFT methodology:

```
DEFAULT_TEMPLATE = DFT/Coverage
```

You can also set a custom goal as the default goal. Just ensure that the full path to the methodology directory is specified by using the *I* command. For example, you have goal named `mygoal1` located in /usr/john/myGoals directory. Then, set the `DEFAULT_TEMPLATE` key as follows:

```
DEFAULT_TEMPLATE = myGoals/mygoal1
```

Also, specify the following command in the project file:

```
set_option I /usr/john ...
```

NOTE: *The I command is not recommended to be used in the project file. For details, see Options Not Recommended. However, you should specify the directory paths using the -I option on the command-line itself while invoking console or sg_shell.*

The DEFAULT_TEMPLATE key can be overridden by the *-template <goal-names>* command-line option.

DEFAULT_TEMPLATE_DIRECTORY

Use this key to set a goal directory.

You can set the `DEFAULT_TEMPLATE_DIRECTORY` key to the following values:

Value	Indicates
`GUIDEWARE_NEW_RTL`	(Default) The GuideWare goals for the New_RTL methodology (installed at <your-inst-dir>/ SPYGLASS_HOME/GuideWare/New_RTL directory)
`GUIDEWARE_IP_RTL`	The GuideWare goals for the IP_RTL methodology (installed at <your-inst-dir>/SPYGLASS_HOME/ GuideWare/IP_RTL directory)
`GUIDEWARE_IP_NETLIST`	The GuideWare goals for the IP_RTL methodology (installed at <your-inst-dir>/SPYGLASS_HOME/ GuideWare/IP_netlist directory)
`GUIDEWARE_SOC`	The GuideWare goals for the SoC methodology (installed at <your-inst-dir>/SPYGLASS_HOME/ GuideWare/SoC directory)

The `DEFAULT_TEMPLATE_DIRECTORY` key can be overridden by the *-templatedir* command-line option.

DEFAULT_POLICY

NOTE: *This key is applicable only for classic batch. The classic batch mode of SpyGlass has entered in its End-of-Life period and will not be supported starting the SpyGlass 5.4.0 release.*

Use this key to lock the license of product/products to be run while invoking SpyGlass if none of the *-policies | -policy*/and *-template <goal-names>* command-line option is specified directly at the command-line.

While invoking SpyGlass, if you do not use the DEFAULT_POLICY key and try to run a product, it might be possible that the license for the same product is checked out by some other user.

To avoid such a situation, you can lock the license for such product/ products to be run by specifying them using the DEFAULT_POLICY key.

You should specify the product names separated by a space. The DEFAULT_POLICY key can be overridden by the *-policies | -policy* or *-template <goal-names>* command-line options.

DEFAULT_POLICY_FOR_SPYEXPLAIN

Use this key to set a default product to search with the spyexplain utility if the *-policies | -policy* command-line option is not specified directly on the command-line.

SpyGlass searches all installed products by default if you have not specified any product on the command-line.

To search a specified product by default, set the product mnemonic as the value. You can also specify multiple product mnemonics as a space or comma-separated single-line list.

You can set the value to none to disable searching any product including the SpyGlass Built-in products.

The `DEFAULT_POLICY_FOR_SPYEXPLAIN` key can be overridden by the *-policies | -policy* command-line option.

DEFAULT_REPORT_FORMAT

Use this key to set a default report format in which messages will be reported at end of SpyGlass run if the *report* command is not specified in project file or through *-f* file(s) on the command-line.

By default, the SpyGlass run generates the *moresimple* report.

You can set the value to none to disable this setting and not print any report at the end of SpyGlass run (that is, same behavior as *noreport* command).

The `DEFAULT_REPORT_FORMAT` key can be overridden by the *report* command.

DEFAULT_REPORT_FORMAT_FOR_SLC

Use this key to set a default report format in which messages will be reported at end of a SpyGlass Library Compiler run if the *report* command is not specified in project file or through *-f* file(s) on command-line.

DEFAULT_PRAGMA

Use this key to set pragma keywords.

By default, SpyGlass assumes synopsys as pragma keyword for Verilog designs and synopsys and pragma as pragma keywords for VHDL designs.

Set the name of the default pragma keyword as the value. You can also specify multiple pragma keywords as a space or comma-separated single-line list.

You can set the value to none to disable this setting (that is, same behavior as `set_option` *pragma* `nopragma` command in a project file).

DEFAULT_BBOX_MODEL

Use this key to define BBOX_MODEL types to be recognized for SpyGlass rule-checking.

By default, the DEFAULT_BBOX_MODEL configuration key is set to BBOX_LIBCELL so that only the SpyGlass Library Compiler-generated cells are considered as the BBOX_MODEL type.

You can set the DEFAULT_BBOX_MODEL configuration key to BBOX_ILM for user-defined BBOX_MODEL types, BBOX_CELLDEFINE for Verilog 'celldefine modules, BBOX_LIBCELL for the SpyGlass Library Compiler generated cells, BBOX_ENCRYPTED_LIB for the Precompiled and encrypted library cells, or any combination of these values as a space or comma-separated list.

You can also set the DEFAULT_BBOX_MODEL configuration key to NONE to disable the feature.

DEFAULT_VERBOSITY

Use this key to control the verbosity option for all SpyGlass runs.

Set the value of the key to one of the following values: 0, 1, 2, or 3.

NOTE: *Use of debug flags will automatically dump the verbosity level to 3. Even if config key DEFAULT_VERBOSITY is set to some lower level, and DEBUG options is specified, the verbosity will be changed to level 3.*

AUTOENABLE_RULEGROUP_PARAMETER_CONTROL

Use this key to specify whether rules of a rule group specified with the *rules* command are always run or are run according to their rule-running condition.

By default, the AUTOENABLE_RULEGROUP_PARAMETER_CONTROL configuration key is set to no so that the rules of a rule group specified with the *rules* command are run irrespective of their rule-running conditions. In addition, rules that are enabled/disabled by a Boolean-type rule parameter (for example, the fast rule parameter in the SpyGlass lint solution disables some rules) are run irrespective of the rule parameter status.

You can set the AUTOENABLE_RULEGROUP_PARAMETER_CONTROL configuration key to yes so that the rules of a rule group specified with the *rules* command are run according to their rule-running condition. A rule that is switched off by default will not be run. In addition, rules that are enabled/disabled by a Boolean-type rule parameter are run based on the rule parameter status only.

AUTOENABLE_MEMORY_HANDLING

Use this key to enable the memory reduction feature.

By default, the AUTOENABLE_MEMORY_HANDLING key is set to no and the Memory Reduction feature is not enabled.

AUTOENABLE_HUGE_SCHEMATIC_DISPLAY

Use this key to load huge schematics in *The Modular Schematic Window*.

By default, such schematics are not loaded.

AUTOENABLE_BUILTIN_CHECKS_FOR_POLICY

Use this key to run built-in rules of products specified in its value list, irrespective of rule set specified in a project file or goals. Products specified in this list are loaded and built-in rules are run in following situations:

- While precompiling your design using the command `set_option` *noelab* `yes` specified in a project file, the `set_goal_option` *norules* `yes` command is specified in a project file.
- Rules are run without the noelab option. For products that are common in the list specified through the goals and that specified in the configuration key, built-in rules are enabled irrespective of the rule set through commands (such as *rules*) or goals.

ABSTRACT_FILE_NAME_STYLE

The ABSTRACT_FILE_NAME_STYLE key controls the naming style of the abstract files generated by SpyGlass. You can specify one of the following values specified below:

- **short**: Generates short file name in the <module_name>_<product>_abstract.sgdc format. For example, deep_cdc_abstract.sgdc.
- **comapt**: Generates unique file names with parameter details in the <module_name>_<parameter_details>_<product>_abstract.sgdc. For example, deep_BUS_1_WIDTH_1_cdc_abstract.sgdc.

By default, the value of the *ABSTRACT_FILE_NAME_STYLE configuration key* is set to `compat`.

Configuration Settings for VHDL Designs

The following keys are used to configure settings for VHDL designs:

VHDL_LIB_MAP	*AUTOENABLE_VHDL_SORT*
DEFAULT_VHDL_SORT_METHOD	

VHDL_LIB_MAP

Use this key to set default VHDL library mappings.

This configuration setting is equivalent to the *lib* command in the project file.

By default, the VHDL_LIB_MAP key is set to the following value:

```
SYNOPSYS $SPYGLASS_HOME/vhdl_libs/$SPYGLASS_PLATFORM/SYNOPSYS
```

The VHDL_LIB_MAP configuration setting can be repeated for specification of multiple logical library maps. However, for any one logical library name, only one setting is taken as per order of precedence defined earlier. The value specified for a logical library map in a project file or the *lib* command has higher precedence than this specification.

When SpyGlass is run, library map of all logical libraries as defined in either configuration file or command file is taken. For example, if library L1 and L2 are mapped in the configuration File and library L3 and L4 are mapped in a specified project file, SpyGlass is run with all four library map arguments.

The VHDL_LIB_MAP Configuration Setting also has an additive effect. Therefore, a VHDL library mapping specified using the VHDL_LIB_MAP configuration setting that is not specified either directly through the *lib* command or through a project file is added to SpyGlass command-line options in addition to other VHDL library mappings specified either directly on command-line or through a project file.

The VHDL_LIB_MAP Configuration Setting has an additive effect for multiple-level Configuration Files. Thus, different VHDL library mappings specified using the VHDL_LIB_MAP Configuration Setting at different Configuration File levels, are all added to SpyGlass command-line options.

AUTOENABLE_VHDL_SORT

Use this key to specify automatic sorting of VHDL source files.

By default, the AUTOENABLE_VHDL_SORT key is set to no and the automatic sorting feature is disabled.

You can set the value of the AUTOENABLE_VHDL_SORT key to yes to enable the automatic sorting feature and specify the sorting algorithm using the DEFAULT_VHDL_SORT_METHOD key.

The AUTOENABLE_VHDL_SORT key can be overridden by the *sort* option in the project file.

DEFAULT_VHDL_SORT_METHOD

Use this key to set an algorithm type for SpyGlass automatic VHDL file sorting feature.

By default, the special automatic sorting algorithm is enabled. You can set the values of the DEFAULT_VHDL_SORT_METHOD key to lexical.

In case you are okay with the default sorting technique specified in the configuration file(s) (the last one takes the priority), simply specify the sort option to enable sorting of VHDL files based on this default sorting technique.

Configuration Settings for Verilog Designs

The following keys are used to configure settings for Verilog designs:

AUTOENABLE_INFERBLACKBOX	*AUTOENABLE_PRECOMPILED_VLOG*

AUTOENABLE_INFERBLACKBOX

Use this key to specify whether the SpyGlass Inferblackbox feature is enabled for all design phases, is enabled for a particular design phase, or is disabled.

By default, the AUTOENABLE_INFERBLACKBOX key is set to yes and the feature is enabled for both RTL and Netlist phases.

You can set the following values to the AUTOENABLE_INFERBLACKBOX key:

Value	**Effect**
yes or yes_netlist	Equivalent to specifying the set_option *inferblackbox* yes command
no	Do nothing

The AUTOENABLE_INFERBLACKBOX key can be overridden by the inferblackbox or disable_inferblackbox options.

AUTOENABLE_PRECOMPILED_VLOG

Use this key to support precompiled Verilog libraries.

To enable precompiled Verilog library support, you need to supply the `set_option` *enable_precompile_vlog* `yes` command in project file.

Setting the `AUTOENABLE_PRECOMPILED_VLOG` key to `yes` is equivalent to supplying the enable_precompile_vlog option.

Other Configuration Settings

The following configuration settings require you to set values as they do not have a default value:

COMMAND_OPTION_FILENAME	*COMMAND_FILE_ARGS*
SGDC_INCLUDE_FILE_PATH	*OVERLOAD*
DEFAULT_SLF_CONFIG_FILE	*AUTOENABLE_GATESLIB_AUTOCOMPILE*
DEFAULT_METHODOLOGY	*METHODOLOGY_SEARCH_PATH*

COMMAND_OPTION_FILENAME

Use this key to set a default command file.

This configuration setting is equivalent to the `read_file -type` *sourcelist* `<file-names>` command. The file specified with the `COMMAND_OPTION_FILENAME` key is always read before any actual command-line options.

You can supply this configuration setting multiple times in one Configuration File.

The `COMMAND_OPTION_FILENAME` configuration setting also has an additive effect. Thus, command files specified at any level (Configuration File(s), command-line, or in a command file) are all supplied to SpyGlass.

COMMAND_FILE_ARGS

Use this key to specify contents of a `read_file -type` *sourcelist* `<file-names>` command inside the configuration file.

A COMMAND_FILE_ARGS configuration setting must have its value in the same line where the key is specified and it cannot span multiple lines. However, you can have multiple specifications of the COMMAND_FILE_ARGS key in the same configuration file and the result of adding all these specifications will be used during the SpyGlass run. Also, the contents of this key in various configuration files are added (and not replaced as true for most of the other keys) and used for the SpyGlass run. This key is more convenient to use when you have concise -f listing for the configuration file and this option saves the overhead of creating a separate -f file and then using it inside the configuration file.

SGDC_INCLUDE_FILE_PATH

Use this key to specify a path from where included SpyGlass Design Constraints files can be picked.

By default, the SGDC_INCLUDE_FILE_PATH key is not set and the included SGDC files (specified using the INCLUDE directive in an SGDC file) are searched and included as follows:

1. If included SGDC file name is an absolute file name, the specified file at the specified location is included.
2. If included SGDC file name is a relative file name, the included SGDC file is searched in the relative directory location with respect to the location of the parent SGDC file.
3. If the included SGDC file name is any other type of file name (that is the file name does not start with a forward slash (first case above) or a period (second case above), the included SGDC file is searched with respect to the location of the parent SGDC file.

Use the SGDC_INCLUDE_FILE_PATH key to specify a different location for the last case above. Then, you can include company-, project-, or user-specific SGDC files without needing to specify exact path in each INCLUDE specification.

You can specify a space-separated list of directory names with the SGDC_INCLUDE_FILE_PATH key as in the following example:

```
SGDC_INCLUDE_FILE_PATH = /usr/corporate/sgdc
     /usr/projectfiles/sgdc /usr/john/mySGDC
```

You can also specify comma-separated or colon-separated lists of directory names.

The included SGDC files are searched in the specified directories in the same order in which they are specified with the SGDC_INCLUDE_FILE_PATH key.

NOTE: *In the last case, the included SGDC files are first searched with respect to the location of the parent SGDC file. Only if they are not found in these locations, the directories specified with the SGDC_INCLUDE_FILE_PATH key are searched.*

OVERLOAD

Use this key to set default named overloads.

By default, SpyGlass assumes no named overload. You can specify the named overloads as a space-separated list as in the following example:

```
OVERLOAD = CAD BOB METEOR
```

To disable named overloads, specify the OVERLOAD key with none value.

To set an additive behavior of the OVERLOAD key, use the OVERLOAD value. Consider the following example configuration files:

Configuration File	OVERLOAD Key Setting
$SPYGLASS_HOME/.spyglass.setup	OVERLOAD = CAD2
$HOME/.spyglass.setup	OVERLOAD = OVERLOAD METEOR
$CWD/.spyglass.setup	OVERLOAD = BOB OVERLOAD

The OVERLOAD value (shown in green color) in the $HOME/.spyglass.setup file indicates that SpyGlass should include the OVERLOAD key setting from the lower-precedence configuration file (that is, $SPYGLASS_HOME/ .spyglass.setup file). Similarly, the OVERLOAD value (shown in red color) in the $CWD/.spyglass.setup file indicates that SpyGlass should include the OVERLOAD key setting from the lower-precedence configuration file (that is, $HOME/.spyglass.setup file). Therefore, the effective value of the OVERLOAD key is as follows:

```
OVERLOAD = BOB CAD2 METEOR
```

DEFAULT_SLF_CONFIG_FILE

Use this key to specify the name of a text file that contains names of library attributes not to be reported as un-supported by the SpyGlass Library Compiler.

The format of the text file contents is as follows:

```
IGNORE_LIB_CONSTRUCT =
{
  [<attr-group>::]<attr-name>
  [<attr-group>::]<attr-name>
  [<attr-group>::]<attr-name>
...
}
```

Here, <attr-group> is the name of a library attribute group and <attr-name> is the name of a library attribute that belongs to the library attribute group <attr-group>. Thus, the following example suppresses the warnings for all instances of time_unit library attribute under the pin library group:

```
IGNORE_LIB_CONSTRUCT =
{
...
pin::time_unit
...
}
```

Specifying <attr-group> is optional. Thus, you can just specify the library attribute name to ignore it under all applicable library attribute groups. Thus, the following example causes library attribute direction to be ignored under all its applicable library attribute groups:

```
IGNORE_LIB_CONSTRUCT =
{
...
direction
...
}
```

You can add comments in the text file using the #-type comment format.

AUTOENABLE_GATESLIB_AUTOCOMPILE

Use this key to enable compilation of gate libraries (.lib) automatically to SpyGlass-compatible format library files (.sglib).

By default, the value of this key is set to no and the auto compilation of gate libraries does not occur.

You can set the value to yes or yes_forced to enable auto compilation of the gate libraries.

If you set the value to yes, the specified .lib files are compiled to .sglib file unless there is an up-to-date copy in the cache directory. However, if you set the value to `yes_forced`, any criteria for re-compilation of gate libraries are not evaluated. In such case, the specified .lib files are always compiled and they overwrite the existing .sglib file present in the cache directory.

The `AUTOENABLE_GATESLIB_AUTOCOMPILE` key value set to `yes` or `yes_forced` can be overridden by the `set_option` *enable_gateslib_autocompile* `no` command in a project file. In this case, the auto-compilation is triggered by specifying the `set_option` *enable_gateslib_autocompile* `yes` and/or `set_option` *enable_gateslib_autocompile* `yes` commands in project file. Similarly, if value of this key is `no` then you can enable automatic compilation by specifying the *enable_gateslib_autocompile* command.

DEFAULT_METHODOLOGY

Use this key to set a directory path of default methodology files.

By default, the DEFAULT_METHODOLOGY key points to the directory containing the GuideWare New_RTL methodology files.

The DEFAULT_METHODOLOGY key takes the following values:

- Absolute path of the directory where the methodology resides.
- Relative path to the SPYGLASS_HOME directory. For example, $SPYGLASS_HOME/GuideWare/New_RTL
- Relative path to the current working directory. For example, ../GuideWare/New_RTL

METHODOLOGY_SEARCH_PATH

Use this key to specify paths of directories containing custom methodologies.

All such custom methodologies appear in the drop-down list adjacent to the *User Specified Methodology* option in the *Select Methodology* dialog.

Following is the example of setting the `METHODOLOGY_SEARCH_PATH` key:

```
METHODOLOGY_SEARCH_PATH=/path1/GuideWare
```

If a directory name appearing in the path contains spaces, enclose the path within double-quotes.

UI_WAIVER_DEFAULT_REGEXP_EXCLUDE_FIELDS

Use this key to specify the sub-fields to be excluded while generating waivers with regexp enabled.

You can specify one or more of the following values:

- du
- msg
- file
- ip
- severity

Specify multiple values as a comma-separated list.

Following is the example of setting the `UI_WAIVER_DEFAULT_REGEXP_EXCLUDE_FIELDS` key:

```
UI_WAIVER_DEFAULT_REGEXP_EXCLUDE_FIELDS = du, msg, file
```

DC_DWARE_FILES_PATH

Use this key to pick designware `dware` packages from a customized path other than the path where dc has been installed.

DC_DW_FILES_PATH

Use this key to pick designware `dw` packages from a customized path other than the path where dc has been installed.

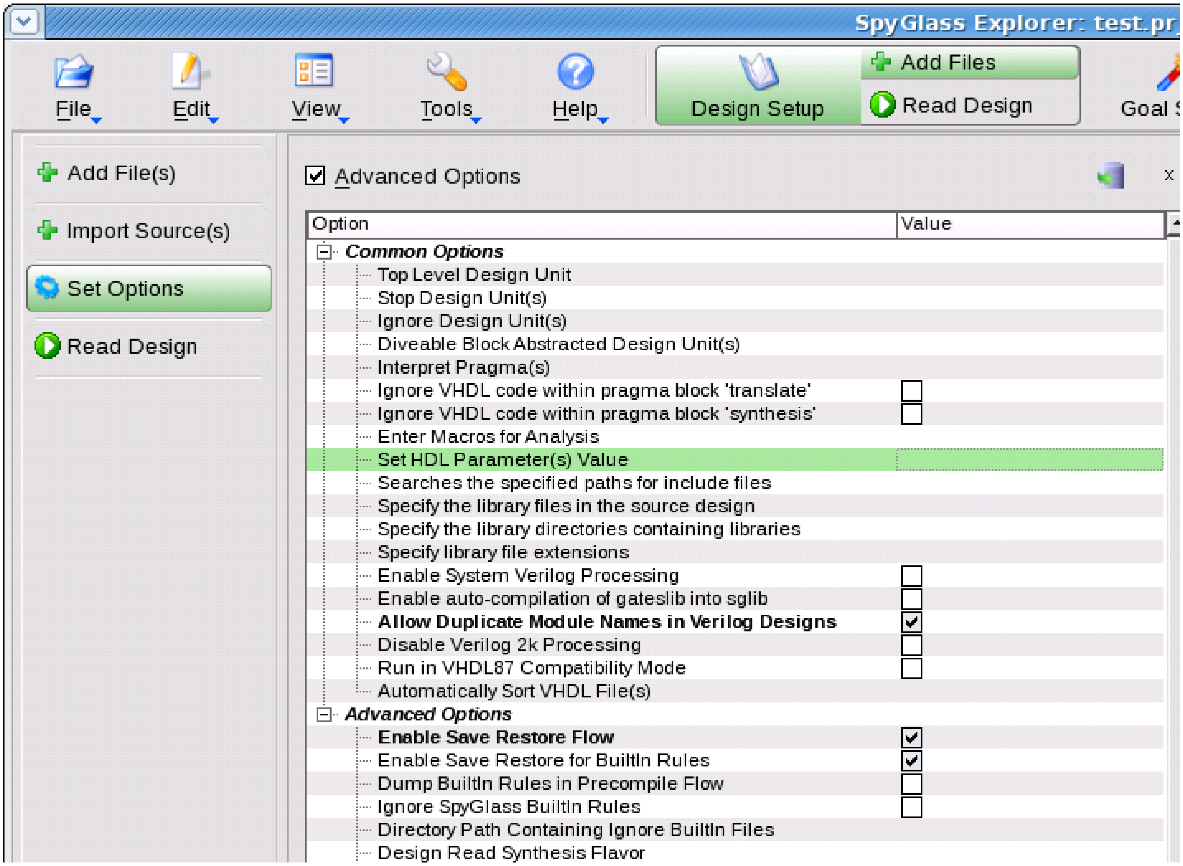
SKIP_PLATFORM_CHECK

Use this key to not perform platform checks by default. Set the SKIP_PLATFORM_CHECK configuration key to `no` to enable SpyGlass to perform platform check.

Design Read Options in SpyGlass Explorer

The design-read options enable you to specify various settings during the design setup stage. For example, you can specify top-level modules of your design and set the logical working directory.

All these options are present under the *Set Read Options* tab, as shown in the following figure:



☑ Advanced Options

Option	Value
⊟ *Common Options*	
Top Level Design Unit	
Stop Design Unit(s)	
Ignore Design Unit(s)	
Diveable Block Abstracted Design Unit(s)	
Interpret Pragma(s)	
Ignore VHDL code within pragma block 'translate'	□
Ignore VHDL code within pragma block 'synthesis'	□
Enter Macros for Analysis	
Set HDL Parameter(s) Value	
Searches the specified paths for include files	
Specify the library files in the source design	
Specify the library directories containing libraries	
Specify library file extensions	
Enable System Verilog Processing	□
Enable auto-compilation of gateslib into sglib	□
Allow Duplicate Module Names in Verilog Designs	☑
Disable Verilog 2k Processing	□
Run in VHDL87 Compatibility Mode	□
Automatically Sort VHDL File(s)	
⊟ *Advanced Options*	
Enable Save Restore Flow	☑
Enable Save Restore for BuiltIn Rules	☑
Dump BuiltIn Rules in Precompile Flow	□
Ignore SpyGlass BuiltIn Rules	□
Directory Path Containing Ignore BuiltIn Files	
Design Read Synthesis Flavor	

FIGURE 202. Design Read Options

To set a design-read option, click that option and set it to a value in the *Value* field corresponding to that option. Alternatively, use the following command in the project file to set a design-read option:

```
set_option <option-name> <value>
```

There are certain SpyGlass setting options that are determined when a project is created or saved. These options are displayed in the following table:

Option	Fixed Value	Default Value
project_directory	-	<CWD>
current_methodology	-	value of DEFAULT_METHODOLOGY in .spyglass.setup
language_mode	verilog/vhdl/mixed	mixed

NOTE: *Arguments using $ or other meta-characters that are meant to be passed directly to SpyGlass should be enclosed in curly brackets to prevent evaluation by the Tcl interpreter.*

SpyGlass design-read options are divided into the following categories:

- *Common Design-Read Options*
- *Advanced Design-Read Options*

Common Design-Read Options

Common design-read options are used to set additional options important for design analysis that are not associated with any goal.

By default, the *Show Common Options Only* check box is selected and SpyGlass displays only the commonly-used design-read options under the *Set Read Options* tab.

Whenever you save your project, SpyGlass saves only the modified design-read options in the project file by default. If you also want to save the unmodified design-read options in the project file, perform the following steps:

1. Select the *File -> Project Properties* menu option.

   The *Project Properties* dialog appears.
2. Select the *Write unmodified options in Project File* check box in the *Project Properties* dialog.
3. Click the *OK* button.

The structure of the common options section in the project file: is as follows:

```
##Common Options Section
set_option <option-name> [<option-arguments>
```

Language Mode

Use this design-read option to set a language for the current project. The default language mode is `mixed`.

Other possible language modes are `verilog` and `vhdl`.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option language_mode <verilog | vhdl | mixed>
```

Top Level Design Unit

Use this design-read option to specify a top-level module so that all design units instantiated directly or indirectly under this module are included in the scope of SpyGlass analysis.

Modules outside the hierarchy of the top-level module are considered as black boxes during SpyGlass analysis. Only syntax checking and lexical rule checking is performed on such modules.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option top <name>
```

Example

The following command defines `alu` as the top-level module:

```
set_option top alu
```

Stop Design Unit(s)

Use this design-read option to specify design units which you want to skip from SpyGlass analysis.

Such design units are considered as black boxes during SpyGlass analysis.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option stop <module-name>
```

When this command is run, SpyGlass generates the *stop_summary report*.

Format of Specifying Design Units with the stop Command

Specify design unit names in any of the following formats:

VHDL	`<entity-name>`	Skip rule-checking on the specified entity and all its architectures (in all logical libraries)
	`<entity-name>.<arch-name>`	Skip rule-checking on the specified architecture of the specified entity (in all logical libraries)
	`<lib-name>.<ent-name>`	Skip rule-checking on the specified entity and all its architectures (for the specified logical library)
	`<lib-name>.<ent-name>.<arch.name>`	Skip rule-checking on the specified architecture of the specified entity in the specified logical library
	`<lib-name>`	Skip rule-checking on all design units in the specified logical library
	`ALL`	Skip rule-checking on all design units in all logical libraries
	`ALL.<arch-name>`	Skip rule-checking on the specified architecture in all logical libraries
Verilog	`<module-udp-name>`	Skip rule-checking on the specified module or UDP
	`<lib-name>.<module-udp-name>`	Skip rule-checking on the specified module or UDP from the specified logical library

Examples of using the stop Command

Following are some examples:

- The following command specifies `alu` as the design unit to be skipped from SpyGlass analysis:

```
set_option stop alu
```

- The following command specifies block1 and block2 as the design units to be skipped from SpyGlass analysis:

  ```
  set_option stop {block1 block2}
  ```

- The following commands use wildcard expressions to specify design units:

Command	Action
`set_option stop {lib1.*}`	Skips all design units from the lib1 logical
`set_option stop {e.*}`	Skips all architectures of the e entity
`set_option stop {*.e}`	Skips all entities named e from any library

  If you specify escaped names, you must escape wildcard characters. For example, \a123* should be specified as the following to avoid any unexpected matches:

  ```
  set_option stop {\a123\*}
  ```

  In addition, the following examples have different meanings:

  ```
  set_option stop {a1*}
  set_option stop {\a1\*}
  ```

  Here, the first example matches a11, a12, and a13 while the second example matches \a1*.

Making Project File Read Only

Use this design read option to make the project file read only.

Sometimes a project file contains many environment variables and include paths. In such a case, you may not want to overwrite the project file. When working with GUI, the GUI prompts you to save the project file when exiting. If you click yes, the project file is overwritten.

To prevent overwriting the project file, add the following option to the project file:

```
set_option project_read_only yes
```

Specifying this option makes the project file as read only and the GUI does not prompt you to save any changes in this mode. You can still change values of parameters or design options and run the design, without saving them in the project file. The results of the run are saved and available when the project is loaded next time.

Also, if the value of this option is set to `yes`, SpyGlass generates the following three files in the project directory on exit:

- **<project_wdir>Sel/<project_name>/project_sources.f**: Contains all the design sources used in the project.
- **<project_wdir>Sel/<project_name>/project_options.f**: Contains all the design options that you may have set.
- **<project_wdir>Sel/<project_name>/project_goals.f**: Contains all the goal related setup, such as, parameters and constraints.

You can source these files in the original project file, to preserve any changes in the GUI for the next session.

Diveable Block Abstracted Design Unit(s)

Use this design-read option to specify a list of abstract modules such that when you double-click on these modules in the schematic, SpyGlass shows the hierarchy within these modules. By default, SpyGlass does not allow you to view the hierarchy within such modules.

Setting this design-read option is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_abstract_blocks_schematic <space-or-comma-
separated-list-of-modules>
```

Consider that you specify the following command:

```
set_option enable_abstract_blocks_schematic E2
```

Now, consider the following schematic showing the E2 module:

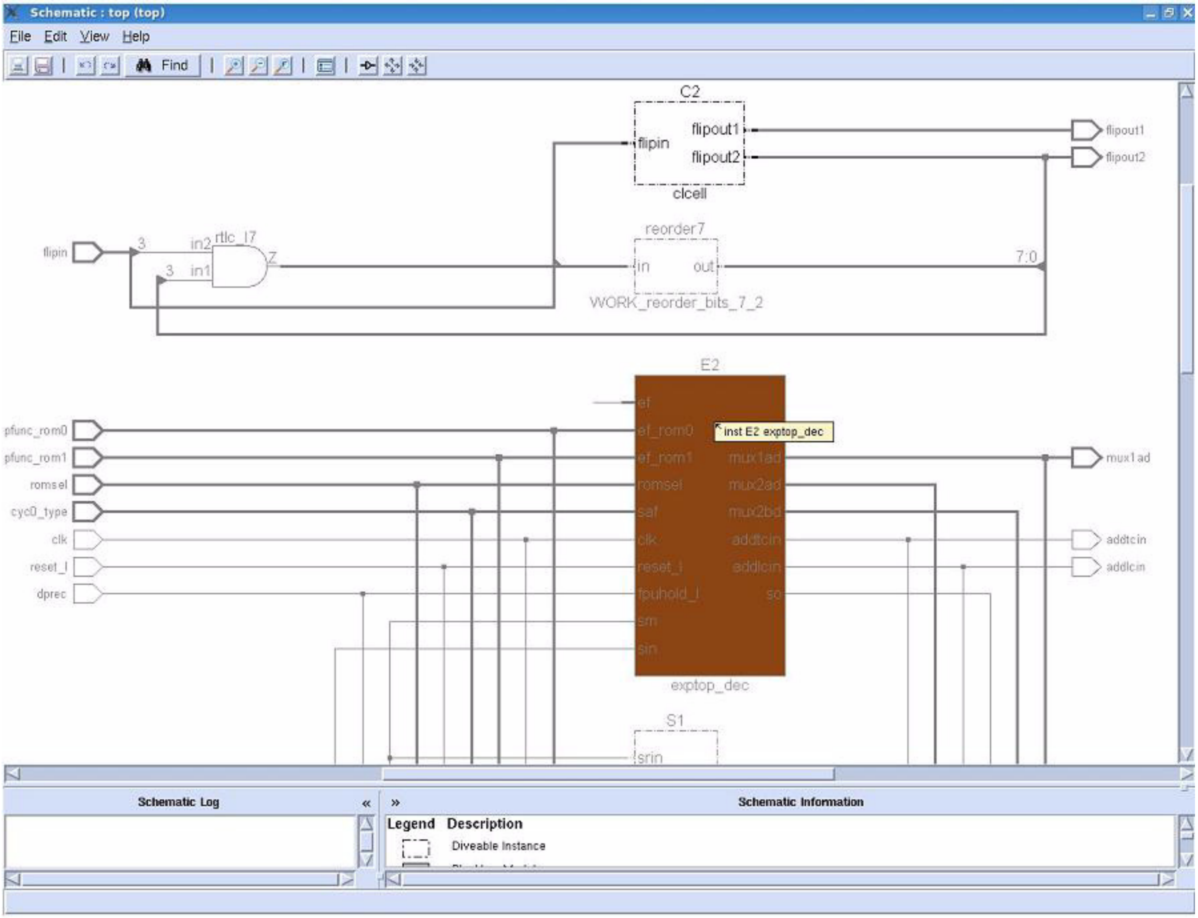


FIGURE 203. Abstract Blocks Schematic

To view the hierarchy within E2, run **Design Read** or **Run Goal** again and double-click on the E2 module in the schematic.

The following figure shows the hierarchy within E2:

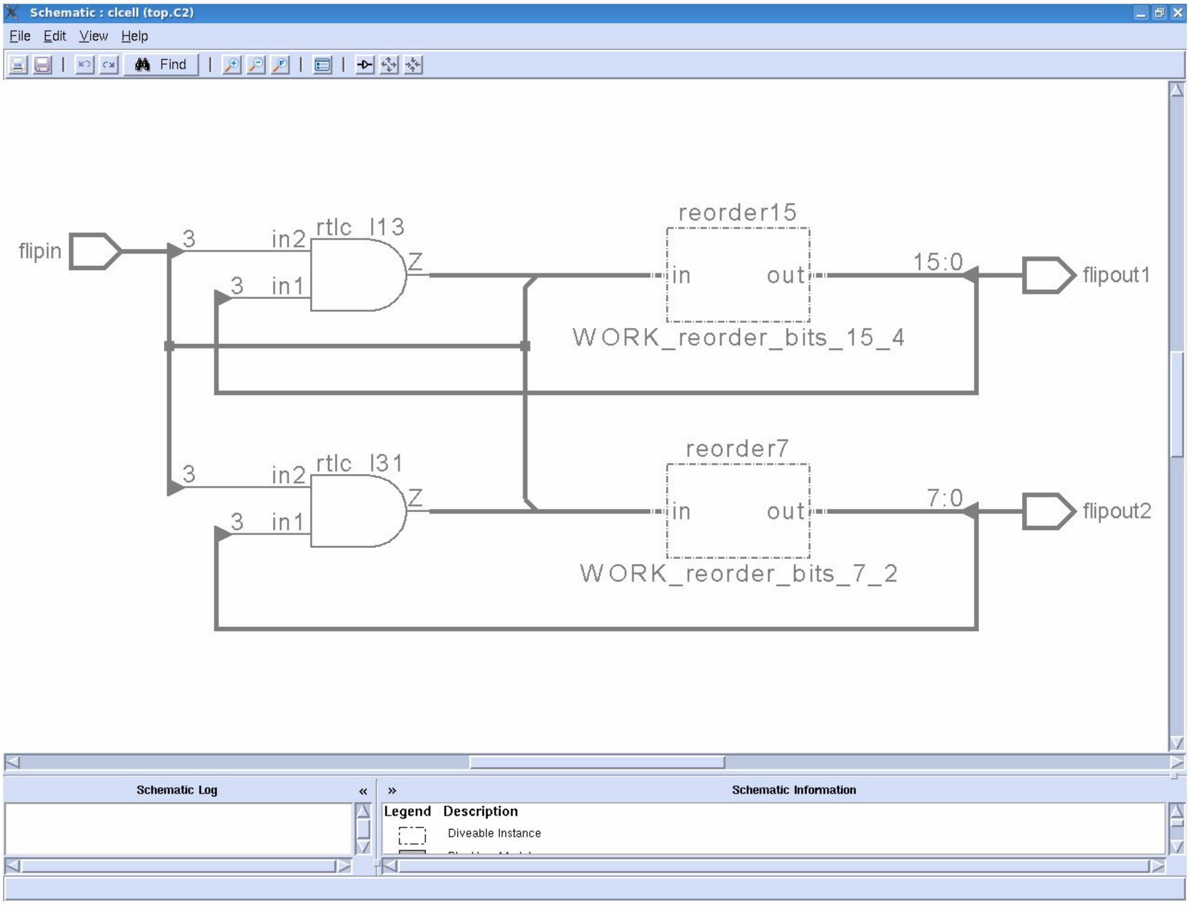


FIGURE 204. Hierarchical Schematic for E2 Module

Block Abstract Directory

Use this design read option to specify a directory in which the abstract view of a block should be saved.

By default, the directory value is set to <project_dir>/<top>/<goal>/spyglass_reports.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option block_abstract_directory <directory>
```

Examples

```
sg_shell>set_option block_abstract_directory my_abstract_out
sg_shell>current_goal soc_abstraction/lint_abstract -top top
sg_shell>run_goal
```

In the above example, an abstract view of the block for SpyGlass lint solution is generated in the following path:

```
<current_working_directory>/my_abstract_out/
spyglass_reports/abstract_view/<top>_lint_abstract.sgdc
```

Ignore Design Unit(s)

Use this design-read option to ignore the specified VHDL design units or Verilog modules during the design-read (parsing) stage.

NOTE: *For details, see Ignoring Individual Design Units.*

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option ignoredu { <design-unit-names> }
```

When this command is run, SpyGlass generates the *ignore_summary report*.

SpyGlass does not consider the ignorefile and ignoredu specifications, if they are specified in a source file that is passed to the set_option libhdlf <logical-library-name> <source-files> project file command.

If you specify an ignore specification by using the `ignorefile` or `ignoredu` commands on modules having generic parameters, the *ErrorAnalyzebbox* rule reports such modules as black boxes.

Interpret Pragma(s)

Use this design-read option to specify pragmas that should be set in your Verilog/VHDL analysis run.

For Verilog, the default pragma is set to `synopsys`, `$s`, and `$S`. For VHDL, the default pragma is set to `synopsys`, `$s`, and `$S`, and `pragma`.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option pragma { <list-of-pragmas> }
```

Interpreting Pragmas in Source Code

A design may contain pragmas, which are runtime commands for specific EDA tools. These commands are embedded as comments in the source code.

Following are some pragmas that SpyGlass recognizes during the design synthesis step:

translate_on	translate_off	full_case	parallel_case

By default, SpyGlass assumes that you are using pragmas from `Synopsys`. For example, following is the `Synopsys` pragma that SpyGlass automatically recognizes and interprets:

```
//synopsys full_case
```

Using Pragma Families

You can specify pragmas of other vendors. However, if you use the `pragma` command to define one or more pragma families, the default pragma support no longer applies. You must therefore include `pragma synopsys`, `pragma $s`, and `pragma $S` in the list of pragma families if you plan to use these default pragmas along with the newly defined pragma sources.

Using Single Pragma Family

To enable SpyGlass recognize a pragma family of a specific vendor, use the pragma command followed by the first word of the pragma family.

For example, to enable SpyGlass recognize the Quickturn pragma family in the test.v file under the PIC directory PIC, use the following project file commands:

```
read_file -type verilog PIC/test.v
set_option pragma quickturn
```

In this case, SpyGlass recognizes the following Quickturn pragmas:

```
//quickturn translate_off
//quickturn translate_on
```

Using Multiple Pragma Families

The following example enables SpyGlass to recognize the Quickturn and Synopsys pragma families:

```
read_file -type verilog PIC/test.v
set_option pragma quickturn
set_option pragma synopsys
```

Here, you need to enable the Synopsys pragma family because it is no longer recognized by default once you define a new pragma family (the Quickturn family in this case).

Ignoring All Pragma Families

To enable SpyGlass not recognize any pragma family, including the default Synopsys pragma family, specify the nopragma argument with the pragma command.

For example, to disable pragma recognition for the test.v file under the PIC directory, specify the following project file commands:

```
read_file -type verilog PIC/test.v
set_option pragma nopragma
```

Ignore VHDL code within pragma block 'translate'

Use this design-read option to ignore VHDL code within the pragma block, Synopsys `translate_off/translate_on`.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option hdlin_translate_off_skip_text yes
```

By default, this option is enabled. To disable this option, set the value of the *disable_hdlin_translate_off_skip_text* command to `yes`.

Ignore VHDL code within pragma block 'synthesis'

Use this design-read option to ignore VHDL code within the pragma blocks, Synopsys `synthesis_off/synthesis_on`.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option hdlin_synthesis_off_skip_text yes
```

By default, this option is enabled. To disable this option, set the value of the *disable_hdlin_synthesis_off_skip_text* command to `yes`.

Enter Macros for Analysis

Use this design-read option to specify macro definitions in your Verilog analysis run.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option define <list-of-options>
```

Purpose of Specifying Macro Definitions

A Verilog design may contain source code that should be compiled if certain conditions are met. Following is the example of such code:

```
'ifdef MacroName ...
'else ...
'endif
```

Alternatively, you can create text macros that are substituted with actual values during compilation. You can specify such values by using the *define* command.

Examples of using the define Option

Following are some examples:

- The following command sets the `State0` macro to `3`:

  ```
  set_option define {State0=3}
  ```

- The following command sets the `State0` and `State1` macros to `3` and `5`, respectively:

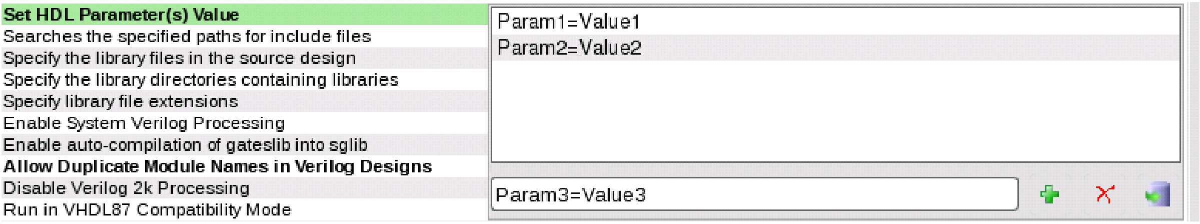
  ```
  set_option define { {State0=3} {State1=5} }
  ```

Set HDL Parameter(s) Value

Use this design-read option to specify parameters used during Verilog/ VHDL analysis run.

To specify parameters, perform the following steps:

1. Click ⬇ adjacent to this option. The following dialog appears:



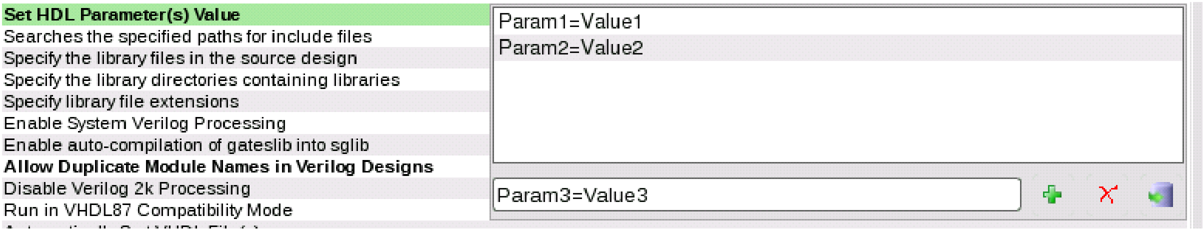
Set HDL Parameter(s) Value
Searches the specified paths for include files
Specify the library files in the source design
Specify the library directories containing libraries
Specify library file extensions
Enable System Verilog Processing
Enable auto-compilation of gateslib into sglib
Allow Duplicate Module Names in Verilog Designs
Disable Verilog 2k Processing
Run in VHDL87 Compatibility Mode

Param1=Value1
Param2=Value2

Param3=Value3

FIGURE 205. Specify HDL Parameter(s)

2. In the above dialog, click the *Add* button. A new row appears in the dialog, as shown in the following figure:



Set HDL Parameter(s) Value
Searches the specified paths for include files
Specify the library files in the source design
Specify the library directories containing libraries
Specify library file extensions
Enable System Verilog Processing
Enable auto-compilation of gateslib into sglib
Allow Duplicate Module Names in Verilog Designs
Disable Verilog 2k Processing
Run in VHDL87 Compatibility Mode

Param1=Value1
Param2=Value2

Param3=Value3

FIGURE 206. Add HDL Parameters

3. In the newly added row, specify the generic/parameter name in the *Key* column and the corresponding value in the *Value* column.
4. Repeat step 2 to set more generics/parameters.
5. Click the *Close* button to close the dialog.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option param <name>
```

In a normal design hierarchy, it is possible to define design units that can be parameterized, where the value of the parameter (in Verilog) or generic (in VHDL) is defined when the design unit is instantiated in the hierarchy.

If you then choose to run SpyGlass from the top of such design unit, this value is obviously not visible to SpyGlass. To be able to define a value for each parameter or generic, you can use this option.

Defining a Value for a Generic/Parameter

To define a value for a parameter, specify an instance of the parameter you want to define along with the value you want to set. In VHDL, entity_name.generic_name is considered while in Verilog module_name.parameter_name is considered.

The param command supports only integer constants, binary, octal, decimal, and hexadecimal. However, SpyGlass reports an error, if the parameters are overridden with a string value.

For Verilog, you can specify a parameter in the following format:

```
<width>'<base> <value>
```

Where:

- <width> and <base> are optional.
- <base> can take the following values:
  - ❒ b for binary,
  - ❒ o for octal
  - ❒ d for decimal
  - ❒ h for hexadecimal
- <value> is a valid number for the specified base.

For VHDL, you can specify a parameter in the following format:

```
<base>#<value>#
```

Where:

- <base> is optional.
- <base> can take following values:
  - ❒ 2 for binary
  - ❒ 8 for octal
  - ❒ 10 for decimal
  - ❒ 16 for hexadecimal
- <value> is a valid number for the specified base.

NOTE: *Do not override parameters with a string value.*

For example, to set the VHDL generic width in the entity control to 8, specify the following command in the project file:

```
set_option param { control.width=8 }
```

To set the Verilog parameter limit in the module block1 to 4, specify the following command in the project file:

```
set_option param { block1.limit=4 }
```

Overriding Generics in VHDL

In VHDL, you can override the following type of values:

- Integer, positive, or a natural value, as shown in the following examples:

Integer value	`set_option param { entity_name.generic_name=20 }`
Binary value	`set_option param { entity_name.generic_name=2#10# }`
Octal value	`set_option param { param entity_name.generic_name=8#76# }`
Decimal value	`set_option param { entity_name.generic_name=10#93# }`
Hexadecimal value	`set_option param { entity_name.generic_name=16#AF# }`

- Boolean/enum value, as shown in the following examples:

Boolean value	`set_option param { entity_name.generic_name true }`
Enum value	`set_option param { entity_name.generic_name red }`

- Bits, as shown in the following examples:

Bit value	`set_option param { entity_name.generic_name="0" }` Note that if double quotes around 0 are not present, the value is considered as the decimal value 0.
bit_vector value	`set_option param { entity_name.generic_name="0000" }`

- std_logic, as shown in the following examples:

std_logic	`set_option param { entity_name.generic_name="1" }` Note that if double quotes around 1 are not present, the value is considered as the decimal value 1.
std_logic_vector	`set_option param { entity_name.generic_name="0000" }`

- String value, as shown in the following example:

```
set_option param { entity_name.generic_name=ABC }
```

Overriding Parameters in Verilog

In Verilog, you can override parameter values with the following type of values:

- Integer

  When you override a parameter value with an integer value or a value in a valid based number format, it is considered as overriding the parameter value with an integer value.

  Following are some examples:

Integer	`set_option param { module_name.parameter_name=123 }`
Binary	`set_option param { module_name.parameter_name=16'b10101010 }`
Octal	`set_option param { module_name.parameter_name=32'o657 }`
Decimal	`set_option param { module_name.parameter_name=16'd94 }`
Hexadecimal	`set_option param { module_name.parameter_name=32'hAB4 }`

NOTE: *If you pass a value in an invalid based number format, SpyGlass reports a violation.*

- String

  When you override a parameter with a value containing alphabets and/or special characters (with or without numerical values), it is considered as overriding with a string value.

  Following are some examples:

```
set_option param { module_name.parameter_name=12ab3 }
set_option param { module_name.parameter_name=abc }
```

Re-definition of a Generic/Parameter at Module Instantiation

If a generic/parameter is re-defined at module instantiation, the value specified during module instantiation is given preference over the value specified by the `param` command.

For example, consider the following code (line numbers are also highlighted):

```
module sr(srin, clk, srout);

1.   parameter SIZE1 = 8;
2.   input srin, clk;
```

```
3.   output srout;
4.   reg [SIZE1:0] sr, srout;
5.     integer i;
6.   always @(posedge clk)
7.     for ( i = 0 ; i < SIZE1 ; i=i+1 )
8.     if ( i==0 ) sr[i] <= srin;
9.     else if ( i < SIZE1-1 ) sr[i] <= sr[i-1];
10.     else
11.       srout <= sr[i-1];
12. endmodule
13. module srtop(in, clk, out);
14.   input in, clk;
15.   output out;
16. sr #(32)__sr1(.srin(in),.clk(clk),.srout(out));
17. sr __sr2(.srin(in),.clk(clk),.srout(out));
18. endmodule
```

In line 16 of the above example, the value, `8`, of the `SIZE1` parameter is overridden with the value `32`. However, for the functionality specified in line17, following cases may arise:

- If the `param` command is specified, SpyGlass overrides the value, `8`, of the `SIZE1` parameter with the specified value.
- If the `param` command is not specified, SpyGlass retains the value, `8`, of the `SIZE1` parameter.

Searches the specified paths for include files

Use this design-read option to include a relative path name of a directory that contains include files.

NOTE: *This option is available only if your design files also contain Verilog files.*

Include files contain code that defines some frequently performed actions. You can include such files at required locations in your code by using the `'include` compiler directive.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option incdir <path-name>
```

NOTE: *It is recommended to use relative path names instead of absolute path names to ensure portability of include files.*

If your `incdir` directory name includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as a literal. For example, if your incdir directory name is abc*d, you need to refer to it as {{abc*d}}. However, if you want to refer to two directories, for example abc1d and abc2d, you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}}, in this case. For details on pattern matching support, see *Pattern Matching Across Features.*

SpyGlass first searches the current directory for the `'include` files. It then searches the directories specified by the `incdir` option in the sequence in which you list them. SpyGlass returns an error message and terminates if it cannot find all the required `'include` files in your directory structure.

While specifying relative path names using the `incdir` command, ensure that no duplicate file names are present in the directory list. The file corresponding to the second instance of the duplicate file name is never read because SpyGlass searches the directory structure until it finds the first instance of the file and then stops.

Specify the library files in the source design

Use this design-read option to specify Verilog library files used in a source design.

To specify library files, click ⤓ adjacent to this option. The following dialog appears:

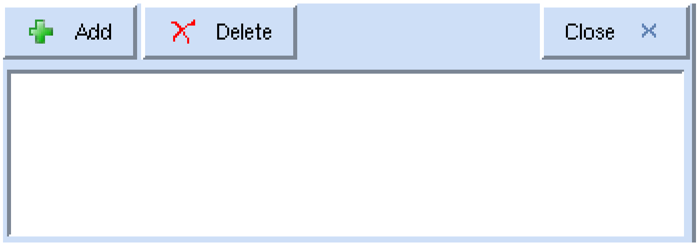


FIGURE 207. Specify Library Files

In the above dialog, click *Add* and browse to the directory containing library files. When you are done, click *Close*.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option v {space-separated list of lib names}
```

Modules or User-Defined Primitives (UDPs) missing from your Verilog source code are usually present in a single library file or in files stored in a library directory. In such cases, use the *v* command (to locate the library) or the *y* command (to locate the library directory) so that SpyGlass can compile your Verilog design correctly.

SpyGlass checks the current directory for such libraries first. If it cannot find them, SpyGlass searches the path you specify with the *v*/*y* command.

Key Points

- This option is available only if your design files also contain Verilog files.
- If your filename includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your filename is "abc*d", you need to refer to it as {{abc*d}}. However, if you want to refer to two files, for example "abc1d" and "abc2d", you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}} in this case. For details on pattern matching support, see *Pattern Matching Across Features*.
- By default, SpyGlass performs rule-checking on all the cells specified by the *v* option. To disable rule-checking on such cells, specify the *ignorelibs* option.

Specify the library directories containing libraries

Use this design-read option to specify directories containing libraries to compile your Verilog design correctly.

To specify directories, click ⬇ adjacent to this option. The following dialog appears:

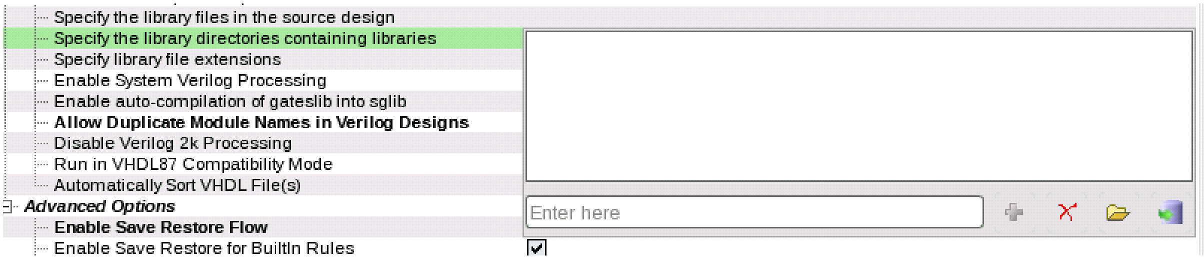

Specify the library files in the source design
Specify the library directories containing libraries
Specify library file extensions
Enable System Verilog Processing
Enable auto-compilation of gateslib into sglib
Allow Duplicate Module Names in Verilog Designs
Disable Verilog 2k Processing
Run in VHDL87 Compatibility Mode
Automatically Sort VHDL File(s)
Advanced Options
Enable Save Restore Flow
Enable Save Restore for BuiltIn Rules
Enter here
☑

FIGURE 208. Specify Library Directories

In the above dialog, click *Add* and browse to the directory containing the required library files. When you are done, click *Close*.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option y { space-separated path-names of directories }
```

NOTE: *You must also specify library file extensions using the libext command to enable SpyGlass to read library files.*

Modules or User-Defined Primitives (UDPs) missing from your Verilog source code are usually present in a single library file or in files stored in a library directory. In such cases, use the v command (to locate the library) or the y command (to locate the library directory) so that SpyGlass can compile your Verilog design correctly.

SpyGlass checks the current directory for such libraries first. If it cannot find them, SpyGlass searches the path you specify with the *v*/*y* command.

NOTE: *If your directory name includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your directory name is "abc*d", you need to refer to it as {{abc*d}}.*
However, if you want to refer to two directories, for example "abc1d" and "abc2d",

*you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}} in this case. For details on pattern matching support, see Pattern Matching Across Features.*

NOTE: *By default, SpyGlass performs rule-checking on all the cells specified by the y command. To disable rule-checking on such cells, use the ignorelibs command.*

Specify library file extensions

Use this design-read option to specify the library file extension. You can add multiple library extensions by specifying a space-separated list of extensions in the text field provided.

NOTE: *You must specify the library file extension when specifying the library file directory.*

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option libext { space-separated list of extensions }
```

Enable SystemVerilog Processing

Use this design-read option to enable or disable SystemVerilog compatibility mode. By default, this option is disabled.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enableSV <yes |no>
```

Enable auto-compilation of gateslib into sglib

Use this design-read option to compile the Synopsys liberty files (.lib files) automatically to a SpyGlass-compatible format library file (.sglib files).

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_gateslib_autocompile <yes | no>
```

If there is a pre-existing .sglib file, such .sglib file is not recompiled, if you change the value of the `set_option include_opt_data <true|false|yes|no>` command in the subsequent runs.

However, if the .sglib file is recompiled because of reasons, such as .lib files changed, the `include_opt_data` command is taken into consideration.

Allow Duplicate Module Names in Verilog Designs

Use this design-read option to allow duplicate module/UDP definitions.

Only the last found module/UDP definition is processed, and earlier definitions with the same name are ignored.

By default, duplicate module definitions result in STX_589 syntax error.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option allow_module_override <yes |no>
```

Disable Verilog 2k Processing

Use this design-read option to enable or disable Verilog 2001 compatibility mode. By default, this preference is disabled.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option disablev2k yes
```

Run in VHDL87 Compatibility Mode

Use this design-read option to enable or disable VHDL (IEEE Std 1987) compatibility mode.

By default, this preference is disabled, and SpyGlass considers VHDL (IEEE Std 1993) as the VHDL language standard.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option 87 yes
```

Automatically Sort VHDL File(s)

Use this design-read option to sort VHDL files before analyzing them and print the sorted file order in the log file.

NOTE: *The Automatically Sort VHDL File(s) option is available only when your design files also include VHDL files for analysis.*

For a VHDL design to compile correctly, it must be analyzed in the correct order (that is, lower-level dependent design units and libraries must be analyzed before the top-level or primary units).

You can select the following sorting options from GUI:

- *DU based sort* (default): Select this option to sort the VHDL files by design units.

  Selecting this option is equivalent to the following project file command:

  ```
  set_option sortmethod du
  ```

- *Lexical*: Select this option to quickly tokenize a given set of VHDL files and identify dependency of a file on remaining files. The dependency information can then be used to build a sorted list where dependent files are compiled prior to the files using them.

  Selecting this option is equivalent to the following project file command:

  ```
  set_option sortmethod lexical
  ```

NOTE: *Use the* `lexical` *option only when you encounter a problem with DU-based sorting. Lexical-based sorting is not able to handle dependency specified through the configuration file and so on. Therefore, VHDL sorting may not be correct is such cases.*

- *None*: Select this option if you do not want VHDL files to be sorted. When you select this option, you should specify source files in the correct order for VHDL compilation.

  Selecting this option is equivalent to the following project file command:

  ```
  set_option sort no
  ```

If you want SpyGlass to automatically sort design units, specify the following project file command:

```
set_option sort yes
```

Printing Sorted VHDL Files

To view the order of sorted VHDL files, specify the following project file command after specifying the sort command:

```
set_option print_sortorder_only yes
```

NOTE: *SpyGlass prints the order automatically in the log file when you use the* sort *command. The* print *command is retained for backward compatibility.*

Cases when the sort Command is not Used

If you do not use the sort command, you must specify source files in correct order of VHDL compilation.

The order must have design unit definitions preceding any VHDL files that use the design units, which is standard VHDL ordering. Following is the example of an explicitly defined order:

```
read_file -type hdl bottom_ent.vhd bottom_arch.vhd
upper_ent.vhd upper_arch.vhd
```

Where, the upper design unit contains instances of the bottom design unit.

Points to be Noted

Please note the following points:

- Use the sort command only if you do not know the correct design order. However, there are design configurations in which the sort command cannot reliably infer the correct order, no matter how good the sort algorithm is.
- If you use the *.vhd wild card without the sort option, SpyGlass compiles the files in the order they are returned from the shell (that is, alphabetically). The dependency order determined by SpyGlass for the design is printed in the log file.
- During sort operation, if SpyGlass is unable to determine the files to be updated, SpyGlass reports an error indicating that one or more files

need to be recompiled. To resolve this issue, delete your work directory and any library directories and rebuild them.

- The specified VHDL libraries are checked for existence before SpyGlass attempts to determine any dependency. If any such libraries do not exist, SpyGlass reports an error and exits.

Advanced Design-Read Options

To view the advanced design-read options, de-select the *Show Common Options Only* check box.

Define Verbosity Level for Log File

Enables you to control the verbosity of the log file thereby ensuring that the spyglass.log file is easily readable and content is controllable.

This option can have take one of the following four values: 0, 1, 2, or 3.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option verbosity <0 | 1 | 2 | 3>
```

Following table describes the various options for the verbosity option:

Level	Output
3	Represents the output with DEBUG option set. Generates the maximum information in the spyglass.log file.
2	Represents the default output in the current scenario.
1	Represents the reasonable output
0	Represents the minimal output

Enable Abstract Blocks Schematic

Enables schematic debugging of abstracted modules. This option accepts abstracted module names, for which you want to view schematic to debug issues in the abstracted IPs of SoC.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_abstract_blocks_schematic
<abstract_module_names>
```

Enable Save Restore Flow

Use this design-read option to enable the design save-restore feature.

By default, the save-restore feature is on. To disable this feature, set the value of this option to `no`.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_save_restore <yes | no>
```

Enable Save Restore for BuiltIn Rules

Use this design-read option to view built-in messages during the restore run that were reported during the save run.

By default, built-in message that were reported in the save run are also reported in the restore run. However, if you do not want to view such messages during the restore run, set the value of this option to `no`.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_save_restore_builtin <yes | no>
```

SpyGlass saves built-in messages (parsing, elaboration, and synthesis) reported during the save run as a part of the saved design database. These messages are then displayed during the restore run if the `enable_save_restore_builtin` option is set to `yes`.

Dump BuiltIn Rules in Precompile Flow

Use this design-read option to view built-in messages while using precompiled RTL dump. These messages are same that were reported when that precompiled RTL dump was generated.

By default, SpyGlass reports such built-in messages when you use the precompiled dump created earlier. If you do not want to view such messages, set the value of this option to No.

Setting this field is equivalent to specifying the following command in the

SpyGlass project file:

```
set_option dump_precompile_builtin <yes | no>
```

Built-in message reported during the generation of RTL precompiled dump are stored as a part of that dump in a file. You can later view these messages during usage of that precompiled dump if the dump_precompile_builtin option is set to yes.

NOTE: *Please note the following points:*

- *Messages may change across SpyGlass releases. Therefore, you are recommended to precompile your libraries for each release in which you want to use the precompiled dump.*
- *If you want some product-specific built-in checks to be reported on the usage of precompiled design units, then during the RTL precompilation step, you must set the value of the AUTOENABLE_BUILTIN_CHECKS_FOR_POLICY configuration key to an appropriate product name in the .spyglass.setup file.*

Ignore SpyGlass BuiltIn Rules

Use this design-read option to ignore certain built-in rules reported during SpyGlass analysis.

When you set this option to *Yes*, the *Directory Path Containing Ignore BuiltIn Files* option is enabled using which you can specify a file mentioning built-in rules to be ignored.

By default, built-in rules to be ignored are picked from the ignore_builtin file present under the $SPYGLASS_HOME/auxi/policy_data/spyglass/ directory.

By default, this option is set to No, and built-in rules are not ignored during SpyGlass analysis.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option ignore_builtin_rules <Yes | No>
```

Directory Path Containing Ignore BuiltIn Files

Use this design-read option to specify the path of a directory that contains built-in files.

Each built-in file contains the `-ignorerules` command to specify the built-in rules to be ignored, as shown in the following example:

```
-ignorerules builtin_rule1
-ignorerules builtin_rule 2
-ignorerules builtin_rule3
```

By default, the directory path is set to $SPYGLASS_HOME/auxi/policy_data/spyglass/ that contains the ignore_builtin file.

You can specify a different directory path containing your own built-in files for each language, such as Verilog, VHDL, and mixed. However, you must ensure that the name of all such built-in files should have the following naming convention:

ignore_builtin-<*language*>.spq

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option ignore_builtin_spqdir <path>
```

design-read Synthesis Flavor

Use this design-read option to specify the type of synthesis you wish to perform during the design-read process.

SpyGlass considers this option only if you enable synthesis during the design-read process. To enable synthesis, select the *Synthesize Netlist* option under the *Run design-read* tab or specify the following command in the project file:

```
set_option designread_enable_synthesis yes
```

You can specify the type of synthesis by specifying the following command in the project file:

```
set_option designread_synthesis_mode <mode>
```

Where, <mode> can accept any of the three values, as specified in the following table:

Mode	Description
base	Executes the CLASSIC mode rules only (which run on non-optimized netlist) and disables rules of EST and ESYNTH mode.
opt	Executes the ESYNTH mode rules only (which run on optimized netlist) and disables rules of CLASSIC and EST mode.
techmap	Executes the EST mode rules only (which run on optimized and technology-mapped netlist) and disables rules of CLASSIC and ESYNTH mode.

Goal Run Synthesis Flavor

As part of the goal run, the synthesis mode is determined by the rules in `<goal>.spq` file.

If the synthesis modes are conflicting, spyglass reports the following violation message with FATAL severity:

```
ERROR [294] Rules corresponding to multiple synthesis mode
selected in current run.
```

If above error is flagged, then the given rule set is not compatible, and you need to run the compatible product/rule set together.

Though majority of the product's rules use classic synthesis, the products that use other modes are mentioned below:

- *SpyGlass Power Estimation and Reduction*: This product uses the EST mode. The rules of this product are compatible with the following modes:
  - ❒ base
  - ❒ opt
  - ❒ optimized and techmap
  - ❒ All of the above
- *SpyGlass SEC*: This product uses the ESYNTH mode by default.
- *SpyGlass DFT and SpyGlass DFT DSM:* These products use classic mode by default. Most of their rules are compatible with base and opt mode, and only a few are compatible with just opt mode.

The behavior of multi-mode rule is driven by the mode of the other rules it is run with. For example, consider a rule, R1, is compatible with Classic and Esynth synthesis modes and rule, R2, is compatible with Esynth and techmap synthesis mode. Also, consider that R1 and R2 are run together, then the run proceeds with the Esynth synthesis mode since it is common to both the rules.

Enable Incremental Mode for All Goals

Use this design-read option to enable reporting of incremental messages so that you can compare results of a previously run goal with the current goal.

By default, this option is set to *No*, and SpyGlass does not perform incremental reporting of messages.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option report_incr_messages <Yes | No>
```

Logical Working Directory

Use this design-read option to specify a logical working directory for compilation of Verilog/VHDL libraries.

By default, the working directory is set to the WORK directory in the current directory.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option work <value>
```

After setting this option, ensure that you define a logical library of the same name in the *HDL Libraries* section under the *Add Design Files* tab.

For example, to use the working library `work2` present in the my_work_lib directory, specify the following commands:

```
set_option work work2
set_option lib work2 /u/<user_name>/my_work_lib
```

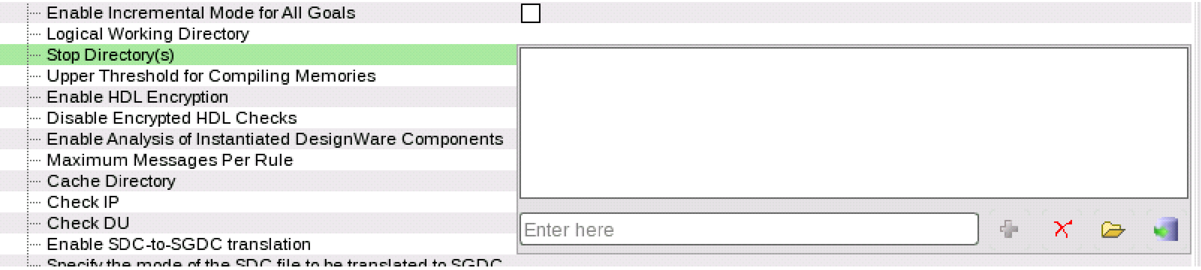
NOTE: *Libraries are platform-specific. For example, you cannot compile on Sun platform, use a library on Linux platform. If you switch platforms, you need to delete the working library and other user-defined VHDL libraries and rebuild them. Atrenta supplies versions of all four default libraries for all supported platforms. These should not need to be rebuilt.*

Stop Directory(s)

Use this design-read option to specify directories that you want to skip for rule-checking during SpyGlass analysis.

To specify a directory that you want to skip for rule-checking, perform the following steps:

1. Select the *Stop Directory(s)* option. The following text box appears in the *Value* column:



Enable Incremental Mode for All Goals □
Logical Working Directory
Stop Directory(s)
Upper Threshold for Compiling Memories
Enable HDL Encryption
Disable Encrypted HDL Checks
Enable Analysis of Instantiated DesignWare Components
Maximum Messages Per Rule
Cache Directory
Check IP
Check DU
Enable SDC-to-SGDC translation
Specify the mode of the SDC file to be translated to SGDC

Enter here

FIGURE 209. Stop Directory(s)

2. Click ⬇. This displays the following dialog:

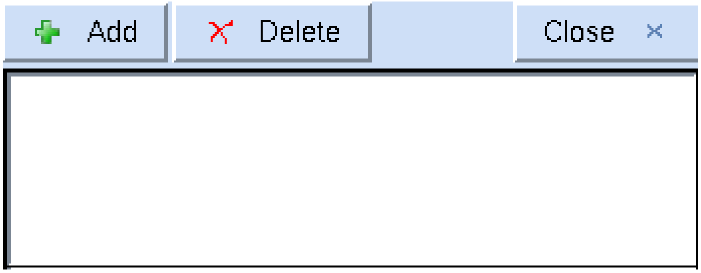


FIGURE 210. Add Directories

3. In the above dialog, click the *Add* button. This displays the *Select Directory* dialog.
4. In the *Select Directory* dialog, select the directory that you want to skip from rule-checking.
5. Click the *OK* button to close the *Select Directory* dialog.

After performing the above steps, the specified directory is not considered for rule-checking.

If a file added under the *Add Design Files* tab exists in the directory selected by performing the above steps or if you later add a file from this directory, the ✋ icon appears before the name of that file to indicate that this file is not considered for rule-checking.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option stop_dir <name>
```

Upper Threshold for Compiling Memories

Use this design-read option to set an upper threshold for compiling memories. You must specify a positive integer value in this field. By default, the value of this option is set to 4096.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option mthresh <value>
```

If the individual net/variable size (bit-count) is more than the specified threshold, the module is not compiled and is treated as a black box and the SYNTH_ERROR[5273] message is reported. To solve the problem, specify the threshold value greater than or equal to the value displayed in the error message. However, this higher value may not be feasible due to system memory size limitations.

NOTE: *If handlememory option is specified, the individual memory size is reduced memory size.*

Synthesizing memories (2-dimensional arrays) in RTL designs has always been a resource- and time-consuming task; some designs even run out of system memory. When the memory is synthesized, each bit location of the synthesized memory is represented by a flip-flop or latch in the synthesized netlist. This can easily consume an appreciable amount of system memory, resulting in design capacity problems.

To control compilation of memories in a module, SpyGlass does not compile modules where the total bit-count (post-elaboration) of the memories in a module exceeds the value specified in this field.

Enable Handle Memories

Use this design-read option to enable processing of memories in an optimized manner. By default, the value is set to *No*.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option handlememory <yes | no>
```

Enable HDL Encryption

Use this design-read option to enable the encryption of VHDL/Verilog libraries during compilation. By default, the encryption feature is disabled.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_hdl_encryption <yes | no>
```

By default, RTL rule-checking on encrypted precompiled design units is enabled. Any highlighting information within such modules is shown on the library module boundary only. You can disable RTL rule-checking on such design units by setting the *Disable Encrypted HDL Checks* option to `yes`.

NOTE: *When you specify the* `enable_hdl_encryption` *command, a precompile dump is created on both 32-bit and 64-bit platforms irrespective of the platform on which SpyGlass is run. By default, the dump_all_modes command is enabled with the* `enable_hdl_encryption` *command.*

Disable Encrypted HDL Checks

Use this design-read option to disable RTL rule-checking on encrypted design units.

When you set this option to *Yes*, SpyGlass internally removes messages coming from post-flattening rules that do not provide any highlighting information in the schematic.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option disable_encrypted_hdl_checks <yes | no>
```

NOTE: *Please note the following points:*

- *The Enable RTL Checking of Precompiled HDL Libraries option does not have any effect on the above rule-checking behavior for encrypted modules.*
- *All design units instantiated in an encrypted design unit are treated as encrypted even if they are not encrypted.*
- *Encryption using any robust encryption algorithm is outside the scope of SpyGlass.*

Enable Analysis of Instantiated DesignWare Components

In case your design has instances of Synopsys® DesignWare® components, set this option to map these components in terms of technological gates.

By default, this option is set to *No*.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option dw <yes | no>
```

You can also use the *dw_options* command to specify multiple options for the DesignWare components.

The following table explains the results when different values are specified for the *dw* and *dw_options* commands:

Value of the dw option	**Value of the dw_options option**	**SpyGlass Behavior**
yes	{hide_all_dw_violations}	It waives all violations across all rules using the waive –ip -delete_internal_use_only command and doesn't report anything in the waiver report
yes	{report_violations, enable_waiver} OR {enable_waiver} OR {enable_waiver, report_violations}	Enables DW waivers

Hierarchical SGDC Modes

Use this design-read option to specify a hierarchical SGDC flow. You can specify the required flow by selecting any of the following options from the adjacent drop-down list.

- *Generation Mode*

  Selecting this option enables hierarchical migration of block-level SGDC commands to the chip-level.

- *Validation Mode*

  Selecting this option enables validation of hierarchically migrated block-level SGDC commands to the chip-level.

- *None*

  (Default) Selecting this option means that none of the flow is specified.

  In this case, the *Synthesize Netlist* option (under *Run design-read* tab) is enabled. You can select this option to enable synthesis during the design-read process.

By default, the *Synthesize Netlist* option is disabled during *Generation Mode* and *Validation Mode* as synthesis always occurs in these modes.

Maximum Messages Per Rule

Use this design-read option to set an upper limit for messages to be reported per rule.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option lvpr <value>
```

Cache Directory

Use this design-read option to specify a cache directory where you want library compilation to be performed.

By default, the value of this option is set to `spyglass_cache`.

SpyGlass considers <cwd>/spyglass_cache as the default cache directory if you leave this field blank.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option cachedir <directory-name>
```

Exit on Detecting Blackboxes in the Design

Use this design-read option to force SpyGlass to stop analysis and exit if a black box is found in a design.

By default, the option is set to *No*.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option nobb yes
```

Enable RTL Checking of Precompiled HDL Libraries

Use this design-read option to perform design-read checks (including lexical checks) on precompiled HDL libraries.

By default, this option is set to *no*.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option hdllibdu yes
```

Check IP

Use this design-read option to specify design units on which rule-checking should be done. For the rest of the design units, SpyGlass skips rule-checking.

When you specify this command, SpyGlass not only considers the design unit specified by this command, but also considers all those design units starting from the top in the hierarchy till the design unit specified by this command.

For example, consider the design hierarchy shown in the following figure:

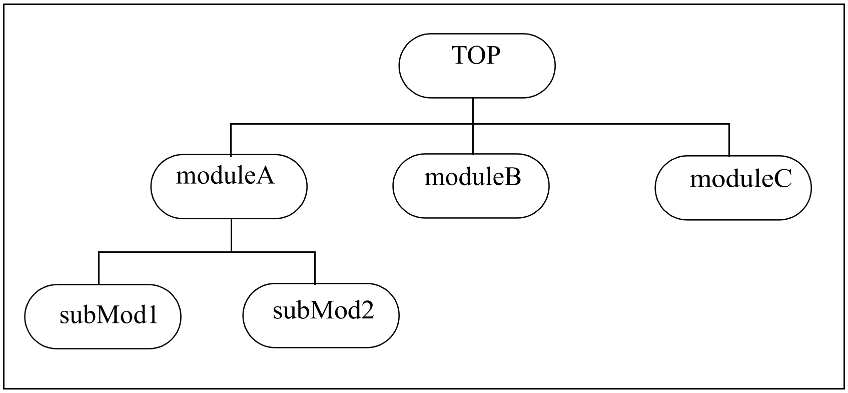


FIGURE 211. Sample Design Hierarchy

Now, if you set the `checkip` command to `ModuleA`, SpyGlass synthesizes the modules, `TOP`, `ModuleA`, `subMod1`, and `subMod2`, and consider only these modules for rule-checking.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option checkip <ip-name>
```

Check DU

Use this design-read option to specify a design hierarchy for which rule-checking should be done for interconnects.

All design units instantiated under the design unit specified by this option are treated as grey boxes. For the rest of the design units, SpyGlass skips rule-checking.

When you specify this option, SpyGlass not only considers the design unit specified by this option for rule-checking, but also considers all those design units starting from the top in the hierarchy till the design unit specified by this option.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option checkdu <du-name>
```

Consider the design hierarchy shown in the following figure:

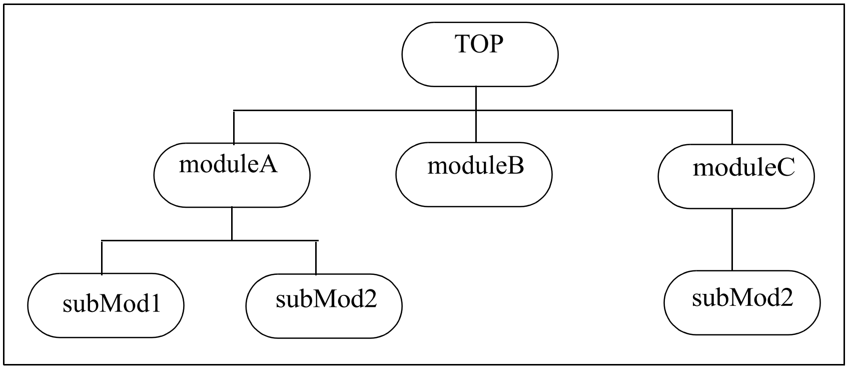


FIGURE 212. Sample Design Hierarchy

Now, if you set the `checkdu` option to `moduleA`, SpyGlass synthesizes the modules, `TOP` and `ModuleA`, and considers only these modules for rule-checking.

Consider another design hierarchy, as shown in the following figure:

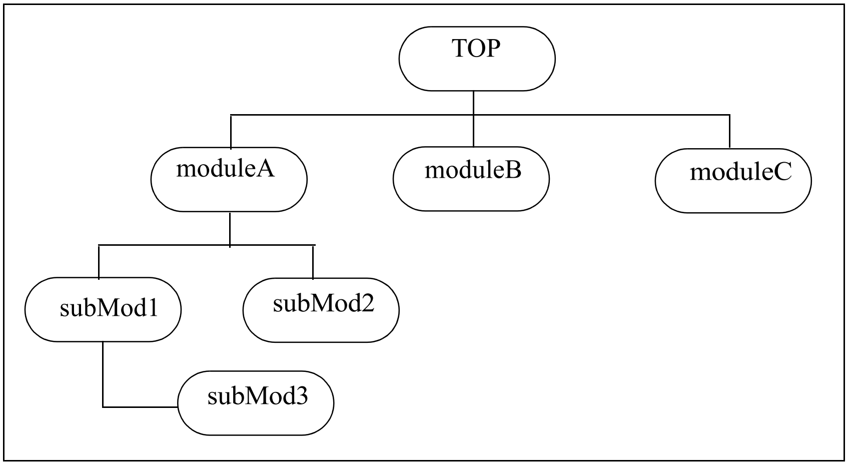


FIGURE 213. Sample Design Hierarchy

Based on the above design hierarchy, consider that you specify the following commands:

```
set_option checkdu moduleA
set_option checkip subModule1
```

In this case, SpyGlass synthesizes the modules, `TOP`, `moduleA`, `subModule1`, and `subModule3`, and consider only these modules for rule-checking.

NOTE: *The checkdu option is given preference over the checkip option, if these options are specified for the same design unit.*

Please note the following points for the `checkdu` and `checkip` options:

- The *stop*, *stopfile*, and *stopdir* options are given preference over the *checkdu* and *checkip* options. For example, SpyGlass ignores the *checkdu* and/or *checkip* options on design units for which the `stop` option is specified.
- If a design unit specified with the *checkdu* and *checkip* options contains an overlapping spanning tree (hierarchical tree rooted at that design unit), the option with the design unit at a higher hierarchical level (in the tree) is given higher priority.

The *checkip* and *checkdu* options cannot be used to skip synthesis and rule-checking on library cells. All library cells used inside the design units specified by the *checkip* and *checkdu* options are also considered a part of the reduced design and are considered for rule-checking.

Enable SDC-to-SGDC translation

Use this design-read option to translate design attributes from SDC format to SGDC format. These attributes are then used during SpyGlass analysis.

By default, the output of SDC-to-SGDC translation is saved in the file sdc2sgdc_*<mode>*.sgdc.*<pid>* in the $CWD/*<vdb-name>*_reports/sdc2sgdc directory.

NOTE: *By default, the sdc2sgdc generated constraint files are retained in the subsequent SpyGlass run. Set the retain_old_sgdc parameter to no to remove the SGDC file generated in the previous SpyGlass runs.*

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option sdc2sgdc <yes | no>
```

Specify the mode of the SDC file to be translated to SGDC

Use this design-read option to specify a mode of the SDC file to be translated to an SGDC file.

Consider an example in which you specify two different modes in an SGDC file, as shown below:

```
sdc_data -file one.sdc -mode one
sdc_data -file two.sdc -mode two
```

Now, under the *Set Read Options* tab, if you specify the mode as *one*, only SDC data given under the mode, *one*, is translated. As a result, only one.sdc file is converted into SGDC.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option sdc2sgdc_mode <mode-name>
```

Specify the file to save output of SDC-to-SGDC translation

Use this design-read option to specify a file to save the output of SDC-to-SGDC translation.

If the file specified in this field already exists, SpyGlass overwrites the existing file.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option sdc2sgdcfile <file-name>
```

Dump SDC Generated Clocks in SGDC

Use the `sdc_generated_clocks` parameter to dump the generated clocks having same domain in an uncommented form in the sgdc file.

By default, the generated clocks having the same domain, if not given on black box, are dumped in a commented form in the sgdc file.

Set the value of the `sdc_generated_clocks` parameter to `yes` to dump all the generated clocks in an uncommented form in the sgdc file.

Used by	SDC2SGDCPARSE
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_parameter sdc_generated_clocks yes`
Usage in goal/source files	`-sdc_generated_clocks=yes`

Define a Mode to Infer Domains from SDC

You can use the `sdc_domain_mode` parameter to specify the mode of domain inference. By default, the domain inference mode is `sta_compliant`. In this mode, the clock domains are extracted as per the following guidelines:

- A path verified by STA implies that the source and destination have the same domain and therefore the clocks in such a path will be assigned the same domain. SpyGlass CDC therefore does not verify such paths.
- A path not verified by STA implies that the source and destination have different domains and therefore the clocks in such a path will be assigned different domains. SpyGlass CDC therefore verifies such paths.

NOTE: *If none of the* `set_clock_group`*,* `set_clock_uncertainty`*,* `set_false_path` *constraints is specified for a clock pair, they are considered synchronous.*

The following table lists the additional details about the `sdc_domain_mode` parameter:

Used by	sdc2sgdc flow
Options	sta_compliant, pessimistic, strict, async, strict_sta, sta_scg
Default value	sta_compliant
Example	
Console/Tcl-based usage	`set_parameter sdc_domain_mode strict`
Usage in goal/source files	`-sdc_domain_mode=strict`

When the mode is set to `strict`, you must provide all clock relationships in a single `set_clock_groups` command. SpyGlass reports a FATAL error if domain is not inferred.

When the mode is set to `pessimistic`, the behavior is similar to that of sta_compliant with the exception that if none of the `set_clock_group`, `set_clock_uncertainty`, or `set_false_path` constraint is specified for a clock pair, they are considered asynchronous.

NOTE: *The values, strict and pessimistic, will be deprecated in a future release.*

When the mode is set to `async`, then no domain inference from sdc constraints is done and clocks are considered asynchronous to each other.

When the mode is set to `strict_sta` or `sta_scg`, only the user-specified `set_clock_group` command would be considered and all the other commands related to domain computation are ignored. Also, a spreadsheet (`.csv`) file showing clock relationship is generated.

NOTE: *The value, strict_sta, for the sdc_domain_mode parameter is deprecated and will be removed in a future release.*

Inferring cdc_false_path for Clocks in Different Domains

Use the *sdc_generate_cfp* option to infer *cdc_false_path* in case we have different domains that were assigned to the clocks but no asynchronous relationship was specified between these clocks.

Used by	SDC2SGDCPARSE
Options	yes, no
Default value	no
Example	
Console/Tcl-based usage	`set_option sdc_generate_cfp yes`
Usage in goal/source files	`-sdc_generate_cfp=yes`

For example, consider the following SDC declaration:

```
create_clock -name Clk1 -period 10.00 in1
create_clock -name Clk2 -period 10.00 in2
create_clock -name Clk3 -period 10.00 in3
create_clock -name Clk4 -period 10.00 in4

set_clock_group -asynchronous -group { Clk1 } -group { Clk3 }
```

Following corresponding SGDC commands are generated when you set the value of the *sdc_generate_cfp* option to yes:

```
cdc_false_path -from Clk1 -to Clk2
cdc_false_path -from Clk1 -to Clk4
cdc_false_path -from Clk2 -to Clk1
cdc_false_path -from Clk2 -to Clk3
cdc_false_path -from Clk3 -to Clk2
cdc_false_path -from Clk3 -to Clk4
cdc_false_path -from Clk4 -to Clk1
cdc_false_path -from Clk4 -to Clk3
```

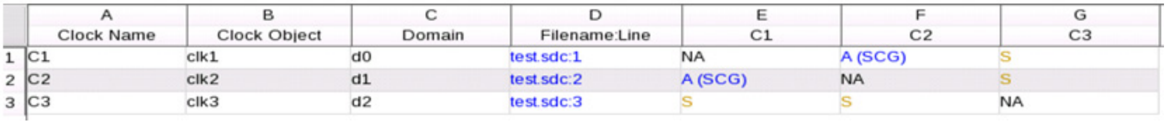
Capturing Domain Inferring Results

The following rules capture the domain inferring results:

- `Domain_Missing01`: Reports clocks for which no domain relationship could be inferred from SDC commands. The domain assigned to these clocks depends on the mode specified in the `sdc_domain_mode` parameter. The severity of this rule is Error by default. In case of `strict` mode, it reports a FATAL violation.
- `Domain_Conflict01`: Reports conflicts found during domain inference. The severity of this rule is Error by default. In case of `strict` mode, it reports a FATAL violation.
- `Domain_Matrix01`: Generates spreadsheet to show clock relationships and the inferred domain depending upon the mode specified in the `sdc_domain_mode` parameter. It is an informational rule. For example, consider the following input in the sdc file:

  ```
  create_clock -name C1 -period 10 { clk1 }
  create_clock -name C2 -period 10 { clk2 }
  create_clock -name C3 -period 10 { clk3 }
  set_clock_group -asynchronous -group { C1 } -group { C2 }
  ```

  For the above input, the spreadsheet generated by the `Domain_Matrix01` rule when the `sdc_domain_mode` parameter is set to `sta_compliant` and the `sdc_generate_cfp` parameter is set to `yes`, is shown in the following figure.



	A	B	C	D	E	F	G
	Clock Name	Clock Object	Domain	Filename:Line	C1	C2	C3
1	C1	clk1	d0	test.sdc:1	NA	A (SCG)	S
2	C2	clk2	d1	test.sdc:2	A (SCG)	NA	S
3	C3	clk3	d2	test.sdc:3	S	S	NA

FIGURE 214. Domain_Matrix01 Spreadsheet

NOTE: *If you generate the **cdc_false_path** constraint through the **sdc_generate_cfp** command, SpyGlass CDC ignores the asynchronous crossings for the paths specified in the **cdc_false_path** constraints.*

Changing the Default Hierarchy Separator of the SDC2SGDC Constraints

Use the use_hier_sep_slash parameter to change the default hierarchy separator of the SGDC constraints.

By default, the hierarchy separator used in SGDC constraint’s name field is '.'

Example

Consider the following create_clock command in SDC.

```
create_clock -name "CLK" -add -period 10.0
-waveform {0.0 5.0} [get_pins buf_cts/clkout]
```

The above command is converted into following SGDC clock using the default hierarchy separator, '.', as shown below:

```
clock  -name "test_top.buf_cts.clkout"  -domain CLK
-edge { 0.000000 5.000000}  -period 12 -tag CLK
```

When the value of the use_hier_sep_slash parameter is set to yes, the same clock is converted to the following SGDC clock:

```
clock  -name "test_top/buf_cts/clkout"  -domain CLK
-edge { 0.000000 5.000000}  -period 12 -tag CLK
```

Handling False Paths

The *false_path* constraint is generated in the SGDC file when the following commands are specified:

- **set_false_path**

  Consider the following SDC commands:

  ```
  set_false_path -from [get_pins f1/q] -through [get_pins
  A2/out] -to [ get_pins f2/in]
  set_false_path -from [get_pins f1/q] -through [get_pins
  A2/out] -to [ get_pins f2/in]
  ```

  The following false_path constraints are generated corresponding to the above SDC commands:

  ```
  false_path -from f1/q -to f2/in -through A2/out  -type sfp
  false_path -from f1/q -to f2/in -through A2/out  -type sfp
  ```

- **set_clock_group**

  Consider the following SDC commands:

  ```
  set_clock_group -logically_exclusive
  set_clock_group - physically_exclusive
  set_clock_group - asynchronous
  ```

  The following false_path constraints are generated corresponding to the above SDC commands:

  ```
  false_path -scg_logically_exclusive
  false_path -scg_physically_exclusive
  false_path -scg_asynchronous
  ```

Handling Multi-cycle Paths

The sg_multicycle_path constraint is generated in the sgdc file when the set_multicycle_path command is given.

Consider the following SDC commands:

```
set_multicycle_path  -from [get_pins f1/q] -through [get_pins
```

```
A2/out] -to [ get_pins f2/in] 2
```

The following sg_multicycle_path constraint is generated corresponding to the above SDC commands:

```
sg_multicycle_path -from f1/q -to f2/in -through A2/out
-path_multiplier 2
```

Handling Mutually Exclusive Clocks

The cdc_false_path constraint is generated in the sgdc file when the -logically_exclusive option is given with the set_clock_groups command.

Consider the following SDC commands:

```
create_clock -name Clk1 -period 10.00 in1
create_clock -name Clk2 -period 15.00 in2
set_clock_group -logically_exclusive
-group{ Clk1 } -group { Clk2 }
```

The following cdc_false_path constraints are generated corresponding to the above SDC commands:

```
cdc_false_path -from Clk1 -to Clk2
cdc_false_path -from Clk2 -to Clk1
```

Handling of Directional Clocks

The cdc_false_path constraint is generated in the sgdc file when the set_clock_uncertainity command is used but the clock is inferred as asynchronous.

Consider the following SDC commands:

```
create_clock -name Clk1 -period 10.00 in1
create_clock -name Clk2 -period 15.00 in2
set_clock_groups -asynchronous
-group { Clk1 } -group { Clk2 }
set_clock_uncertainty -from Clk1 -to Clk2
```

The following cdc_false_path constraint is generated corresponding to the above SDC commands:

```
cdc_false_path -from Clk1 -to Clk2
```

Translating set_clock_sense command

The sdc *set_clock_sense* SDC command is converted to the *clock_sense* SGDC command during the sdc2sgdc flow.

Currently, the *set_clock_sense* command is translated only when you specify the -*stop_propagation* option.

For example, consider the following SDC command:

```
set_clock_sense -stop_propagation -clocks Clk2  [get_pins
orinst1/Z ]
```

Following is the converted SGDC command:

```
clock_sense -pins "top.orinst1.Z" -tag  Clk2
```

Translating set_disable_timing command

The set_disable_timing SDC command is converted to the *disable_timing* SGDC command when:

- Values for the -from and -to fields of the command is specified
- Object list in the command is a lib cell module or lib cell instance

For example, consider the following SDC command:

```
set_disable_timing -from A -to Z [ get_lib_cells lsi_10k/OR2
]
```

Following is the converted SGDC constraint:

```
- disable_timing -name OR2 -from A -to Z
```

Translating set_mode command

The *set_mode* SDC command in translated into the *set_lib_timing_mode* SGDC command, when the type of *set_mode* is cell.

For example, consider the following SDC command:

```
set_mode -type cell mode1 U1
```

Following is the converted SGDC constraint:

```
set_lib_timing_mode -modes mode1 -instances U1
```

Specify the manner in which virtual-to-real clock mapping to be done

Use this design-read option to specify a manner in which virtual to real clock-mapping should be done.

If you set this field to *No* or do not specify any value in this field, a traversal method is used in which involves finding clocks in the fan-in/fan-out of output/input delays by matching clock characteristics, that is, period, waveform, followed by name-mapping.

Set this field to *Yes* to implement name mapping only.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option mapVirtualClkByName <yes | no>
```

Specify parameter to give the list of suffix strings

Use this design-read option to specify a list of suffix strings used for mapping virtual clocks to real clocks.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option mapSuffixList <comma-separated-list>
```

Consider the following example:

```
set_option mapSuffixList virtual
```

In this case, `CLKAvirtual` maps to `CLKA`.

If you do not specify this command or if none of the real clocks satisfies the matching criteria for the concerned virtual clock name, no mapping is done. However, virtual clocks are still dumped, but with their actual names only.

Specify parameter to give the list of prefix strings

Use this design-read option to specify a list of prefix strings used for mapping virtual clocks to real clocks.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option mapPrefixList <comma-separated-list>
```

Consider the following example:

```
set_option mapPrefixList virtual
```

In this case, `virtualCLKA` maps to `CLKA`.

If you do not specify this command or if none of the real clocks satisfies the matching criteria for the concerned virtual clock name, no mapping is done. However, virtual clocks are still dumped, but with their actual names only.

Enable the physical aware power estimation flow

Use this design read option to enables Physical Aware Power Estimation flow and check out required licenses.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option enable_physical_aware_pe <yes|no>
```

If you do not specify this option, Spyglass runs normally.

Path (physical_dbdir) where design database is saved

Use this design read option to specify directory path to the design database for save & restore runs in the Physical Aware Power Estimation flow.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option physical_dbdir <directory_path>
```

If you do not specify this option, SpyGlass uses default location for the design database Save and Restore runs.

Is the current design a netlist design

Use this design-read option to specify an input design as a netlist design that contains no behavioral construct.

When you set this option to *Yes*, certain runtime optimizations are enabled during elaboration.

If this option is set to *Yes* and the design is not a netlist design, SpyGlass skips the behavioral constructs, and synthesizes only the structural constructs.

A netlist design can only contain the following:

- Instantiations of other modules or library cells
- Simple assign statements

This option is not allowed when any of the following options are specified:

- `enable_mmdelete/higher_capacity`

  This set enables module-by-module deletion that is not allowed for a netlist design.
- `enable_precompile_vlog/libhdlfiles/libhdlf`

  This set allows precompile dump for Verilog designs, and it is not supported with this option.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option netlist <yes | no>
```

The default value of this options is no. Set the value of this option to yes to specify an input design as a netlist design.

Reports Max Count Size

Use this design-read option to specify the maximum number of messages for sorted reports (simple, moresimple, and waiver reports).

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option report_max_size <value>
```

Reports Name

Use this design-read option to specify the name of the report to be generated.

The specified report is written to the <report-name>.rpt file (default name) in the current working directory. To specify a different file name and location, use the `reportfile` project file command.

NOTE: *If you generate the same report again with the same settings, the existing report file is overwritten by the new information.*

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option report <name>
```

For example, the following command generates the summary report for the current design:

```
set_option report { "summary" }
```

The following example generates the summary and inline report for the current design:

```
set_option report { {summary} {inline} }
```

Report File

Use this design-read option to specify the name and location of the report file.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option reportfile <file-name>
```

For example, to send results from the SpyGlass review of your design file to the file myoutput, which you intend to edit later, use the following command:

```
set_option reportfile myoutput
set_option report simple
```

NOTE: *If you generate the same report again with same settings, the existing report file is overwritten by the new information.*

Report Style

Use this design-read option to specify the style in which you want to customize a report.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option report_style <style>
```

Where, *<style>* can accept any of the following values:

```
[flat | grouped] | [display_msgid | hide_msgid] |
[display_rulegroup | hide_rulegroup] | display_sdcgroup |
display_taggroup
```

NOTE: *You can specify only one value from each set of values (mentioned above) in a given run.*

The following table describes the purpose of each of the above values:

Value	**Description**
flat	Displays the report in an ungrouped format
grouped	Groups the content of the report (for example, by goals)
display_msgid	Enables the display of the message index column in the reports
hide_msgid	Hides the message index column in the reports
display_rulegroup	Allows grouping of rule messages in the reports by rule group
hide_rulegroup	Disallows grouping of rule messages by rule group in the report

Value	Description
display_sdcgroup	Groups messages of the schema-based rules in the SpyGlass Constraints solution based on the SDC schema specified in the SGDC file. This option is enabled by default.
display_taggroup	Groups messages of the *Ac_sync_group* rules of the SpyGlass CDC solution based on instance names or user-specified names

The default values of the `report_style` command are `grouped`, `display_msgid`, and `hide_rulegroup`.

If you specify conflicting values to this command, only the last specified value is considered. For example, consider that you specify conflicting values, as shown in the following command:

```
set_option report_style { hide_msgid display_msgid }
```

In the above example, the `display_msgid` value is considered.

Aggregated Report(s) Name

Use this design-read option to specify the type of aggregate reports (project_summary, datasheet, or dashboard) to be generated.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option aggregate_report {<report-names>}
```

Specify Configuration File to be Used to Create the Report

Use this design-read option to specify a configuration file that contains a list of projects and run directories generated by batch console or GUI. This data is used to generate the specified aggregate report.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option aggregate_report_config_file <config-file-path>
```

Report Output Directory Path

Use this design-read option to specify a directory in which you want to generate data for the specified aggregate report.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option aggregate_reportdir <report-directory-path>
```

Disable HTML Reports generation

Use this design-read option to disable the auto-generation of the specified reports. You can specify 'datasheet', 'dashboard' and/or 'html' to stop auto-generation of either of the reports.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option disable_html_report {datasheet dashboard}
```

Generating waive Commands for Non-designread Rules

Use this design-read option to generate the waive commands for non-designread rules in commented form.

Setting this field is equivalent to specifying the following command in the SpyGlass project file:

```
set_option exhaustive_pragma2Waiver <yes | no>
```

By default, when the designread goal runs, the waiver for rules that are not part of the designread goal are not generated in the `pragma2Waiver.swl` file because the rule is not part of the designread goal.

Set the `exhaustive_pragma2Waiver` option to `yes` to enable SpyGlass to generate the waive commands for non-designread rules in commented form. For example, consider the following specified in the RTL code:

```
//spyglass_enable W123 DetectTopDesignUnits
```

```
...
...
...
// spyglass_disable W123 DetectTopDesignUnits
```

If the `exhaustive_pragma2Waiver` option to `yes`, for the above RTL code, SpyGlass generates the waiver for the W123 rule in commented form. However, DetectTopDesignUnits is generated in uncommented form.

Product-Specific Interfaces in SpyGlass Explorer

SpyGlass Explorer provides dedicated user interface for the following products:

- *SpyGlass Auto Verify Product*
- *SpyGlass CDC Product*
- *SpyGlass Power Verify Product*

SpyGlass Auto Verify Product

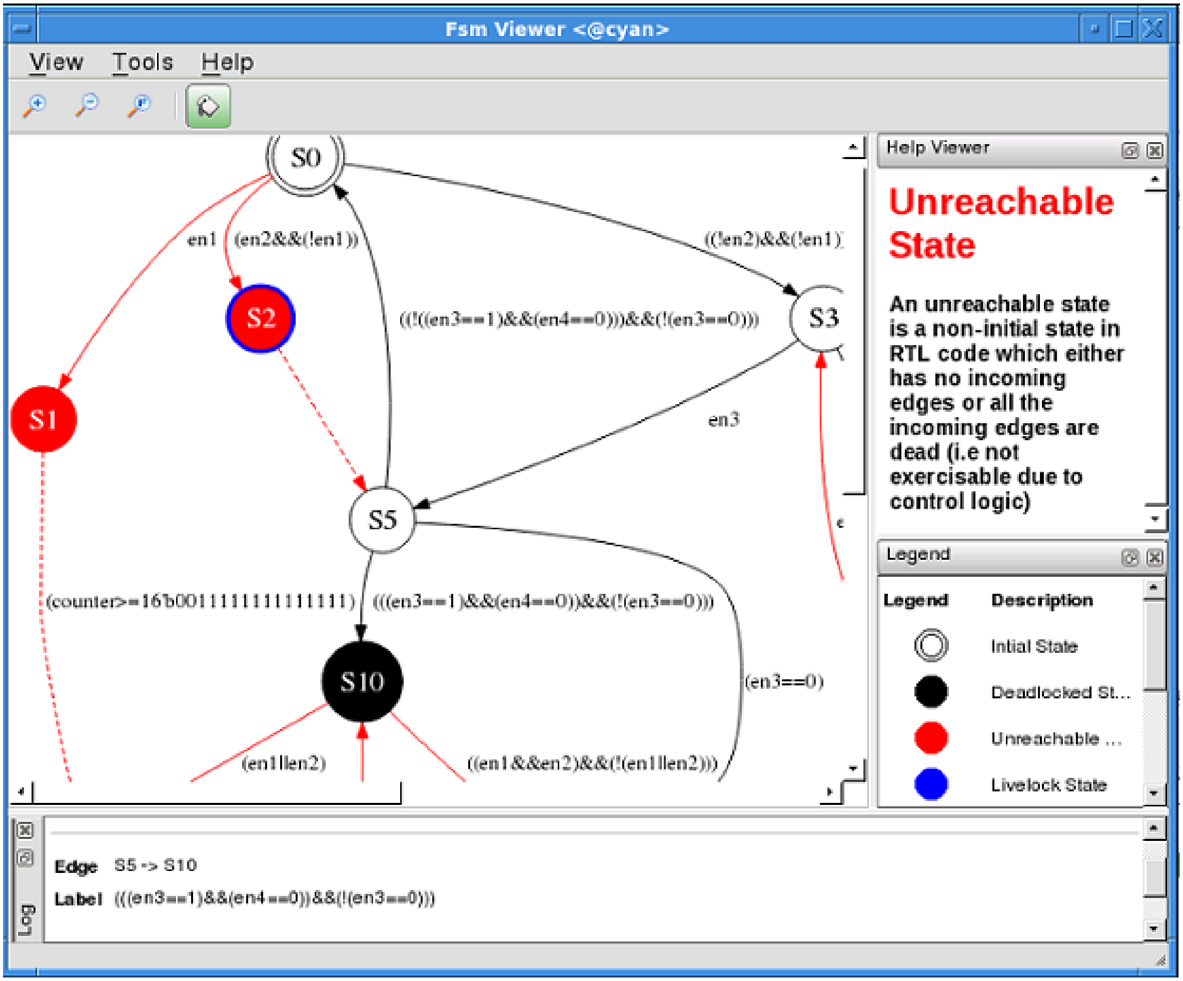
The **FSM Viewer** in SpyGlass Explorer allows you to analyze the FSM states and transitions in the design.

You can invoke the **FSM Viewer** in SpyGlass Explorer by double-clicking on the violation message for the *Av_fsm_Analysis* rule.

The **FSM Viewer** in SpyGlass Explorer now provides following additional capabilities for analyzing FSM states and transitions in the design:

- Zoom-in, zoom-out, and zoom-to-fit capabilities
- Switch on/off transition labels
- Legend information for FSM states
- Help viewer window
- Switch on/off log viewer

Figure 215 displays a sample **FSM Viewer** in SpyGlass Explorer:


Fsm Viewer <@cyan>
View
Tools
Help
S0
en1
(en2&&(!en1))
((!en2)&&(!en1)
S2
((!((en3==1)&&(en4==0)))&&(!(en3==0)))
S3
S1
en3
S5
(counter>=16'b0011111111111111)
(((en3==1)&&(en4==0))&&(!(en3==0)))
S10
(en3==0)
(en1llen2)
((en1&&en2)&&(!(en1llen2)))
Help Viewer
Legend
Log

Unreachable State

An unreachable state is a non-initial state in RTL code which either has no incoming edges or all the incoming edges are dead (i.e not exercisable due to control logic)

Legend	Description
◎	Intial State
●	Deadlocked St...
●	Unreachable ...
●	Livelock State

Edge S5 -> S10

Label (((en3==1)&&(en4==0))&&(!(en3==0)))

FIGURE 215. FSM Viewer

SpyGlass CDC Product

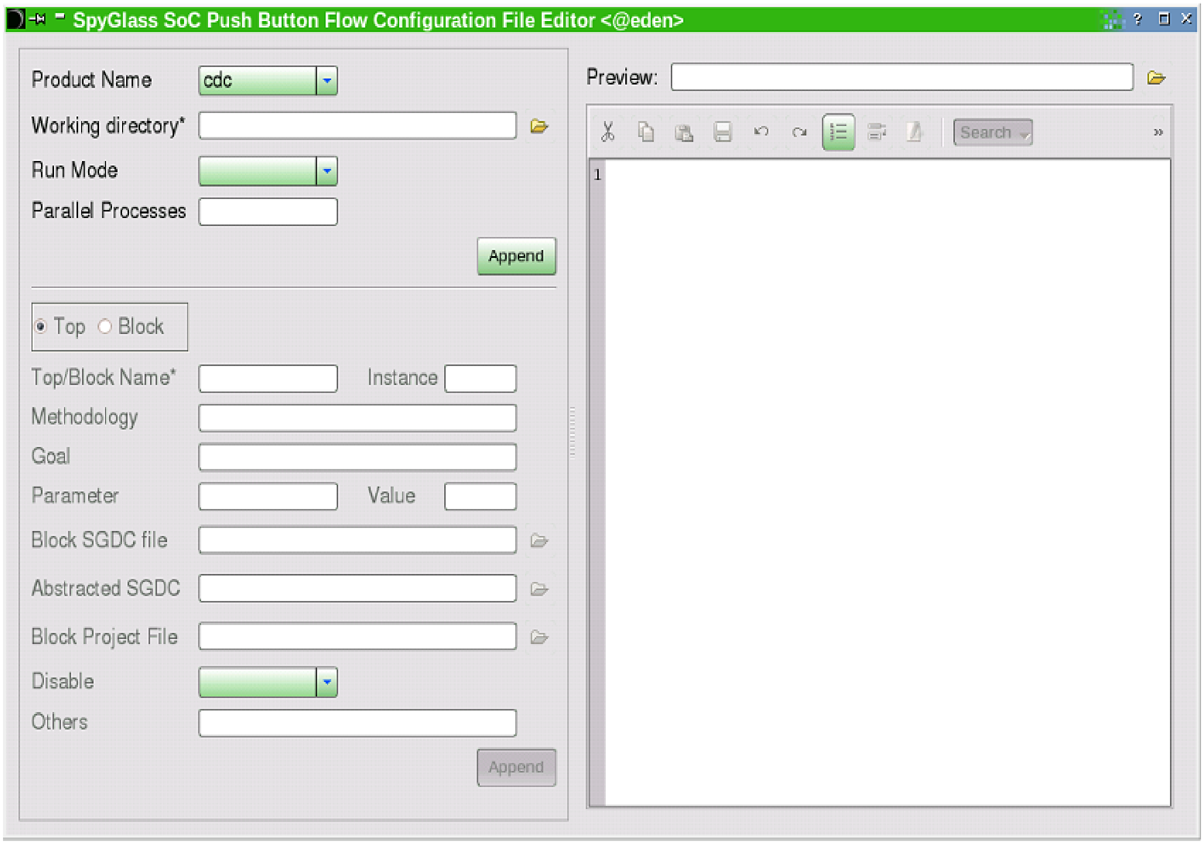
Following are the user interfaces available for the SpyGlass CDC product in SpyGlass Explorer:

- *SoC Push Button Flow Configuration File Editor*
- *Multi-mode Support*
- *CDC Explorer*

SoC Push Button Flow Configuration File Editor

The SoC Push Button Flow Configuration File Editor window allows you to generate configuration file to run the auto_soc Tcl command

The following figure shows a sample SoC Push Button Flow Configuration File Editor:



Product Name: cdc

● Top ○ Block

1

FIGURE 216. SoC Push Button Flow Configuration File Editor

In the above figure, the Push Button Flow Configuration File Editor window is divided into two panes:

- **Form pane**: Fill up the relevant form fields to create a Tcl file. After filling the first four fields of the form, click the Append button to add this information to the Tcl file and enable the remaining form. Select either Top or Block option, if you want to add information for top modules or blocks, respectively. Click Append to add the specified information to the Tcl file. You can specify information for multiple top and block modules.
- **Preview pane**: Provides you a preview of the Tcl file. You can also edit the Tcl file in the preview area. Also, you can change the Tcl file or

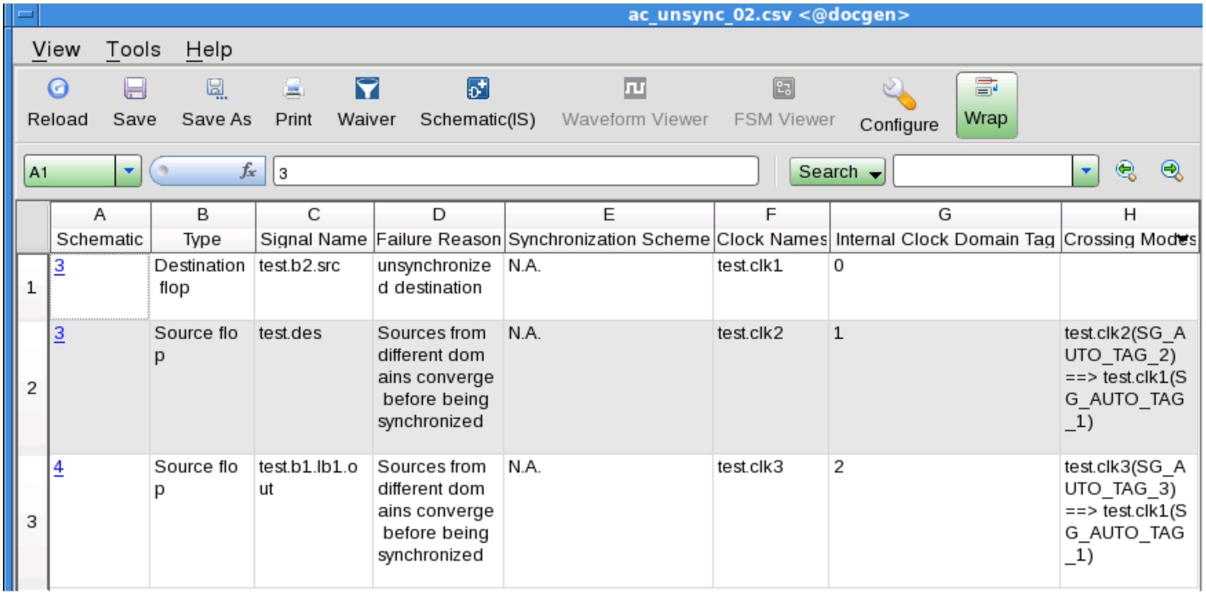
create a new one using the browse button provided above the preview area.

Multi-mode Support

Enables you to view the crossing modes in Incremental Schematic.

You can enable the multi-mode support by setting the value of the *enable_mode_computation* parameter to yes.

When the *enable_mode_computation* parameter is set to yes, the spreadsheet reports generated by the *Ac_sync_group* rules, that is, *Ac_sync01*, *Ac_sync02*, *Ac_unsync01* and *Ac_unsync02*, displays the **Crossing Modes** column. The **Crossing Modes** column lists all the crossing clock pairs as shown in *Figure 217*.



	A	B	C	D	E	F	G	H
	Schematic	Type	Signal Name	Failure Reason	Synchronization Scheme	Clock Names	Internal Clock Domain Tag	Crossing Modes
1	3	Destination flop	test.b2.src	unsynchronized destination	N.A.	test.clk1	0	
2	3	Source flop	test.des	Sources from different domains converge before being synchronized	N.A.	test.clk2	1	test.clk2(SG_AUTO_TAG_2) ==> test.clk1(SG_AUTO_TAG_1)
3	4	Source flop	test.b1.lb1.out	Sources from different domains converge before being synchronized	N.A.	test.clk3	2	test.clk3(SG_AUTO_TAG_3) ==> test.clk1(SG_AUTO_TAG_1)

FIGURE 217. Spreadsheet showing the Crossing Modes

On clicking a spreadsheet row, Incremental Schematic lists all the modes. Clicking on any mode displays the path between the source clock path and the destination clock path as shown in *Figure 218*:

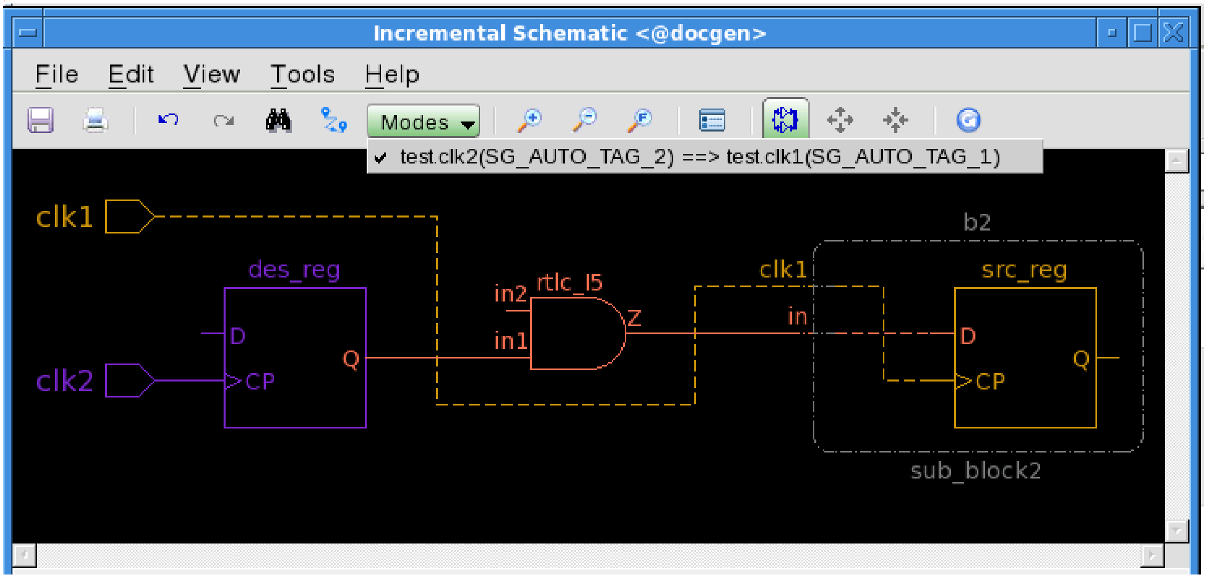


FIGURE 218. Selected Mode in Incremental Schematic

CDC Explorer

CDC Explorer consists of three browsers that display clock tree exploration and clock domain crossings view based on hierarchies, clock, and domains.

To view the CDC Explorer in SpyGlass Explorer, set the value of the *enable_cdc_explorer* parameter to yes.

CDC Explorer consists of following browser windows:

- *Clock Browser*
- *Design Browser*
- *Domain Browser*

Clock Browser

Displays the abstracted view of the clock tree as shown in *Figure 219*. It displays the number of flops driven by the respective clock.

NOTE: *Run the Setup_Clock01 rule to view the* ***Clock Browser*** *window.*

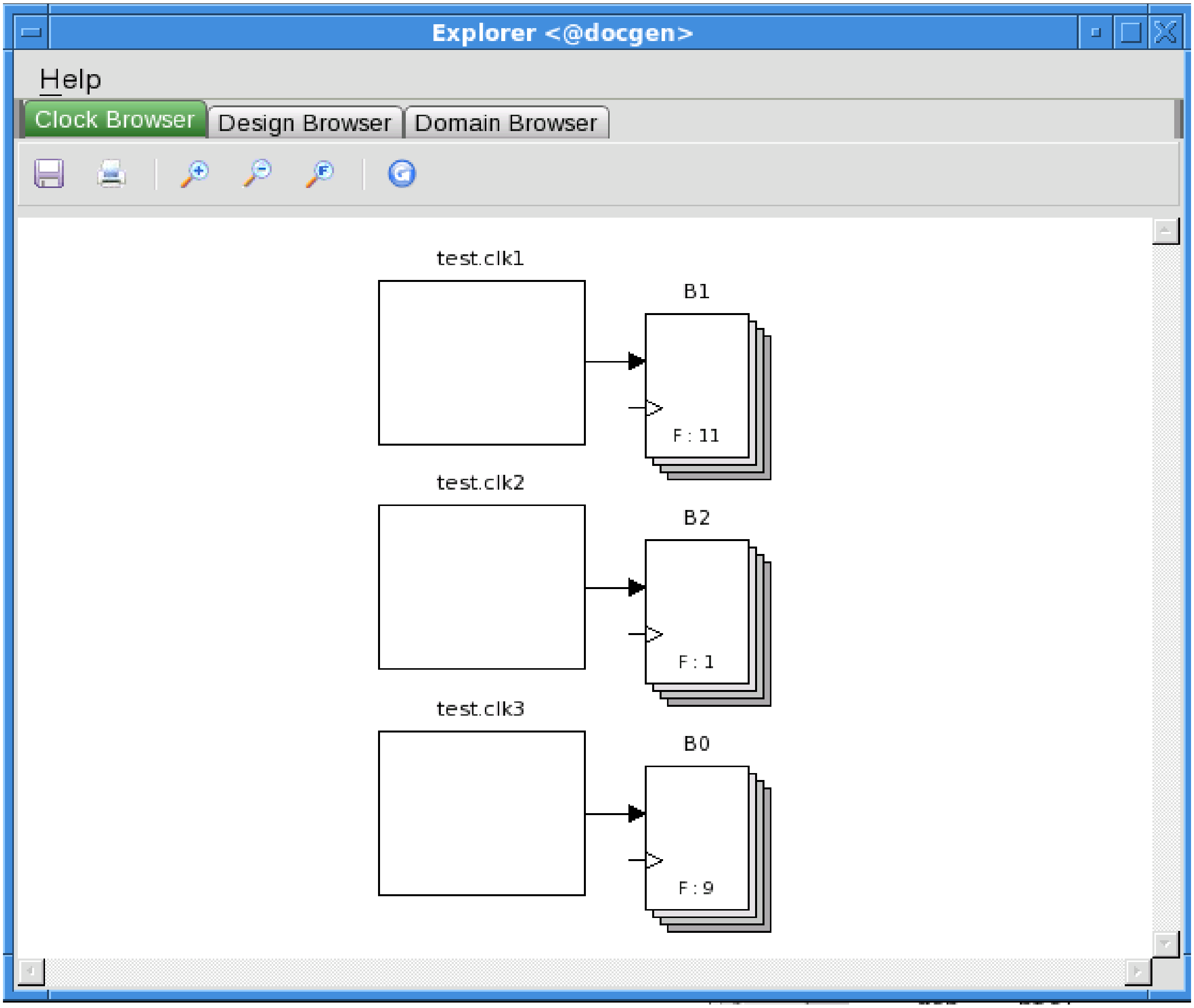


FIGURE 219. Clock Browser Window

Design Browser

Displays the hierarchical view of the clock domain crossings, clocks, and resets as shown in *Figure 220*.

NOTE: *Run the Ac_sync01, Ac_sync02, Ac_unsync01, Ac_unsync02 rules to view the Design Browser window.*

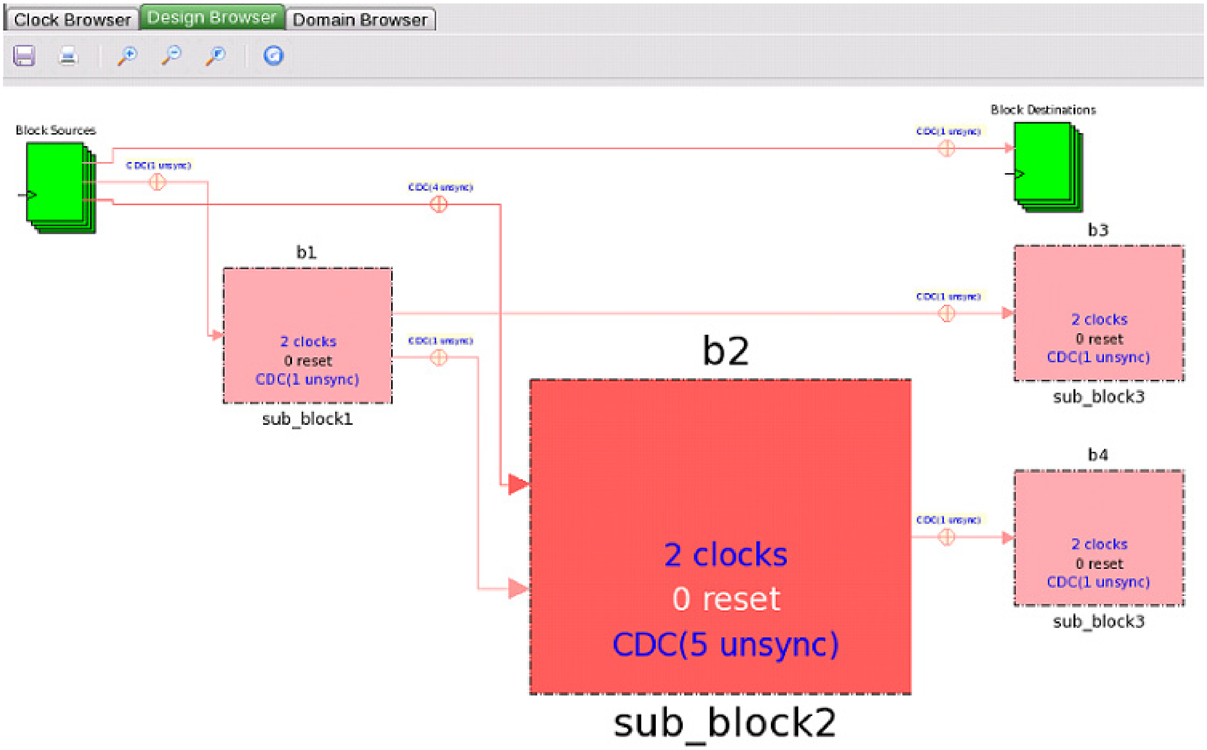


FIGURE 220. Design Browser

You can double-click on any of the block to view the hierarchical view of that block. The connections between the blocks displayed in the Design Browser window represents the number of crossings between two blocks.

The blocks in the above figure display the following information:

- Number of clocks and resets reaching the block
- Number of clocks and resets present in the block

- Number of clock domain crossings (CDC) present inside the block

Clicking on any of the above information displays a spreadsheet, which displays the details of the clocks, resets, and CDCs.

Domain Browser

Displays the clock domain crossings between domain pairs as shown in *Figure 221*.

NOTE: *Run the Ac_sync01, Ac_sync02, Ac_unsync01, Ac_unsync02 rules to view the* ***Domain Browser*** *window.*

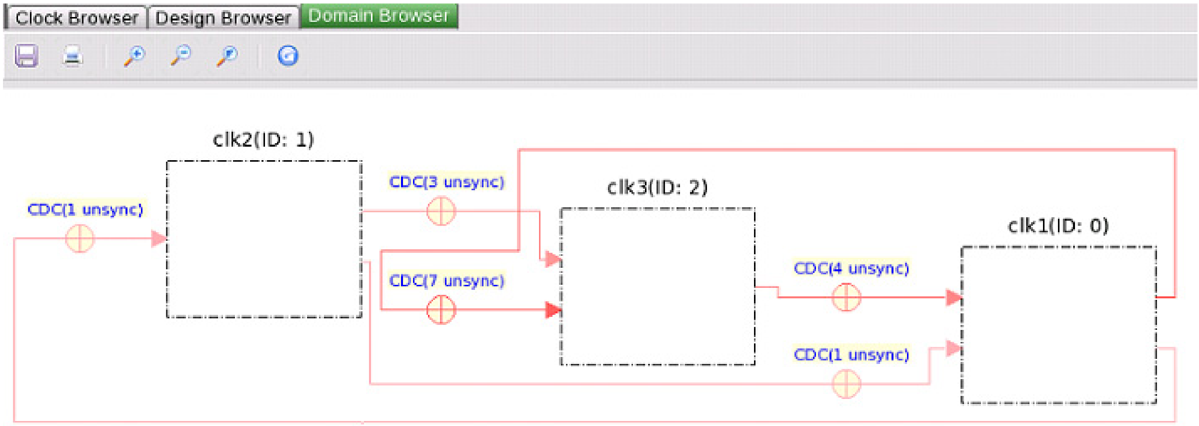


FIGURE 221. Domain Browser Window

You can double-click on any block to view the clocks domain crossings between the clock in pairs as shown in *Figure 222*.

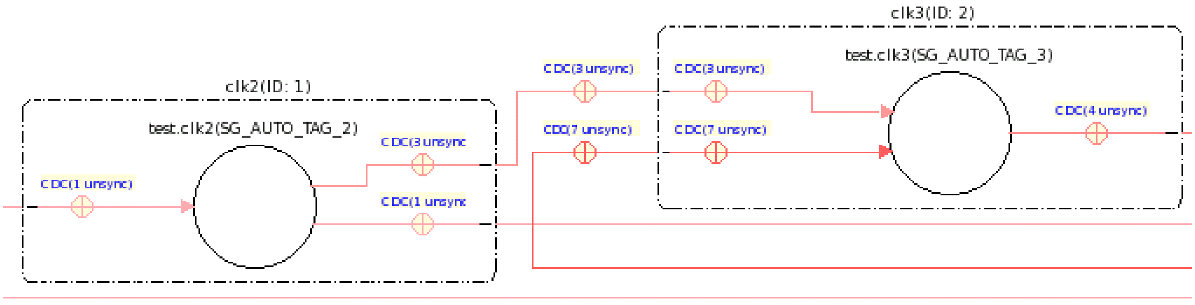


FIGURE 222. CDC between Clock Pairs

SpyGlass Power Verify Product

Following are the user interfaces available for the SpyGlass Power Verify product in SpyGlass Explorer:

- *Enhanced Incremental Schematic*
- *Improved Power Intent View*
- *Enhanced PID Browser*
- *PST Widgets*
- *Miscellaneous*

NOTE: *The Power Intent View interface in SpyGlass Explorer is enhanced in the 5.5.0 release to provide you more cross-linking options between violation message and UPF/PIH. See the Usability Enhancements in the SpyGlass 5.5.0 Release section to know about these enhancements.*

Enhanced Incremental Schematic

You can view the debug data for the SpyGlass Power Verify product in the Incremental Schematic window. The following capabilities are provided:

- Placing mouse on a port, pin, instance, or net in Incremental Schematic automatically updates the debug data
- Clicking on a message label (for example, net, source, or destination pin) in Message in the Schematic Legend pane (*Figure 223*), highlights the selected power object in the Incremental Schematic window and Power Intent View window.
- Clicking on a message label (for example, net, source, destination pin or power domain) in message in the Schematic Legend pane (Figure 218), highlights/cross probes to the relevant object in the incremental schematic or the Power Intent View window.

  If the label refers to a design object like an instance or a signal, the corresponding design object gets highlighted in the incremental schematic.

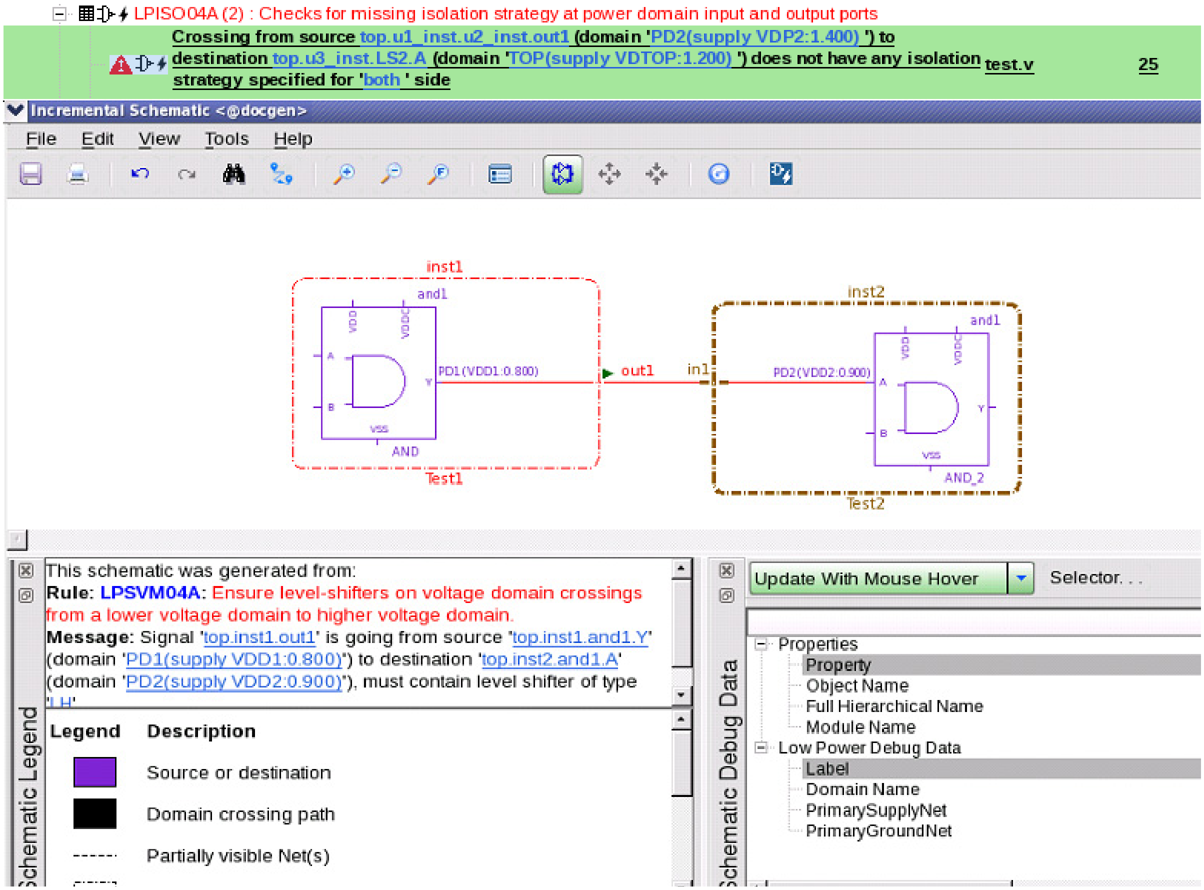
  If the label refers to a power object like an domain, supply net or an isolation/level shifter strategy, the corresponding power object gets highlighted in the Power Intent View window.
- The UPF-related information is available in debug data

- The hyperlinks from the debug data in Incremental Schematic to UPF and PIH are available. The following table lists the UPF attributes to which the hyperlinks are available from the PIH/PID Browser window:

UPF Attribute	**Hyperlink to UPF**	**Hyperlink to PIH**
set_isolation	Yes	Yes
set_level_shifter	Yes	Yes
set_retention	Yes	Yes
create_power_switch	Yes	Yes
connect_supply_net	Yes	No
set_port_attributes	Yes	No

NOTE: *The UPF data is available when the lp_set_upf_attributes parameter is set during the PV rule checking.*

A sample violation message with the respective Incremental Schematic is shown in *Figure 223* illustrating the above enhancements:



LPISO04A (2) : Checks for missing isolation strategy at power domain input and output ports

Crossing from source top.u1_inst.u2_inst.out1 (domain 'PD2(supply VDP2:1.400) ') to destination top.u3_inst.LS2.A (domain 'TOP(supply VDTOP:1.200) ') does not have any isolation strategy specified for 'both ' side test.v 25

Incremental Schematic <@docgen>

This schematic was generated from:
Rule: LPSVM04A: Ensure level-shifters on voltage domain crossings from a lower voltage domain to higher voltage domain.
Message: Signal 'top.inst1.out1' is going from source 'top.inst1.and1.Y' (domain 'PD1(supply VDD1:0.800)') to destination 'top.inst2.and1.A' (domain 'PD2(supply VDD2:0.900)'), must contain level shifter of type 'LH'

Legend	Description
🟪	Source or destination
⬛	Domain crossing path
------	Partially visible Net(s)

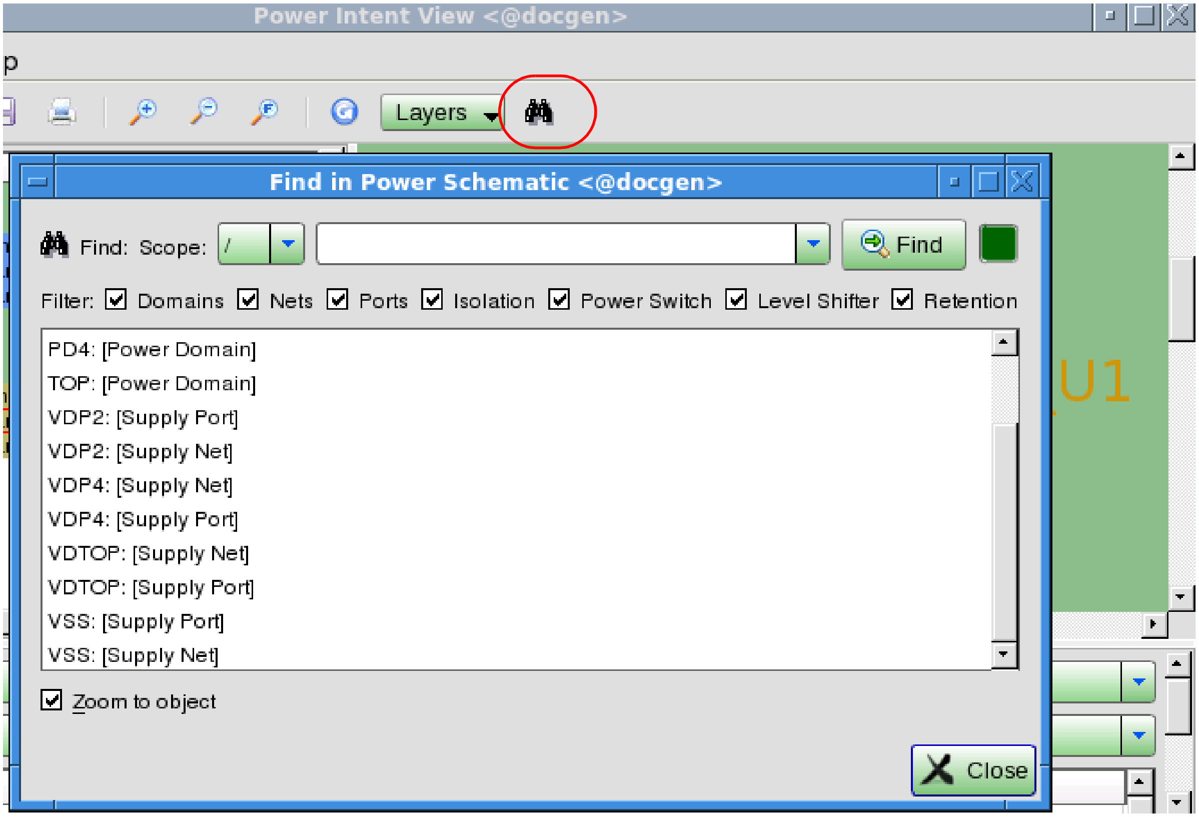
Update With Mouse Hover

FIGURE 223. Updated Incremental Schematic

Improved Power Intent View

The following enhancements have been done in the **Power Intent View** window:

- You can now search for instances using the **Find** button as shown in *Figure 224*:

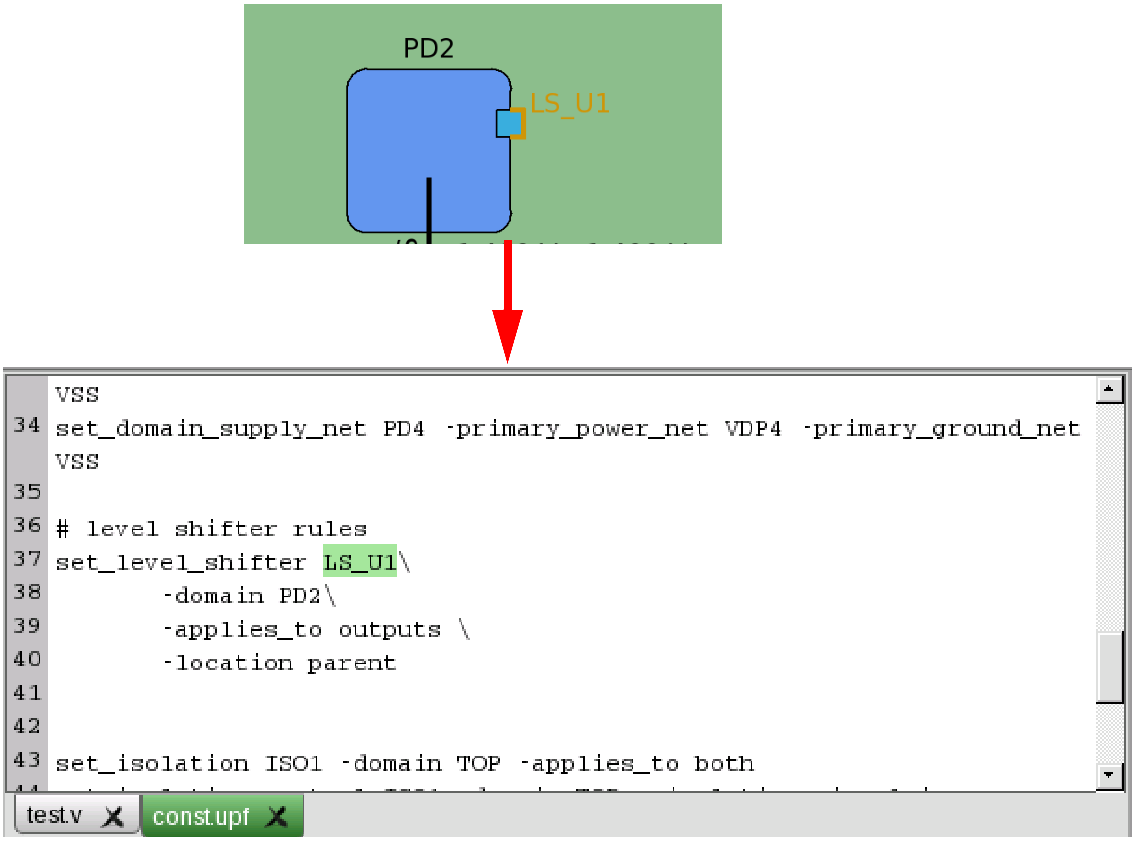


Power Intent View <@docgen>
Layers
Find in Power Schematic <@docgen>
Find: Scope: /
Find
Filter: ☑ Domains ☑ Nets ☑ Ports ☑ Isolation ☑ Power Switch ☑ Level Shifter ☑ Retention
☑ Zoom to object
Close
U1

PD4: [Power Domain]
TOP: [Power Domain]
VDP2: [Supply Port]
VDP2: [Supply Net]
VDP4: [Supply Net]
VDP4: [Supply Port]
VDTOP: [Supply Net]
VDTOP: [Supply Port]
VSS: [Supply Port]
VSS: [Supply Net]

FIGURE 224. Find in Power Schematic

- Domain or strategy selected in PIH Schematic is back-referenced to corresponding command in the UPF file as shown in *Figure 225*:



```
VSS
set_domain_supply_net PD4 -primary_power_net VDP4 -primary_ground_net
VSS

# level shifter rules
set_level_shifter LS_U1\
        -domain PD2\
        -applies_to outputs \
        -location parent


set_isolation ISO1 -domain TOP -applies_to both
```

FIGURE 225. Schematic to UPF Referencing

Enhanced PID Browser

The following enhancements have been done for the PID Browser window in the Power Intent View window:

- The instances selected in the PID browser window are displayed in Incremental Schematic as shown in *Figure 226*:

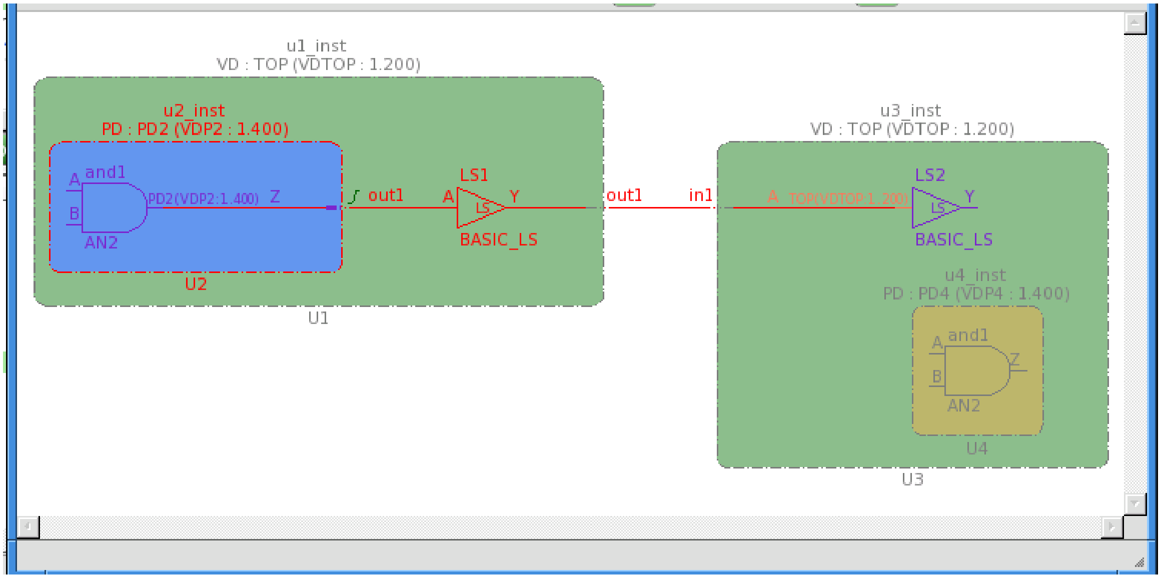
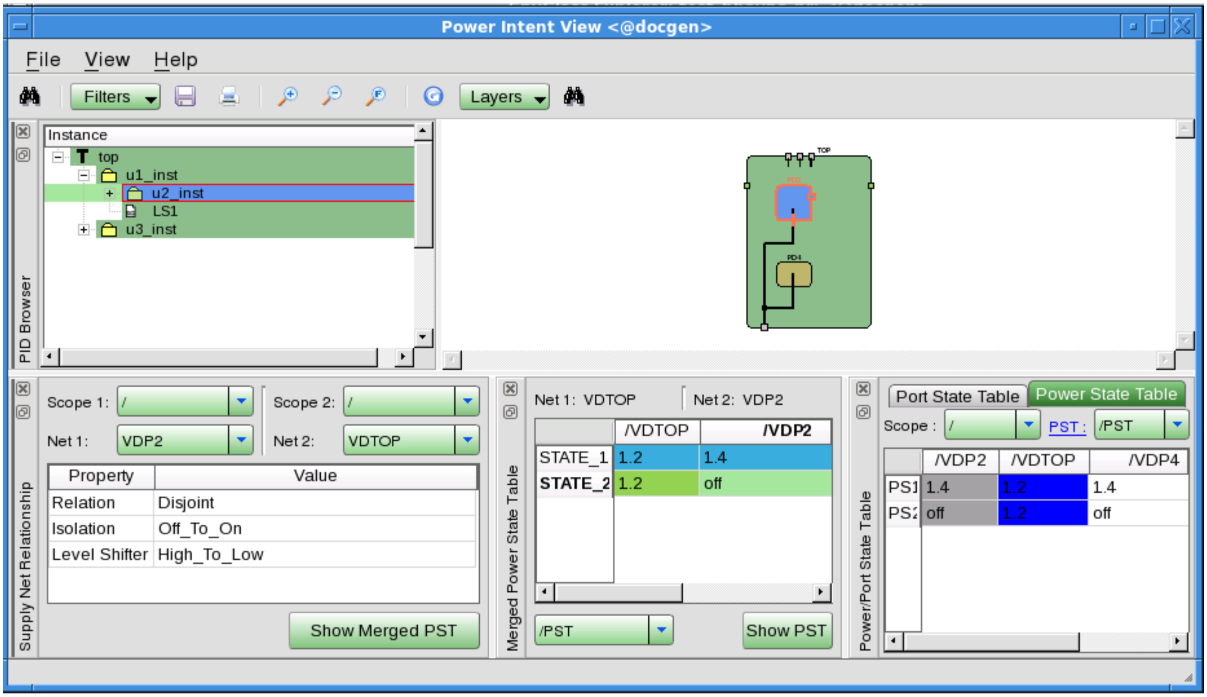


FIGURE 226. Selected Instance in Incremental Schematic

- If an instance is selected in PID, its domain will be highlighted in the PIH window as shown in *Figure 227*:



Scope 1: /
Scope 2: /
Net 1: VDP2
Net 2: VDTOP

Property	Value
Relation	Disjoint
Isolation	Off_To_On
Level Shifter	High_To_Low

Net 1: VDTOP
Net 2: VDP2

	/VDTOP	/VDP2
STATE_1	1.2	1.4
STATE_2	1.2	off

/PST

Scope : /
PST : /PST

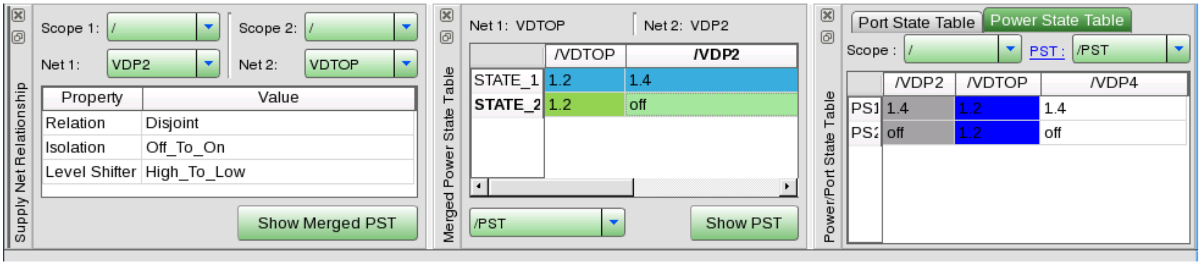
	/VDP2	/VDTOP	/VDP4
PS1	1.4	1.2	1.4
PS2	off	1.2	off

FIGURE 227. PID Browser

PST Widgets

The merged PST window displays power state table between two nets. States requiring level shifting are highlighted in blue color, while the ones requiring isolation are shown in green color

Figure 228 displays a consolidated view of the Power Intent View window.



Property	Value
Relation	Disjoint
Isolation	Off_To_On
Level Shifter	High_To_Low

	/VDTOP	/VDP2
STATE_1	1.2	1.4
STATE_2	1.2	off

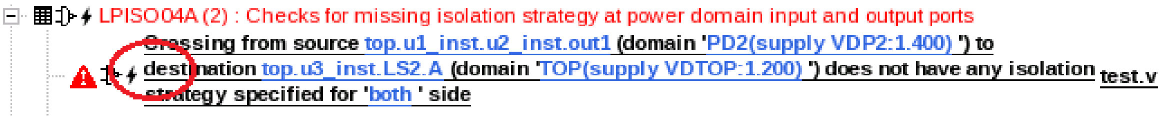
	/VDP2	/VDTOP	/VDP4
PS1	1.4	1.2	1.4
PS2	off	1.2	off

FIGURE 228. PST Widgets in Power Intent View

Miscellaneous

Following miscellaneous enhancements have been done for the SpyGlass Power Verify Product in SpyGlass Explorer:

- The violation message in SpyGlass Explorer shows a Power Intent icon, if the Power Intent View is enabled for that message. *Figure 229* shows a sample violation message for which the Power Intent View is enabled:



LPISO04A (2) : Checks for missing isolation strategy at power domain input and output ports

Crossing from source top.u1_inst.u2_inst.out1 (domain 'PD2(supply VDP2:1.400) ') to destination top.u3_inst.LS2.A (domain 'TOP(supply VDTOP:1.200) ') does not have any isolation strategy specified for 'both ' side test.v

FIGURE 229. Violation Message with Power Intent View Enabled

- Power domains are shown in consistent colors across Incremental schematic, PIH, and PID browser
- You can cross-probe to a signal from the Violation message. To do so, right-click on a violation message, click **Cross Probe** option and select the object that you want to highlight in the **HDL Viewer** pane.

Usability Enhancements in the SpyGlass 5.5.0 Release

The Power Intent View interface is now enhanced and enables you to better ascertain the reasons of a violation being reported. More cross-linking options between the violation message and UPF/PIH have been provided.

These *enhancements are*:

- *Back-Referencing from LP Debug Data to UPF/PIH*
- *Displaying Important LIB attributes*
- *Support for Supply Net Relationship Widget*
- *Direct Back-Referencing from Message to PIH*

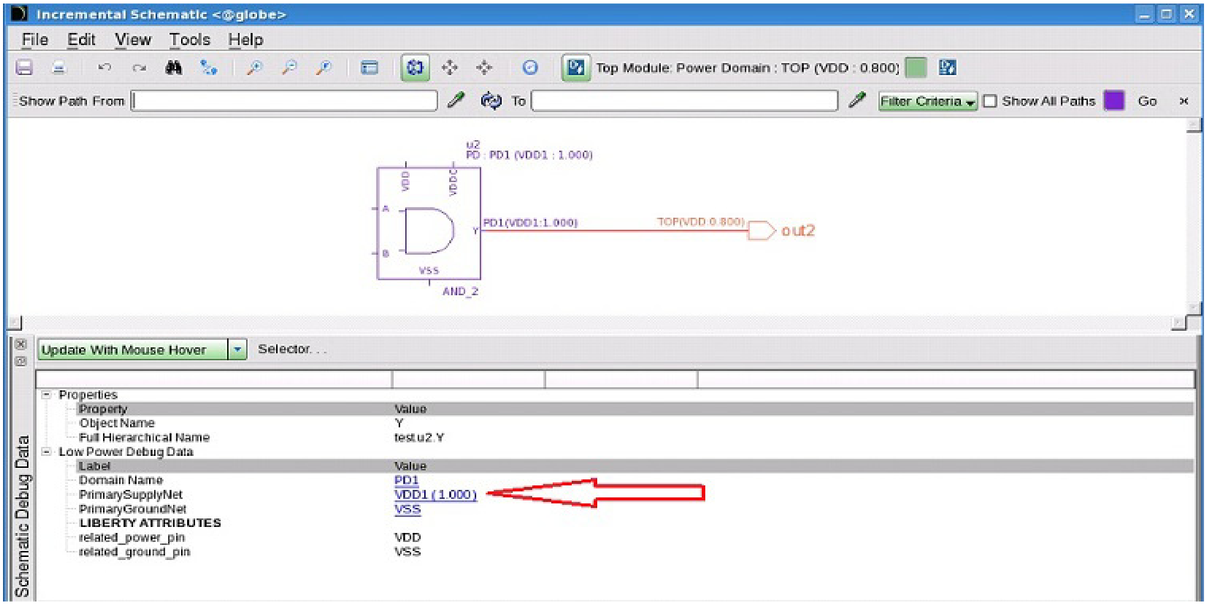
NOTE: *The above enhancement are available when the* `lp_set_upf_attributes` *parameter is set during the Power Verify rule checking.*

Back-Referencing from LP Debug Data to UPF/PIH

The ***Low Power Debug Data*** section now displays hyperlinks in the ***Value*** column, as shown in *Figure 230*, for the following labels:

- Domain name
- Primary supply net
- Primary ground net
- Related supply net
- Related ground net

When you click on any of these links, the corresponding command in the UPF is highlighted. In addition, the same supply net or domain is highlighted in the **Power Intent View** (PIH) window.



Properties

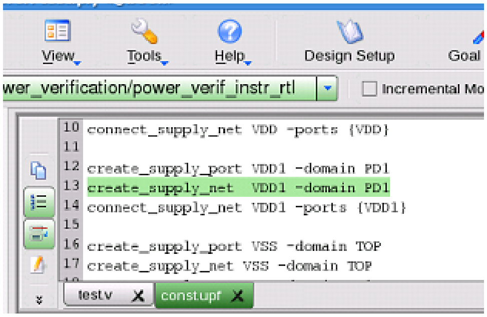
Property	Value
Object Name	Y
Full Hierarchical Name	test.u2.Y

Low Power Debug Data

Label	Value
Domain Name	PD1
PrimarySupplyNet	VDD1 (1.000)
PrimaryGroundNet	VSS
LIBERTY ATTRIBUTES	
related_power_pin	VDD
related_ground_pin	VSS

FIGURE 230. Low Power Debug Data with cross-linked values

For example, if you click on the *VDD1* link, shown in *Figure 230*, the corresponding `create_supply_port` command is highlighted in UPF, as shown in *Figure 231*. The same supply net is highlighted in PIH window, as shown in *Figure 232*.



```
connect_supply_net VDD -ports {VDD}

create_supply_port VDD1 -domain PD1
create_supply_net  VDD1 -domain PD1
connect_supply_net VDD1 -ports {VDD1}

create_supply_port VSS -domain TOP
create_supply_net VSS -domain TOP
```

FIGURE 231. Back-referenced command in the UPF

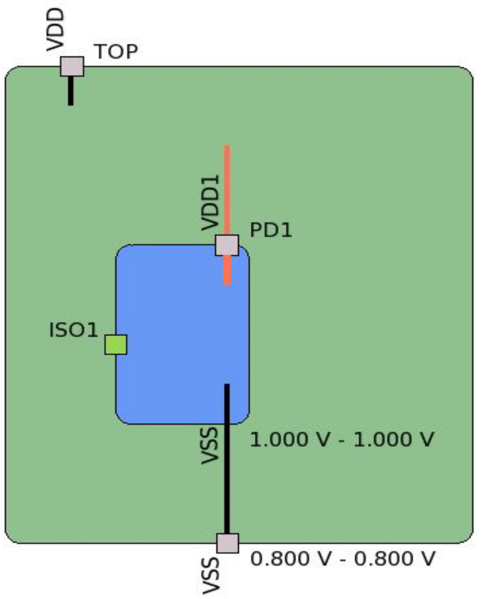


FIGURE 232. Back-referenced supply net (VDD1) in PIH

Displaying Important LIB attributes

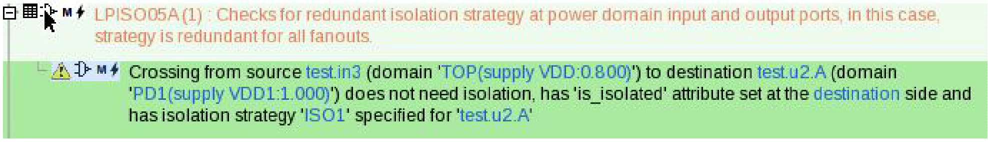
The ***Low Power Data Debug*** section has been enhanced to displayed library attributes along with domain and supply information, under the *LIBERTY ATTRIBUTES* heading.

The following attributes are displayed under the *LIBERTY ATTRIBUTES* heading:

Attribute	Description
is_isolated	For library cell pins having `is_isolated` attribute set to `true` in their library
always_on	For library cell pins having `always_on` attribute set to `true` in their library

Attribute	Description
isolation_cell_data_pin	For data pins of isolation cells, only if attribute `isolation_cell_data_pin` is set to `true` in library
isolation_cell_enable_pin	For enable pins of isolation cells, only if attribute `isolation_cell_enable_pin` is set to `true` in library
level_shifter_cell_data_pin	For data pins of level shifter cells, only if attribute `level_shifter_data_pin` is set to `true` in library
level_shifter_cell_enable_pin	For enable pins of enable level shifter cells, only if attribute `level_shifter_enable_pin` is set to `true` in library
switch_pin	For enable pins of power switch cells, only if attribute `switch_pin` is set to `true` in library
retention_pin	For save/restore pins of retention cells, only if `retention_pin` attribute is set on them in the library

Consider the following violation message:



LPISO05A (1) : Checks for redundant isolation strategy at power domain input and output ports, in this case, strategy is redundant for all fanouts.

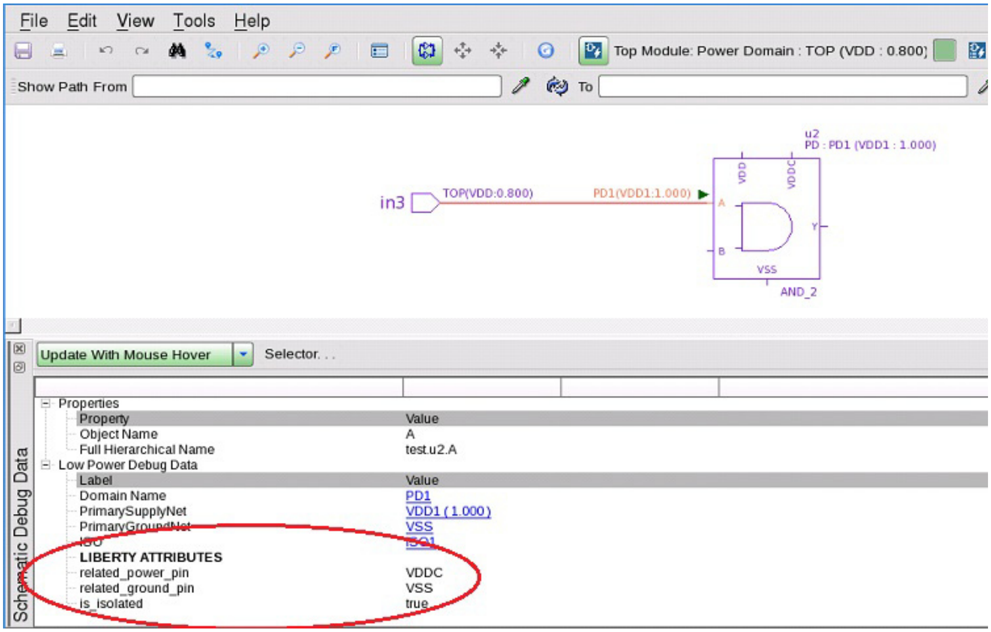
Crossing from source test.in3 (domain 'TOP(supply VDD:0.800)') to destination test.u2.A (domain 'PD1(supply VDD1:1.000)') does not need isolation, has 'is_isolated' attribute set at the destination side and has isolation strategy 'ISO1' specified for 'test.u2.A'

FIGURE 233. Violation message

In the above violation message, the destination is a library cell where the input pin `A` has the `is_isolated` attribute already set on it. The following is a snapshot of cell description in the technology library:

```
cell (AND_2) {
      pg_pin (VDD) { pg_type : primary_power; }
      pg_pin (VDDC) { pg_type : backup_power; }
      pg_pin (VSS) { pg_type : primary_ground; }
      pin (A) { direction : input;related_power_pin:
VDDC;is_isolated : true;  }
      pin (B) { direction : input;related_power_pin:
VDDC;is_isolated : false;  }
      pin (Y) { direction : output; function :
"A&B";related_power_pin: VDD;is_isolated : false;  }
   }
```

For the above violation message, when you open the ***Incremental Schematic*** window, the `is_isolated` attribute is displayed in the ***Low Power Debug Data*** section, under the *LIBERTY ATTRIBUTES* heading, as shown in *Figure 234*:



Properties

Property	Value
Object Name	A
Full Hierarchical Name	test.u2.A

Low Power Debug Data

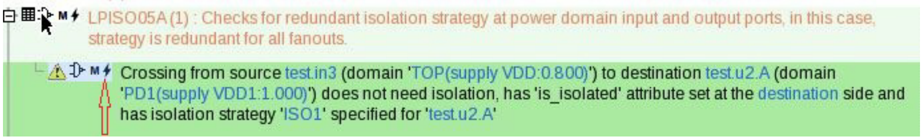
Label	Value
Domain Name	PD1
PrimarySupplyNet	VDD1 (1.000)
PrimaryGroundNet	VSS
ISO	ISO1
LIBERTY ATTRIBUTES	
related_power_pin	VDDC
related_ground_pin	VSS
is_isolated	true

FIGURE 234. Library attribute displayed under the LIBERTY ATTRIBUTES heading

Support for Supply Net Relationship Widget

The support for **Supply Net Relationship Widget** has been extended to more rules, for example LPERC01B, LPPLIB19A, etc.

The violation message in SpyGlass Explorer shows a Power Intent icon, if the **Power Intent View** is enabled for that message. *Figure 235* shows a sample violation message for which the back-referencing from the message to PIH is enabled.



LPISO05A (1) : Checks for redundant isolation strategy at power domain input and output ports, in this case, strategy is redundant for all fanouts.

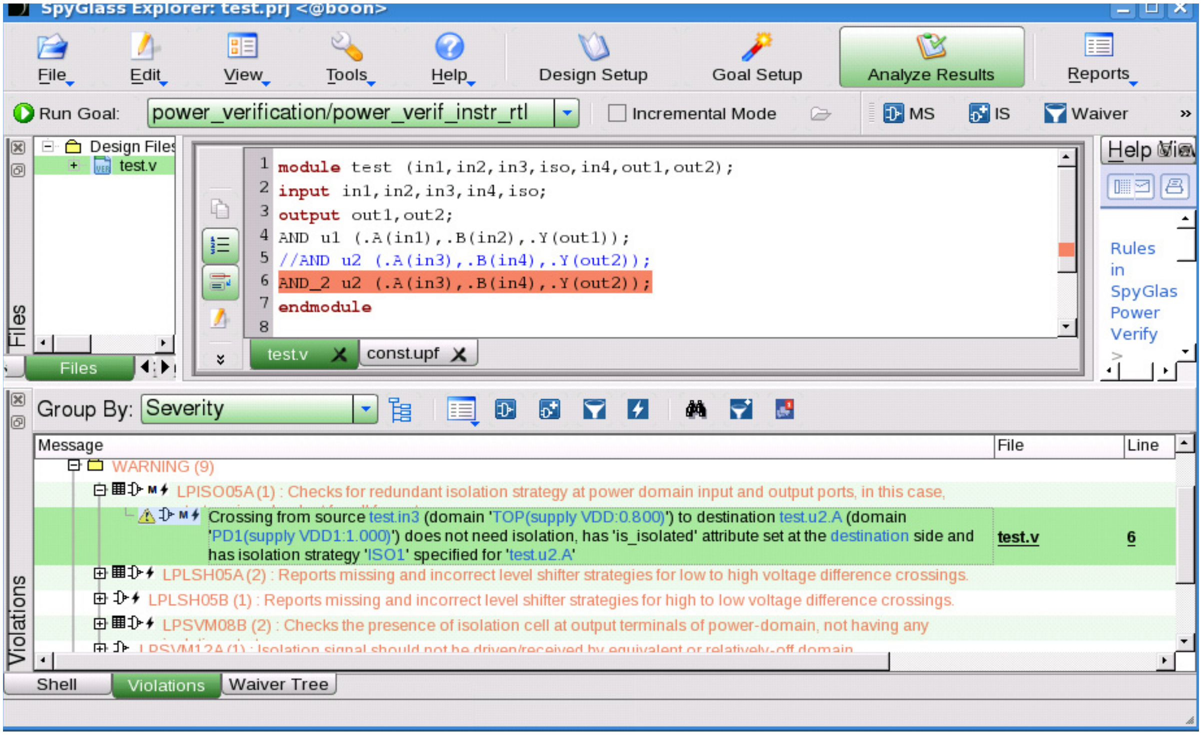
Crossing from source test.in3 (domain 'TOP(supply VDD:0.800)') to destination test.u2.A (domain 'PD1(supply VDD1:1.000)') does not need isolation, has 'is_isolated' attribute set at the destination side and has isolation strategy 'ISO1' specified for 'test.u2.A'

FIGURE 235. Violation message with Power Intent View enabled

Direct Back-Referencing from Message to PIH

This enhancement enables you to view the relevant domains, power supplies, and strategies (isolation/level shifter), by highlighting them in the **PIH** window.

Double-clicking on a violation message highlights the corresponding power domains, power supplies, and isolation/level shifter strategy (if any), in the **PIH** window.



```
module test (in1,in2,in3,iso,in4,out1,out2);
input in1,in2,in3,in4,iso;
output out1,out2;
AND u1 (.A(in1),.B(in2),.Y(out1));
//AND u2 (.A(in3),.B(in4),.Y(out2));
AND_2 u2 (.A(in3),.B(in4),.Y(out2));
endmodule
```

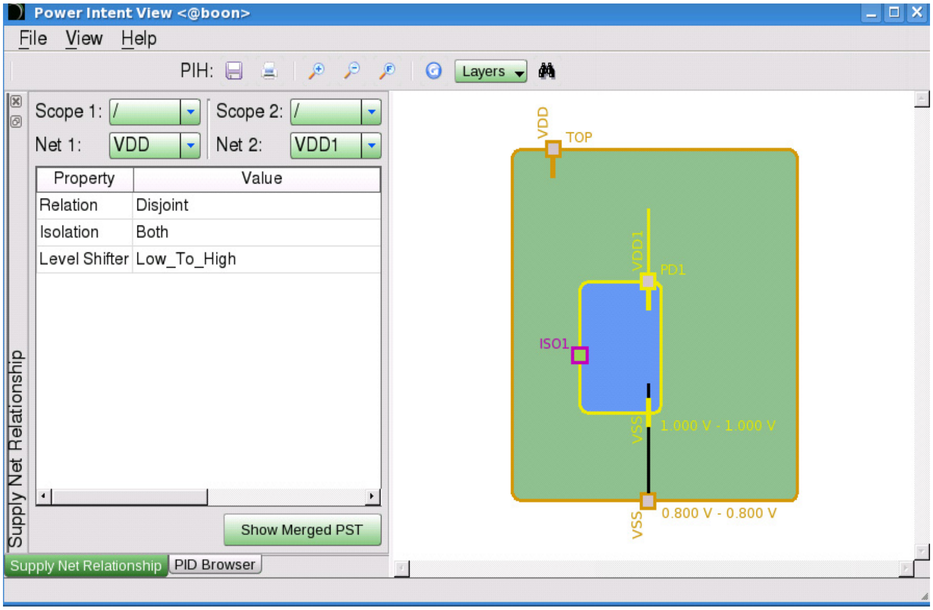
Message	File	Line
WARNING (9)		
LPISO05A (1) : Checks for redundant isolation strategy at power domain input and output ports, in this case,		
Crossing from source test.in3 (domain 'TOP(supply VDD:0.800)') to destination test.u2.A (domain 'PD1(supply VDD1:1.000)') does not need isolation, has 'is_isolated' attribute set at the destination side and has isolation strategy 'ISO1' specified for 'test.u2.A'	test.v	6
LPLSH05A (2) : Reports missing and incorrect level shifter strategies for low to high voltage difference crossings.		
LPLSH05B (1) : Reports missing and incorrect level shifter strategies for high to low voltage difference crossings.		
LPSVM08B (2) : Checks the presence of isolation cell at output terminals of power-domain, not having any		
LPSVM12A (1) : Isolation signal should not be driven/received by equivalent or relatively-off domain		

FIGURE 236. Violation message to PIH back-referencing

Click the ⚡ icon, provided on top of the *Violation Browser,* to open the ***Power Intent View*** window.

The ***Power Intent View*** window shows automatically highlighted power domains, supplies and strategies, which are relevant to the message being checked.

In *Figure 237*, you can see that the `TOP` domain, `PD1` domain, `VDD` power net, `VDD1` power net, and `ISO1` isolation strategy are highlighted.



Scope 1: /

Scope 2: /

Net 1: VDD

Net 2: VDD1

Property	Value
Relation	Disjoint
Isolation	Both
Level Shifter	Low_To_High

FIGURE 237. Highlighted domains, power supplies, and strategies

Batch Mode in SpyGlass

To use SpyGlass in the batch mode, specify the `-batch` command-line option along with the other commands, as shown below:

`spyglass -batch [-project <project-file>] [` *SpyGlass Explorer-Specific Options* `] [` *Batch, Goal, and Source File List Command-Line Options* `]`

NOTE: *If you do not specify a project file in the batch mode, SpyGlass Explorer runs in a normal batch mode in which SpyGlass Explorer-Specific Options do not work.*

This chapter provides details on various command-line options used in SpyGlass.

SpyGlass Explorer-Specific Options

Following are the SpyGlass Explorer-specific command-line options:

-goals <goals>

Use this command to run the specified goals.

Please note the following points:

- SpyGlass Explorer searches goals in the default methodology directory as set in the project file.
- You can specify goal names by using wildcard expressions. For details on wildcard search, see *Pattern Matching Across Features*.

NOTE: *The -goal option is given in the Linux shell command-line. Therefore, special characters (such as *, ?) in wildcard expressions must be used with the '\' escape character or enclosed within single or double quotes as appropriate so that the Unix shell does not interpret the expressions. See your Linux shell documentation for details.*

- If you do not specify a valid goal, SpyGlass uses the best possible matching alternatives to the given goal name.

Consolidated Goal Results Summary

When you run multiple goals by using the `-goals` command-line option, then at the end of the run, SpyGlass Explorer displays a consolidated results summary that contains the run result, message count per severity class, and the total messages reported, waived and generated for each goal run. In addition, the consolidated goals results summary displays the paths pointing to the spyglass.log file and various reports.

The sample consolidated goal results summary generated by SpyGlass is as follows:

```
-------------------------------------------------------------
Consolidated Results Summary for multi-goal run
-------------------------------------------------------------
Goal : Initial_RTL/Connectivity     Status : PASS
======================================================
    3 ERROR Severity Messages
    24 WARNING Severity Messages
    15 INFO Severity Messages

    Total Messages Generated        :        42
    Total Messages Waived           :        0
    Total Messages Reported         :        42

Run LogPath : /atrenta/test/VI-27088/Project-410/
Initial_RTL/Connectivity/spyglass.log

Run Results : /atrenta/test/VI-27088/Project-410/
Initial_RTL/Connectivity/spyglass_reports/


Goal : Initial_RTL/Simulation     Status : PASS
=====================================================
  4 ERROR Severity Messages
  157 WARNING Severity Messages
  17 INFO Severity Messages


  Total Messages Generated        :       178
  Total Messages Waived           :       0
  Total Messages Reported         :       178

 Run LogPath : /atrenta/test/VI-27088/Project-410/
 Initial_RTL/Simulation/spyglass.log

 Run Results : /atrenta/test/VI-27088/Project-410/
 Initial_RTL/Simulation/spyglass_reports/
```

-designread

Use this command to perform design read in the batch run of SpyGlass.

NOTE: *When the -designread option is given along with the -goals option, design read is performed first followed by goal run.*

Example

```
%> spyglass -project myproject.prj -batch -designread or
%> spyglass -project myproject.prj -batch -designread
-goals="Initial_RTL/Connectivity"
```

-showgoals

Use this command to print all the goals in the default methodology directory along with their recommended setup status and run status.

For regression goals, the `-showgoals` command prints the name of the regression goals in the Goal Status Summary report. To view the details of the regression goals, use the `-verbose` command.

If the goal run is complete, this command prints the corresponding message count by severity (Error/Info/Warning counts of messages per goal is displayed in the goal summary).

The following is the sample goal summary:

```
=======================================================================
Goal Status Summary
======================================================================
Goal                            Setup Status   Run Status    Results
                                                             by Severity
======================================================================
initial_rtl/lint/connectivity   Setup Optional Run Complete  (0/0/2/3)
initial_rtl/lint/simulation     Setup Optional Not Run Yet
initial_rtl/lint/synthesis      Setup Optional Not Run Yet
initial_rtl/lint/structure      Setup Optional Not Run Yet
initial_rtl/audit/block_profile Setup Optional Not Run Yet
initial_rtl/audit/rtl_audit     Setup Optional Not Run Yet
myReg3                              -                -        Regression
                                                              Goal
myReg1                               -               -        Regression
                                                              Goal
=======================================================================
Note: The format of the "Results By Severity" column is: ( Fatal / Errors
      / Warnings / Info )
      It contains "Regression Goal" keyword for regression goals, use
      -verbose option to get details on these goals.
* -  Goal(s) has one or more prerequisite goals which must be run
    prior to running this goal. The batch process has been configured to
```

```
     enforce this and will indicate which, if any, goals need to be run
    first. For details on the prerequisites requirements, please see the
     documentation for this methodology.
========================================================================
```

NOTE: *If you have selected the* Show Optional Goals *option in the* Preferences *dialog, the* `-showgoals` *option also prints the optional goals along with their recommended setup status and run status.*

-verbose

Regression goals summary is not shown by default with -showgoals. Specify this option to get Setup, Run status and Result summary for each goal specified as part of regression goals. This option is valid with -showgoals option only.

Following is a sample output from -showgoals in batchConsole run without -verbose option:

```
========================================================================
Goal Status Summary
========================================================================
Goal        Setup Status       Run Status     Results by Severity
========================================================================
testgoal0   Setup Optional    Not Run Yet
testgoal1   Setup Optional    Not Run Yet
testgoal2   Setup Optional    Not Run Yet
testgoal3   Setup Optional    Not Run Yet
myReg3      -                 -               Regression Goal
myReg1      -                 -               Regression Goal
========================================================================
Note: The format of the "Results By Severity" column is: ( Fatal / Errors
     / Warnings / Info )
     It contains "Regression Goal" keyword for regression goals, use
     -verbose option to get details on these goals.
* - Goal(s) has one or more prerequisite goals which must be run
    prior to running this goal. The batch process has been configured
   to enforce this and will indicate which, if any, goals need to be run
    first. For details on the prerequisites requirements, please see the
    documentation for this methodology.
```

```
========================================================================
```

If you specify -verbose option with the -showgoals option, then status is shown for each goal inside regression goals. The following tables are additionally decompiled prior to **Note:** in the above report, if -verbose option is also specified.

```
Regression Goal 'myReg3' Status Summary
========================================================================
Goal         Setup Status        Run Status     Results by Severity
========================================================================
testgoal1    Setup Optional      Not Run Yet
testgoal2    Setup Optional      Not Run Yet
========================================================================

Regression Goal 'myReg1' Status Summary
========================================================================
Goal         Setup Status        Run Status     Results by Severity
========================================================================
testgoal2    Setup Optional      Not Run Yet
testgoal3    Setup Optional      Not Run Yet
========================================================================
```

-group

Use this command to run the goals specified using the *-goals <goals>* option.

When you use this option, combined results for all goals is generated and stored in a common project working directory Group_Run. Use this -group command-line when you want to analyze the results of different goals together.

There are some limitations when running SpyGlass in batch mode with the -group option. These are:

- The –showgoals option does not report that the goals have been completed

- In goal selection, the goals are shown as been run, however, the summary data is not available
- Since a group has a single result directory, when analyzing the results of a goal from a group the run with show all results for that group
- The results of the group run cannot be used for the aggregated reports for a project, or consolidated management reports

-gen_aggregate_report

Use this command to generate aggregated report, such as project summary, DataSheet, or DashBoard report, in the batch mode. You can generate only one type of aggregated report at a time by using this command.

Following is the syntax of using the `-gen_aggregate_report` command:

```
-gen_aggregate_report <project_summary | datasheet |
dashboard>
-config_file <config-file> [-reportdir <dir>] [-DEBUG]
[ -LICENSEDEBUG ] [-project <prj-file>]
```

The above command requires you to specify two mandatory arguments, report type (`project_summary`, `datasheet`, or `dashboard`) and the configuration file (`-config_file` *`<config-file>`*).

Various options of this command are described in the following table:

TABLE 17

Option name	Description
-config_file	Specifies the name of a configuration file that contains a list of projects and run directories generated by goal/ source files or GUI. For the *Project Summary Report*, you can only specify a list of project files, because this report is specific to projects only. For the rest of the aggregated reports, such as the *DataSheet Report* and the *DashBoard Report*, you can specify a list of projects as well as run directories.
-project	(Optional) Specifies the name of a single project file. This option is considered only if you do not specify the `-config_file` option. In such cases, SpyGlass accepts the specified project file as the input. If you have specified both the `-config_file` and `-project` options, SpyGlass ignores the `-project` option and considers the`-config_file` option.
-reportdir	(Optional) Specifies the directory containing result files. By default, SpyGlass considers the value of this option as the current working directory.
-DEBUG	(Optional) Prints useful debug messages on STDOUT. This information includes various details such as the current project accessing, time stamps at various stages, etc. Information printed on STDOUT is also be dumped in the log file, aggregate_reports.log.
-LICENSEDEBUG	Prints license debug information.

-parallel_run

Use this command to run goals in parallel.

-host_config_file

Use this command to specify a host configuration file while running goals in parallel.

The host configuration file contains details, such as login type, maximum number of process that can run at a time and machine names on which goals should run.

Following is the sample host configuration file:

```
LOGIN_TYPE: lsf
MAX_PROCESSES: <num>
LSF_CMD: <bsub-command>
```

-enable_cmdline_debug

Use this command to generate the spyglass_cmdline_debug.log file that enables you to understand how SpyGlass arrives at the final set of command line options using the initial option set provided by you.

-oem_mode

Use this command to run SpyGlass Explorer in the OEM (Original Equipment Manufacturer) mode.

In the OEM mode, you will notice the following differences as compared to the normal SpyGlass Explorer run:

- The *Run Design Read* tab is disabled.
- You cannot run non-OEM goals in this mode. If you try loading such goals, SpyGlass Explorer reports an error and does not load these goals.
- The save/restore feature is disabled.
- Parallel goal run is not supported.
- The *Common Setup* tab under the *Goal Setup and Run* tab is disabled.

-I

See *I*.

-shell

Use this command to invoke `sg_shell` from SpyGlass as shown below:

```
% spyglass -shell
```

You can perform the following tasks while invoking `sg_shell` from SpyGlass:

- Load a project file using `sg_shell` as shown below:

  ```
  spyglass -shell -project <project_file_name>.prj
  ```

- Run one or more goals from an existing project file using `sg_shell` as shown below:

  ```
  % spyglass -shell -project<project_file_name.prj> -goals
    "<goal1>,<goal2>,..."
  ```

- Playback a set of Tcl commands in the `sg_shell` as shown below:

  ```
  % spyglass -shell -tcl <file_name>.tcl
  ```

Batch, Goal, and Source File List Command-Line Options

This section explains the various batch, goal, and source file list command-line options.

These SpyGlass Explorer-specific commands can be:

- Design-related commands which are stored in a project file.
- Rule/product specific commands that are stored in a methodology file.

NOTE: *The following classic-batch options are converted into the batch format, but are not mapped directly to the project or methodology files:*

-f	*-template <goal-names>*	*-templatedir*	*-wdir*	*-logfile*	-namevdb

Specifying Source Files through Command-Line

Provide a space-separated list of HDL files on command-line.

You can provide file names with or without a path (actual or relative).

If you provide only file name, the file is searched in the current directory.

You can also use wildcard expressions to specify file names. For example, specifying *.vhd will result in processing all files with extension .vhd in the current directory.

Specifying Rule Parameters through Command-Line

Specify rule parameters in the goal or source files in the following format:

```
-<name>=<value>
```

Where:

- *<name>* is the name of the rule parameter.
- *<value>* is the parameter value.

NOTE: *To know parameters and their corresponding usage for each product, refer to product documentation.*

If you specify a command-line option that SpyGlass does not recognize, it creates a parameter of that name. If the option has a value, it assigns that value to the parameter.

NOTE: *Specifying parameters in goal or source files is equivalent to using the following project file command:*

```
set_parameter <param-name> <param-value>
```

Classic Batch Command-Line Options and their Corresponding Project File Commands

The following table lists the classic-batch command-line options and their equivalent project file commands:

Classic-Batch Command-Line Options	Corresponding Project File Commands
-87	*87*
-addrules <rule-list>	*addrules*
-allow_celldefine_as_top	*allow_celldefine_as_top*
-allow_module_override	*allow_module_override*
-cell_library	*cell_library*
-check_celldefine	*check_celldefine*
-checkdu="<du-name>"	*checkdu*
-checkip="<IP-name>"	*checkip*
-checkTopDu <rule-names>	*checkTopDu*
-configfile <file-name>	*configfile*
-consolidate_reportdir	*consolidate_reportdir*
-convert_udp_to_latch	*convert_udp_to_latch*
-def <file-name>	*def* option of the *read_file* command
+define{+<macro-name>}	*define*
-define_incr_dirmap	*define_incr_dirmap*
-define_severity <severity-expression>	*define_severity*
-disable_encrypted_hdl_checks	*disable_encrypted_hdl_checks*
-disable_hdllibdu_lexical_checks	*disable_hdllibdu_lexical_checks*

Classic-Batch Command-Line Options	**Corresponding Project File Commands**
-disable_report	*disable_report*
-disallow_view_delete	*disallow_view_delete*
-dump_all_modes	*dump_all_modes*
-dw	*dw*
-enable_const_prop_thru_seq	*enable_const_prop_thru_seq*
-enable_hdl_encryption	*enable_hdl_encryption*
-enable_module_based_reporting	*enable_module_based_reporting*
-enable_pass_exit_codes	*enable_pass_exit_codes*
-enable_precompile_vlog	*enable_precompile_vlog*
-enable_save_restore	*enable_save_restore*
-enable_sgdc_debug	*enable_sgdc_debug*
-enable_sglib_debug	*enable_sglib_debug*
-enableSV	*enableSV*
-f	*sourcelist* option of the *read_file* command
-gateslib <file-name>	*gateslib* option of the *read_file* command
-handlememory	*handlememory*
-hdlin_synthesis_off_skip_text	*hdlin_synthesis_off_skip_text*
-hdlin_translate_off_skip_text	*hdlin_translate_off_skip_text*
-hdllibdu	*hdllibdu*
-higher_capacity	*higher_capacity*
-ignore_builtin_rules	*ignore_builtin_rule*
-ignore_builtin_spqdir <dir>	*ignore_builtin_spqdir*
-ignorerules {<rule-name>}	*ignorerules*
-ignorelibs	*ignorelibs*
-ignore_undefined_rules	*ignore_undefined_rules*
-ignorewaivers	*ignorewaivers*
-inferblackbox	*inferblackbox*
-inferblackbox_iterations <int-value>	*inferblackbox_iterations*
-lef <file-name>	*lef* option of the *read_file* command

Classic-Batch Command-Line Options	**Corresponding Project File Commands**
-lib <logical-lib-name> <physical-lib-name>	*lib*
[+libext{+<ext-name> }	*libext*
-LICENSEDEBUG	*LICENSEDEBUG*
-lvpr (<number>	*lvpr*
-macro_synthesis_off	*macro_synthesis_off*
-mapSuffixList <suffix-names>	*mapSuffixList*
-mapPrefixList <prefix-names>	*mapPrefixList*
-mapVirtualClkByName = <yes | no>	*mapVirtualClkByName*
-mixed	`mixed` option of the *language_mode* command
-mthresh	*mthresh*
-net_osc_count_limit <limit>	*net_osc_count_limit*
-nobb	*nobb*
-nodefparam	*nodefparam*
-noelab	*noelab*
-nopreserve	*nopreserve*
-noreport	*noreport*
-norules	*norules*
nosavepolicies <product-list> -nosavepolicy <product-name>	*nosavepolicy/nosavepolicies*
-overload {<named-overload>}	*overload*
-overloadpolicy {<product-name>	*overloadpolicy*
-overloadrules <expression>	*overloadrules*
-ovl_verilog <file-name>	*ovl_verilog*
-ovl_vhdl <file-name>	*ovl_vhdl*
--perflog	*perflog*
-plib <file-name>	*plib* option of the *read_file* command
-portparam <list-of-space-separated-strings>	*portparam*
-pragma <name>	*pragma*
-prefer_tech_lib	*prefer_tech_lib*

Classic-Batch Command-Line Options	**Corresponding Project File Commands**
-preserve_mux	*preserve_mux*
-print_sortorder_only	*print_sortorder_only*
-relax_hdl_parsing	*relax_hdl_parsing*
-remove_work	*remove_work*
-report { <formats> }	*report*
-report_adjustment_waiver	*report_adjustment_waiver*
-reportfile <file-name>	*reportfile*
-report_inst_backref	*report_inst_backref*
-report_ip_waiver	*report_ip_waiver*
-report_max_inst <value>	*resetall*
-report_max_size=<value>	*report_max_size*
-report_per_policy	*report_per_policy*
--report_style=<list-values>	*report_style*
+resetall	*resetall*
-rules {<rule-name>}	*rules*
-savepolicies <product-list> -savepolicy <product-name>	*savepolicy/savepolicies*
-sfcu	*sfcu*
-sdc2sgdc	*sdc2sgdc*
-sdc2sgdcfile <file-name>	*sdc2sgdcfile*
-sdc2sgdc_mode <mode-name>	*sdc2sgdc_mode*
-sgdc <file-name>	*sgdc* option of the *read_file* command
-sglib <file-name>	*sglib* option of the *read_file* command
-sgsyn_clock_gating	*sgsyn_clock_gating*
-sgsyn_clock_gating_threshold <num>	*sgsyn_clock_gating_threshold*
--sgsyn_enable_latch_removal	*sgsyn_enable_latch_removal*
--sgsyn_loop_limit <value>	*sgsyn_loop_limit*
-show_sdc_progress	*show_sdc_progress*
-show_lib	*show_lib*
-skip_rules_for_fast_restore	*skip_rules_for_fast_restore*

Classic-Batch Command-Line Options	Corresponding Project File Commands
-sort	*sort*
-sortrule	*sortrule*
-stop <module-name>	*stop*
-stopdir <dir-name>	*stopdir*
-stopfile <file-name>	*stopfile*
-support_sdc_style_escaped_name	*support_sdc_style_escaped_name*
-target <lib-name-list>	*target*
-testsynth	*designread_enable_synthesis*
-top <module-name>	*top*
-use_goal_rule_sort	*use_goal_rule_sort*
-use_scan_flops	*use_scan_flops*
-v <lib-name>	*v*
-verilog	*verilog* option of the *language_mode* command
-vhdl	*vhdl* option of the *language_mode* command
-w	*w*
-waiver <file-name>	*waiver* option of the *read_file* command
-wdir	*projectwdir*
-work <dir-name>	*work*
-write_sdc	*write_sdc*
-y <lib-dir-name>	*y*

Other Command-Line Options

This section describes other classic batch command-line options, such as informational options and mode-selection options.

NOTE: *The class-batch feature has entered the End-of-Life stage and will not be supported from now on. This section is listed only for the purpose of backward-compatibility.*

Informational Command-Line Options

The SpyGlass informational options provide information about SpyGlass software environment and do not require you to specify a design file.

NOTE: *SpyGlass does not create a violation database file and log file when you invoke SpyGlass with any of the informational options.*

NOTE: *You do not need to specify the* `-batch` *command-line option with the informational command-line options.*

-h

(Optional) Prints site-specific SpyGlass help (that is, information in <your-inst-dir>/SPYGLASS_HOME/doc/site-help.txt) to the stdout and exits.

You can access site-specific help from SpyGlass by entering the `-h` or *-help* option as in the following example:

```
spyglass -h
```

```
spyglass -help
```

The information you see depends on what customer-specific SpyGlass help information has been set up for your company. If no such help has been set up, a message appears saying site-specific help does not exist.

-help

Same as the *-h* option.

-quickstart

(Optional) Prints the SpyGlass Quick Start help (that is, information in <your-inst-dir>/SPYGLASS_HOME/doc/quickstart.txt) to the stdout and exits.

-usage

(Optional) Prints the help of various options based on the mode (SpyGlass Explorer or batch) in which you want to run SpyGlass.

- If `-usage` is specified alone, SpyGlass help is shown and a reference of getting batch help is shown.
- If `-usage` is specified with any SpyGlass Explorer-related option, help on SpyGlass Explorer is displayed.

- If `-usage` is specified with any SpyGlass Explorer-related options as well as a batch-related option, a message is displayed indicating that the options specified are not compatible, and only SpyGlass Explorer-related help is shown.
- If `-usage` is specified with any batch-related option then batch-related help is displayed.

-version

(Optional) Prints the SpyGlass version to the stdout and exits.

If several versions of SpyGlass are available on your company's network, it is important to know which version of SpyGlass you are using.

To print the release number of the SpyGlass version you are using, enter the `-version` option in a SpyGlass command line:

```
spyglass -version
```

When the optional argument `-policies` (or `-policy`) is also specified with one or more registered product names, SpyGlass prints the version and the minimum required SpyGlass version for each specified product to stdout and exits.

Mode Selection Command-Line Options

The SpyGlass mode selection command-line options decide the SpyGlass operating mode (the GUI or the batch mode) and the SpyGlass operating language.

-32bit

Specifies SpyGlass to run in 32-bit mode.

-64bit

Specifies SpyGlass to run in 64-bit mode.

By default, 64-bit SpyGlass binaries are executed on 64-bit architectures

-batch

(Optional) Specifies to run SpyGlass in batch mode.

By default, SpyGlass starts in the GUI mode.

-gui

Specifies the GUI mode of SpyGlass.

When you specify `-gui` or `-gui=se`, SpyGlass Explorer is invoked.

Other Command-Line Options

-fullpolicy

(Optional) Specifies to run all rules (except those rules that are controlled by parameters, which run based on the parameter values) of all selected products.

All Atrenta standard products have been reorganized to run only a selected set of rules when you select to run the complete product (by specifying the product name using the `-policy`/`-policies` command-line option and not specifying one or more rules of that product using the `-rules` command-line option). The selected set of rules has been chosen to quickly highlight the value of the particular product.

Earlier, such specification used to run all rules of the specified product. You can still run all rules (except those rules that are controlled by parameters, which run based on the parameter values) of a product by specifying the `-fullpolicy` command-line option.

NOTE: *There is no change in SpyGlass behavior when you use the* `-rules` *command-line option to specify rules to be run or when you run a goal.*

-policies | -policy

(Optional) Specifies a comma-separated list of products to be loaded.

SpyGlass comes with a number of standard products. Each product file is named as *<product-name>*`-policy.pl`. where *<product-name>* is the product name and is located in its own directory <your-inst-dir>/SPYGLASS_HOME/policy/<product-name>. For example, the product file for SpyGlass lint solution is lint-policy.pl, and is located in the <your-inst-dir>/SPYGLASS_HOME/policy/lint directory.

NOTE: *If you specify a policy.pl file at the command-line that has an error, a fatal error message is displayed and the SpyGlass GUI is not invoked.*

NOTE: *Use of rule parameters at the command-line may affect the functional behavior of* *some rules and may determine if those rules should be run or not run. Refer to the corresponding product documentation for details on the rule parameters and the effect of the parameters on some rules.*

More information is available in the respective Rule Reference document for each product that can be accessed using *The spydocviewer utility* or *The spyhelpviewer utility*.

In addition, you can create your own customized products and use them just like Atrenta Standard products.

You can tell SpyGlass to use multiple products at one time.

When you apply a product, you are telling SpyGlass to check your design source files against the rules that are in this product.

To apply one product

To apply a single product on your design files, use the `-policy` command-line option or `-policies` command-line option as follows:

```
spyglass -batch -policy=<product-name> ...
spyglass -batch -policies=<product-name> ...
```

For example, to apply SpyGlass OpenMore solution, use the following command-line:

```
spyglass -batch -policy=openmore ...
```

To apply SpyGlass STARC solution, use the following command-line:

```
spyglass -batch -policies=starc ...
```

To apply more than one product

You can apply multiple products to a design file during a single run.

To apply multiple products on your design files, use the `-policy` command-line option or `-policies` command-line option as follows:

```
spyglass -batch -policy=<product-name-list> ...
spyglass -batch -policy=<product1-name>
  -policy=<product2-name> ...
spyglass -batch -policies=<product-name-list> ...
```

Where *`<product-name-list>`* is a comma-delimited list of product names (**with no spaces**).

For example, to apply SpyGlass lint Solution and SpyGlass OpenMore solution, use any of the following command-line options:

```
spyglass -batch -policy=lint,openmore ...
spyglass -batch -policy=lint -policy=openmore ...
spyglass -batch -policies=lint,openmore ...
```

NOTE: *You can run SpyGlass without any products by specifying the* `none` *value (for example,* `-policies=none`*). Then, only the SpyGlass built-in rules are run on your design files (HDL files, SpyGlass Design Constraints files, Library files, etc.), that is, your design files are effectively syntax-checked.*

NOTE: *Also, see the* `norules` *command that has a similar function.*

-run

Analyzes a design with the specified settings after invoking SpyGlass GUI.

Use the `-run` command-line option when you want to work in the SpyGlass GUI and want the design to be immediately analyzed after invoking the SpyGlass GUI.

You must provide all essential SpyGlass command-line options with the `-run` command-line option. Otherwise, the design will not be analyzed.

-template <goal-names>

NOTE: *Using this command is equivalent of using the -goals <goals> command or the* `current_goal` *<goal-name> project file command.*

Runs the specified goals.

Specify a goal name in the following manner:

```
-template <method-name>/<goal-name> ...
```

Where:

- *<method-name>* is the name of the methodology.
- *<goal-name>* is the name of a goal under the *<method-name>* methodology.

To run multiple goals, use the `-template` command in any of the following ways:

- Method 1

```
-template <method1-name>/<goal1-name>
-template <method2-name>/<goal2-name> ...
```

- Method 2

```
-template "<method1-name>/<goal1-name>
          <method2-name>/<goal2-name>"
```

Methodology names are the names of directories containing goal files. The goal name is the first part of the name of a goal file.

For example, `clock_reset_integrity` goals present in the SpyGlass CDC methodology (that is, the <your-inst-dir>/SPYGLASS_HOME/Methodology directory) contains the following files for each language mode:

- `clock_reset_integrity-verilog.spq`
- `clock_reset_integrity-vhdl.spq`
- `clock_reset_integrity-mixed.spq`

To run these goals, specify the following commands:

```
spyglass -batch -vhdl -template Clock-reset/
clock_reset_integrity ...
spyglass -batch -verilog -template Clock-reset/
clock_reset_integrity ...
spyglass -batch -mixed -template Clock-reset/
clock_reset_integrity ...
```

Depending on the language specification, SpyGlass selects and runs the corresponding goal file.

SpyGlass searches for the specified goals in the following order:

1. The directory specified using the *-templatedir* command-line option (overrides the *DEFAULT_TEMPLATE_DIRECTORY* configuration file setting, if any)
2. Paths specified using the *-I* command-line option
3. In the <your-inst-dir>/SPYGLASS_HOME/Methodology directory

NOTE: *If the specified goals are not found in any of the above paths, SpyGlass prints the* *best possible matches for the goals, which are present in the above mentioned paths.*

You can specify the `-template=none` command-line option to indicate that no goals should be run during SpyGlass analysis. All goals specified after specifying the `-template=none` option are ignored. However, the goals specified before specifying the `-template=none` option are processed as before.

-templatedir

(Optional) Specifies the directory where the goals specified using the *-template <goal-names>* command-line option are to be searched.

If the `-templatedir` command-line option is specified, SpyGlass searches for the specified goals in the following order:

1. The directory specified using the `-templatedir` command-line option
2. The paths specified using the `-I` command-line option
3. In the <your-inst-dir>/SPYGLASS_HOME/Methodology directory

If the `-templatedir` command-line option is not specified, SpyGlass searches for the specified goals first in the directory specified using the *DEFAULT_TEMPLATE_DIRECTORY* configuration file setting and then in the other directories mentioned above.

NOTE: *The goal directory structure may contain one or more methodology directories within which the goals are present. In such cases, you should set the* `-templatedir` *option one level up than the methodology directory to see goals below each methodology. For example, if a goal,* `T`*, is present under the methodology directory,* `M`*, which is further present under the directory,* `D1/D2`*, you should set the value of the* `-templatedir` *option as* `D1/D2` *and the value of the -template option as* `M/T`*.*

-classic_mode

(Optional) Executes the CLASSIC mode rules only (which run on non-optimized netlist) and disables rules of EST and ESYNTH mode.

-est_mode

(Optional) Executes the EST mode rules only (which run on optimized and technology-mapped netlist) and disables rules of CLASSIC and ESYNTH mode.

-esynth_mode

(Optional) Runs only ESYNTH mode rules (which run on optimized netlist) and disables rules of CLASSIC and EST mode.

NOTE: *SpyGlass displays an error message if you specify a combination of -est_mode, -esynth_mode, or -classic_mode options.*

NOTE: *While running SpyGlass in -enable_save_restore mode along with any of the -est_mode/-esynth_mode/-classic_mode option, it is recommended that you specify different precompile work directory (using the -lib WORK <WORK-DIR> option) for each mode. If you do not specify a different precompile work directory for each mode, SpyGlass may synthesize the design every time you run SpyGlass in a different synthesis mode and re-save the database for each synthesis mode.*

-f

NOTE: *This command is equivalent of using the following project file command:*

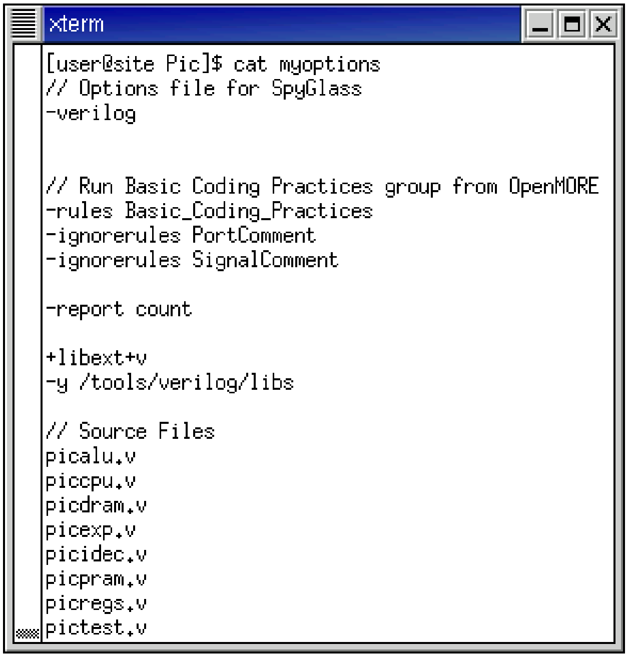
```
read_file -type sourcelist <file-names>
```

Example of Using the -f Command in Classic batch

The following example specifies the myoptions file present in the MYFILES directory:

```
spyglass -batch -f MYFILES/myoptions -verilog design.v
```

Where, myoptions contains the following details:



```
[user@site Pic]$ cat myoptions
// Options file for SpyGlass
-verilog


// Run Basic Coding Practices group from OpenMORE
-rules Basic_Coding_Practices
-ignorerules PortComment
-ignorerules SignalComment

-report count

+libext+v
-y /tools/verilog/libs

// Source Files
picalu.v
piccpu.v
picdram.v
picexp.v
picidec.v
picpram.v
picregs.v
pictest.v
```

FIGURE 238. myoptions File

-logfile

(Optional) Sets the name and location of the SpyGlass log file.

By default, the log file name is spyglass.log and it is created in the current working directory as specified with the *-wdir* command-line option (default is the current directory).

When you specify the `-logfile` command-line, the SpyGlass log file name and location are determined as follows:

- If the specified log file name is a simple file name (for example mylog.txt), the log file will be created as mylog.txt in the current working directory as specified with the *-wdir* command-line option (default is the current directory).
- If the specified log file name is an absolute file path name (for example /usr/john/logs/mylog.txt), the log file will be created as mylog.txt in the /usr/

john/logs directory irrespective of the *-wdir* command-line option specification.

NOTE: *The specified directory (the* /usr/john/logs *directory in the example) must exist.*

- If the specified log file name is a relative file path name (for example myprojects/mylog.txt), the log file will be created as mylog.txt in the myprojects directory under the current directory irrespective of the *-wdir* command-line option specification.

NOTE: *The specified directory (the* myprojects *directory in the example) must exist under the current directory.*

-rulegroup_display_depth

(Optional) Enables you to specify the number of the subgroups to be displayed in reports and message tree.

By default, the value of the `-rulegroup_display_depth` option is set to 2. You can change this value to specify different number of subgroups to be displayed.

If you specify the value of this option as 0 then all the subgroups are displayed.

NOTE: *The* `-rulegroup_display_depth` *option can be specified only if the* `display_rulegroup` *option is used.*

-wdir

NOTE: *This command is equivalent to the following project file command:*

```
set_option projectwdir <dir-name>
```

Specifies the output directory for SpyGlass if you want output to be generated at a location different from the current working directory.

Then, the SpyGlass output (SpyGlass schematics, Violation Database file, Log file, and Reports files) will be written to this new path.

Another application of the `-wdir` command-line option is to run SpyGlass from a read-only area and create the output files in another location.

If you do not specify this option, SpyGlass uses the current directory as the output directory.

NOTE: *The Verilog/VHDL precompiled libraries are always controlled by the* `-lib` *specification only.*

SpyGlass produces a number of different output files when it runs, including the Violation Database file, the log file and files for creating the schematics of the synthesized design for use in the SpyGlass (these are stored in a hidden directory <vdbfilename>.spysch). By default, all of these files are stored in the current directory from where SpyGlass was invoked.

It is possible to change the storage location of these output files using the `-wdir` option.

To change the current working directory

To have all SpyGlass output files stored in a different location from the current context, enter the `-wdir` option followed by the path to the new working directory, followed by the language, product to be checked, name of the design and any other desired options.

For example, to direct SpyGlass to store output files in directory mydir, enter:

```
spyglass -batch -wdir ../mydir -vhdl design.vhd
```

-gateslib

NOTE: *This command is equivalent to the following project file command:*

read_file `-type` *gateslib* `<file-name>`

(Optional) Specifies gate libraries containing functionality information about gate cells instantiated in a design.

NOTE: *If a file name includes wildcard characters (*, or ?), the name should be enclosed in single quotes, and the wildcard characters should be preceded by a backslash (\) to be treated as literal. For example, if your file name is '`abc*d`', you need to refer to it as '`abc\*d`'.*

However, if you want to refer to two files, for example '`abc1d`' and '`abc2d`', you can specify them using SpyGlass pattern matching support, that is, you can specify "`abc*d`" in this case. For details on pattern matching support, see Pattern Matching Across Features.

Example of Using the -gateslib Command in Classic Batch

The examples are described below. These examples are also applicable for the `spyglass_lc` utility.

- Specifying a single gate library

The following example specifies the mygates.lib Synopsys Liberty-format file to be analyzed:

```
spyglass -batch -gateslib mygates.lib
-enable_gateslib_autocompile -verilog -policy=erc
mydesign.v
```

- Specifying multiple gate libraries

  The following example specifies the mygates1.lib and mygates2.lib gate libraries:

```
// Method 1 (Using multiple -gateslib commands)

spyglass -batch -gateslib mygates1.lib \
  -gateslib mygates2.lib -enable_gateslib_autocompile
-verilog -policy=erc mydesign.v


// Method 2 (Using a single –gateslib command in which a
// space-separated list of library names are specified)

spyglass -batch -gateslib “mygates1.lib mygates2.lib” \
-enable_gateslib_autocompile -verilog -policy=erc
mydesign.v


// Method 3: Combining the above two methods:

spyglass -batch -gateslib “a.lib b.lib” \
-enable_gateslib_autocompile -verilog -policy=erc
mydesign.v -gateslib c.lib
```

- Specifying functionality information of a gate cell in a Verilog library file

  SpyGlass gives preference to a Verilog library file over Synopsys Liberty format file (.lib file) to pick the functionality information of a gate cell.

Example:

Consider the x module declared in the x.v file. The functionality information of this module is present in the lib1.v library file as well as the mylib1.lib file.

Now consider you specify the following command:

```
spyglass x.v -v lib1.v -gateslib mylib1.lib -verilog
```

In this case, SpyGlass picks the functionality information of the `x` module from the lib1.v library rather than the mylib1.lib file.

- Specifying the functionality information of a gate cell in a Verilog/VHDL design file

  Similar to the Verilog library file, SpyGlass gives preference to a Verilog/VHDL design file over Synopsys Liberty-format file (.lib file) to pick the functionality information of a gate cell.

  Example:

  Consider that the functionality information of the `m` module (declared in the m.v file) is present in the mod1.v file and mylib1.lib file.

  Now consider the following command:

  ```
  spyglass m.v mod1.v -gateslib mylib1.lib -verilog
  ```

  In this case, SpyGlass picks the functionality information of the `m` module (declared in the m.v file) from the mod1.v file rather than the mylib1.lib file.

  Alternatively, you can use the following command:

  ```
  spyglass_lc -gateslib <libfile-name>
     -lib <logical-lib-name> <physical-lib-name>
  ```

- Mixed-Language

  Re-run the SpyGlass Library Compiler specifying the Verilog and VHDL RTL description files directly (for both Verilog and VHDL), using the `-v/-y/+libext` command-line option (for Verilog), or using the `-lib` command-line option (for both Verilog and VHDL).

  In this case, the SpyGlass Library Compiler searches for cell descriptions first in the Verilog domain, then in the VHDL domain, and finally in the gates library files.

In case, the functional description for a cell is available both in the library and in a user-specified HDL source file, SpyGlass stores the functional view as follows:

Library	HDL	Functional View
Available and can be translated	Not specified	Library description is stored
Available and cannot be translated	Not specified	Only port interface from the library is stored
Available and can be translated	Specified	HDL description is stored
Available and cannot be translated	Specified	HDL description is stored
Available and can be translated	Specified but is not synthesizable	Library description is stored
Available and cannot be translated	Specified but is not synthesizable	Only port interface from the HDL is stored

-param

Specifies parameters used during Verilog/VHDL analysis run.

NOTE: *For SpyGlass Explorer-specific usage of this command, see Set HDL Parameter(s) Value.*

Defining a Value for a Generic/Parameter

To set a value for a parameter, you must specify an instance of the parameter you wish to define together with the value you wish to set. In VHDL, this means the `entity_name.generic_name`, while in Verilog it means the `module_name.parameter_name`.

For example, to set the VHDL generic `width` in entity `control` to 8, enter the following as part of the SpyGlass invocation:

```
-param control.width=8
```

To set the Verilog parameter `limit` in module `block1` to 4, enter the following as part of the SpyGlass invocation:

```
-param block1.limit=4
```

Overriding the Value of a Generic/Parameter

In batch, you can override a generic/parameter value present in RTL by using the `-param` command-line option.

In case if any issues are encountered while overriding generic/parameter values, SpyGlass reports appropriate messages, as shown below:

```
Incorrect argument passed to -param. Correct usage is : '-param
<key>=<value>', where <key> is either <ent>.<gen> or
<mod>.<param>


'-param' option specified multiple times on the command line
for the entity/module "<entity/module-name>" generic/param
"<generic/parameter-name>" using the last value: '<value>'
```

Other than the above messages, SpyGlass also reports violation of the *CMD_param01*, *CMD_param02*, *CMD_param03*, *CMD_param04*, *CMD_param05*, and *CMD_param06* rules depending upon different cases. For details on these rules, refer to *BuiltIn Rules Reference Guide*.

Overriding Generics in VHDL

In VHDL, you can override the following type of values:

- Integer, positive, or a natural value, as shown in the following examples:

Integer value	`-param entity_name.generic_name=20`
Binary value	`-param entity_name.generic_name="2\"10\""`
Octal value	`-param entity_name.generic_name="8\"76\""`
Decimal value	`-param entity_name.generic_name="10\"93\""`
Hexadecimal value	`-param entity_name.generic_name="16\"AF\""`

- Boolean/enum value, as shown in the following examples:

Boolean value	`-param entity_name.generic_name=true`
Enum value	`-param entity_name.generic_name=red`

- Bits, as shown in the following examples:

Bit value	`-param 'entity_name.generic_name="0"'` Note that if single quotes around "0" are not present, the value is considered as the decimal value 0.
bit_vector value	`-param 'entity_name.generic_name="0000"'`

- std_logic, as shown in the following examples:

std_logic	`-param 'entity_name.generic_name="1"'` Note that if single quotes around "1" are not present, the value is considered as the decimal value 1.
std_logic_vector	`-param 'entity_name.generic_name="0000"'`

- String value, as shown in the following example:

String Value	`-param entity_name.generic_name=ABC`

- Aggregate value, as shown in the following example:

Aggregate Value	`-param entity_name.generic_name="others=>\"0\""`

Overriding Parameters in Verilog

In Verilog, you can override parameter values with the following type of values:

- Integer

  When you override a parameter value with an integer value or value in a valid based number format, it is considered as overriding the parameter value with an integer value.

  Following are some examples:

Integer	`-param "module_name.parameter_name=123"`
Binary	`-param "module_name.parameter_name=16'b10101010"`
Octal	`-param "module_name.parameter_name=32'o657"`

Decimal	`-param "module_name.parameter_name=16'd94"`
Hexadecimal	`-param "module_name.parameter_name=32'hAB4"`

NOTE: *If you pass a value in an invalid based number format, SpyGlass reports a violation.*

- String

  When you override a parameter with a value containing alphabets and/ or special characters (with or without numerical values), it is considered as overriding with a string value.

  Following are some examples:

  ```
  -param "module_name.parameter_name=12ab3"
  -param module_name.parameter_name=abc
  -param "module_name.parameter_name=abc"
  ```

-testsynth

NOTE: *This command is equivalent to the designread_enable_synthesis project file command:*

(Optional) Causes SpyGlass to elaborate and synthesize the design and report elaboration and synthesis messages.

Use the `-testsynth` command-line option to check the design for elaboration/synthesis issues without running any rules.

Thus, you must specify the norules command along with the `-testsynth` command-line option.

-debug_proc

(Optional) Dumps procedure call trace information for errors inside the procedure definition.

By default, the value of this command-line option is set to `no`, and SpyGlass does not dump any procedure call trace information.

To dump the procedure call trace information, specify `-debug_proc=yes` or `-debug_proc` on command-line.

Consider the following example (with line numbers highlighted):

```
1    # file1.sdc
2    proc proc2 { period_arg2 } {
```

```
3    create_clock -name CLK1 in1 $period_arg2
4    create_clock -name CLK2 in1
5    }
```

```
1    # file2.sdc
2    proc proc1 { period_arg1 } {
3    proc2 $period_arg1
4    }
5    proc proc0 { period_arg0 } {
6    # First proc
7    proc1 $period_arg0
8    }
9    proc0 10
```

For the above example, SpyGlass flags the following SDC errors for the `create_clock` command in the moresimple report:

```
SDC_106  Error   file1.sdc   3      10   Incorrect argument
"10" for "create_clock" (too many arguments?)
```

```
SDC_145  Error   file1.sdc   4      10   -period value
missing
```

However, if you want more information about the procedure call-trace, use the `-debug_proc` option. When you specify this option, SpyGlass generates the debugProcInfo file in the <wdir>/spyglass_spysch/spyglass_sdc/ directory and dumps the procedure call-trace information in this file in the following format:

```
SDC_106 File/Line : file1.sdc/3 (Procedure Trace : proc0
(file2.sdc,9) --> proc1 (file2.sdc,7) --> proc2 (file2.sdc,3)
)
```

```
SDC_145 File/Line : file1.sdc/4 (Procedure Trace : proc0
(file2.sdc,9) --> proc1 (file2.sdc,7) --> proc2 (file2.sdc,3)
)
```

+incdir

Searches the specified path for include files.

NOTE: *This command is equivalent to the incdir project file command.*

Examples of Using +incdir Command in Classic-Batch

Following are some examples:

- To specify directories containing `'include` files, use the `+incdir` option followed by the paths to the directories separated by + signs, the `-verilog` option and your design file name.

  For example, to specify the global directory, use the following command:

  ```
  +incdir+/u/<user_name>/global -verilog mydesign.v
  ```

- If the name of an `'include` directory contains the + character, specify that directory by using the `-incdir` command instead of using the `+incdir` command. For example, if the directory name is `abc+bcd`, specify this directory by using the following command:

  ```
  -incdir abc+bcd
  ```

- To specify multiple directories by using the `-incdir` command, specify a space-separated list of directories, as shown in the following example:

  ```
  -incdir "abc+bcd xyz"
  ```

- You can use the `+incdir` and `-incdir` commands together, as shown in the following example:

  ```
  -incdir abc+bcd +incdir+xyz+
  ```

NOTE: *If your directory name includes wildcard characters (*, or ?), the name should be enclosed in single quotes, and the wildcard characters should be preceded by a backslash (\) to be treated as literal. For example, if your directory name is '`abc*d`', you need to refer to it as '`abc\*d`'.*

However, if you want to refer to two directories, for example '`abc1d`' and '`abc2d`', you can specify them using SpyGlass pattern matching support, that is, you can specify "`abc*d`" in this case. For details on pattern matching support, see Pattern Matching Across Features.

+define

Adds the specified macro definitions.

NOTE: *This command is equivalent to the define project file command.*

Examples of Using +define Command in Classic-Batch

Following are some examples:

- To set the value of a ‘define macro, specify the `+define` command followed by the macro name and a value (separated by a + sign), the `-verilog` option and the design file name.

  For example, to set the macro State0 equal to 3, enter:

  ```
  spyglass -batch +define+State0=3 -verilog mydesign.v
  ```

- The following command sets the value of the `State0` and `State1` macros to `3` and `5`, respectively:

  ```
  +define+State0=3+State1=5 -verilog <file-name>
  ```

- The ‘define macros can also be included in the *.v files. However, these files must be analyzed by SpyGlass first before analyzing the remainder design files. Therefore, such files must be listed first on the command-line. For example, consider a define.v file that has the following format:

  ```
  // comment
  'define State0 3
  'define State1 5
  etc.
  ```

  In this case, define.v file must be listed first on the SpyGlass command-line, as shown below:

  ```
  spyglass -batch -verilog define.v design.v
  ```

--gdb

(Optional) Invokes GDB (GNU Project Debugger) during the SpyGlass run.

Use the `--gdb` option to debug your custom rules.

You must have the GDB tool suite installed in your file system.

The GDB tool suite is searched in the following order:

1. Path to the GDB executable set using the `SPYGLASS_GDB_PATH` environment variable
2. The /usr/local/bin directory
3. The /usr/bin directory
4. Path set in your `PATH` environment variable

SpyGlass Configuration File Setting Override Options

SpyGlass has the Configuration File feature using which you can specify configuration settings like default startup mode (the SpyGlass GUI or batch), default product to be run, default language setting, default report format etc.

The Configuration File settings can be overridden by specifying certain command-line options directly on the command-line or indirectly in a command file as follows:

Configuration File Setting	Value	Overriding Command-line Option
DEFAULT_STARTUP_MODE	gui	*-batch*
	batch	-gui
USE_32_BIT_EXECUTABLE_ONLY	no	
	yes	-32bit
DEFAULT_LANGUAGE_MODE	VHDL	-verilog, -def, or -mixed
	Verilog	-vhdl, -def, or -mixed
	Mixed	-verilog, -vhdl, or -def
	DEF	-verilog, -vhdl, or -mixed
	none	-verilog, -vhdl, -def, or -mixed
DEFAULT_TEMPLATE	Any	-template
DEFAULT_POLICY_FOR_SPYEXPLAIN	Any	-policies | -policy
DEFAULT_REPORT_FORMAT	default, <report-name>	-report_per_policy, -noreport
	none	-report_per_policy
DEFAULT_PRAGMA	default, <pragma-name-list>	-pragma=<pragma-name-list>, -pragma=nopragma
	none	-pragma=<pragma-name-list>
VHDL_LIB_MAP	Any	-lib

Configuration File Setting	Value	Overriding Command-line Option
COMMAND_OPTION_FILENAME		Additive effect hence not possible
COMMAND_FILE_ARGS		Additive effect hence not possible
SYSTEMVERILOG_SUPPORT	no	-enableSV
	yes	-disableSV
AUTOENABLE_INFERBLACKBOX	no	-inferblackbox, +libext
	yes	+libext, -disable_inferblackbox
	yes_netlist	-disable_inferblackbox
	yes_rtl	-inferblackbox, -disable_inferblackbox
AUTOENABLE_VHDL_SORT	no	-sort
	yes	-disable_sort
DEFAULT_VHDL_SORT_METHOD	lexical	-sort
	no argument	-sort=lexical

NOTE: *The command-line options that do not have an associated description have been provided only to override the Configuration File settings.*

Command-Line Utilities

The spyexplain Utility

SpyGlass provides the spyexplain utility that displays information about specified rules or rule parameters. Sometimes, you will need to find which product contains which rules, for instance. Moreover, you may not know exactly what rules are available for checking clocks in a particular product. Then, you can use the spyexplain utility to display a report that shows where rules are defined and gives a brief description of their function.

NOTE: spyexplain *is a separate utility with its own options and it is NOT an option to the* spyglass *application.*

Searching Rules

The spyexplain utility has a number of options, allowing you to define the HDL language, the product (or products you wish to search) and a keyword for which you wish spyexplain to search in the rule description and name.

The syntax of using the spyexplain command to search rules is as follows:

```
spyexplain
  -verilog | -vhdl | -mixed | -def
  [ -policies | -policy = {<product-name>,} ]
  [ { <rule-name> } ]
    | [ -k <search-string> [ -searchlonghelp ] ]
  [ -I <path> ]
  [ -include_builtins ]
```

Where the -policy, -policies, and -I command-line options work same as the corresponding SpyGlass command-line options, *<rule-name>* is a rule name or rule alias name (case-insensitive), and *<search-string>* is a valid string.

If you do not specify a product in the command-line, the spyexplain utility searches the installed products.

Use the -include_builtins argument to search the SpyGlass Built-in rules (HDL Parsing rules, SpyGlass Design Constraints file Parsing rules, and Library (.lib) Files Parsing rules).

The spyexplain utility searches both the rule name and short help message fields for matches but does not search extended help message fields unless the -searchlonghelp command-line option is supplied. The search mechanism is not case-sensitive, but partial words are also located during the search.

The spyexplain utility reports the following information for each rule that matches the specified search criteria:

- The rule name
- The language to which that rule applies
- The product in which the rule is described
- A short description of the rule
- An extended description of the rule

Examples of Searching Rules

For example, to see the rules in the VHDL SpyGlass OpenMore solution, use the following command-line:

```
spyexplain -vhdl -policies=openmore
```

To see the rules in the Verilog SpyGlass lint solution, use the following command-line:

```
spyexplain -verilog -policies=lint
```

To find out about a particular rule

For example, to see information about the W703 rule in a Verilog SpyGlass lint solution, use the following command-line:

```
spyexplain -verilog -policies=lint W703
```

To locate a particular rule in a product

To locate a rule in a product or to search for all rules in a product that relate to a required check (for example, all clock-related rules), you need to search for a word that is contained in the rule name or the short help message.

To search for a specific word in a product, enter the `-k` (keyword) command-line option of the `spyexplain` utility.

For example, to search for the keyword `Reset` in the Verilog SpyGlass lint solution, use the following command-line:

```
spyexplain -k Reset -verilog -policies=lint
```

To search for the keyword `Clock` in the VHDL SpyGlass lint solution, use the following command-line:

```
spyexplain -k Clock -vhdl -policies=lint
```

Searching Rule Parameters

The `spyexplain` utility has a number of options, allowing you to search a rule parameter in a specified product (or products).

The syntax of using the `spyexplain` command to search rule parameters is as follows:

```
spyexplain
  -verilog | -vhdl | -mixed | -def
  [ -policies|-policy={<product-name>,} ]
  [ -I <path> ]
  [ -param <param-name-list> ]
```

Where the `-policy`, `-policies`, and `-I` command-line options work same as the corresponding SpyGlass command-line options and *`<param-name-list>`* is a space-separated list of valid strings.

If you do not specify a product using the `-policies/-policy` option, the `spyexplain` utility searches all products. If you specify a product name using the `-policies/-policy` option and specify the `-param` option without any string, the `spyexplain` utility reports all rule parameters in the specified product.

NOTE: *You must specify at least one of the* `-policies/-policy` *and* `-param` *options.*

The `spyexplain` utility searches both the rule parameter name and short help message fields for matches. The search mechanism is not case-sensitive, but partial words are also located during the search.

The `spyexplain` utility reports the following information for each rule parameter that matches the specified search criteria:

- The rule parameter name
- The language to which that rule parameter applies
- The product in which the rule parameter is described
- The description of the rule parameter

Searching Goals

You can also list the description of all rules in a specified goal using the `-template` option of the `spyexplain` utility.

The syntax of using the `spyexplain` command to list description of rules in a goal is as follows:

```
spyexplain
  -verilog | -vhdl | -mixed | -def
  [ -I <path> ]
  [ -template <methodology-name>/<goal-name> ]
```

Where the `-I` command-line option works same as the corresponding SpyGlass command-line option, *`<methodology-name>`* is the name of the methodology and *`<goal-name>`* is the name of the goal.

For example, if you want the description of all rules in the Atrenta standard Block-Design/Creation goal then specify as follows (for different languages):

```
spyexplain -template Block-Design/Creation -verilog
spyexplain -template Block-Design/Creation -vhdl
spyexplain -template Block-Design/Creation -mixed
```

The `spyexplain` utility searches for the specified goal in the user-specified -I paths.

Searching SpyGlass Design Constraints

You can search the description of standard SpyGlass Design Constraints using the spyexplain utility.

The syntax of using the spyexplain command to search standard SpyGlass Design Constraints is as follows:

```
spyexplain
  [ -policies|-policy={<product-name>,} ]
  [ -I <path> ]
  [ -sgdc <constraint-name-list> ]
```

Where the -policy, -policies, and -I command-line options work same as the corresponding SpyGlass command-line options and *<constraint-name-list>* is a space-separated list of valid strings.

If you do not specify a product using the -policies/-policy option, the spyexplain utility searches all products. If you specify a product name using the -policies/-policy option and specify the -sgdc option without any string, the spyexplain utility reports all SpyGlass design constraints in the specified product.

NOTE: *You must specify at least one of the* -policies/-policy *and* -sgdc *options.*

The spyexplain utility first searches for the exact match by the design constraint name. If not found, the utility searches in the registration and long help of all design constraints in the specified products for matches. The search mechanism is not case-sensitive, but partial words are also located during the search.

The spydocviewer utility

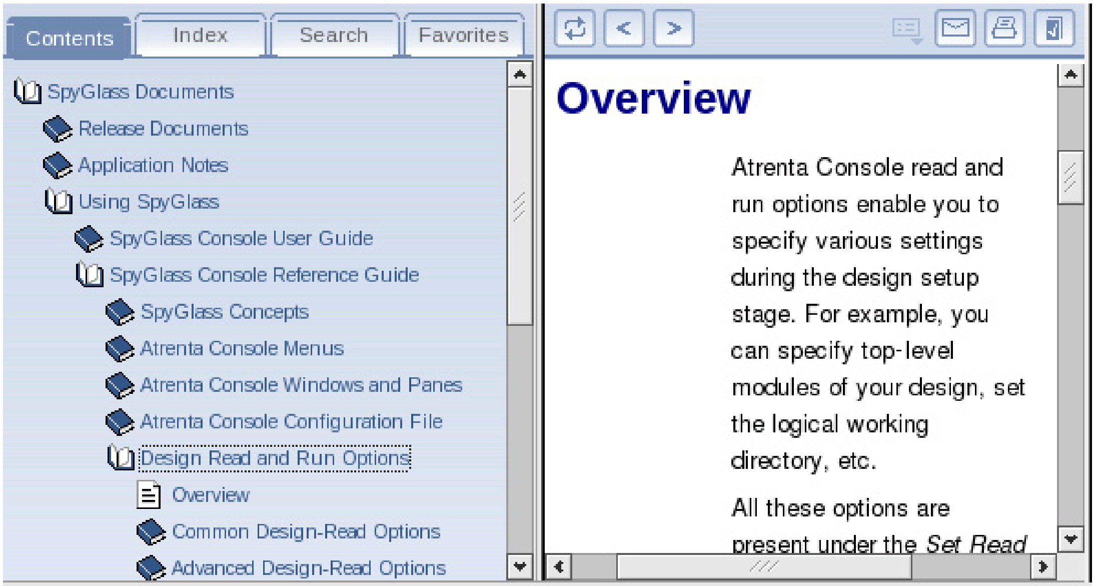
The spydocviewer utility displays the SpyGlass documentation in a tree format for easy access. In addition, the Atrenta Standard Rule-Primitive documentation (in text format) is also accessible.

By default, the spydocviewer utility searches for the acroread or xpdf executables in your machine’s path for displaying the PDF files. Set the SG_PDF_VIEWER environment variable to set your PDF viewer.

The spyhelpviewer utility

The spyhelpviewer utility displays the SpyGlass documentation in HTML format arranged in a tree format for easy access.

By default, the spyhelpviewer utility searches for the netscape executable in your machine's path for displaying the HTML files. Use the SG_HTML_BROWSER environment variable to set your HTML Browser.


Contents
Index
Search
Favorites
SpyGlass Documents
Release Documents
Application Notes
Using SpyGlass
SpyGlass Console User Guide
SpyGlass Console Reference Guide
SpyGlass Concepts
Atrenta Console Menus
Atrenta Console Windows and Panes
Atrenta Console Configuration File
Design Read and Run Options
Overview
Common Design-Read Options
Advanced Design-Read Options

Overview

Atrenta Console read and run options enable you to specify various settings during the design setup stage. For example, you can specify top-level modules of your design, set the logical working directory, etc.

All these options are present under the *Set Read*

FIGURE 239. SpyGlass HTML-based On-line Help System

The HTML-based online Help system has the following:

- The Contents tab that shows all topics arranged in a hierarchical tree
- The Index tab that has the index entries for the complete documentation set
- The Search tab for search across the complete documentation set
- The Favorites tab for collecting a set of related topics for viewing

- The Topic display page

System Requirements

The HTML-based On-line Help system works on a UNIX computer running version 4 or later of Internet Explorer or Netscape or a current version of Mozilla, or Safari. The underlying engine has also been tested with Mozilla. JavaScript must be enabled in the user's browser.

To view the Java implementation of the Help system, Java must be enabled in the user's browser.

Known Limitations

1. Netscape 6.0 is not supported on any platform; Netscape 6.1 and later are supported.
2. The underlying engine may also work with Opera and other browsers, but it has been tested only with Internet Explorer, Netscape, Mozilla, and Safari.

Project File Commands

The project file commands are described below.

87

Usage:	`set_option 87 <yes	no>`

For details on this command, see *Run in VHDL87 Compatibility Mode*.

abstract_file_name_style

Use this command to control the naming style of the abstract files generated by SpyGlass.

The following table describes the valid input values that you can specify, their description, and related example:

Permissible Value	Description	Example
short	Generates short file name in the *<module_name>_<product>_abstract.sgdc* format. *For example, deep_cdc_abstract.sgdc*	deep_cdc_abstract.sgdc
compat	Generates unique file names with parameter details in the *<module_name>_<parameter_details>_<product>_abstract.sgdc format.*	deep_BUS_1_WIDTH_1_cdc_abstract.sgdc

By default, the value of the *abstract_file_name_style* command is set to `compat`.

The following is the usage of the *abstract_file_name_style* command:

Usage:	`set_option abstract_file_name_style <compat	short>`

abstract_searchpath

Usage:	`set_option abstract_searchpath { <dir_path1> <dir_path2> }`

Use this command during the SoC flow to locate block SGDC files during abstract view use run.

If the block SGDC files are moved or relocated to some other path, use this option to specify the new block SGDC path.

The following example shows the usage of this command:

```
set_option abstract_searchpath {./block1_sgdc/}
```

active_design

Usage:	`set_goal_option active_design <reference	implement>`

Use this command to specify the type of design, `reference` or `implement`.

In the scope of the current goal, when you set this command to `reference` and specify a source list file by using the `reference_design_sources` command, SpyGlass considers that source list file for the current goal run and ignores the files specified by the `read_file` command. However, if you set the `active_design` command to reference outside the scope of the current goal or do not specify this command at all, SpyGlass ignores the source list file specified by the `reference_design_sources` command and considers the files specified by the `read_file` command.

For example, consider the following commands in a project file:

```
read_file -type verilog test.v
read_file -type sgdc test.sgdc

current_goal G1
set_goal_option active_design reference
set_goal_option reference_design_sources a.f
```

```
current_goal G2
set_goal_option reference_design_sources b.f
```

In the above example, SpyGlass picks the a.f file for the G1 goal. However, it picks the test.v and test.sgdc file for the G2 goal.

However, in the scope of the current goal, when you set this command to `implement` and specify a source list file by using the `implement_design_sources` command, SpyGlass considers that source list file in the current goal run and ignores the files specified by the `read_file` command at the global level. However, if you set the `active_design` command to implement outside the scope of the current goal, SpyGlass ignores the source list file specified by the `implement_design_sources` command and considers the files specified by the `read_file` command.

addrules

Usage:	`set_goal_option addrules` *`<rule-names>`*

Use this command to:

- Enable a rule
- Add a rule in the current goal

The following command enables the *W448* rule that is switched off by default:

```
set_goal_option addrules W448
```

allow_celldefine_as_top

Usage:	`set_option allow_celldefine_as_top <yes	no>`

Use this command to perform rule-checking on the top hierarchy of a top module.

By default, a top that is inside a 'celldefine module is ignored for rule-checking.

allow_constraints_override

Usage: `set_option allow_constraints_override <yes | no | 1 | 0 >`

Use this constraint to override or replace the current constraint command with the overriding constraint command, which is specified on the same object.

For overriding the current constraint commands, specify the -override argument in the overriding constraint commands for the same object.

You can also override the value specified for one object using one constraint commands with the value specified for the same object through different constraint commands.

If multiple constraints are provided for same object, but -override option is not specified, then all constraints are applied in the usual manner.

NOTE: *SpyGlass reports the SGDCWRN_126 warning if any SGDC command is overridden by another constraint command specified using the -override option.*

The following table lists the constraints and their respective arguments for which the support for the -override option is available:

TABLE 18 Support for -override option

Constraint	Argument
abstract_port	-ports
clock	-name
set_case_analysis	-name
quasi_static	-name

Example 1

Consider the following example:

```
abstract_port -ports A* -clock clkA -combo yes -start
set_case_analysis -name AHI -value 1 -override
```

In the above example, the value for port, AHI is set to 1 and no domain is assigned for analysis. Also, no violation message is reported for multiple constraint assignments on AHI.

Example 2

Consider the following example:

```
clock -name clkA -domain A1
clock -name clkA -domain A2 -override
```

In the above example, clkA is assigned domain A2. However, no violation messages are reported for multiple clock constraint assignments on clkA.

Example 3

Consider the following example:

```
quasi_static -name A
set_case_analysis -name A -value 0 -override
```

In the above example, the value of port A is set to 0. Also, no violation message is reported for multiple constraint assignments on A.

Example 4

Consider the following example:

```
set_case_analysis -name A -value 0
quasi_static -name A -override
```

In the above example, port A is quasi_static. No violation message is reported for multiple constraint declarations on port A.

allow_fatal_downgrade

Usage:	`set_option allow_fatal_downgrade <yes	no	1	0 >`

Cancels overloading of the SGDC rules, which are not built-in and have a FATAL severity. This option is available for backward compatibility only.

allow_incr_save_restore

Usage: `set_option allow_incr_save_restore <yes | no>`

Use this command to enable incremental save restore feature. This command is valid only when specified with the *enable_save_restore* command.

NOTE: *The allow_incr_save_restore command can not be used with the dw command in the designware flow.*

For details on this command, see *Incremental Save and Restore*.

allow_module_override

Usage: `set_option allow_module_override <yes |no>`

For details on this command, see *Allow Duplicate Module Names in Verilog Designs*.

allow_non_lrm

Usage: `set_option allow_non_lrm <1 | 0>`

Enables parsing of constructs, described in *Table 19*, in Verilog parser. These constructs are not standard lrm.

By default, the value of this command is 0. In this case, SpyGlass flags non-standard lrm constructs as syntax errors.

TABLE 19 Constructs Supported by the allow_non_lrm Command

Construct	Example

Support for use of port/net before its complete declaration	module top (a, b, sel, out1, out2, out3
	);
	input a, b, sel;
	output wire out3, out2;
	assign out3 = out2 ? a : out1 ;
	output wire out1;
	endmodule

Support for re-declaration of genvar STX_VE_600 for genvar re-declaration is replaced by WRN_1048	genvar i; genvar i; // Warning

Support for non-standard label style of assignment pattern
WRN_1047 is flagged for each non-standard usage
	int a[4] = {default:0}; // Warning
	[non-standard]
	int a[4] = '{default:0}; // Fine
	[standard]

Support for assert final
WRN_1046 is flagged for each assert final
	a1: assert final (out1 == ~in1)
	$fatal (2,"System Task FATAL executed");
	  is treated as
	     a1: assert #0 (out1 == ~in1)

Support for macro argument replacement within double quotes	`define A(a) "a" module top; int a = `A(X); endmodule With switch `A(X) is replaced by "X" whereas by default it is replaced by "a"

Support to relax STX_VE_800 by truncating evaluated value of expression used to represent ranges if they exceed 32bit limit	parameter NUM = 1'b0;
	input wire [0: 85'b0 - 1] q_sec;

TABLE 19 Constructs Supported by the allow_non_lrm Command

Construct	Example
Removed STX_810 fatal by considering functions having a parameter affected by a defparam statement as constant	NA
Relaxing STX_299 into WRN_1059 due to same type from same name package compiled into different libraries	NA

Relaxing the sign and state mismatched layout for array assignment compatibility checks
It will result in replacement of some violations of STX_VE_279, STX_VE_467, STX_VE_292 and STX_VE_299 by WRN_1462
	`module top;`
	`    logic signed conn1[3:0];`
	`    logic conn2[3:0];`
	`    bottom inst1(.a(conn1));        //`
	`Error :`
	`   incompatible port connection due to`
	`    mismatch in sign`
	`    bottom inst2(.a(conn2));        //`
	`Fine  :`
	`    compatible port connection`
	`endmodule`
	
	`module bottom(input wire a[3:0]);`
	`endmodule`

Support to allow assignment of concatenation to unpacked type
It will result in replacement of some violations of STX_VE_462, STX_VE_467, STX_VE_292 and STX_VE_299 by INFO_994
	`int a[4] = {0,1,2,3};  // Info`
	`int a[4] = '{0,1,2,3}; // Fine`

allow_pre_packaged_goals

Usage: `set_option allow_pre_packaged_goals <yes|no>`

Use this command to run all the optional and mandatory rules of the *SpyGlass lint* product, without explicitly editing the goal spq files.

allow_recursion_limit

Usage:	`set_option allow_recursion_limit <recursion_limit>`

Use this command to specify the recursion limit for a VHDL function. You can specify any positive integer value to specify the recursion limit. If the VHDL function call recursion exceeds the value specified using this parameter, the STX_VH_437 rule reports a violation and exits recursion. The default value for this option is 10420.

aggregate_report

Usage:	`set_option aggregate_report {<report-names>}`

Use this command to specify the type of aggregate reports, that is, *project_summary*, *datasheet*, or *dashboard*, that you want to generate.

aggregate_report_config_file

Usage:	`set_option aggregate_report_config_file <config-file-path>`

Use this command to specify a configuration file that contains a list of projects and run directories generated by batch console or GUI. This data is used to generate the specified aggregate report.

aggregate_reportdir

Usage:	`set_option aggregate_reportdir <report-directory-path>`

Use this command to specify a directory in which you want to generate data for the specified aggregate report.

auto_restore

Usage:	`set_option auto_restore <yes |no>`

Enables restore of design query data as part of the `current_goal/ open_project` Tcl commands. You can automatically restore the design information saved earlier by enabling this option in the project file. It is useful for executing Tcl procedures accessing this design information.

If auto_restore option is enabled, the design query data for the active goal is restored as part of the current_goal and open_project commands. In this case, only design data is restored and the product attributes are restored when user queries for a specific attribute.

auto_save

Usage:	`set_option auto_save <yes |no>`

Enables save of design query data as part of the goal execution. The design attributes are computed during the goal execution. These attributes along with the synthesized/flattened design view are saved when the auto_save option is enabled in the project file. You can retrieve this information at a later point without a need to re-run the goal.

block_abstract

Usage:	`set_option block_abstract <yes	no	true	false>`

Use this command to generate an abstract view during block-level verification in the SpyGlass CDC flow.

The following example shows the usage of this command:

```
set_goal_option block_abstract  yes
```

block_abstract_directory

Usage:	`set_option block_abstract_directory <directory>`
	`set_goal_option block_abstract_directory <directory>`

Use this command to specify a directory in which the abstract view of a block should be saved.

For details of this command, see *Block Abstract Directory*.

cachedir

Usage:	`set_option cachedir <directory-name>`

Use this command to specify a cache directory where you want library compilation to be performed.

By default, the value of this option is set to `spyglass_cache`.

cell_library

Usage:	`set_option cell_library <library-name>`
	`set_goal_option cell_library <library-name>`

Use this command to skip rule-checking on design units (Verilog/VHDL) that are loaded from precompiled libraries specified by this command. However, such units are synthesized by SpyGlass.

For example, consider that you have four precompiled libraries, `L1`, `L2`, `L3`, and `L4`, and suppose that you want to skip rule-checking on design units loaded from the libraries, `L2` and `L4`. In this case, specify the following command:

```
set_option cell_library { "L2"  "L4" }
```

check_celldefine

Usage:
```
set_option check_celldefine <yes | no>
set_goal_option check_celldefine <yes | no>
```

Use this command to enable rule-checking on all the `'celldefine` modules.

By default, Verilog `'celldefine` modules are not checked by SpyGlass as they are leaf-level library cells, and any independent checks (such as style checks like indentation) on these cells are not desired by most of the users. However, if you still want to check for such modules, use the `check_celldefine` command.

If you do not use this command, SpyGlass reports a warning specifying that rule-checking for `'celldefine` modules is off.

NOTE: *Also see the* `allow_celldefine_as_top` *command.*

checkip

Usage:
```
set_option checkip <ip-name>
```

For details on this command, see *Check IP*.

checkdu

Usage:
```
set_option checkip <du-name>
```

For details on this command, see *Check DU*.

checkTopDu

Usage:
```
set_parameter checkTopDu <rule-names>
```

Use this command to specify a space-separated list of rules that should run for the top-level design.

For example, consider the following command:

```
set_parameter checkTopDu "UndrivenOutPort-ML UndrivenNet-ML"
```

In the above example, the UndrivenOutPort-ML and UndrivenNet-ML rules will check top-level design unit only.

You can also set the value of the `checkTopDu` command to `yes` so that all rules check at the top level.

NOTE: *Behavior of post-flattening rules is not affected by the* `checkTopDu` *command. For such rules, create waive commands to waive hierarchical violations.*

configfile

Usage: `set_option configfile <file-name>`

Use this command to specify a configuration file with the highest priority.

For details, see *The Configuration File in SpyGlass*.

consolidate_reportdir

Usage: `set_option consolidate_reportdir <directory-path>`

Use this command to specify a directory path where SpyGlass should keep the physical files of the reports generated after SpyGlass run.

By default, SpyGlass stores the reports at the following path:

<project-working-directory>/<project-name>/consolidated_report/

For details, see *Directory Structure of Generated Reports*.

convert_udp_to_latch

Usage: `set_option convert_udp_to_latch <yes | no>`

Use this command to infer UDP as a latch while translating the UDP with both edge and level sensitiveness.

By default, SpyGlass infers a UDP as a flip-flop in such cases.

debug_comments

Usage: `set_goal_option debug_comments {"<text>"}`

Use this command to specify the debug comments to identify a goal/ scenario in the correlation view. The text/debug comment specified using this option is visible in the top row of each column of trials matrix.

decompile_block_constraints

Usage: `set_option decompile_block_constraints <yes |no |1 |0>`

Use this command to decompile the block file specified using the abstract_file command.

This facilitates porting of individual abstract SGDC files from creation directory to separate directory under the source control.

default_waiver_file

Usage: `set_option default_waiver_file <file-name>`

Use this command to specify a default waiver file for saving interactive waiver commands.

If you do not specify a default waiver file by using this command, SpyGlass Explorer considers the *<project-wdir>/<project_name>.awl* file as the default waiver file.

define

Usage: `set_option define <list-of-options>`

For details on this command, see *Enter Macros for Analysis*.

define_cell_sim_depth

Usage:
```
set_option define_cell_sim_depth
<cell_simulation_depth_depth>
```

Enables you to increase the threshold limit for the size of the macro in order to completely simulate the macro.

Synthesis Macro Logic Evaluation is done partially, if the macro size crosses the threshold limit. The default threshold limit is data bus size of 1024, and macros with data bus size > 1024 are partially simulated.

In order to completely simulate such big macros to improve the accuracy of the logic simulation, you can increase the threshold limit using this command.

This command takes the threshold limit in terms of size of data bus given as a power of 2. For example, if the value of this parameter is specified as 12, then data bus of size of 4096 (2^12) is completely evaluated.

The default value for this command is 10. You can set the value in the range of 1 to 30, both limits inclusive.

A lower value indicates a quick evaluation but improper result. However, a higher value indicates a longer evaluation period but more accurate result.

You can increase the threshold limit during evaluation for the following synthesis macros using the define_cell_sim_depth command:

- M_RTL_ARITH_SHIFT
- M_RTL_MUX_N
- M_RTL_PRIM_MUX
- M_RTL_LSHIFT
- M_RTL_RSHIFT
- M_RTL_SHIFT
- M_RTL_ROTATE

If the size of any of the above macros exceeds the threshold limit and these macros are partially simulated, then SpyGlass reports FLAT_504 message. This message indicates the value, which is required for the define_cell_sim_depth command, in the current run to completely simulate these macros.

define_incr_dirmap

Usage:

```
set_option define_incr_dirmap <dir1> <dir2>
set_goal_option define_incr_dirmap <dir1> <dir2>
```

Use this command to map different locations of RTL files. It is used by an incremental algorithm to find if RTL files in the current run are the same as the RTL files in old SpyGlass run with respect to the incremental analysis that is being performed.

define_regression

Usage:

```
define_regression <regression-name> -goals {<comma-
separated goal-list>}
```

For more information on the Tcl-based usage of the *define_regression* command, refer to the *define_regression* section of the *SpyGlass Tcl Shell Interface User Guide*.

Use this command to define a regression by:

- Specifying a list of goals to be run in the sequential mode, and
- Assigning a name *<regression-name>* to the goals list.

Once you define a regression, specify the regression name to the *-goals <goals>* command-line option to run the goals of that regression in the sequential mode.

This command is useful when the list of goals to be run in the sequential mode is huge. Specifying such huge list of goals every time you run the *-goals <goals>* command-line option can be time-consuming. Therefore, specify this list in one go by using the `define_regression` command

and later specify the list name (*<regression-name>*) to the *-goals <goals>* command-line option.

NOTE: *If a regression name specified by the -goals <goals> batch command is not declared in the project file, SpyGlass searches the specified goal in the current methodology.*

Example of Using the define_regression Command

The following example shows the usage of this command:

```
// Project1.prj
define_regression Reg1 -goals {G1,G2,G3}
```

Now consider that you specify the above project file in batch by using the following command:

```
spyglass -projectfile Project1.prj -goals Reg1
```

When you run the above command, SpyGlass searches `Reg1` in the project file. As the project file contains the declaration of `Reg1` through the `define_regression` command, SpyGlass runs the `G1`, `G2`, and `G3` goals in the sequential mode.

NOTE: *If you want to exempt any goal from the list of goals belonging to a regression, create a new regression by using the define_regression command such that the new regression does not have that goal.*

Creating a Regression File

A regression file is a Tcl file in which you can define regressions by using the `define_regression` commands. You must keep this file parallel to the order file in the current methodology.

After creating the regression file, when you specify a regression name to the *-goals <goals>* command-line option, SpyGlass looks for that regression name in the regression file and runs the goals of that regression in the sequential mode.

Consider an example in which Methodology/New_RTL is the current methodology and you have created the following regression_run.tcl file in this methodology directory:

```
define_regression Reg1 -goals {G1 G2 G3}
define_regression Reg2 -goals {G4 G5 G6}
```

Now, when you specify `Reg1` to the *-goals <goals>* command-line option, SpyGlass searches `Reg1` in the regression_run.tcl file and runs the `G1`, `G2`, and `G3` goals in the sequential mode.

However, if a regression name defined in a regression file matches with the regression name defined in a project file, SpyGlass gives preference to the regression defined in the project file. For example, if a regression file defines the `A`, `B`, and `C` regressions and a project file defines the `C` and `D` regressions, SpyGlass considers all the `A`, `B`, `C`, and `D` regressions where the `C` regression is picked from the project file.

define_severity

Usage:	`set_goal_option define_severity <product-name>+<severity-label>+<severity-class>` Where: ● <product-name> is the product mnemonic for the product for which you are specifying the new severity label. ● <severity-label> is the new severity label being defined. ● <severity-class> is one of the pre-defined rule severity classes. Do not add spaces between values and the intervening + characters.

Use this command to define a severity label for the current SpyGlass run.

This command is usually used in conjunction with the `overloadrules` command.

The following example defines the new severity label `myFATAL` under the pre-defined severity class `FATAL` for SpyGlass lint solution (product mnemonic is `lint`):

```
set_goal_option define_severity lint+myFATAL+FATAL
```

designread_disable_flatten

Usage: `set_option designread_disable_flatten <yes | no>`

Disables flattening during the *compile_design* command in the sg_shell. Also disables flattening while opening a project, if the project was closed with a flattened view.

designread_enable_synthesis

Usage: `set_option designread_enable_synthesis <yes | no>`

For details on this command, see *design-read Synthesis Flavor*.

designread_synthesis_mode

Usage: `set_option designread_synthesis_mode <mode>`

For details on this command, see *design-read Synthesis Flavor*.

disable_auto_migrate_waiver

Usage:

```
set_option disable_auto_migrate_waiver
<true|false|on|off|yes|no|1|0>
```

Use this command to disables the automatic migration of waivers.

disable_encrypted_hdl_checks

Usage:

```
set_option disable_encrypted_hdl_checks <yes|no>
set_goal_option disable_encrypted_hdl_checks <yes|no>
```

For details on this command, see *Disable Encrypted HDL Checks*.

disable_hdllibdu_lexical_checks

Usage:

```
set_option disable_hdllibdu_lexical_checks <yes|no>
set_goal_option disable_hdllibdu_lexical_checks
<yes|no>
```

Use this command to disable lexical rule checking on precompiled libraries.

disable_hdlin_synthesis_off_skip_text

Usage:

```
set_option disable_hdlin_synthesis_off_skip_text <yes|no>
```

Use this command to disable interpretation of VHDL design code between Synopsys synthesis_off/synthesis_on pragma pair as comments.

By default, SpyGlass interprets the VHDL design code between Synopsys synthesis_off/synthesis_on pragma pair as comments. This command disables this behavior, and the code between Synopsys synthesis_off/ synthesis_on pragma pair is checked by SpyGlass.

If this command is specified along with the hdlin_synthesis_off_skip_text command, then it is ignored, and the VHDL design code between Synopsys synthesis_off/synthesis_on pragma pair is treated as comments.

This option can also be specified as part of sources.f supplied with the libhdlf command, while precompiling a given logical library. It enables parsing of VHDL code between Synopsys synthesis_off/synthesis_on pragma pair for design files precompiled as part of the given logical library.

disable_hdlin_translate_off_skip_text

Usage:

```
set_option disable_hdlin_translate_off_skip_text <yes|no>
```

Use this command to disable interpretation of VHDL design code between Synopsys translate_off/translate_on pragma pair as comments.

By default, SpyGlass interprets the VHDL design code between Synopsys translate_off/translate_on pragma pair as comments. This command disables this behavior, and the code between Synopsys translate_off/ translate_on pragma pair is checked by SpyGlass.

If this command is specified along with the hdlin_translate_off_skip_text command, then it is ignored, and the VHDL design code between Synopsys translate_off/translate_on pragma pair is treated as comments.

This option can also be specified as part of sources.f supplied with the libhdlf command, while precompiling a given logical library. It enables parsing of VHDL code between Synopsys translate_off/translate_on pragma pair for design files precompiled as part of the given logical library.

disable_html_report

Usage:

```
set_option disable_html_report
 { < datasheet | dashboard | html > }
```

Use this command to disable the auto-generation of the specified reports. You can specify 'datasheet', 'dashboard' and/or 'html' to stop auto-generation of either of the reports.

The following command disables the datasheet and dashboard reports from being generated automatically:

```
set_option disable_html_report {datasheet dashboard}
```

disable_multiple_elaboration_check

Usage:

```
set_option disable_multiple_elaboration_check < yes | no >
```

Use this command to waive fatal messages reported by the `checkSGDC_existence` rule for non-existing constraint nodes that are resolved from wild cards and migrated due to elaboration of parameterized modules.

By default, the values is set to `no`. Set the parameter value to `yes` to waive such fatal errors.

disable_report

Usage: `set_goal_option disable_report { <report-names> }`

Use this command to disable the generation of the default reports other than the following reports:

moresimple report	*no_msg_reporting_rules report*	*stop_summary report*
elab_summary report	*ignore_summary report*	

The following command disables the waiver and summary reports:

```
set_goal_option disable_report {R1 R2}
```

NOTE: *Do not disable a report that is specified by using the report command.*

disable_infer_async_rst_latch

Usage: `set_option disable_infer_async_rst_latch { <module-names> }`

By default, Spyglass synthesis infers latches with asynchronous set/reset. When this command is set for specified modules, Spyglass synthesis infers latches without asynchronous set/reset, for such modules.

disable_sgdc_dump

Usage: `set_option disable_sgdc_dump <sg_multicycle_path | false_path>`

Use this command to disable the conversion of certain SDC commands to their respective SGDC commands in the sdc2sgdc flow.

The following table describes the valid values available for this command and the respective behavior:

TABLE 20 disable_sgdc_dump values

Value	Behavior	Command
sg_multicycle_path	Disables the conversion of the set_multicycle_path SDC constraint to the sg_multicycle_path SGDC constraint	`set_option disable_sgdc_dump sg_multicycle_path`
false_path	Disables the conversion of the set_false_path SDC constraint to false_path SGDC constraint	`set_option disable_sgdc_dump false_path`
false_path sg_multicycle_path	Disables the conversion of set_false_path and set_multicycle_path SDC constraints to false_path and sg_muticycle_paths SGDC constraints	`set_option disable_sgdc_dump {false_path sg_multicycle_path}`

disable_Vlog2005_lrm_naming

Usage: `set_option disable_Vlog2005_lrm_naming <0 | 1>`

Use this command to disable the Verilog-2005 LRM naming style.

The following example shows the usage of this command:

```
set_option disable_Vlog2005_lrm_naming 1
```

disallow_view_delete

Usage: `set_option disallow_view_delete <yes | no>`

Use this command to disable large design processing mode.

When you disable large design processing mode:

- All rules are run as in versions before SpyGlass version 3.2.0.
- Both RTL and netlist views remain in the memory for all types of rules.

NOTE: *You cannot use the* `higher_capacity` *and* `disallow_view_delete` *commands together as they are complementary.*

dnc_param

Usage: `set_option dnc_param <val_list>`

Sets the specified VHDL generics and Verilog parameters as do not care during abstract model generation.

You can specify the key name in the `module-name.parameter-name` format for Verilog and in the `entity-name.generic-name` format for VHDL. The names are case-sensitive for Verilog parameters and case-insensitive for VHDL generics.

In a normal design hierarchy, it is possible to define parameterizable design units, where the value of the parameter (in Verilog) or generic (in

VHDL) is defined when the design unit is instantiated in the hierarchy. In order to define a value for a parameter or a generic as do not care during abstract model generation, use the `dnc_param` option.

To set the `dnc_param` option for a generic or parameter, specify the instance of the parameter you wish to define.

For example, to set the VHDL generic width in entity control as do not care, specify the following command as part of the SpyGlass invocation:

```
<sg_shell>> set_option dnc_param "control.width"
```

To set the Verilog parameter limit in module block1 as do not care, specify the following command as part of the SpyGlass invocation:

```
<sg_shell>> set_option dnc_param "block1.limit"
```

dont_save_deleted_waivers

Usage: `set_option dont_save_deleted_waivers <yes | no>`

Use this command to specify if deleted waivers should be printed in waiver files. If option is set to `yes`, deleted waivers are not preserved as comments during save_waiver.

dummy_top

Usage: `set_option dummy_top <yes | no>`

Use this option along with the *top_instance* option to change the synthesis top.

The *dummy_top* option accepts the name of the dummy top module that needs to be elaborated. The *dummy_top* option is honored only if the top is specified. If the top is not specified, SpyGlass ignores the option and reports the following Warnings in the spyglass.log file:

- If no `top` is specified:

```
Warning: Ignoring options 'dummy_top' and 'top_instance' as
option top is not specified
```

- If multiple tops are specified:

  ```
  Warning: Ignoring options 'dummy_top' and 'top_instance' as
  multiple top specified
  ```

- If either of the *dummy_top* or the *top_instance* options are not specified, one of the following messages, as appropriate is reported:

  ```
  Warning: Ignoring option 'dummy_top' as option
  'top_instance' is not specified
  ```

  ```
  Warning: Ignoring option 'top_instance' as option
  'dummy_top' is not specified
  ```

  ```
  Warning: Ignoring option 'dummy_top' as options 'top' and
  'top_instance' are not specified
  ```

  ```
  Warning: Ignoring option 'top_instance' as options 'top' and
  'dummy_top' are not specified
  ```

- If the specified dummy top is not present in the design:

  ```
  Warning : Dummy Top specified in option dummy_top not found
  in the design
  ```

- If the specified top or dummy_top is config:

  ```
  Warning : Dummy top not supported if top or dummy top is
  config
  ```

dump_all_modes

Usage: `set_option dump_all_modes <yes | no>`

Use this command to create both 32-bit and 64-bit versions of the precompiled HDL sources irrespective of the architecture.

dump_report_dir

Usage: `set_option dump_report_dir <custom-dir-name>`

Use this command to specify the directory where the spyglass_run_stat.log, spyglass_violations_report.xml, and the spyglass_rules_report.xml reports should be saved.

dump_inactive_rules

Usage: `set_option dump_inactive_rules <yes | no>`

Use this command to print inactive rules in the spyglass_rules_report.xml file.

dw

Usage: `set_option dw <yes | no>`

For details on this command, see *Enable Analysis of Instantiated DesignWare Components*.

dw_options

Usage:

```
set_option dw_options <hide_all_dw_violations |
dont_use_tpts  | report_violations | enable_waiver>
```

Use this command to specify following options for the DesignWare components:

- **hide_all_dw_violations**: This is the default option. In this case, none of the violations, even if waived, are reported in the waiver report.
- **dont_use_tpts**: Disables use of third-party tools in the dw run.
- **report_violations**: Disables all DW waivers, so they show as any other violations.
- **enable_waiver**: Waives violations flagged on dw components. Such waived violations are reported in the waiver report.

If none of report_violations or enable_waiver options are specified, then violations on dw components are waived. Additionally, the violations are not reported in the waiver report.

elab_precompile

Usage:
```
set_option elab_precompile <yes | no>
```

Use this command to enable elaboration in single-step precompilation.

enable_amg

Usage:
```
set_option enable_amg <yes | no>
```

Use this command to enable the AMG flow in base synthesis.

enable_abstract_blocks_schematic

Usage:
```
set_option enable_abstract_blocks_schematic
<abstract_module_names>
```

For details of this command, see *Enable Abstract Blocks Schematic*.

enable_const_prop_thru_seq

Usage:
```
set_option enable_const_prop_thru_seq <yes | no>
set_goal_option enable_const_prop_thru_seq <yes|no>
```

Use this command to enable constant propagation beyond sequential elements during logic simulation.

By default, constant propagation stops at sequential elements.

If you set the value of the enable_const_prop_thru_seq command to yes in the project file, the set_case_analysis/test_mode values propagate beyond sequential elements. Constant propagation from flop-D happens only if one of the following conditions is true:

- Flip-flop does not have preset/clear pin.
- Data is tied to 0, and flip-flop has only clear pin.
- Data is tied to 1, and flip-flop has only preset pin.

NOTE: *The* enable_case_anal_seq_prop *option has been renamed to* enable_const_prop_thru_seq*.*

enable_fpga

Usage: `set_option enable_fpga <yes | no>`

This command converts the non-synthesizable FPGA constructs into synthesizable Verilog RTL gate constructs. The following table shows the mapping of the non-synthesizable constructs and their corresponding synthesizable constructs in Verilog:

Non-synthesizable construct	Corresponding synthesizable construct
Nmos	bufif1
Pmos	bufif0
Rtran	buf
Tri1	wire
Tri0	wire
Time	reg [63:0]

enable_non_sgdc_naming

Usage: `set_option enable_non_sgdc_naming <0 | 1 | no | yes>`

Use this command to specify non-SGDC style names, that is, SDC style names, in the SGDC file.

You can specify an instance, port, or, net name in an RTL in either SDC or SGDC formats.

Following table lists the key differences in the usage of the SDC and SGDC formats:

Difference	SDC Format	SGDC Format

Escape Character	For SDC compliant names, there is no need of giving an escape character for names containing special character as a part of the name.
	Example:
	`"mid/lower@leaf"`
	`(Hierarchy separator is '/')`
		An SGDC name needs an escape character ('\' backward slash in our case) plus a white space at the end to mark the end of the name.
		Example:
		`"top.mid.\lower@leaf "`
		`(Hierarchy separator is '.' And also note that '\' is added to identify '@' as a part of the name and '  ' white space at the end to mark the end of the name.)`

RTL Names	No special characters needed in the path **Example:** `gen_block[0]/U1/q_reg[0]`	Full hierarchial name specified using the special character, such as, '.' **Example:** `top.\gen_block[0] .U1.q_reg[0]`

Using the *enable_non_sgdc_naming* command to facilitate the search for the above specified SDC style naming conventions.

enable_gateslib_autocompile

Usage: `set_option enable_gateslib_autocompile <yes | no>`

The default value of *enable_gateslib_autocompile* option is yes.

For details on this command, see *Enable auto-compilation of gateslib into sglib*.

Also see *force_gateslib_autocompile*.

enable_hbo

Usage:

```
set_option enable_hbo <yes|no|1|0|on|off|true|false>
```

Use this command to enable hierarchical boundary optimization in synthesis.

NOTE: *This options is valid only for est mode of synthesis.*

By default, in optimized synthesis flow, optimizations are performed at module level only. Across module/hierarchy optimizations are not performed in default flow.

In hierarchical boundary optimization, information about constant and unloaded nets on instance interface is propagated across boundaries and corresponding logic optimization is performed.

enable_hdl_encryption

Usage:

```
set_option enable_hdl_encryption <yes | no>
```

For details on this command, see *Enable HDL Encryption*.

enable_inactive_rtl_checks

s

Usage:

```
set_option enable_inactive_rtl_check <yes | no>
set_goal_option enable_inactive_rtl_check <yes | no>
```

Use this command to enables semantic checking capability in SpyGlass.

enable_instance_based_reporting

Usage: `set_option enable_instance_based_reporting <yes|no>`

Set the value of this option to yes to group the instances together in the moresimple report.

enable_ip_based_report

Usage: `set_option enable_ip_based_report <yes|no>`

Set the value of this option to `yes` to generate IP-based reports.

For details, see *IP-Based Report*.

enablePackaging

Usage: `set_option enablePackaging  <yes|no>`

Use this command to enable packaging of block SGDC files along with abstract views.

enable_pass_exit_codes

Usage:

```
set_option enable_pass_exit_codes <yes |no>
set_goal_option enable_pass_exit_codes <yes |no>
```

Use this command to print detailed exit status codes and messages. For details, see *SpyGlass Exit Status*.

enable_pgnetlist

Usage: `set_option enable_pgnetlist <yes |no>`

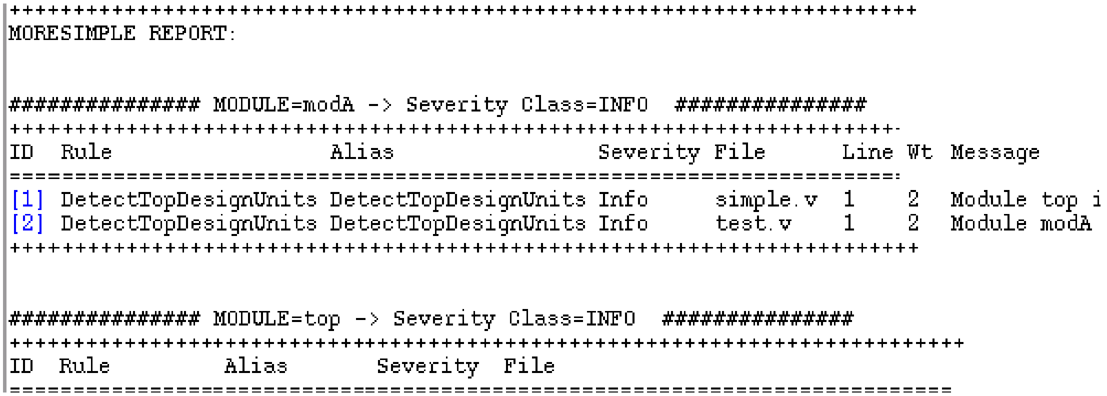
Use this command to enable power and ground pin information to consider from the specified physical libraries.

enable_module_based_reporting

Usage: `set_option enable_module_based_reporting <yes | no>`

Use this command to group violations based on module names in SpyGlass reports.

The following figure shows the moresimple report in which violations are grouped based on module names when you set this command to `yes`:



```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
MORESIMPLE REPORT:


############### MODULE=modA -> Severity Class=INFO  ###############
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
ID  Rule                 Alias                Severity File      Line Wt  Message
=====================================================================:
[1] DetectTopDesignUnits DetectTopDesignUnits Info     simple.v  1    2   Module top i
[2] DetectTopDesignUnits DetectTopDesignUnits Info     test.v    1    2   Module modA
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


############### MODULE=top -> Severity Class=INFO  ###############
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ID  Rule         Alias       Severity  File
==========================================================================
```

enable_power_platform_flow

Use this command to enable the Power Explorer feature of the SpyGlass Power Estimation product. This command enables various views that contains data related to register and memory instance of design. By Default this feature is off.

Usage: `set_option enable_power_platform_flow <yes | no>`

enable_precompile_vlog

Usage: `set_option enable_precompile_vlog <yes | no>`

Use this command to enable the precompiled Verilog library feature. For more information, see *Precompiling Verilog Libraries*.

By default, the precompiled Verilog library feature is not enabled.

enable_physical_aware_pe

Usage: `set_option enable_physical_aware_pe <yes | no>`

Use this command to enable physical aware power estimation flow and check out required licenses. It is used in `physical_power_postfloorplan` and power_est_average goals.

By default, this option is not set and SpyGlass runs normally.

enable_rule_category_in_moresimple

Usage:
```
set_option enable_rule_category_in_moresimple <yes | no |
1 | 0>
```

Use this command to include the `Rule Category` column in the moresimple.rpt report.

enable_rule_mnemonic

Usage: `set_option enable_rule_mnemonic <yes | no | 1 | 0>`

Use this command to enable the support for rule mnemonics in the SpyGlass run.

For more information on the support of rule mnemonics, see *Using Rule Mnemonics*.

enable_save_restore

Usage: `set_option enable_save_restore <yes | no>`

The default value of the *enable_save_restore* option is `yes`.

For details on this command, see *Enable Save Restore Flow*.

enable_save_restore_builtin

Usage: `set_option enable_save_restore_builtin <yes | no>`

The default value of the *enable_save_restore_builtin* option is yes.

For details on this command, see *Enable Save Restore for BuiltIn Rules*.

enable_sgdc_debug

Usage: `set_option enable_sgdc_debug <yes | no>`

Set this command to `yes` to generate the debug_sgdc report under the spyglass_reports/SpyGlass directory.

The debug_sgdc report contains decompiled SGDC commands with back reference information.

enable_sglib_debug

Usage:

```
set_option enable_sglib_debug <yes | no>
set_goal_option enable_sglib_debug <yes | no>
```

Use this command to generate the debug_sglib and sglib_version_summary reports.

The debug_sglib report contains the inferred functionality for each gate that was successfully synthesized by the SpyGlass Library Compiler.

The sglib_version_summary report lists the sglib names, their compiling library compiler version, and their status. The report also lists enhancements made in subsequent SpyGlass library compiler releases starting from the oldest version of library files used in the current SpyGlass run.

NOTE: *You must recompile your gates library files with SpyGlass 3.8.2 or higher to generate the* debug_sglib *report.*

enable_turbo_rules

Allows you to run turbo rules selectively.

NOTE: *Ensure that the Turbo option should be enabled in order to use this selective option.*

Usage: `set_option enable_turbo_rules {W123, W336}`

Example

Consider the following commands:

```
set_option turbo yes
set_option enable_turbo_rules {W123,W336}
```

In the above example, turbo is enabled for rule W123 and W336. Other turbo rules are executed in the default flow.

enable_unified_naming_search

Usage: `set_option enable_unified_naming_search <0 | 1 | no | yes>`

Use this option to specify SGDC as well as non-SGDC (SDC) style names in the SGDC file. You can specify an instance, port, net or hierarchical terminal in either SDC or SGDC format. All input formats are supported, such as, wildcard, bit-select, part-select, bus-complete, complex data-types, define-changes names, and multi-dimensional names.

The following table lists the examples for different module names and their respective SDC and SGDC compliant names:

TABLE 21 SDC and SGDC Compliant Naming

Example	SDC Format	SGDC Format
Consider the hierarchy, mid (master `MIDDLE`), instantiated in top module named `top`. lower (master `LOWER`) refers to instance instantiated in `MIDDLE`.	No need to specify the top module's name **SDC compliant name**: `mid/lower` Here, hierarchy separator is '/'.	Every hierarchical name needs the top module's name. **SGDC compliant name**: `top.mid.lower` Here, hierarchy separator is '.'.
Consider the hierarchy, `mid` (master `MIDDLE`) instantiated in top module named `top`. `lower@leaf` ('@' is part of the instance name) refers to an instance of module `LOWER` instantiated in `top`.	The name is written with hierarchy separator as '/' (forward slash) **SDC compliant name**: `mid/lower@leaf` Here, hierarchy separator is '/'.	The name is written with hierarchy separator as '/' (forward slash) **SDC compliant name**: `mid/lower@leaf` Hierarchy separator is '/'.
RTL Names: For a hierarchical name for an instance generated by a generate_block. For example, (gen_block[0] , gen_block[1] ...) are the instances generated corresponding to a generate_block.	No need of giving an escape character for names containing special character as a part name **SDC compliant name**: `gen_block[0]/U1/q_reg[0]`	Needs an escape character ('\' backward slash) plus a white space at the end to mark the end of the names containing special characters. **SGDC compliant name**: `top.\gen_block[0].U1 .q_reg[0]`

enable_vlog_config

Enables the support for verilog configuration. For more information on verilog configuration, see *Support for Verilog Configuration*.

Usage: `set_option enable_vlog_config <yes | no>`

enableSV

Usage: `set_option enableSV <yes | no>`

For details on this command, see *Enable SystemVerilog Processing*.

enable SVA

Usage: `set_option enableSVA <1 | 0>`

Enables parsing of SystemVerilog constructs and also handles the SV Assert logic. The default value for this option is 0. Therefore, SpyGlass considers system verilog constructs as syntax errors.

enableSV05

Usage: `set_option enableSV05 <0 | 1 | yes | no>`

Use this command to enable parsing of verilog2005 constructs.

enableSV09

Usage: `set_option enableSV09 <0 | 1 | yes | no>`

Use this command to enable parsing of SystemVerilog constructs.

By default, SpyGlass reports SystemVerilog constructs as syntax errors.

filter_block_violation

Usage: `set_option filter_block_violation {<space-separated-list>}`

Where <space-separated-list> can have the following:

- errors
- warnings
- waived_errors
- waived_warnings
- builtin_errors
- builtin_warnings
- builtin_errors_waived
- builtin_warnings_waived

Use this command during the SoC flow to filter violations reported during block SGDC generation.

The following example shows the usage of this command:

```
set_option filter_block_violation { errors warnings
builtin_errors builtin_warnings builtin_errors_waived
waived_errors }
```

force_gateslib_autocompile

Usage: `set_option force_gateslib_autocompile <yes|no>`

Set this command to `yes` to forcefully compile Synopsys Liberty(TM) files (.lib files) to SpyGlass-compatible format library files (.sglib files).

Using this command automatically implies the use of the *enable_gateslib_autocompile* command.

If you set the `force_gateslib_autocompile` command to `yes` to automatically compile gate libraries, any criteria for re-compilation of gate libraries is not evaluated. In such cases, the specified .lib files are always compiled and these files overwrite the existing .sglib file present in the cache directory.

gen_block_options

Usage: `set_option gen_block_options { <list-of-blocks> }`

Use this command to generate the block dependency report for certain design units.

gen_block_expand_lib_sources

Usage: `set_option gen_block_expand_lib_sources <true | false>`

Lists specification information about source files, as comments, for precompiled libs, incdir, and -y/v directories. It also filters out irrelevant .lib specifications for the specified design unit or a block. Following examples explain the difference in the information reported using the *gen_block_options* and *gen_block_expand_lib_sources* commands.

Example 1

Consider a top module, top, in a design instantiates a module, mid, which has been precompiled into library, P1. Module, mid, further instantiates two modules, bottom1 and bottom2, that have been compiled in the library P2.

When you specify the *gen_block_options* command, it de-compiles all libraries, including WORK, in the report, as shown below:

```
read_file -type verilog {top.v}
set_option lib L1 "../P1"
set_option lib L2 "../P2"
set_option lib WORK "./top/WORK"
```

However, when you specify *gen_block_expand_lib_sources* command, it reports only relevant .lib specifications as shown below:

```
read_file -type verilog {top.v}
set_option lib L1 "../P2"
# read_file -type hdl {dir1/bottom1.v}
# read_file -type hdl {dir1/bottom2.v}
set_option lib L2 "../P1"
# read_file -type hdl {dir1/mid.v}
```

Example 2

Consider a top module, top, instantiates module, mid, defined in the file, y_dir1/mid.v. The module, mid, further instantiates another module, bottom, defined in the file, y_dir2/bottom.v. Also assume that the file, bottom.v, uses a parameter ‘REGSIZE’ defined in inc1/include1.v.

When you specify the *gen_block_options* command, the top module generates the following report:

```
set_option define {REGSIZE=4}
read_file -type verilog {top.v}
set_option incdir {inc1/}
set_option y {y_dir1/}
set_option libext {.v}
set_option y {y_dir2}
set_option libext {.v}
```

However, when you specify the *gen_block_expand_lib_sources* command, the top module generates the following report:

```
set_option define {REGSIZE=4}
read_file -type verilog {top.v}
set_option incdir {inc1/}
# read_file -type hdl {inc1/include1.v}
set_option y {y_dir1/}
set_option libext {.v}
# read_file -type verilog {./y_dir1/mid.v}
set_option y {y_dir2}
set_option libext {.v}
# read_file -type verilog {./y_dir2/bottom.v}
```

gen_blk_sgdc

Usage:

```
set_option gen_blk_sgdc <yes | no>
set_goal_option gen_blk_sgdc <yes | no>
```

Set this command to `yes` to enable generation of an abstract view of a block.

NOTE: *This command is currently used by SpyGlass CDC solution only.*

generate_run_stat

Usage: `set_option generate_run_stat <yes | no>`

Use this command to create the `spyglass_run_stat.log` file.

generate_violations_report

Usage: `set_option generate_violations_report <yes | no>`

Use this command to create the `spyglass_violations_report.xml` file.

generate_rules_report

Usage: `set_option generate_rules_report <yes | no>`

Use this command to create the `spyglass_rules_report.xml` file, which contains all the active rules.

gensys_compatible_dump

Usage: `set_option gensys_compatible_dump <0 | 1>`

Use this command during the generation of a precompiled dump to generate extra information required by Atrenta GenSys platform.

generate_consolidated_waiver_file

Usage: `set_option generate_consolidated_waiver_file <yes | no>`

Use this command to generate a consolidated waiver file, consolidated_waiver_file.swl. This file contains all the user-specified waivers for a specific goal. The consolidated_waiver_file.swl file is generated in the <current_working_directory>\<project_name>\<goal_name> directory.

Example

Consider the following commands specified in the project file:

```
read_file -type waiver fsm.awl
read_file -type waiver top_waivers.awl
set_option generate_consolidated_waiver_file yes
```

As an output, the following sample `consolidated_waiver_file.swl` file is generated:

```
#############################################################
#### Original Waive File - /remote/abc/abc1/Test/2016.06/
1001010/case00/fsm.awl ####

#############################################################
waive -msg \
     q%Flip-flop 'st_art_timer' has neither asynchronous set
nor asynchronous reset. [Hierarchy: 'TOP.fsm']% \
      -rule q%STARC-2.3.4.3% -comment \
      q% Created by user on 10-May-2016 17:52:55 %
waive -msg \
      q%Flip-flop 's2f_wr_reset_test_clken' has neither
asynchronous set nor asynchronous reset. [Hierarchy:
'TOP.fsm']% \
      -rule q%STARC-2.3.4.3% -comment \
      q% Created by user on 10-May-2016 17:53:16%

# waive -import "soc_sdm_fpga_bridges" "fsm_import.awl"
waive -du "soc_sdm_fpga_bridges" -msg \
      q%Multiple statements (if/case/while/for/forever/
repeat) in a single always block% \
      -rule q%STARC-2.6.2.1% -comment \
      q% Created by user on 10-May-2016 17:54:26 %

#############################################################
```

```
#### Original Waive File - /remote/abc/abc1/Test/2016.06/
1001010/case00/top_waivers.awl ####

############################################################
# waive -import "Divider" \
#      "Divider/Divider_files/Divider_test_goal/Divider.awl"
waive -msg \
      m%Flip\-flop '.*pos_cnt1\[31:0\]' has neither
asynchronous set nor asynchronous reset\. \[Hierarchy:
'.*'\]% \
      -rule "STARC-2.3.4.3" -comment \
      q%Created by user on 11-Dec-2015 22:01:51%
# waive -import "Timer" "Timer/Timer_files/Timer_test_goal/
Timer.awl"
waive -msg \
      m%Flip\-flop '.*running' has neither asynchronous set
nor asynchronous reset\. \[Hierarchy: '.*'\]% \
      -rule "STARC-2.3.4.3" -comment \
      q%Created by user on 11-Dec-2015 22:19:34%
```

Note that in the generated consolidated_waiver_file.swl file, the -import commands are automatically commented and the imported commands are dumped below the original commented waive -import command.

GLOBAL_SPY_DW_WORK

Usage:	`set_option GLOBAL_SPY_DW_WORK <yes	no>`

Use this command to reuse Netlist of DesignWare modules during SpyGlass analysis. The command enables a global read-only directory to read the DesignWare VHDL components and packages and Verilog netlist of DesignWare components that are compiled by SpyGlass in a previous run and copied to the GLOBAL_SPY_DW_WORK directory with the same directory structure as that of the generated SPY_DW_WORK directory.

This enables you to save disk space while reusing Netlists and pre-compiled dump of DW component declarations.

Use the following command to enable the global directory:

```
set_option lib GLOBAL_SPY_DW_WORK <dir-name>
```

SpyGlass checks for existing netlists in GLOBAL_SPY_DW_WORK first and then in SPY_DW_WORK. SpyGlass generates the remaining netlists required by the design in SPY_DW_WORK.

Note the following for the GLOBAL_SPY_DW_WORK option:

- If the DW VHDL components and packages are not found in GLOBAL_SPY_DW_WORK (or if empty), the components are created in SPY_DW_WORK (in mixed and VHDL mode).
- If Verilog netlist is not available in GLOBAL_SPY_DW_WORK, it will be generated in SPY_DW_WORK and read from it in that SpyGlass run.
- If GLOBAL_SPY_DW_WORK needs to be recompiled, the libraries are compiled in SPY_DW_WORK. The libraries can then be copied to the global directory for subsequent SpyGlass run.
- If GLOBAL_SPY_DW_WORK contains netlists for only some DW modules, SpyGlass reads these and generates the remaining in SPY_DW_WORK and read them in that current run.

handle_large_param

Usage: `set_option handle_large_param <yes | no>`

Use this option to avoid truncation of the parameter value if the value is greater than 32 bits and is not specified with the bit-width.

For example, consider the following command:

```
set_option param {toper.SYS_RAM_FIFO_DEPTH1=115605798470427
```

In this case, the abstracted model contains the truncated value of the parameter.

To avoid this, set the value of the *handle_large_param* command to yes. When set to yes, the abstracted model contains the entire value of the parameter. In addition, the Elab_summary report shows the entire value (including the bit width) of the parameter.

handlememory

Usage:
```
set_option handlememory <yes | no>
```

Use this command to process memories in an optimized manner.

See the *Memory Reduction Feature* section for more details.

NOTE: *The handlememory option is ignored (for individual memories only), if the memory size is less than the specified mthresh value.*

You can also use the AUTOENABLE_MEMORY_HANDLING configuration setting to enable this feature.

hdlin_synthesis_off_skip_text

Usage:
```
set_option hdlin_synthesis_off_skip_text <yes | no>
```

For details on this command, see *Ignore VHDL code within pragma block 'synthesis'*.

hdlin_translate_off_skip_text

Usage:
```
set_option hdlin_translate_off_skip_text <yes | no>
```

For details on this command, see *Ignore VHDL code within pragma block 'translate'*.

hdllibdu

Usage:
```
set_option hdllibdu <yes | no>
set_goal_option hdllibdu <yes | no>
```

Use this command to perform RTL and lexical rule-checking on precompiled Verilog/VHDL design units that are directly or indirectly instantiated in the design being processed.

By default, this command is disabled, and SpyGlass does not perform RTL and lexical rule-checking on precompiled Verilog/VHDL design units.

When you set the hdllibdu option to yes, SpyGlass enables the lexical rule-checking on precompiled design units. This can result in an increased peak memory.

If the increase in peak memory is significant resulting in an out of-memory situation, use the *disable_hdllibdu_lexical_checks* project file command while running precompiled design units. This will disable lexical rule checking on precompiled design units.

higher_capacity

Usage: `set_option higher_capacity <yes | no>`

Use this command to:

- Disable rules that are designed for SpyGlass version 3.2.0 that require both RTL and netlist views, if any.
- Delete RTL view from memory before running rules designed for SpyGlass version 3.2.0 that require a netlist view only.

hw_variant_name

Usage: `set_option hw_variant_name <hw_variant>`

Use this command to view results of the same top module separately in DashBoard when executed under different hardware environments.

Each hardware variant may have different set of defines, file lists, parameters and may lead to a different synthesis for the same top module.

When user sets this command in the project files and the result directories are explicitly set, the SoC DashBoard shows all such hardware variant analysis results separately for tracking. Without this command, the SoC Dashboard accumulates all results of the same top module and displays them as one entity.

For example, assume that you want to execute a top module, my_top, under two different environments/projects, say, master2port and slave2port. To track these environments separately, include the following in the respective project files:

- In the `master2port` project:

```
<other settings>
set_option top my_top
set_option hw_variant_name master2port
set_option projectwdir ./my_top_master2port
<other settings …>
```

- In the `slave2port` project:

```
<other settings>
set_option top my_top
set_option hw_variant_name slave2port
set_option projectwdir ./my_top_slave2port
<other settings …>
```

NOTE: *To avoid overwriting results, if the project files are located in the same directory or the same project is being used for different hardware definitions, set the value of the projectwdir command.*

include_block_interface

Usage:	set_option include_block_interface <encrypted | abstract>

Use this command to capture block interface information in the abstract model.

You can set the value of the include_block_interface command to either encrypted, or abstract, or both encrypted and abstract.

The following table describes the valid options supported by the include_block_interface command:

Option	Description	Example
abstract	Enables you to capture block interface information in the form of SGDC (abstract_interface_port, abstract_interface_param) in the abstract model	set_option include_block_interface { abstract }
encrypted	Enables you to capture block interface information encrypted in the abstract model	set_option include_block_interface { encrypted }
abstract encrypted	Enables you to capture block interface information in the form of both SGDC and encrypted information in the abstract model	set_option include_block_interface { abstract encrypted }

Recommendations when working with encrypted abstract model

Consider the following points when working with encrypted abstract model:

- In the encrypted abstract model, do not tamper or edit the encrypted part. This may corrupt the abstract model and, therefore, make it unusable for top-level SoC run.
- Avoid copy-paste of the encrypted part from one model to another model. This may corrupt the abstract model and, therefore, make it unusable for the SoC verification.
- Use the encrypted route for the HDL blocks having very complex interface.

include_opt_data

Usage: `set_option include_opt_data <true|false|yes|no>`

Use this command with the *enable_gateslib_autocompile* option to enable SpyGlass Library Compiler to generate advanced optimized data used in the SpyGlass Power Estimation for the specified gates library (.lib) files only, and store them as part of the resulting sglib (.sglib) files.

Using the `include_opt_data` command enables the preparation and storage of advanced optimized data inside the .sglib file for faster processing. It also helps in detecting errors in the .lib files early in the process resulting in better performance. Though performance improvement is specifically seen in SpyGlass power estimation flow, use of the .sglib file with advanced optimized data is not limited to this SpyGlass flow only. You can use it in all SpyGlass runs without any problem.

If SpyGlass fails to generate the advanced optimized data for the specified library file, SpyGlass does not generate the .sglib file and reports a fatal violation indicating the failure reason.

If any .sglib file used in the SpyGlass power estimation flow does not contain optimized data, that is, it was not compiled by using the `include_opt_data` command, SpyGlass reports an informational message, using the *CMD_sglib04* rule, recommending the use of this command for better performance. For more information on the *CMD_sglib04* rule, refer to the *SpyGlass Built-in Rules Reference Guide*.

ignore_builtin_rule

Usage: `set_option ignore_builtin_rules <yes | no>`

For details on this command, see *Ignore SpyGlass BuiltIn Rules*.

ignore_case_analysis

Usage: `set_option ignore_case_analysis <yes | no>`

Use this command to ignore set_case_analysis constraint, if specified. The *checkCMD_ignore_case_analysis* rule reports an INFO message, when this command is set to yes and at least one *set_case_anlysis* constraint is specified in the design.

For more information on the *checkCMD_ignore_case_analysis* rule, refer to *SpyGlass Built-in Rules Reference Guide.*

ignoredir <dir-name>

Usage: `set_option ignoredir { <directory-name> }`

Use this command to ignore all files in a directory and its subdirectories from SpyGlass analysis.

When you ignore a file, the design units described in that file are assumed to be black boxes for the purpose of SpyGlass analysis. For details, please refer *ignorefile*.

ignoredu

Usage: `set_option ignoredu { <design-unit-names> }`

For details on this command, see *Ignore Design Unit(s)*.

ignore_builtin_spqdir

Usage: `set_option ignore_builtin_spqdir <path>`

For details on this command, see *Directory Path Containing Ignore BuiltIn Files*.

ignorefile

Usage: `set_option ignorefile <file-name>`

Use this command to ignore a file from SpyGlass analysis.

When you ignore a file, the design units described in that file are assumed to be black boxes for the purpose of SpyGlass analysis.

The following example ignores the design file `myaddlfile.v`:

```
set_option ignorefile myaddlfile.v
```

NOTE: *You can specify a relative or absolute path of a file in the above command.*

When this command is run, SpyGlass generates the *ignore_summary report*.

SpyGlass does not consider the ignorefile and ignoredu specifications, if they are specified in a source file that is passed to the set_option libhdlf <logical-library-name> <source-files> project file command.

However, SpyGlass considers the ignorefile and ignoredu specifications in the following cases:

- If they are directly specified in a project file
- If they are specified in a source file that is different from the file specified in libhdlf specification

To ignore design units from precompilation, perform the following steps:

- Perform precompilation of design files in a separate SpyGlass run along with the ignorefile and ignoredu commands.
- Use the precompiled dump for further analysis.

If the you ignore a file that includes design units that are referred in another file, SpyGlass may report an `STX_*` error.

For example, consider the following `ignorefile` specification in which the specified file, `file1.vhd`, includes an entity and the architecture of the same entity is defined in another file, `file2.vhd`:

```
set_option ignorefile file1.vhd
```

In this case, the architecture is not ignored and SpyGlass may report an STX_* error.

Using Wildcard while Ignoring Files

The following table shows some examples of using expressions with this command:

Example	Description
`set_option ignorefile {a*}`	Ignores all files (in the current directory) whose names match the wildcard a* expression. For example, a1, aa1, and abc.
`set_option ignorefile {dir1/*}`	Ignores all files in the dir1 directory
`set_option ignorefile {dir?/*}`	Ignores all files in directories that match the wildcard expression dir?. For example, dir1, dir2, and dir3.

If your filename includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your filename is "abc*d", you need to refer to it as {{abc*d}}.
However, if you want to refer to two files, for example "abc1d" and "abc2d", you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}} in this case. For details on pattern matching support, see *Pattern Matching Across Features.*

ignorelibs

Usage:

```
set_option ignorelibs <yes | no>
set_goal_option ignorelibs <yes | no>
```

Use this command to skip rule-checking on modules in library files specified by using the `v` or `y` commands.

When you set this command to `yes`, SpyGlass does not report violations (except ELAB/SYNTH/InfoAnalyzeBBox/WarnAnalyzeBBox/ErrorAnalyzeBBox/FatalAnalyzeBBox errors) on these modules.

However, functional model of these modules is synthesized/flattened and is available during any checks performed on other modules.

NOTE: *If any part of incremental schematic for a rule violation lies outside an IP boundary (module passed as a library file), SpyGlass does not waive off the violation of that rule even with the* `ignorelibs` *command.*

ignore_reference_project_sgdc

Usage:

```
set_goal_option ignore_reference_project_sgdc <yes | no>
```

Set this command to `yes` to ignore the SGDC files specified in the project file specified by the *reference_design_projectfile* command.

ignorerules

Usage:

```
set_option ignorerules { rule-names }
set_goal_option ignorerules { rule-names }
```

Use this command to specify name of rules or rule groups to be ignored in the current SpyGlass run. You can specify multiple rule names, rule group names, or a combination of both.

NOTE: *Rule and rule group names are case sensitive. You can specify multiple rules to ignore as space-separated list. The comma-separated list is not allowed.*

NOTE: *There are certain rules in SpyGlass that cannot be ignored. If you specify such rules with the* `ignorerules` *command, SpyGlass reports WARNING [38]. For example, most of the Built-In and prerequisite rules cannot be ignored.*

NOTE: *You can ignore semantic checks for SpyGlass classic-batch options and SGDC commands, even though they are mandatory checks. Some of these checks are CMD_define_severity03, checkCMD_wildcardMatch03, SGDC_clock01, and checkSGDC_existence. Do not ignore these checks unless you encounter any problems with them.*

Effect of the RULE_SELECTION_ON_CL_POSITION Command

`RULE_SELECTION_ON_CL_POSITION` is the command of the .spyglass.setup file. Based on the value of this command, the tool decides the priority of the `ignorerules` and `rules` commands.

The `RULE_SELECTION_ON_CL_POSITION` command accepts any of the following values:

- `yes`

  In this case, the `rules` command is given higher priority over the `ignorerules` command. Consider the following example:

  ```
  set_goal_option ignorerules R1
  set_goal_option rules R1
  ```

  In this example, the `R1` rule will not be ignored and will run during SpyGlass analysis.

- `no` (default)

  In this case, the `ignorerules` command is given higher priority over the `rules` command. Therefore, in the above example, the `R1` rule will be ignored and it will not run during SpyGlass analysis.

Example of using the ignorerules Command

Following are some examples:

- The following command ignores the `R1` and `R2` rules from SpyGlass analysis:

  ```
  set_option ignorerules {R1 R2}
  ```

- The following example ignores the `R1` rule of the `grp1` rule group:

```
set_goal_option rules grp1
set_goal_option ignorerules R1
```

In the above example, except the `R1` rule, all rules of the `grp1` group are run.

ignorewaivers

Usage:

```
set_option ignorewaivers <yes | no>
set_goal_option ignorewaivers <yes | no>
```

Use this command to ignore waivers supplied as embedded SpyGlass waiver pragmas.

ignore_undefined_rules

Usage:

```
set_goal_option ignore_undefined_rules <yes | no>
```

Use this command to continue SpyGlass analysis after reporting a warning message if an undefined rule is specified.

By default, SpyGlass exits with an error if you specify a rule that does not exist in any of the specified products.

incdir

Usage:

```
set_option incdir <directory-path>
```

For details on this command, see *Searches the specified paths for include files*.

inferblackbox

Usage: `set_option inferblackbox <yes | no>`

Use this command to infer black box module interface based on black box instances in the synthesized netlist and write to the sgBlackbox.v file in the current output directory.

For details, see *Inferring Black Boxes*.

inferblackbox_iterations

Usage: `set_option inferblackbox_iterations <int-value>`

Use this command to specify an effort in terms of the total number of iterations that SpyGlass should make before finalizing port directions for black boxes.

In a single iteration, SpyGlass may not be able to assign a definite port direction to a black box based on connectivity with non black box instances. To achieve a definite port direction, few iterations are required.

Typically, three to four iterations are enough to converge on a good estimate of port direction.

NOTE: *You can specify the name of this command as inferblackbox_iterations or inferblackbox_iteration.*

infer_enabled_flop

Usage: `set_option infer_enabled_flop <on | off>`

Use this command to enable synthesis to create flip-flops without an enable signal.

By default, the value of this command is on. In this case, synthesis creates flip-flops with enable.

Set the value of this command to off to enable synthesis to create flip-flops without enable together with a MUX to get enable functionality.

ipdbdir

Usage: `set_option ipdbdir {list of db's}`

Use this command to specify the list of db's wherein each individual db corresponds to an individual IP block. Depending on the impact of change across save-restore run, the netlist modules are either restored from their respective databases or synthesized afresh incrementally in the restore run.

For more information on this option, see *Restoring From Multiple Databases*.

language_mode

Usage: `set_option language_mode <value>`

Use this command to specify the operating language for the current SpyGlass run. The allowed values are:

- `verilog`

  Specifies that the language mode as Verilog. In this case, you can process only Verilog design files.

- `vhdl`

  Specifies that the language mode as Verilog. In this case, you can process only VHDL design files.

- `mixed`

  Specifies that the language mode as mixed. In this case, you can process either Verilog designs, VHDL designs, or mixed language designs.

For details on specifying a language mode through GUI, see *Language Mode*.

lib

Usage: `set_option lib <logical-lib-name> <physical-path>`

Use this command to define a mapping between the logical name of a library and the actual physical location where the compiled library is present.

By default, SpyGlass considers the WORK directory (in the current working directory) as the path of the working library. The physical location must be an existing directory.

If you map a single logical library to multiple physical locations, the last specified location is used.

If libraries are completely debugged, there should be no need to run SpyGlass rule checks on them and the `norules` option should be used.

Examples of Using the lib Command

Following are some examples:

- The following command maps the `alu` library to the physical directory mylibs:

  ```
  set_option lib { "alu" "~libs/mylibs" }
  ```

- The following command maps the `NEW` and `MATH` libraries to their physical VHDL files

  ```
  set_option lib { "NEW" "~libs/new" }
  set_option lib { "MATH" "~libs/math" }
  ```

Handling of Incorrect Library Path Specifications

In case of incorrect library path specifications, SpyGlass behavior is as follows:

- If a library path is not specified and the library is used only in the `use` clause and not used in any design unit, SpyGlass generates a warning message and continues. If the library is being used in a design unit, SpyGlass generates additional error message and may abort depending on the criticality of library usage.

- If the library path does not exist, SpyGlass generates an error message and aborts before analysis. This is irrespective of whether the specified library is used or not.

NOTE: *The 32-bit version of user-compiled libraries are created in a sub-directory named* 32 *under the specified working directory. The 64-bit version of user-compiled libraries is created in a sub-directory named* 64 *under the specified working directory.*

Precompiled VHDL Libraries

The SpyGlass VHDL environment comes with the following precompiled libraries:

- IEEE
- STD
- SYNOPSYS

By default, these libraries are visible SpyGlass.

Reusing Netlist of DesignWare modules

SpyGlass enables you to reuse a Netlist of DesignWare modules during SpyGlass analysis by supporting a global read-only directory to read the DesignWare VHDL components and packages and Verilog netlist of DesignWare components that are compiled by SpyGlass in a previous run and copied to the `GLOBAL_SPY_DW_WORK` directory with the same directory structure as that of the generated `SPY_DW_WORK` directory.

This enables you to save disk space while reusing Netlists and pre-compiled dump of DW component declarations.

Use the following command to enable the global directory:

```
set_option lib GLOBAL_SPY_DW_WORK <dir-name>
```

SpyGlass checks for existing netlists in `GLOBAL_SPY_DW_WORK` first and then in `SPY_DW_WORK`. SpyGlass generates the remaining netlists required by the design in `SPY_DW_WORK`. Note the following for the GLOBAL_SPY_DW_WORK option:

- If the DW VHDL components and packages are not found in GLOBAL_SPY_DW_WORK (or if empty), the components are created in SPY_DW_WORK (in mixed and VHDL mode)
- If Verilog netlist is not available in GLOBAL_SPY_DW_WORK, it will be generated in SPY_DW_WORK and read from it in that SpyGlass run.
- If GLOBAL_SPY_DW_WORK needs to be recompiled, the libraries are compiled in SPY_DW_WORK. The libraries can then be copied to the global directory for subsequent SpyGlass run.
- If GLOBAL_SPY_DW_WORK contains netlists for only some DW modules, SpyGlass reads these and generates the remaining in SPY_DW_WORK and read them in that current run.

libhdl_extmap

Usage: `set_option libhdl_extmap <file-extension> <file-type>`

Use this command to provide mapping from extension-name to the compilation language in a project file.

This option is same as specifying the LIBHDL_EXTMAP key in the SpyGlass Configuration file. If libhdl_extmap is specified in a project file and the LIBHDL_EXTMAP key is specified in the SpyGlass Configuration file, the extension map from both will be unified for the run.

Examples for existing LIBHDL_EXTMAP in SpyGlass config file, as mentioned in Console guide:

Consider the following keys specified in the SpyGlass Configuration file:

```
LIBHDL_EXTMAP = .v   verilog
LIBHDL_EXTMAP = .v2K  verilog2000
LIBHDL_EXTMAP = .sv  systemverilog
LIBHDL_EXTMAP = .vh87  vhdl87
LIBHDL_EXTMAP = .vh93  vhdl93
```

The corresponding project file commands for the above keys is as follows:

```
set_option libhdl_extmap .v  verilog2000
set_option libhdl_extmap .v2K  verilog2000
set_option libhdl_extmap .sv  systemverilog
set_option libhdl_extmap .vh87  vhdl87
set_option libhdl_extmap .vh93  vhdl93
```

libmap

Usage:	`set_option libmap <logical-lib-name> <intermediate-lib-name>`

Use this command to specify mapping between logical libraries and an intermediate library.

libext

Usage:	`set_option libext <text>`

For details on this command, see *Specify library file extensions*.

library_gen_clock_naming

Usage:	`set_parameter library_gen_clock_naming <yes	no>`

Use this command to switch the naming conventions for the clock names.

By default, the value of this parameter is set no. In this case, SpyGlass generates clock names as per the Liberty Reference Manual. Following is a sample clock naming scheme in SpyGlass when the value of this parameter is no:

```
cell  (namestring {
  generated_clock  (namestring)  { //  This  is  the  name  of  the
generated clock by default
  ...clock data...
    }
}
```

You can set the value of the parameter to yes to generate the clock names in accordance with other industry tools. That is, the hierarchical name in the instance name or pin name format.

LICENSEDEBUG

Usage: `set_option LICENSEDEBUG <yes | no>`

Use this command to print the following license debug information on the screen and in the SpyGlass log file, spyglass.log, in the given order:

- Values of `SNPSLMD_LICENSE_FILE` and `LM_LICENSE_FILE`

  The host IP set in these variables is used to search for licenses.

NOTE: `SNPSLMD_LICENSE_FILE` *is given preference over* `LM_LICENSE_FILE`*.*

- License checkout start message followed by end message for each license checked out

  Here, the time elapsed during the checkout process is also printed. This information is useful if the license checkout time needs to be bench marked.

  In addition, the checkout server name (and server host id) is also printed. This information helps in identifying the servers from where various features have been checked out. On reviewing this data, you may want to rearrange the settings in the `SNPSLMD_LICENSE_FILE` /`LM_LICENSE_FILE` variable to get the features checkout from the nearest server(s), if the license servers are geographically distributed, for quick turnaround time.

- Summary for licenses that are checked out when products are being loaded in the initial stages of SpyGlass run. This information is printed in the following order:

  Total number of features successfully checked out

  Total number of denied licensing calls

  Total time between first and last licensing call

  Total time elapsed in successful licensing calls

  Total time elapsed in failed licensing calls

  Total time elapsed in licensing calls

  Here, some rules also checkout few licenses when they run. The summary for such license checkouts will not be printed. However, license checkout start and end messages are printed for these licenses.

All the license debug information starts with ##LICENSEDEBUG: to aid in quick scanning of license related debugging information.

To stop creation of license jobs for the license files/servers, specified with LM_LICENSE_FILE, specify the following option in the .spyglass.setup file.

```
IGNORE_LM_LICENSE_FILE = yes
```

lvpr

Usage:

```
set_option lvpr <rule-name>=<num>
set_option lvpr <num>
set_goal_option lvpr <rule-name>=<num>
set_goal_option lvpr <num>
```

Where:

- <rule-name> is the name of the rule for which you want to limit the number of messages.
- <num> is the maximum number of messages to be reported.

Use this command to specify the maximum number of messages to be reported for a particular rule. This way, you can limit the number of repetitions of individual rule messages, which makes it easier to focus on other messages.

This command is useful when errors are difficult to identify because some rules report multiple messages that are similar in nature. For example, multiple rules may report a name that is frequently used in a design and that name violates the defined naming convention.

Examples of using the lvpr Command

Following are the examples:

- The following command limits the number of messages to 20 for the *SigName* rule of SpyGlass OpenMore solution but keeps an infinite limit for all other rules checked in a Verilog design file:

  ```
  set_goal_option lvpr SigName=20
  ```

- The following command limits the number of messages reported for any rule of the product being used to 30 in a VHDL design file:

  ```
  set_option language_mode vhdl
  set_goal_option lvpr 30
  ```

- The following command limits the number of messages to 30 for the *SigName* rule of SpyGlass OpenMore solution, and limits the number of all the other messages of this product to 20:

  ```
  set_goal_option lvpr 20
  set_goal_option lvpr SigName=30
  ```

NOTE: *The maximum limit per rule (30 for the SigName rule in this case) overrides the overall limit set (20 in this case).*

- The following command limits the number of messages to 10 for the *SigName* rule 20 for the *ClkName* rule:

  ```
  set_goal_option lvpr SigName=10
  set_goal_option lvpr ClkName=20
  ```

- The following command reports all messages of all rules:

  ```
  set_goal_option lvpr -1
  ```

macro_synthesis_off

Usage: `set_option macro_synthesis_off <yes | no>`

Use this command to disable the `SYNTHESIS` macro. By default, SpyGlass includes the `SYNTHESIS` macro.

mapSuffixList

Usage:

```
set_option mapSuffixList <list>
set_goal_option mapSuffixList <list>
```

For details on this command, see *Specify parameter to give the list of suffix strings*.

mapPrefixList

Usage:

```
set_option mapPrefixList <list>
set_goal_option mapPrefixList <list>
```

For details on this command, see *Specify parameter to give the list of prefix strings*.

mapVirtualClkByName

Usage:

```
set_option mapVirtualClkByName <yes | no>
set_goal_option mapVirtualClkByName <yes | no>
```

For details on this command, see *Specify the manner in which virtual-to-real clock mapping to be done*.

max_msg_len

Usage: `set_parameter max_msg_len <-1 | 0 | <n>>`

Use this parameter to specify the maximum length of the violation messages reported by SpyGlass in VDB file.

The following table lists the permissible values for the max_msg_len parameter:

TABLE 22 max_msg_len parameter values

Value	Description
0 (Default)	The maximum message limit is set to 10000.
-1	SpyGlass prints the complete violation messages.
<n>	Any positive integer value. SpyGlass prints the violation message upto the specified limit.

mthresh

Usage: `set_option mthresh <num>`

For details on this command, see *Upper Threshold for Compiling Memories*.

max_err_count

Usage: `set_option max_err_count <num>`

Use this option to control the number of syntax fatal errors reported during design analysis. You can set this option to any positive integer, which limits the number of syntax fatal errors that will be reported.

By default, this option is set to 200.

When you set this option to 0, no limit is set and all the fatal errors are reported.

migrate_non_flat_ports

Usage: `set_option migrate_non_flat_ports <yes | no>`

In SpyGlass, port names for ports reporting violations below the FLAT_DU level, are not migrated. Set this option to yes to migrate the ports in all block-level violations to the top-level.

module_inst_mapping_file

Usage: `set_option module_inst_mapping_file <file-name>`

Use this option to specify the instance names and the IP names that you need to include in the IP-Based report.

Ensure that the IP name and the instance names are specified in the following format in the file-name mapping file:

```
<ipname>:<full_instance_path>
```

For example, consider the following input specified in the mapping file:

```
ahb2wb:wb_subsystem.ahb2wb_u0
```

The `Moresimple_<ip_name>_<instpath>.rpt` files and `waive_<ipname>_<instpath>.swl` files are generated automatically. In the above example, the `Moresimple_ahb2wb_wb_subsystem.ahb2wb_u0.rpt` files and `waive_ahb2wb_wb_subsystem.ahb2wb_u0.swl` files are generated.

Note that you need to invoke the waiver report generator in project file, as shown below, to generate the `waiver_<ip_name>_<instpath>.rpt` reports:

```
set_option report {waiver}
```

The IP-based report generation is enabled during run_goal only. Incremental report generation is not supported.

net_osc_count_limit

Usage: `set_option net_osc_count_limit <num>`

Use this command to specify the number of oscillations allowed to get a stable value on a particular net within SpyGlass logic evaluator.

By default, the limit for the oscillation count for any net is 100. You can override this default value by using the `net_osc_count_limit` command, as shown in the following example:

```
set_option net_osc_count_limit 10
```

netlist

Usage: `set_option netlist <true | false>`

Use this command to specify a design as a netlist design to enable various run-time optimizations.

Ensure that this option is specified only for the netlist design. Spyglass behavior is undefined when this option is specified for the non-netlist or RTL designs.

Note that you cannot perform the following tasks using this command:

- Enable module by module deletion by using the `enable_mmdelete` or `higher_capacity` command.
- Generate a precompile dump by using the `enable_precompile_vlog` command or single step precompile feature by using the `libhdlf` or `libhdlfiles` commands.

netlist_clock_polarity

Usage: `set_option netlist_clock_polarity <yes | no>`

Use this parameter to improve the calculation of number of inversions inside the library cells.

By default the value of this parameter is set to yes. In this case, the waveform of the generated clocks is shifted appropriately, depending on whether odd number of inversions are found.

Set the value of this parameter to no to ignore the waveform of the generated clocks.

nobb

Usage: `set_option nobb <yes | no>`

For details on this command, see *Exit on Detecting Blackboxes in the Design*.

noelab

Usage: `set_option noelab <yes | no>`

Set this command to `yes` to exit after design analysis without elaborating the design. In this case, only built-in rules are checked on the design.

Set this command to `yes` when you need to compile only Verilog/VHDL files without rule-checking.

NOTE: *Do not set the* `noelab` *command to* `yes` *when the* `top` *command is also specified. If you specify these commands together, SpyGlass exits and reports an error.*

nodefparam

Usage: `set_option nodefparam <yes | no>`

Use this command to ignore explicit parameter re-definition given by a `defparam` Verilog construct.

Verilog parameters can be redefined within a module instance by using `defparam` statements, as shown in the following example:

```
module ram (...);
  parameter WIDTH = 8;
  parameter SIZE = 256;
  ...
endmodule
module my_chip (...);
  ...
  //Explicit parameter redefinition by name
  RAM ram1 (...);
  defparam ram1.SIZE = 1023;
  ...
endmodule
```

If you want to disable these parameter re-definitions and want to use original parameter values, use the `nodefparam` command.

nopreserve

Usage: `set_option nopreserve <yes | no>`

Use this command to remove hanging/unconnected instances and nets.

SpyGlass RTL synthesis engine usually retains hanging nets (open-ended interconnections among logic gates) and hanging/unconnected instances. If you want to remove such instances and nets from your design so that the result matches with that of your main synthesis engine, specify this command.

Preserving all instances and nets makes it easier to relate inferred logic back to your source code.

noreport

Usage:

```
set_option noreport <yes | no>
set_goal_option noreport <yes | no>
```

Use this command to suppress report generation.

Use this command if you want to analyze results separately (for example in the SpyGlass GUI).

If you do not specify a report format, SpyGlass uses the moresimple report, which is the default SpyGlass report.

Sometimes you might not need any extensive report. For example, you might want only a quick status check of your design to make sure no errors remain, or you might prefer to analyze errors using the SpyGlass GUI. In such cases, use this command to suppress all reporting.

Example of using the noreport Command

The following command suppresses report generation for the Verilog file test1.v:

```
set_option language_mode verilog
read_file -type verilog test1.v
set_option noreport yes
```

norules

Usage:

```
set_goal_option norules <yes | no>
```

Use this command to suppress rule-checking during SpyGlass analysis.

When you set this command to `yes`, SpyGlass analyzes a design for syntax errors and warnings only. In addition, SpyGlass reports only RTL description-level built-in messages.

You should set this command to `yes` to check that:

- All files, libraries, include files, macro definition files, etc. are all present and correctly defined before any rules are checked.
- VHDL libraries are precompiled and checked for HDL syntax errors before any rules are checked.

nosavepolicy/nosavepolicies

Usage:

```
set_option nosavepolicy <product-name>
set_option nosavepolicies <product-list>
```

Use these commands to specify a product (`nosavepolicy`) or a list of products (`nosavepolicies`) that should not be saved during design save.

Under Design Save-Restore feature, you specify products during design save and run any subset of these products during design restore. Entire base polices and the four advanced products, that is, SpyGlass Constraints solution, SpyGlass DFT solution, SpyGlass Power Verify solution, and SpyGlass TXV solution, are saved by default. In addition, the design view related to the products being used, which are specified indirectly through goals is also saved. If you do not want to save certain products, specify such products by using the `nosavepolicy/nosavepolicies` command.

Prerequisite

The *nosavepolicies* command works only when the *enable_save_restore* command is set to `yes`.

Exception

The *nosavepolicies* command is ignored during design restore mode.

Keywords supported by nosavepolicy/nosavepolicies

You can specify the following keywords with `nosavepolicy/nosavepolicies`:

Keyword	Products Covered by the Corresponding Keyword			
basepolicy	timing	starcad-21	starc2005	starc2002
	starc	simulation	openmore	morelint
	miscellaneous	lint	latch	erc
	area	Audits		
advpolicy	txv	lowpower	power_est	dft
	dft_dsm	const_intern1	constraints	clock-reset
all	Union of basepolicy and advpolicy			

To check for the product names to be used as a keyword for this command, refer to product name directory in your installation directory.

To disable the SpyGlass Constraints product, also specify the const_intern1 and Spyglass Txv product with the nosavepolicy/nosavepolicies option as shown below:

```
set_option nosavepolicy { constraints const_intern1 txv }
```

Similarly, to disable the SpyGlass DFT product, also specify the SpyGlass DFT DSM product with the nosavepolicy/nosavepolicies option as shown below:

```
set_option nosavepolicy { dft dft_dsm }
```

However, if you need to disable the SpyGlass Txv product or SpyGlass DFT DSM product, you need not specify the SpyGlass Constraints or SpyGlass DFT product, respectively, with the nosavepolicy/nosavepolicies.

For example, to discard the design data for the lint product, specify the lint keyword as mentioned in the <your-inst-dir>/SPYGLASS_HOME/policies/lint directory, with the `set_option nosavepolicy` command.

Also, consider the following examples:

```
set_option nosavepolicy dft
```

```
set_option nosavepolicy {basepolicy dft}
```

The first example listed above discards the design data for the SpyGlass DFT product. In the second example, the SpyGlass DFT product and all the products covered under the basepolicy keyword are not saved during design save.

NOTE: *Specify the 'advcdc' keyword along with the nosavepolicy option to prevent the checkout of adv-cdc product license, which otherwise gets checked out implicitly if an Advanced CDC rule is running.*

To save a product during design save, use the *savepolicy/savepolicies* command.

non_lrm_options

Usage:	`set_option non_lrm_options < ignore_tokenpasting_space	dft_nettype_on_port	interconnect_type >`

Use this command to enable SpyGlass to support non-LRM constructs.

The command takes the following values:

- **ignore_tokenpasting_space**: Set the *non_lrm_options* command to this value to enable the *INFO_999* rule to not report a violation if white space is found before or after the `` (token pasting operator) during macro expansion.
- **dft_nettype_on_port**: Set the *non_lrm_options* command to this value to enable the *STX_601* rule to report a violation if `default_nettype = NONE` and the nettype is not specified in the IO list. For example, consider the following code:

```
`default_nettype none
module test (
   input wire clk,
   input wire rst_n,
   input      din                  //Flag STX if
dft_nettype_on_port is enabled and nettype of port is not
give while default net type is set to NONE.
)
```

- **interconnect_type**: Set the non_lrm_options command to this value to enable SpyGlass to support the interconnect SV2012 construct. Note that SpyGlass converts interconnect constructs as wire. Specify the set_option enableSV09 option with the interconnect_type option to enable support for interconnect constructs.

overload

Usage: `set_goal_option overload <named-overload>`

Use this command to runs the specified named overloads for all the specified products.

For details on using named overloads, refer to the *SpyGlass Policy Customization Guide*.

overloadpolicy

Usage: `set_goal_option overloadpolicy <product-name>`

Use this command to run the specified products with overloaded components, if any.

NOTE: *Rule-checking results with overloaded products may be different from results from normal products.*

If you do not use this command, overloaded product components, if any, are ignored and normal products are run.

Need to the overloadpolicy Command

This command is used to overload a product to meet local preferences of a user base. It is typically required when you want to use Atrenta-standard products but with certain customization, such as different severity-labels, pod-cut description, and alias name.

Prerequisites for Using the overloadpolicy Command

To use this command, create a file named <product-name>-policy-overload.pl where *<product-name>* is the name of the original product you want to overload. For example, to overload SpyGlass lint solution, create the file lint-policy-overload.pl file.

NOTE: *For information on creating product overload files, refer to the SpyGlass Policy Customization Guide.*

Product overload files are searched using the same *-I* command-line option mechanism as used for custom products.

This command does not work, if the overloaded product and the directory containing this product have the same name. Use a different name than the name of the overloaded product for the directory containing this product.

Using the overloadpolicy Command

Based on the value specified to this command, SpyGlass performs different actions, as discussed in the following table:

Specification	Action
`set_goal_option overloadpolicy`	Apply product overload on all product being selected to run. This is subject to product overload file being found in the specified search path.
`set_goal_option overloadpolicy none` (Or not specified)	Disables search and loading of overload files
`set_goal_option overloadpolicy a,b`	Apply product overload to the a and b products only, assuming one or more of these product are selected to run.
`set_goal_option overloadpolicy all`	Apply product overload on all products selected to run. If product overload file is not found for any product, it is considered as an error condition, and (frequently) may be because of incorrect or no *-I* specification.

overloadrules

Usage:

```
set_option overloadrules <rule-name>+<
lang>+severity=<severity-label>+weight=<value>

set_goal_option overloadrules <rule-name>+<
lang>+severity=<severity-label>+weight=<value>
```

Note that all settings and the intervening + characters must be specified without any spaces.

Use this command to overload the severity or weight of a rule for the current SpyGlass run.

NOTE: *The* `overloadrules` *command is used in conjunction with spyOverload specification. However, the* `overloadrules` *command takes precedence over the spyOverload specification.*

NOTE: *The* `overloadrules` *command is not applicable for rule aliases.*

Arguments of this command are discussed below:

- *`<rule-name>`* is the name of the rule being overloaded.
- *`<lang>`* is the language for which you want to overload a rule. The allowed values are `VHDL`, `Verilog`, `Verilog+VHDL` (case-insensitive).

  Language specified with the `overloadrules` command should be same as used in the corresponding rule registration. For example, if a rule is registered with `Verilog + VHDL` language, its `overloadrules` specification should also be `Verilog + VHDL`.

  NOTE: *If a rule is registered in a single language (Verilog or VHDL), and there is no* `overloadrules` *specification in the same language (through* `overloadrules` *or through spyOverload), the* `overloadrules` *command with Verilog+VHDL is still applied for backward compatibility purposes. However, this is not a recommended use-model and should not be used.*

  The *`<lang>`* setting is optional. If you do not specify a language, the overload values are applicable to all language variations of the rule. Therefore, if you are not sure about the language specification of rule registration, you can omit the language option from `overloadrules` specification.

- *<severity-label>* is the overloaded severity label. The severity label can be one of the SpyGlass predefined severity labels (`Fatal`, `Error`, `Warning`, `Info`, and `Data`), one of the severity labels defined in the corresponding product, or a user-specified severity label defined using the `define_severity` command. If the specified severity label is not any of these labels, SpyGlass auto-registers the severity label under the severity class `ERROR`.

NOTE: *As you overload a rule's severity label, the SpyGlass processing of the rules will be governed by the severity class of the overloaded severity label. For example, when you overload a rule with a severity label of severity class FATAL, SpyGlass aborts when a rule-violation of this rule is encountered.*

- *<value>* is the overloaded rule weight value.

You can specify the *<rule-name>* followed by any combination of the other settings.

Example of using the overloadrules Command

Following are some examples:

- The following example indicates that the severity label for the Verilog version of the W402b rule (of SpyGlass lint solution) should be `Warning` (one of the SpyGlass predefined severity labels) for the current SpyGlass run:

```
set_option overloadrules W402b+Verilog+severity=Warning
```

- The following example indicates that the severity label for all language versions of the Clk_Gen03a rule (of SpyGlass Constraints solution) should be `Error` (one of the SpyGlass predefined severity labels) for the current SpyGlass run:

```
set_option overloadrules Clk_Gen03a+severity=Error
```

- The following example indicates that the rule weight for the Verilog version of the InstNameLength rule (of SpyGlass OpenMore solution) should be 10 (the default weight is 5) for the current SpyGlass run:

```
set_option overloadrules InstNameLength+Verilog+weight=10
```

- The following example indicates that the severity label for the VHDL version of the W116 rule (of SpyGlass lint solution) should be myERROR1 (defined using the define_severity command) under the severity class ERROR for the current SpyGlass run:

  ```
  set_goal_option define_severity lint+myERROR1+ERROR
  set_goal_option overloadrules W116+VHDL+severity=myERROR1
  ```

- The following example indicates that the severity label for the TA_01 rule (of SpyGlass DFT solution) should be Info (one of the SpyGlass predefined severity labels) and the rule weight should be 10 (the default weight is 1) for the current SpyGlass run:

  ```
  set_option overloadrules TA_01+severity=Info+weight=10
  ```

Specifying overloadrules Specification for a Particular Label

You can also specify overloadrule specification for a particular label. For example, consider a multi message rule, *dummyRule*, with the following assumptions:

- The rule that has two labels, Label1 and Label2.
- Severity for Label1 is Warning and severity for Label2 is Info.

Now if you want to change the severities of both the labels to Error, use the following overloadrule specification:

```
set_option overloadrules dummyRule+severity=Error
```

To change the severity of a specific label (say Label1), you must also specify the language for which you want to overload, unless label is present for all the personalities of the rule:

```
set_option overloadrules
dummyRule+severity=Error+msgLabel=LABEL1+Verilog
```

Consider the following message labels for the W123 rule for both Verilog and VHDL:

Verilog

- veCheckUsage_Viol_Msg
- SignalUsageReport_Refer
- veCheckArrayOutBound_Info

VHDL

vhLintArray_Viol_Msg

SignalUsageReport_Refer

If you want to overload a specific message label, you must also specify the language for which you want to overload as shown in the following example:

```
set_option overloadrules
W123+severity=Violation+msgLabel=veCheckUsage_Viol_Msg+Veril
og
```

If a message label is present for all the variants of the rule, both Verilog and VHDL, in that case you do not need to mention the language while overloading this label specifically. Following example demonstrates this condition:

```
overloadrules
W123+severity=Info+msgLabel=SignalUsageReport_Refer
```

overload_rule_category

Usage:	`set_goal_option overload_rule_category <rule_name>+rulecategory=<rule_category>`

Use this command to overload the rule category with a user-specified category.

Overloaded rule categories are included in the moresimple report after run_goal. The overload feature is available only if the `enable_rule_category_in_moresimple option` is enabled for the design. If the `enable_rule_category_in_moresimple` is enabled, all the violations in moresimple report are sorted by severity and then by rule category.

ovl_verilog

Usage:	`set_option ovl_verilog { OVL-file> }`

Use this command to specify an Open Verification Library (OVL) file written in Verilog. This file is used for functional analysis.

ovl_vhdl

Usage:
```
set_option ovl_vhdl { OVL-file> }
```

Use this command to specify an OVL file written in VHDL. This file is used for functional analysis.

param

Usage:
```
set_option param <name>
```

Use this command to specify parameters used during Verilog/VHDL analysis run.

perflog

Usage:
```
set_option perflog <yes | no>
set_goal_option perflog <yes | no>
```

Use this command to generate SpyGlass performance log.

This performance log is written in the spyglass.perflog file in the current working directory. This file contains memory/runtime details of each rule run. In addition, the performance log also contains benchmark data for various stages of SpyGlass run, such as analysis, synthesis, and flattening.

physical

Usage:
```
set_option physical <true | false>
```

Use this command to enable physical library compilation of Synopsys Liberty files (.lib files).

SpyGlass Library Compiler provides the feature to perform physical library compilation of Synopsys Liberty files (.lib files) and other physical related inputs, such as technology LEF files. This is equivalent to the library preparation step for SpyGlass Physical.

This option is meaningful only when automatic compilation of gateslib is enabled and Synopsys Liberty files (.lib files) are provided.

physical_dbdir

Usage: `set_option physical_dbdir <directory_path>`

Use this command to specify the design database directory path for save and restore runs in Physical aware power estimation flow. This switch is used in `physical_power_postfloorplan` and `power_est_average` goals.

By default, if you do not specify a directory path, SpyGlass uses default location for design database save and restore runs.

physical_lib_config_file

Usage: `set_option physical_lib_config_file {<config-file-path>}`

Use this command to specify the path of a configuration file for SpyGlass Physical library preparation through physical library compilation.

This option is meaningful only when you do the following:

- Enable physical library compilation by using the following command:

  `set_option` *physical* `true`

- Enable automatic compilation of gateslib.

Using this command enables you to extend SpyGlass Physical library compilation for advanced capabilities.

For information on the semantics of this configuration file, refer to

SpyGlass Physical documentation.

The following example shows the usage of this command:

```
set_option physical_lib_config_file {./
my_libprep_config_file}
```

physical_target

Usage: `set_option physical_target {space-separated-target-list>}`

Use this command to specify a list of library targets for SpyGlass Physical library preparation through physical library compilation.

This option is meaningful only when you do the following:

- Enable physical library compilation by using the following command:

  ```
  set_option physical true
  ```

- Enable automatic compilation of gateslib.

The following example shows the usage of this command:

```
set_option physical_target {target1 target2 target3}
```

Alternatively, you can specify a list of library targets by specifying the list to the SpyGlass Physical configuration file. You can then specify this file to the *physical_lib_config_file* command.

For information on the semantics of this configuration file, refer to SpyGlass Physical documentation.

portparam

Usage: `set_option portparam { <list-of-space-separated-strings> }`

Use this command to override simple type parameters of top module ports only. The string specified should be in the following format:

```
<module_name>.<port_name>.<param_name>=<param_value>
```

Where:

- <module_name>: Specify only the top module name
- <port_name>: Specify only port name of SV interface type port
- <param_name>: Specify only full name of parameter; do not specify bit select/part select
- <param_value>: Can be an integer or a base number.

Note that there are no spaces between any values in the above format. In addition, a comma separated list of strings is considered as an invalid input.

SpyGlass reports an ELAB_6205 warning with appropriate reason if invalid inputs are specified with the portparam option.

Example

Consider the following RTL:

```
typedef int myint;
interface test_inf ( );
parameter integer W1 = 4;
parameter [3:0] W2 = 4'b0100;
parameter myint W3 = 4;

wire [W1-1:0] in1
wire [W2-1:0] in2;
wire [W3-1:0] in3;
endinterface

module top (test_inf P1, output logic P2);
test I1 (.intf1(P1));
endmodule

module test ( test_inf intf1);
endmodule
```

The override the simple type parameters of top module ports, top.P1.W1 and top.P1.W3, specify the following commands:

```
set_option portparam {"top.P1.W1=7"}
```

```
set_option portparam {"top.P1.W3=7"}
```

pragma

Usage: `set_option pragma { <list-of-pragmas> }`

For details on this command, see *Interpret Pragma(s)*.

prefer_tech_lib

Usage: `set_option prefer_tech_lib <yes | no>`

Use this command to provide higher preference to technology library definitions present in the .lib/.sglib file over user-specified definitions present in source HDL files, precompiled libraries, and simulation models while searching for the master of an instance.

By default, SpyGlass gives higher priority to user-specified definition.

For example, in the following case, the `AN2` definition is picked from .sglib:

```
read_file -type sglib AN2.sglib
read_file -type verilog top.v AN2.v
set_option prefer_tech_lib yes
```

In the following example, the `AN2` definition is picked from .lib:

```
read_file -type gateslib AN2.lib
read_file -type vhdl top.vhd AN2.vhd
set_option enable_gateslib_autocompile yes
set_option prefer_tech_lib yes
```

In the following example, the `AN2` definition is picked from .sglib even if the library `L` contains `AN2`:

```
read_file -type sglib AN2.sglib
read_file -type verilog top.v
set_option lib L P
set_option prefer_tech_lib yes
```

Please note the following points about the prefer_tech_lib command:

- Highest priority is given to technology library definitions in the form of the .lib/.sglib file. This priority is above source file specifications and not only HDL libraries models/precompile dump.
- Use the prefer_tech_lib command judiciously as there is a time penalty during HDL analysis stage.

  In the absence of this command, whenever there is a duplicate definition in an HDL and technology library, the *IgnoredLibCells* rule reports a violation. In this case, if you want to give higher priority to technology libraries, re-run the design with the prefer_tech_lib command.
- Irrespective of whether a functional view exists for a .lib cell definition, SpyGlass gives higher priority to technology library definitions.

  If you want to overwrite or add functional view of .lib cell definition, you can do it in the library compilation stage.
- If you have specified the ignoredu command in which the specified design unit is present in both HDL and .sglib/.lib file, the ignoredu command is given higher priority. That is, the technology library cell definition is also ignored in this case.
- Presence or absence of the prefer_tech_lib command triggers re-synthesis or re-save in the save-restore flow.
- SpyGlass does not report parsing violations on an HDL design unit definition that has been overridden by a technology library definition due to the prefer_tech_lib command.
- SpyGlass reports analysis violations, such as *INFO_1006* (Verilog) and *INFO_631* (VHDL) while ignoring a design unit definition because it is also present in technology library input.
- If the prefer_tech_lib command is used in an HDL precompile run where the .lib/.sglib files are also passed and duplicate definitions (HDL and .lib/.sglib) exist, SpyGlass does not dump the HDL definitions.

Technology library models are not dumped in the HDL pre-compile dumps.

preserve_hanging_nets

Usage: `set_option preserve_hanging_nets <0 | 1 | yes | no>`

Use this command to preserve the user hanging nets in the generated netlist.

By default, the value of this command is 0/no. Therefore, all the user hanging nets in the generated netlist are deleted during synthesis.

To preserve the user hanging nets in the RTL, set the value of this command to 1/yes.

Consider the following example where SpyGlass will not remove the user hanging nets test2 and test4 in the generated netlist:

```
module test(in1,in2);
input in1,in2;
wire [6:0] out1;
wire [1:0] test1, test3, test4;
wire [1:0] test2;

  assign out1[1]= in1 & in2;
  assign test1[0] = in1 & in2;
  assign test3[1] = 1'b0;
endmodule
```

preserve_mux

Usage:

```
set_option preserve_mux <yes | no>
set_goal_option preserve_mux <yes | no>
```

Use this command to enable SpyGlass to pick mux cells from a technology library for mapping muxes in a user design.

When you set this command to `yes`, SpyGlass gives priority to the mux cells present in the technology library rather than a mux implementation in terms of more basic cells from the technology library.

If mux cells are not present in the technology library, SpyGlass selects the mux implementation in terms of more basic cells from the technology library.

The preserve_mux command can be specified only when the *-est_mode* command is used or EST mode compatible rules are specified.

print_sortorder_only

Usage: `set_option print_sortorder_only <yes | no>`

Use this command to print the list of sorted VHDL files.

NOTE: *The* `print_sortorder_only` *option is used only with the* `sort` *option. Spyglass exits after printing the sorted VHDL files.*

prohibit_waiver

Usage: `set_option prohibit_waiver <rule_names>`

Use this command to prohibit a waiver from waiving the violation messages of specified rules, without editing the waiver commands in the pre-existing waiver files. See *Example 1* for the usage of the *prohibit_waiver* command.

Specify list of rules, whose violation messages you do not want to waive, as an input to this command.

The violation messages for the rule specified in the *prohibit_waiver* command will not be waived, even if you have not specified that rule name in the waive command. See *Example 2* for more information.

Example 1

Consider a waiver file containing the following commands:

```
waive -rule R1
waive -rule R2
```

If you don’t want to waive the violations of the rule R1, then specify the following command in the project file:

```
set_option prohibit_waiver R1
```

Example 2

Consider a waiver file containing the following command:

```
waive -msg M1 -file filename
```

Also consider the following *prohibit_waiver* command specified for the R1 rule:

```
set_option prohibit_waiver R1
```

In the above example, rule name, R1, is not specified in the waiver command, but the violation that is getting waived is for R1. However, this rule is specified in the *prohibit_waiver* constraint command, Therefore, the violation messages for R1 will not get waived.

projectwdir

Usage: `set_option projectwdir <dir-name>`

Use this command to specify a project working directory.

For details on a project working directory, see *Project Current Working Directory*.

project_read_only

Usage: `set_option project_read_only <yes | no>`

Use this command to make the project file as read-only and disable overwriting of the project file.

For more details on this command, see *Making Project File Read Only*.

elab_summary_include_localparam

Usage: `set_option elab_summary_include_localparam <yes | no>`

Use this option to avoid including the localparam parameter in the elab_summary report. By default, this option is set to `yes`.

honor_spq_parameter_with_turbo

Usage:
```
set_option honor_spq_parameter_with_turbo <yes | no>
```

Use this option to read parameters from .spq files in turbo mode. By default, this option is set to `no`.

read_file

Usage:
```
read_file -type <file-type> <file-list>
```

Where,

- <file-type> refers to the type of file being read. It can be any of the following:
  verilog, *vhdl*, *def*, *lef*, *hdl*, *gateslib*, *sglib*, *plib*, *sgdc*, *waiver*, *sourcelist*, adc, awl, *sourcelist_relative*
- <file-list> refers to list of files to be read.

For more information on the Tcl-based usage of the *read_file* command, refer to the *read_file* section of the *SpyGlass Tcl Shell Interface User Guide*.

Use this command to specify the files to be used during SpyGlass analysis.

You can specify any of the following types of files to be read:

def

Specifies Design Exchange Format (DEF) files that are used by SpyGlass Power Verify solution.

These files contain design-specific information of a circuit. For example, these files contain information about instantiated gates in a design. You can specify information about such gates in the Synopsys Liberty™ format library files (.lib files). If you do not specify any information for such gates, SpyGlass considers these gates as black boxes.

You can specify multiple DEF designs in case of hierarchical DEF designs. However, you must:

- Specify these files in a correct compilation sequence.
- Set a top-level design unit by using the *top* command.

NOTE: *You must also specify lef files with this file type.*

NOTE: *Do not mix DEF files with Verilog or VHDL files.*
While analyzing DEF designs, SpyGlass does not accept any other form of RTL, such as VHDL or Verilog design files as input. SpyGlass considers all the specified design files as DEF files. If you specify any Verilog/VHDL file as input, SpyGlass reports an appropriate message.

lef

Specifies Library Exchange Format (LEF) files that are used by SpyGlass Power Verify solution and SpyGlass Power Estimate.

These files provide information about power and ground pins of cells present in library files specified by the *gateslib* or *sglib* commands.

NOTE: *While using SpyGlass Power Verify solution, it is mandatory to specify this file type if you specify the* `set_option enable_pgnetlist` *command. This is required to enable processing of the specified LEF files. However, for SpyGlass Power Estimate, do not specify this file type with the* `set_option enable_pgnetlist` *command.*

vhdl

Specifies VHDL files.

NOTE: *If you specify more than one file, list them in the correct compilation sequence.*

verilog

Specifies Verilog files.

sgdc

Specifies design constraints file (.sgdc). For details on design constraints, see *Working with SpyGlass Design Constraints*.

waiver

Specifies waiver file containing the `waive` constraints.

The `waive` constraints and SGDC constraints processing is as follows:

File contains	read_file -type waiver <file-name>	read_file -type sgdc <file-name>
Constraints other than waive constraint	Other constraints are ignored with an ERROR message	Other constraints are processed normally
Only waive constraints	waive constraints are processed normally	waive constraints are processed with a WARNING message
Both waive constraints and other constraints	Other constraints are ignored with an ERROR message. waive constraints are processed normally	waive constraints are processed with a WARNING message. Other constraints are processed normally.

Files specified by using the waiver command may contain setvar commands. If you specify more than one waiver file containing setvar commands and the setvar command in both the waiver files is for the same variable, SpyGlass considers the latter specified waiver file to be used by the RTL pragma (that is, pragma2waiver.swl file).

NOTE: *You can specify the setvar commands to be used in RTL pragmas in waiver files only and not in SGDC files. The setvar commands in SGDC file (specified through sgdc command) will be considered only for the SGDC files and not for RTL pragmas.*

SpyGlass reports a warning message if the setvar commands specified in a waiver file are not used in that waiver file or RTL pragmas.

sourcelist

Specifies source files (.spp or .f).

These files are ASCII files that contain the following information in separate lines:

- Path of files to be used in the current SpyGlass run.

NOTE: *SpyGlass expands file names specified with wildcard expressions. For example, file name, such as *.vhd is interpreted at runtime.*

- Moving to the next line

  You can continue a command-line to the next line by using the backslash (\) continuation character as in the following example:

```
...
-y \
./libdir
+libext+.v
...
```

The above specification is equivalent to the following:

```
...
-y ./libdir
+libext+.v
...
```

- Specifying Comments

  You can also add comments in any of the following formats in source files:

  - ❐ // format
  - ❐ # format
  - ❐ /* */ format

NOTE: *You can specify nested source files.*

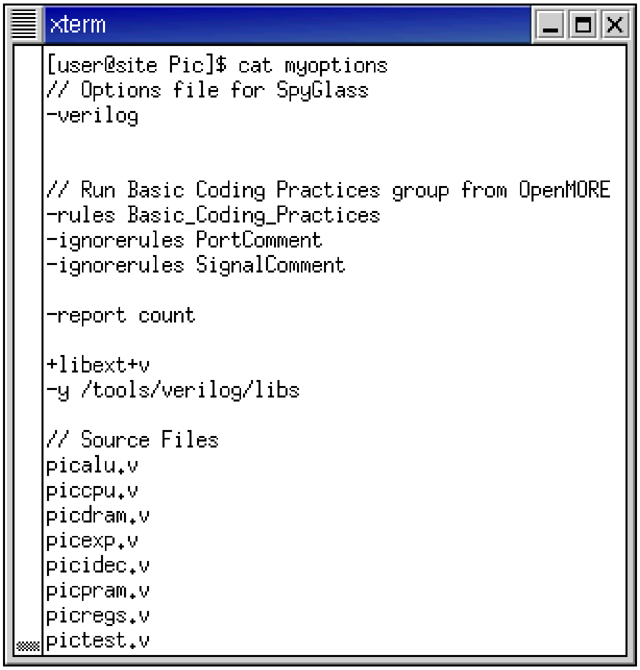
- Type of options

  You can specify the following types of options in a source list file:

TABLE 23 Types of options for Sourcelist File

Option	Label	Syntax
bool	<bool_opt>	''-bool_opt''
scalar/string	<str_opt>	"-str_opt str_opt_val''
list type	<list_opt>	"-list_opt list_opt_val1, list_opt_val2"

Following is an example of a source file:



```
[user@site Pic]$ cat myoptions
// Options file for SpyGlass
-verilog


// Run Basic Coding Practices group from OpenMORE
-rules Basic_Coding_Practices
-ignorerules PortComment
-ignorerules SignalComment

-report count

+libext+v
-y /tools/verilog/libs

// Source Files
picalu.v
piccpu.v
picdram.v
picexp.v
picidec.v
picpram.v
picregs.v
pictest.v
```

FIGURE 240. Sample Source File

sglib

Specifies SpyGlass-compatible format files (`.sglib` files).

hdl

Specifies HDL files, such as Verilog, VHDL, and DEF files.

gateslib

Specifies library files containing functionality information of gates (cells) instantiated in a design.

plib

Specifies the plib format files.

While specifying these files, you must also specify .lib files and .sglib files using the gateslib or sglib options of the `read_file` command.

In addition, to enable processing of plib files, the `set_option` enable_pgnetlist command.

NOTE: *The* plib *option is used by the SpyGlass Power Verify solution only.*

sourcelist_relative

The option honors relative file references specified in the dir/source.f file.

For example, consider that the run directory is /u/top/ and the following is specified in the project file:

```
read_file -type sourcelist_relative xyz/flist.f
```

In addition, consider that the xyz/flist.f file has the following:

```
sample.v
-f dir1/pqr.f
```

Where, the dir1 directory contains the sample1.v file.

In this scenario, SpyGlass reads the files in the following order:

```
/u/top/xyz/flist.f
/u/top/xyz/sample.v
/u/top/xyz/t/pqr.f
/u/top/xyz/sample1.v
```

In addition, the -F option can be used to enable reading files from relative paths. For example, consider that the xyz/flist.f file in the above scenario contains the following:

```
a.v
-F s/v/m.f
```

Where, the m.f file specifies the sample2.v file.

In this scenario, SpyGlass now reads the files in the following sequence:

```
/u/tom/xyz/flist.f
/u/tom/xyz/sample.v
/u/tom/xyz/s/v/m.f
/u/tom/xyz/s/v/sample2.v
```

read_protected_envelope

Usage: `set_option read_protected_envelope <yes | no>`

This command enables the parsing and decryption of Protected Envelope in both VHDL and Verilog.

For VHDL envelopes, this command has lower priority than the `set_option sort yes` command. Although, the parsing and decryption in the same design would work for Verilog files without any issues.

reference_design_projectfile

Usage: `set_goal_option reference_design_projectfile <file-name>`

Specifies a project file containing details, such as design files, design options, and SGDC for the reference design while running a DDR (Dual Design Read) goal.

You can use this command instead of the *reference_design_sgdc* and *reference_design_sources* commands. In this case, the project file specified by this command should contain details that you wanted to specify by using the *reference_design_sgdc* and *reference_design_sources* commands.

reference_design_sgdc

Usage: `set_goal_option reference_design_sgdc <file-name>`

Use this command to specify an SGDC file that contains constraints for the reference design in the Dual Design Read (DDR) flow.

reference_design_sources

Usage: `set_goal_option reference_design_sources <file-name>`

Use this command to specify a source file (.f file) that specifies design files and options for a reference design used in the DDR flow.

This command is used while running a DDR goal where both reference and implement designs need to be loaded together.

relax_hdl_parsing

Usage: `set_option relax_hdl_parsing <yes | no>`

Set this command to yes to enable SpyGlass perform VHDL semantic checking in the following manner:

1. By automatically inferring missing 'library' clauses for user-defined libraries, provided correct library mapping is specified.

   Consider the following example given in two steps:

   Step 1:

   Compile the following code into a user-library, `userlib1`:

   ```
   entity an2 is
     port (
       A  : in  bit;
       B  : in  bit;
       Y  : out bit);
   end an2;

   architecture behav of an2 is
   begin
     Y <= A and B;
   end behav;
   ```

   Step 2:

   Compile the following code using the user-defined library, `userlib1`, and set the `relax_hdl_parsing` command to `yes`:

```
--library userlib1; -- if -relax_hdl_parsing switch --is
used this declaration of library is not needed.

entity top is
  port (
    topA : in  bit;
    topB : in  bit;
    topZ : out bit);
end top;

architecture struct of top is
  component an2
  port (
    A  : in  bit;
    B  : in  bit;
    Y  : out bit);
  end component;

for I1 : an2 use entity userlib1.an2(behav);
  begin
  I1 : an2
    port map (topA,
      topB,
      topZ);
end struct;
```

If you do not set the `relax_hdl_parsing` command to `yes` in the above case, SpyGlass reports the STX_VH_11 error.

2. Relaxes STX_VH_455 (OTHERS must be the only choice in aggregate of non-locally static size)

   For example:

```
entity E is
end;
architecture A of E is
  function ff(a: bit_vector) return integer is
    type mytype is array(a'range) of bit;
    variable j: mytype;
    begin
```

```
          j := (4=>'1', 5=>'0', others=>'1');
          return a'length;
        end;
      begin
    end A;
```

remove_file

Usage:

```
remove_file -type <file-type>
remove_file -type sgdc [<file-list>]
```

For more information on the Tcl-based usage of the *remove_file* command, refer to the *remove_file* section of the *SpyGlass Tcl Shell Interface User Guide*.

Use this command to remove the files that have been added using the *read_file* command. Except for the SGDC type of file, the remove_file command removes all the files of a given type from the project, and it is not context-sensitive.

Following table lists the file types, which you can specify to remove:

Type	Description
hdl	Removes all HDL files (Verilog, VHDL, or DEF). **NOTE**: You cannot remove individual HDL file types, such as Verilog or VHDL files. The complete HDL file set is removed.
gateslib	Removes gateslib files
sglib	Removes sglib files
lef	Removes LEF files
plib	Removes plib files
sgdc	Removes all or given SGDC files
waiver	Removes waiver files

remove_work

Usage:

```
set_option remove_work <yes | no>
```

Use this command to delete contents of the WORK directory (based on the `work` command) and recompiles all design units in the design.

By default, SpyGlass creates the WORK directory if it does not exist. If the specified directory already exists, SpyGlass recompiles the required design units (all or some) in this directory.

When you set the `remove_work` command to `yes`, SpyGlass removes all contents of the WORK directory and recompiles all design units even if some of the existing precompiled design units did not need re-compilation.

When you save a design view by using the `enable_save_restore` command and set the `remove_work` command to `yes`, you would need to set the `remove_work` command to `yes` during design restore also. However, this command is ignored during the full restore mode (all rules are of Rule Type 1 as described in the *Design Save and Restore Feature* section).

replace_rootdir_for_waivers

Usage: `set_option replace_rootdir_for_waivers <path> <string>`

Use this command to make the waivers generated from the SpyGlass GUI position independent.

Use this command to make the path of the -file, -file_line, -file_lineblock, -msg, and -import fields in the waiver command configurable.

Example 1

Consider the following command:

```
set_option replace_rootdir_for_waivers "/nfs/site/disks/
xhdk74.icl.drops.001" "\$env(MODEL_ROOT)"
```

In the above example, the /nfs/site/disks/xhdk74.icl.drops.001 path is replaced as `$env(MODEL_ROOT)` at the waiver command.

Example 2

Consider the following command:

```
set_option replace_rootdir_for_waivers $env(MODEL_ROOT)
"\$env(${DUT}_MODEL_ROOT)"
```

In the above example, `$env(MODEL_ROOT)` is replaced as `$env(USB_MODEL_ROOT)` at the waiver command.

NOTE: *Waiver command modification happens at save_waiver stage. The modified waive* *commands are saved into awl files.*

report

Usage:

```
set_option report <report-name>
set_goal_option report <report-name>
```

For details on this command, see *Reports Name*.

report_adjustment_waiver

Usage: `set_option report_adjustment_waiver <yes | no>`

Set this command to `yes` to print waived messages in the Waiver report even when the `-ignore` argument of the `waive` constraint is specified.

By default, only the waived message count is printed in the Waiver report when the `-ignore` argument of the `waive` constraint is specified.

These waived messages are printed in the *Adjustments Waiver Report* section of the Waiver report.

reportfile

Usage:

```
set_option reportfile <name>
set_goal_option reportfile <name>
```

For details on this command, see *Report File*.

report_incr_messages

Usage: `set_option report_incr_messages <yes | no>`

Use this command to enable reporting of incremental messages so that you can compare results of a previously run goal with the current goal.

report_inst_backref

Usage: `set_option report_inst_backref <yes | no>`

Use this command to print back-reference information, such as file name and line number containing definition for a design and its instances in the elab_summary.rpt report.

report_ip_waiver

Usage:

```
set_option report_ip_waiver <yes | no>
set_goal_option report_ip_waiver <yes | no>
```

Use this command to print the waived messages in the waiver report when the `-ip` argument of the `waive` constraint is specified. By default, only the count of waived messages is reported in the waiver report.

These waived messages are printed in the *IP/Legacy Waiver Report* section of the waiver report.

report_max_inst

Usage:

```
set_option report_max_inst <value>
```

Use this command to specify a maximum number of instances to be displayed for each design unit in the elab_summary.rpt report.

By default, a maximum of five instances per design unit are displayed in this report. Specify a positive integer value to display the required number of maximum instances per design unit.

For example, if you set `report_max_inst` to `2`, the elab_summary.rpt report shows two instances for each design unit.

If you want to view all instances for each design unit in this report, set `report_max_inst` to `-1`.

To specify a negative value, enclose it inside the curly braces as shown in the following example:

```
set_option report_max_inst { -4 }
```

If you specify a negative value other than `-1`, SpyGlass displays a maximum of five instances per design unit. In addition, a warning message appears in the report to indicate that an incorrect value is specified for this command.

report_max_size

Usage:	`set_option report_max_size <value>`
	`set_goal_option report_max_size <value>`

For details on this command, see *Reports Max Count Size*.

report_per_policy

Usage:	`set_option report_per_policy <yes	no>`

Use this command to split the report specified by the *report* command based on products. In this case, a separate report file (<report-name>_<product-name>.rpt) is generated for each product used in the current run.

The following SpyGlass reports can generate individual report for individual products:

simple	moresimple	inline	waiver	count	summary

Example of using the report_per_policy Command

Consider a scenario in which you are using the SpyGlass lint solution, SpyGlass STARC solution, and a custom product.

Now consider that you specify the following project file commands:

```
set_option report { "simple" }
set_option report_per_policy yes
```

In this case, the following report files are generated:

simple_lint.rpt	simple_starc.rpt	simple_custom.rpt	simple_spyglass.rpt

Here, the first three files are the reports specific to SpyGlass lint solution, SpyGlass STARC solution, and custom product. The last file is the report for built-in messages.

You can use the *reportfile* command to change the name of the generated report files. For example, consider the following project file commands:

```
set_option report { "simple" }
set_option report_per_policy yes
set_option reportfile mysimple
```

In this case, the following report files are generated:

mysimple_lint.rpt	mysimple_starc.rpt	mysimple_custom.rpt
mysimple_spyglass.rpt		

NOTE: *The* `report_per_policy` *command has no impact on product-specific reports.*

Special Case: Inline Report

By default, the inline report generates the following files and directories in the current working directory:

- The inline.rpt file, which is a concatenation of all design files with messages embedded between HDL lines.
- The inline directory that contains individual inline reports (.rpt files) for each design file.

  For example, if you have specified the test1.v and test2.v source design files, the inline directory contains the test1.v.rpt file (that has test1.v contents with embedded messages) and the test2.v.rpt file (that has test2.v contents with embedded messages).

Consider the following project file commands when you are using the SpyGlass lint solution and SpyGlass STARC solution:

```
set_option report { "inline" }
set_option report_per_policy yes
```

In this case, the following files and directories are generated in the current working directory:

File/Directory	Description
inline_lint.rpt	Contains a concatenation of all design files with messages of SpyGlass lint solution embedded between HDL lines
inline_lint directory	Contains individual inline reports (.rpt files) for each design file with messages of SpyGlass lint solution
inline_starc.rpt	Contains a concatenation of all design files with messages of SpyGlass STARC solution embedded between HDL lines
inline_starc directory	Contains individual inline reports (.rpt files) for each design file with messages of SpyGlass STARC solution
inline_spyglass.rpt	Contains a concatenation of all design files with SpyGlass built-in messages embedded between HDL lines

report_style

Usage:

```
set_option report_max_size <value>
set_goal_option report_max_size <value>
```

For details on this command, see *Report Style*.

report_unreachable_default_case

Usage:

```
set_option report_unreachable_default_case <true|false>
```

The default value of this option is true.

Use this option to suppress the SYNTH_5039 messages reported by SpyGlass Synthesis for scenarios in which case-conditions are complete and case-default statement is unreachable.

retain_moresimple_in_turbo

Usage:

```
set_option retain_moresimple_in_turbo <yes | no>
```

Set this option to `yes` in turbo mode to generate the moresimple report in the same format as generated in the non-turbo mode.

In turbo mode, the moresimple report contains additional column for the secondary violation count. If the `retain_moresimple_in_turbo` option is set to `yes`, information about the secondary violation count is not included in the moresimple report.

By default, the option is set to `no` and the moresimple report generated in turbo mode includes information about the secondary violation count.

resetall

Usage: `set_option resetall <yes | no>`

Use this command to reset the Verilog compiler directive `default_nettype` to language default, which is `wire`.

Currently, other Verilog compiler directives are not reset by this option.

It is useful while analyzing multiple design files where the user does not want to specify this default in each of these files.

NOTE: *This option is used in Verilog and Mixed mode and is ignored when used in VHDL mode.*

rules

Usage: `set_goal_option rules <rule-names>`

Use this command to specify a space-separated list of rules or rule groups to run during SpyGlass analysis.

NOTE: *Rule names are case sensitive.*

NOTE: *If you try to specify a rule that is not included in the product you are using, SpyGlass reports the 'rule not registered' error at runtime.*

NOTE: *If you specify both the* `ignorerules` *or* `rules` *commands, the precedence of a command is controlled by the* `RULE_SELECTION_ON_CL_POSITION` *variable. For details, see Effect of the RULE_SELECTION_ON_CL_POSITION*

Command.

The rules command option overrides all product rule parameter that enable or disable specific rules. For example, the W546 rule of SpyGlass lint solution is controlled by the verilint_compat rule parameter. However, the W546 rule will run when it is specified with the rules command (directly or indirectly) irrespective of the verilint_compat rule parameter (whether specified or not). Such rule parameters are effective only when you run a complete product without using the rules command.

NOTE: *Use of rule parameters may affect the functional behavior of some rules and may determine if those rules should be run or not run. Refer to the corresponding product documentation for details on the rule parameters and the effect of the parameters on some rules.*

Examples of using the rules Command

Following are the examples:

- The following example specifies the R1 rule to run during SpyGlass analysis:

  ```
  set_goal_option rules R1
  ```

- The following example specifies the R1 and R2 rules to run during SpyGlass analysis:

  ```
  set_goal_option rules {"R1 R2"}
  ```

- The following example specifies the grp1 rule group to run all rules of this group during SpyGlass analysis:

  ```
  set_goal_option rules grp1
  ```

sca_on_net

Usage: `set_option sca_on_net <0|1>`

Use this option to apply set_case_analysis constraints on a net when the port/pin name matches the net name. This impacts the logic cone inside the hierarchy, which is driven by the net connected to pin/port.

By default, the value of this command is set to 0. In this case, the set_case_analysis constraint is applied on port/pin.

To apply the set_case_analysis constraint on a net, set the value of the command to 1.

For example, consider the following design:

```
module Bottom(input x, output c);
….
endmodule
module top (input a,b, output c);
Wire x; // net name: x
Bottom I1 (.x(x), .y (c)); // pin name: x
……
endmodule
```

In the above example, when you specify set_case_analysis constraint on `I1.x`, it is applied on the `I1.x` pin.

To apply set_case_analysis on net x, use the following command:

```
set_option sca_on_net 1
```

savepolicy/savepolicies

Usage:	`set_option savepolicy <product-name>`
	`set_option savepolicies <products-list>`

Use these commands to specify a product (`savepolicy`) or a list of products (`savepolicies`) that are not run during design save but can be run during design restore.

Under Design Save-Restore feature, you specify the products during the design save and then run any subset of these products during design restore. Entire base polices and the four advanced products, that is, SpyGlass Constraints solution, SpyGlass DFT solution, SpyGlass Power Verify solution, and SpyGlass TXV solution, are saved by default.

In addition, the design view related to the products being used, which are specified indirectly through goals is also saved. You can further specify additional products to be saved during the design save by using the `savepolicy/savepolicies` command so that these products are not run during design save but can be run during design restore.

If save/restore is enabled then the synthesis mode for the run is determined by the combination of products specified in the *<goal>.spq* file and the products specified using the set_option savepolicy <policy_list>. If the synthesis mode is conflicting based on this product set, and the rules enabled in the current run, then spyglass FATALs out. For more information, see *Goal Run Synthesis Flavor*.

For more details on Design Save-Restore feature, see *Design Save and Restore Feature*.

NOTE: *The* `savepolicies` *command works only when the* `enable_save_restore` *command is also specified.*

NOTE: *The* `savepolicies` *command is ignored during the design restore mode.*

Keywords supported by savepolicy/savepolicies

You can specify the following keywords with `savepolicy/savepolicies`:

Keyword	Products Covered by the Corresponding Keyword			
basepolicy	timing	starcad-21	starc2005	starc2002
	starc	simulation	openmore	morelint
	miscellaneous	lint	latch	erc
	area	Audits		
advpolicy	txv	lowpower	power_est	dft
	dft_dsm	const_inter n1	constraints	clock-reset
all	Union of basepolicy and advpolicy			

To check for the product names to be used as a keyword for this command, refer to product name directory in your installation directory.

For example, to run the lint product during design restore, specify the lint keyword as mentioned in the <your-inst-dir>/SPYGLASS_HOME/policy/lint directory, with the `set_option savepolicy` command.

Also, consider the following examples:

```
set_option savepolicy dft
```

```
set_option savepolicy {basepolicy dft}
```

The first example listed above runs the SpyGlass DFT product during design restore. In the second example, the SpyGlass DFT product and all the products covered under the basepolicy keyword are run during design restore.

If you do not want to save a product during design save, use the *nosavepolicy/nosavepolicies* command.

sdc2sgdc

Usage:
```
set_option sdc2sgdc <yes | no>
set_goal_option sdc2sgdc <yes | no>
```

The default value of the `sdc2sgdc` option is `no`. However, for `sg_shell` runs, SpyGlass automatically sets this option to yes, when at least one SDC file is specified using the `sdc_data` constraint.

For details on this command, see *Enable SDC-to-SGDC translation*.

sdc2sgdcfile

Usage:
```
set_option sdc2sgdcfile <file-name>
set_goal_option sdc2sgdcfile <file-name>
```

For details on this command, see *Specify the file to save output of SDC-to-SGDC translation*.

sdc2sgdc_mode

Usage:
```
set_option sdc2sgdc_mode <mode-name>
set_goal_option sdc2sgdc_mode <mode-name>
```

For details on this command, see *Specify the mode of the SDC file to be translated to SGDC*.

sfcu

Usage: `set_option sfcu <yes | no>`

Use this command to enable each file to be compiled as a separate compilation unit.

By default, SpyGlass compiles all the specified files in a single compilation unit. Set the `sfcu` command to yes to compile each file as a separate compilation unit.

NOTE: *The* `sfcu` *command can be used only when the* `enableSV` *command is set to* `yes`*.*

sgdc_check_severity

Usage:
```
set_option sgdc_check_severity <COMPAT | DEFAULT | ERROR |
FATAL | WARNING | IGNORE>
```

Use this command to modify the severity or to provide severity control for the *checkSGDC_existence* rule (built-in) and other non-built in rules. Using this command, the specified value is applied on all design constraints globally.

Input Value Description

The following table describes the permissible values for this command and the respective functionality that is triggered for each of the values:

Value	Functionality
COMPAT	This is the default value for the sgdc_check_severity command. When this option is specified, the related rule retains the existing severity. You can perform severity overload for non-built in rules, using this option.
FATAL	Generates violation message with the FATAL severity for the first SGDC command that fails the sanity check.

Value	Functionality
ERROR	Generates violation message with the ERROR severity for any SGDC command that fails the sanity check. The software then ignores or deletes the SGDC command.
WARNING	Generates violation message with the WARNING severity for any SGDC command that fails the sanity check. The software then ignores or deletes the SGDC command.
IGNORE	Ignores and deletes any SGDC command that fails the sanity check.
DEFAULT	Uses the registered severity for the generated violation message. You can not perform severity overload for non-built in rules, using this option

Examples

The following lists some of the examples of using the *sgdc_check_severity* command.

```
sg_shell> set_option sgdc_check_severity {"error"}
sg_shell> set_option sgdc_check_severity {"warning"}
sg_shell> set_option sgdc_check_severity {"compat"}
```

Pragmas to define sgdc severity class

You can also specify individual severity class for one or more constraints, thereby allowing more granularity and block-level control.

The following describes the syntax for specifying sgdc severity class on a block-level:

```
define_sgdc_severity_class -value <option-value>
    constraint <constraint_name>
end_sgdc_severity_class
```

Here, define_sgdc_severity_class -value <option-value> is the pragma for starting a sgdc severity class. Also, end_sgdc_severity_class denotes the pragma for ending an sgdc severity class.

Consider the following sample SGDC file, test.sgdc:

```
sg_shell> cat test.sgdc
current_design test
```

```
constraint 1
define_sgdc_severity_class -value error
    constraint 2
    constraint 3
end_sgdc_severity_class
constraint 4
```

In the above example, constraint 2 and 3 have SGDC existence checks failure reported with severity error. While constraint 1 and constraint 4 either follow the global option value, if specified, or they use the default severity registered with Spyglass.

sgdc_validate

Usage:

```
set_option sgdc_validate <true | false>
```

Use this command to enable the SpyGlass CDC validation flow.

sgsyn_clock_gating

Usage:

```
set_option sgsyn_clock_gating <1 | 0>
set_goal_option sgsyn_clock_gating <1 | 0>
```

Specifies whether clock gating should be enabled or disabled.

By default, value of the *sgsyn_clock_gating* command is set to 0, and clock gating is disabled.

Set the value to 1 to enable clock gating. When you set its value to 1, inference of clock gating for flip-flops with explicit enables or feedback loops, is enabled.

After you enable clock gating, use the *sgsyn_clock_gating_threshold* command to set the threshold for the insertion of clock gate logic.

sgsyn_clock_gating_effort

Usage:
```
set_option sgsyn_clock_gating_effort <low | medium | high>
set_goal_option sgsyn_clock_gating_effort <low | medium |
high>
```

Specifies the effort for clock gating computation.

By default, the `sgsyn_clock_gating_effort` option is set to low, and it uses simple heuristics for finding clock gating candidates. Set the value to either medium or high to adjust the heuristic complexity accordingly. This can lead to more clock gating opportunities but may come at the expense of higher runtime and/or memory consumption.

sgsyn_clock_gating_threshold

Usage:
```
set_option sgsyn_clock_gating_threshold <num>
set_goal_option sgsyn_clock_gating_threshold <num>
```

Specifies the threshold of the clock gating logic.

When you set this command, the *sgsyn_clock_gating* command is automatically set to yes.

By default, the *sgsyn_clock_gating_threshold* command is not set and insertion of clock gating logic is disabled. Set the value of the *sgsyn_clock_gating_threshold* option to zero or a positive integer value. This enables clock gating for registers with width greater than or equal to the specified value.

sgsyn_enable_latch_removal

Usage:
```
set_option sgsyn_enable_latch_removal <yes | no>
```

Use this command for better latch detection and removal of redundant latches.

When you set this command to `yes`, SpyGlass checks if the enable pin of a latch is driven by VCC. If yes, SpyGlass replaces such a latch with a buffer.

NOTE: *This optimization results in an increased runtime for synthesis.*

sgsyn_enable_mux_2_latch

Usage:
```
set_option sgsyn_enable_mux_2_latch <value>
<list_of_modules>
```

Use this command to convert a mux with feedback loop into a latch for a given list of modules. The `<value>` field takes any of the following values:

- 0: No change
- 1: Two input mux with output tied to one of the inputs is converted to latch.
- 2: NxN mux with output tied to one or more of the inputs is converted to latch. Note that setting the value to 2 also enables the behavior listed with value 1.

Specify the list of modules in the `<list_of_modules>` field for which the option will be applicable. An empty list of module indicates that the option will not be applied on any module.

sgsyn_loop_limit

Usage:
```
set_option sgsyn_loop_limit <value>
```

Use this command to specify a loop-rolling limit during design synthesis.

By default, the loop-unrolling limit is set to 2048. When a loop is not completely rolled in the specified number of iterations, SpyGlass marks the module as un-synthesizable.

sgsyn_ungroup_delimiter

Usage: `set_option sgsyn_ungroup_delimiter <value>`

Use this command to override the default delimiter character while un-grouping. The default un-grouping character is forward slash '/'. For more information on un-grouping the instances, see ungroup_cells in the *Consolidated Constraints Application Note.*

Specify a character value as an input to this command.

For example, consider the hierarchy, top.u1.u2 and that the instance, u1 is un-grouped. In this case, the default name of the instance would be '\u1/u2 '.

Now, when you specify the following command:

```
set_option sgsyn_ungroup_delimiter ';'
```

In this case, the instance name would be '\u1;u2 '.

show_all_sdc_violations

Usage: `set_parameter  show_all_sdc_violations  <yes | no>`

Use this parameter to show all the sdc_violations during the SDC parsing.

By default, the value of the show_all_sdc_violations parameter is set to no. During SDC parsing, when the parameter is set to no, SpyGlass reports violations for only those products, which you have selected to run.

During sdc2sgdc translation, the sgdc constraints for only those commands are generated, which are converted used sdc2sgdc translation. Syntax error for other commands is not reported.

Use the show_all_sdc_violations parameter to view violation messages for such the sdc commands.

Set the value of this parameter to yes to view all the SDC violations during the SDC parsing, irrespective of the products selected to run.

show_lib

Usage:	`set_option show_lib <yes	no>`

Use this command to generate messages for each library module (from libraries specified using the `v` and `y` commands) as it is loaded.

By default, these messages are not generated.

show_sdc_progress

Usage:	`set_parameter show_sdc_progress <yes	no>`

Set this command to yes to show the SDC parsing progress bar on standard output during SDC parsing.

NOTE: *The show_sdc_progress parameter will be deprecated in a future release.*

save_single_line_swl_waivers

Usage:	`set_option save_single_line_swl_waivers <yes	no>`

Use this command to save swl waivers as single line waive commands. By default, the saved swl commands are formatted into multiple lines.

skip_methodology_path

Usage:	`set_parameter skip_methodology_path <yes	no>`

Use this command to ignore checking the methodology path. This enables the Waiver Editor window to display all waivers.

skip_rules_for_fast_restore

Usage: `set_option skip_rules_for_fast_restore <yes | no>`

Use this command to skip checking of rules that require design re-parsing and/or re-synthesis during design restore. For details, see *Design Save and Restore Feature*.

Depending on the selected characteristics of rules, SpyGlass may not be able to work with saved design view and therefore, unchanged design re-parsing and/or re-synthesis may be required. You can skip such rules by setting the `skip_rules_for_fast_restore` command to `yes`.

Example of using the skip_rules_for_fast_restore Command

Consider an example in which you run the *Initial_RTL/Ensure_RTL_Block_is_simulation_ready/Connectivity* goal, and save the design by setting the `enable_save_restore` command to `yes`.

Now, in the second SpyGlass run, if you specify the same commands as in the first run, SpyGlass reports the following message:

```
WARNING [237]    Following rules cannot run on restored
design database. Hence, HDL being re-read:
W110
STARC05-2.1.3.1
```

In this example, SpyGlass re-reads the design, runs the `W110` and `STARC-2.1.3.1` rules, and restores the netlist from the disk.

In the third SpyGlass run, if you specify the same commands as in the previous run and set the `skip_rules_for_fast_restore` command to `yes`, SpyGlass reports the following message:

```
WARNING [236]    Following rules not being run (design
database restored & 'skip_rules_for_fast_restore' is set):
W110 (need HDL re-read)
STARC05-2.1.3.1 (need HDL re-read)
```

In this case, SpyGlass disables the `W110` and `STARC-2.1.3.1` rules, skips the design parsing, and loads the netlist straight from the disk.

NOTE: *The* `skip_rules_for_fast_restore` *command works only when the enable_save_restore command is set to* `yes`*.*

skip_sanity_on_blocks_nonhier_nets

Usage: `set_option skip_sanity_on_blocks_nonhier_nets <0 | 1>`

Use this command to skip sanity checks on the constraints specified for the non hierarchical nets of an abstract block.

After block abstraction, such nets become non existent. So when you set this command to 1, SpyGlass marks the constraints for such nets as deleted.

sort

Usage:

```
set_option sort <yes | no>
set_option sortmethod <du | lexical>
```

For details on this command, see *Automatically Sort VHDL File(s)*.

sortrule

Usage:

```
set_option sortrule <language>+<rule-name>+<sortorder>
set_goal_option sortrule <language>+<rule-name>+<sortorder>
```

Use this command to specify a sort order for messages in SpyGlass reports.

For details on using this command, see *Customizing the Sorting Order*.

stop

Usage:

```
set_option stop <module-name>
```

For details on this command, see *Stop Design Unit(s)*.

stopdir

Usage:

```
set_option stopdir <directory-name>
```

Use this command to specify a directory so that SpyGlass skips rule-checking on all design units present in source files present in this directory. Such design units are considered as grey boxes during SpyGlass analysis.

For details, see *Managing the Design Hierarchy*.

When this command is run, SpyGlass generates the *stop_summary report*.

NOTE: *If your directory name includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your directory name is "abc*d", you need to refer to it as {{abc*d}}.*
*However, if you want to refer to two directories, for example "abc1d" and "abc2d", you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}} in this case. For details on pattern matching support, see Pattern Matching Across Features.*

NOTE: *You can specify the* `stopdir` *command with the stop and the stopfile commands.*

NOTE: *The* `stopdir` *command works recursively in the specified directory. Therefore, the following example stops all design units in all source files in the directory,* `mydir`*, and all files in all sub-directories (all levels) under the* `mydir` *directory:*

```
set_option stopdir mydir
```

Wildcard Support

You can specify the directory/file names with the `stopdir` command by using expressions, as shown in the examples in the following table:

`set_option stopdir "dir1/*"`	Stop all files in directory `dir1` and all files in sub-directories (all levels) under directory `dir1` recursively (as in UNIX Shell expansion)
`set_option stopdir "dir1/*/"`	Stop all files in sub-directories (all levels) under directory `dir1` recursively (as in UNIX Shell expansion)

For more information, see *Pattern Matching Across Features*.

stopfile

Usage:	`set_option stopfile <file-name>`

Use this command to specify a file name so that SpyGlass skips rule-checking on all design units specified in that file. Such design units are considered as grey boxes during SpyGlass analysis.

For details, see *Managing the Design Hierarchy*.

When this command is run, SpyGlass generates the *stop_summary report*.

NOTE: *If your filename includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your filename is "abc*d", you need to refer to it as {{abc*d}}.*
*However, if you want to refer to two files, for example "abc1d" and "abc2d", you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}} in this case. For details on pattern matching support, see Pattern Matching Across Features.*

NOTE: *You can specify the* `stopfile` *command with the stop and the stopdir commands.*

Wildcard Support

You can specify the filenames with the stopfile option using SpyGlass Pattern Matching Support, as shown in the examples in the following table:

`set_option stopfile “a*”`	Stop all files in the current directory whose names match the wildcard expression `a*` (for example, a1, aa1 , abc, etc.)
`set_option stopfile “dir1/*”`	Stop all files in directory dir1
`set_option stopfile “dir?/*”`	Stop all files in directories that match the wildcard expression `dir?` (for example, dir1, dir2 etc.)

For more information, see *Pattern Matching Across Features*.

support_sdc_style_escaped_name

Usage:

```
set_option support_sdc_style_escaped_name <yes | no>
set_goal_option support_sdc_style_escaped_name <yes |
no>
```

Set this command to `yes` to enable SpyGlass recognize Synopsys-style escaped names in SGDC files.

By default, SpyGlass recognizes names that start with a backslash character and end with a space (Verilog) or a backslash character (VHDL) as escaped names as per respective HDL conventions. Therefore, you need to specify such escaped names in the set format in SGDC also.

When you set this command to `yes`, you can specify object names without escaped delimiters in SGDC files. For example, you can specify `'ab%c'` instead of `'\ab%c '` or `'\ab%c\'`.

You can specify Synopsys-style escaped names at all levels in a hierarchical name. For example, you can specify `'top.e.&f%g'` instead of `'\top .\e.&f%g '` or `'\top\.\e.&f%g\'`.

You can use both the default format escaped names and the Synopsys-style escaped names in the same SGDC file.

target

Usage:

```
set_option target <lib-name-list>
set_goal_option target <lib-name-list>
```

Use this command to specify libraries to be used for technology-mapping out of the specified .sglib libraries.

By default, all the specified .sglib libraries are used for technology-mapping.

top

Usage: `set_option top <module-name>`

For details on this command, see *Top Level Design Unit*.

top_instance

Usage: `set_option top_instance <instance_name>`

Use this option along with the *dummy_top* option to change the synthesis top.

The *top_instance* option accepts the name of the instance of the top module that is instantiated by using the *dummy_top* option. The *top_instance* option is honored only if the top is specified. If the top is not specified, SpyGlass ignores the option and reports the following Warnings in the spyglass.log file:

- If no top is specified:

  ```
  Warning : Ignoring options 'dummy_top' and 'top_instance' as
  option top is not specified
  ```

- If multiple tops are specified:

  ```
  Warning : Ignoring options 'dummy_top' and 'top_instance' as
  multiple top specified
  ```

- If either of the *dummy_top* or the *top_instance* options are not specified, one of the following messages, as appropriate is reported:

  ```
  Warning : Ignoring option 'dummy_top' as option
  'top_instance' is not specified
  ```

  ```
  Warning : Ignoring option 'top_instance' as option
  'dummy_top' is not specified
  ```

  ```
  Warning : Ignoring option 'dummy_top' as options 'top' and
  'top_instance' are not specified
  ```

  ```
  Warning : Ignoring option 'top_instance' as options 'top' and
  'dummy_top' are not specified
  ```

- If the specified `dummy top` is not present in the design:

  ```
  Warning : Dummy Top specified in option dummy_top not found
  in the design
  ```

- If the specified `top` or *dummy_top* is config:

  ```
  Warning : Dummy top not supported if top or dummy top is
  config
  ```

treat_priority_pin_as_obs

Usage: `set_option treat_priority_pin_as_obs <yes | no>`

Use this command to consider the highest priority asynchronous pin of sequential block as observable/unblocked.

By default, the value of this command is set to no. In this case, under certain conditions, the highest priority asynchronous pin may be marked as un-observable/blocked.

For example, in a flip-flop having highest priority -`clear` pin, when -`data` pin is constrained to value 0, Currently the -`clear` pin is returned as un-observable/blocked in default flow.

To treat highest priority asynchronous pin as observable/un-blocked unconditionally, set the value of the treat_priority_pin_as_obs command to yes.

treat_rtl_macro_as_lib_cell

Usage: `set_option treat_rtl_macro_as_lib_cell <yes | no>`

Use this command to treat all the macros in the design as lib cells.

The default value of this command is no.

Usually, there are size limitations while evaluating a macro, for example, max data size = 1024 (2^10). Therefore, evaluation does not take place completely for a large macro used in the design because it exceeds the size limits specified.

However, when you set the value of the *treat_rtl_macro_as_lib_cell* command to yes, all the macros in the design are treated as lib cells. Therefore, evaluation takes place completely for those large macros.

unify_sdc2sgdc

Usage:

```
set_option unify_sdc2sgdc <1 | 0>
set_goal_option unify_sdc2sgdc <1 | 0>
```

Use this command to enable unification of mutually exclusive information from different sources, that is, SDC and SGDC.

NOTE: *In 5.2, unification is done only for the clock sgdc constraint.*

In this flow, SpyGlass first reads the user-specified SGDC constraints and then reads the sdc2sgdc constraints. When you specify the unify_sdc2sgdc option, SpyGlass creates a single unified constraint by combining the information (options) derived from user-specified SGDC constraints and the SGDC constraints that are generated during translation of SDC to SGDC using the *sdc2sgdc* command.

use_block_interface

Usage: `set_option use_block_interface <1 | 0 | yes | no>`

Use this option to trigger the software flow wherein the block interface information in the abstract model specified using the *include_block_interface* is processed. The abstract model can be either be encrypted or in the form of an SGDC.

If both encrypted and interface information is available through SGDC, Spyglass picks the encrypted part and ignores the abstract interface SGDCs.

use_du_sch_hier

Usage:

```
set_option use_du_sch_hier <yes | no>
set_goal_option use_du_sch_hier <yes | no>
```

Set this command to `yes` to:

- Enable use of hierarchical information while applying IP waivers. Consider a design where a module is instantiated in multiple IPs, but `waive -ip` command is not specified for all the IPs. In this case, if you use the `use_du_sch_hier` command, the violation messages for the specified modules are reported.
- Enable use of schematic information to waive violations on a design unit, using the `waive -du` command. By default, only back-reference information of a design unit is considered while waiving a violation using the `waive -du` command.

By default, SpyGlass waives violations on only those module instances that are present in the IPs specified using the `waive -ip` specification.

use_generate_index_style

Usage: `set_option use_generate_index_style <>`

Use this command to specify the format to generate the names of nets/ instances inside for-generate statements. You can specify a string or multiple strings in the following format, as an input to this command:

```
set_option use_generate_index_style "X%sY%dZ"
```

The following are the arguments for the above command:

- **X**: Represents the start delimiter. This is an optional argument.
- **Y**: Represents the generate index separator. This is an optional argument.
- **Z**: End delimiter. This is an optional argument.
- **%s**: Mapped to for-generate block label name. The input string must have 0 or 1 occurrence of `%s`. This means that the for-generate block label is not used in the name.
- **%d**: Mapped to for-generate bit index. The input string must have exactly 1 occurrence of `%d`.

Default value of this command is `%s[%d]`. This means that the if nothing as specified as an input to this command, then this format is used to generate names.

This command doesn't affect instances or nets inside the `if-generate` and `case-generate` statement. It is only used for naming of `for-generate` objects.

use_generate_separator

Usage: `set_option use_generate_separator <separator string>`

Use this command to specify the separator string to be used between block labels to generate the names of instance/net inside the *generate* statement. The specified string acts as a separator between block and nested blocks and block and nested instance/net.

For example, consider the following command:

```
set_option use_generate_seperator "@^$#&"
```

The default value of this command is '.'.This means, if no value is specified for the command, then '.' is used as generate separator. You can specify single character or a string of multiple characters as an input to this command.

use_goal_rule_sort

Usage:	`set_option use_goal_rule_sort <yes | no>`
	`set_goal_option use_goal_rule_sort <yes | no>`

Use this command to sort violation messages in the moresimple report based on the order of rules specified in a goal file.

use_scan_flops

Usage:

```
set_option use_scan_flops <yes | no>
set_goal_option use_scan_flops <yes | no>
```

Use this command to enable SpyGlass to pick scan flip-flops from a technology library for mapping flops in a design.

When you set this command to `yes`, SpyGlass gives priority to scan flip-flops rather than normal flip-flops in a technology library for mapping. If scan cells are not present in the technology library, SpyGlass selects normal flip-flops for mapping in the design.

The `use_scan_flops` command can be specified only when the *-est_mode* command is used or EST mode compatible rules are specified.

use_hier_sep_slash

Usage:

```
set_parameter use_hier_sep_slash <yes | no>
```

Use this parameter to change the default hierarchy separator of the SGDC constraints.

By default, the hierarchy separator used in SGDC constraint’s name field is '.'

For details on this command, see *Changing the Default Hierarchy Separator of the SDC2SGDC Constraints*.

v

Usage:

```
set_option v <file-names>
```

For details on this command, see *Specify the library files in the source design*.

verbosity

Usage: `set_option verbosity <0 | 1 | 2 | 3>`

Use this command to enable you to control the verbosity of the log file thereby ensuring that the spyglass.log file is easily readable and content is controllable.

For more information on this command, see *Define Verbosity Level for Log File*.

w

Usage:

```
set_option w <yes | no>
set_goal_option w <yes | no>
```

Use this command to enable generation of warnings for Perl-level compilation and the SpyGlass checker activities.

While running SpyGlass, if you find problems, such as a rule failing, you can get more information about these problems by using the `w` command and running SpyGlass again.

This command enables SpyGlass to display more details while running, identifying each rule as it is checked. Using this command, you can easily identify the failing rule, since it will be the last rule listed before the error message.

Once you have identified the rule causing the problem, you can exclude it by using the `ignorerules` command. This enables you to continue analyzing your design while the specific rule problem is being fixed.

NOTE: *If for some reason, there is a problem in SpyGlass, it is sometimes difficult to reproduce the problem outside of your company without the design files. These may however be so confidential that it is impossible to make them available, even with a non-disclosure agreement. In order to provide as much debug information as possible without the design files, SpyGlass supports a* `DEBUG` *option. This saves information about what processes SpyGlass was running at the time of the problem, allowing Atrenta development to gain additional insight into the problem. This information is stored by default in a* spyglass.log *file in the working directory, although this can be changed using the* `logfile` *option. You should send this file to Atrenta Customer support when reporting the problem.*

waivers_translate_generate_name

Usage: `set_option waivers_translate_generate_name <yes | no>`

Use this command to enable use of a non-escaped `generate block` name or an `instance array` name in the `-msg` field of the waive command. See *Waiving Messages by Using the waive Constraint*.

work

Usage: `set_option work <work-dir-name>`

For details on this command, see *Logical Working Directory*.

write_sdc

Usage: `set_parameter write_sdc <yes | no>`

Set this command to yes to generate error free constraints in the TCwritesdcInfo file under the <wdir>/spyglass_spysch/spyglass_sdc/ directory.

SpyGlass generates constraints in the TCwritesdcInfo file in the following format:

```
Input SDC command :
set_ideal_network  {A1/in1 in2}
```

```
Output SDC command:
#ideal.sdc@@28@@
set_ideal_network  [list [get_pins {A1/in1}] [get_ports
in2]]
```

These constraints are generated in the following manner:

- All the SDC commands are generated as un-commented and all the non-SDC commands are generated with the `sg_` prefix and are commented.

- Erroneous commands are not translated.
- Basic Tcl commands, such as `puts`, `if`, and `else` are not translated.
- For all objects that are a part of commands, the corresponding `object_access` commands are used with them, such as `get_ports {p1}`.

By default, this command is set to `no`, and the TCwritesdcInfo file is not created.

Please note the following points:

- Although the aim is to draft a legal SDC file that runs error-free in all the tools, SpyGlass is currently not able to stop translation of some erroneous commands, such as SDC_209, SDC_288, etc.
- The set `bus_naming_style/sdc_version` and `setenv` are the only basic Tcl commands that are translated.
- SpyGlass is not able to insert `object_access` commands, `get_clocks`, `get_libs`, `get_lib_pins`, and `get_lib_cells`. They are decompiled as names only.

write_vlog_config_report

Enables reporting of instances, which are bound using the configuration rules.

For more information on this report, see *Support for Verilog Configuration*.

Usage: `set_option write_vlog_config_report <yes|no>`

strict_vlog_config

Use this command to black box and assign the ELAB error to the instances, which are unable to bind, using the verilog configuration rules.

When the value of this command is set to no, the instances that are unable to bind using the verilog configuration flow, are assigned the ELAB warning. However, their binding is done using default spyglass binding flow.

For more information on verilog configuration, see *Support for Verilog Configuration*.

Usage: `set_option strict_vlog_config <yes | no>`

y

Usage: `set_option y { space-separated path-names of directories }`

For details on this command, see *Specify the library directories containing libraries*.

vlog2005_generate_name

Usage: `set_option vlog2005_generate_name <yes | no>`

Set this command to yes to unroll all Verilog generate statements in the Verilog 2005 syntax. Unrolling occurs even if a design is a SystemVerilog design for which the SystemVerilog mode is enabled by setting the *enableSV* command to yes.

This command is needed when you want to unroll all Verilog generate statements in the Verilog 2005 syntax even without SystemVerilog mode.

vlog2001_generate_name

Usage: `set_option vlog2001_generate_name <yes | no>`

Set this command to `yes` to unroll all Verilog generate statements in the Verilog 2001 syntax. Unrolling occurs even if a design is a SystemVerilog design for which the SystemVerilog mode is enabled by setting the *enableSV* command to `yes`.

vlog2005_lrm_naming

Usage: `set_option vlog2005_lrm_naming <yes | no>`

Set this command to yes to unroll all Verilog generate statements in the Verilog 2005 syntax. Unrolling occurs even if a design is a SystemVerilog design for which the SystemVerilog mode is enabled by setting the *enableSV* command to yes.

Options Not Recommended

Following commands are not recommended to be used in a project-based flow:

Option	Description
I	Used to specify the directories that contain additional files.

I

Usage: `set_option I {space separated list of directory-name}`

Use this command to specify directories that contain additional files, such as customized rule-deck files and customized SpyGlass-compatible library files (.spyso files).

SpyGlass first locates the additional files in the directories specified by this command, before searching in default locations, such as, <your-inst-dir>/SPYGLASS_HOME/lib and <your-inst-dir>/SPYGLASS_HOME/policies.

You can specify multiple directories, which are searched in the same order as they are specified.

This command also adds the specified directories to LD_LIBRARY_PATH environment variable that defines the location from where the shared objects are loaded.

NOTE: *The I command is not recommended to be used in the project file. For details, see Options Not Recommended. However, you should specify the directory paths using the -I option on the command-line itself while invoking console or sg_shell. You can view the custom reports by selecting the Report > Default menu option.*

Handling Wildcard Expressions

If your directory name includes wildcard characters (*, or ?), the name should be enclosed in curly braces, and the wildcard characters should be preceded by a backslash (\) to treat them as literal. For example, if your directory name is "abc*d", you need to refer to it as {{abc*d}}. However, if you want to refer to two directories, for example "abc1d" and "abc2d", you can specify them using SpyGlass pattern matching support, that is, you can specify {{abc*d}} in this case. For details on pattern matching support, see *Pattern Matching Across Features*.

Search Mechanism used by the I Command

When you start SpyGlass, it searches the default directory (<your-inst-dir>/SPYGLASS_HOME/policies) to find the available policies, that is, all files with the filename *-policy.pl. Using the I command, you can redefine the path to the policy directory, and have SpyGlass start looking for its files in your custom path first.

Whatever is defined using the I command is prefixed to the already defined default search path. SpyGlass searches in your directory first to find any subsequently referenced files not qualified by full paths. Similarly, it uses your directory as a starting point for relative path references.

Changing the Default File Search Path

To change the path to the directory where SpyGlass begins file searches, enter the I command followed by the path to your directory.

For example, to tell SpyGlass to start the search in the `myperlfiles` directory for a custom policy called, `mypolicy`, and its subsequent file references, specify the following command:

```
set_option I /mypath/myperlfiles
```

You can specify multiple directories with multiple arguments in the I command. Each argument is prefixed to the existing search path. For example, to search mydir1 followed by mydir2 before searching the default search path, specify the following command:

```
set_option I /mydir1 /mydir2
```

Reports Generated in SpyGlass

When SpyGlass analysis is complete, the following types of reports are generated:

- *General Reports*
- *Default Reports*
- *Custom Reports*
- Product-specific reports

These reports are generated based on selected goals. Refer to the respective product documentation for details on product-specific reports.

You can open the required report to view the summary of design issues, or you can redirect these reports in separate files to review them later.

Generating Reports

To generate a report, perform any of the following actions:

- Select the *Tools -> Report* menu option.

  A sub menu appears listing different categories of reports. Select a report from a category. The selected report appears in a new window.

- Select the *Reports* option or the 💾 icon from the *Violations Pane*. A menu appears listing different categories of reports. Select a report from a category. The selected report appears in a new window.
- Specify the report to be generated by using the *report* command in a project file.
- Specify the report to be generated by using the *Reports Name* option under the *Set Read Options* tab.

General Reports

SpyGlass generates the following reports when you specify them by using the *report* command:

count report	*inline report*
moresimple_filesort report	*moresimple_rulesort report*
moresimple_sevclass report	*score report*
sign_off report	*sign_off_sevlablel report*
simple report	*summary report*

count report

The count report lists the number of times SpyGlass found each type of message. It also displays the total number of messages of rules you previously waived or excluded.

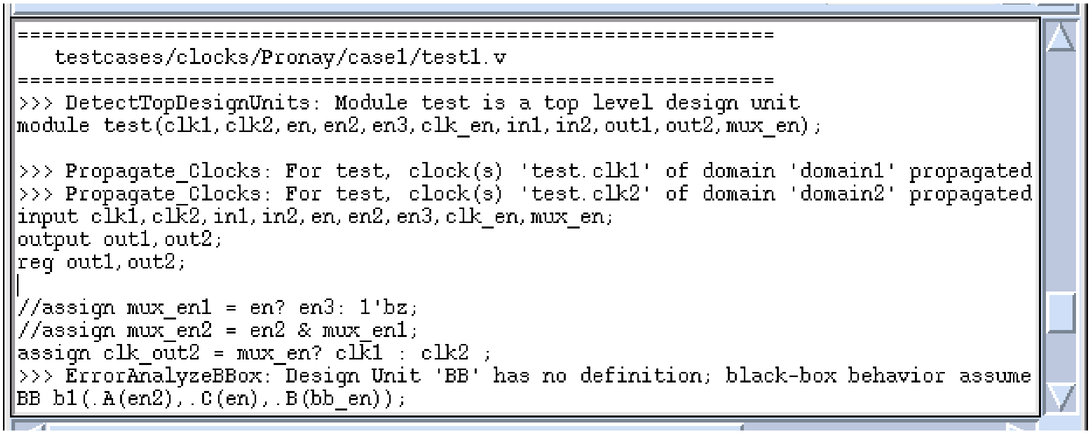
inline report

The inline report displays the contents of the RTL source file annotated with violation messages. These messages are displayed above the violating line in the source code.

To help you spot the messages, SpyGlass prefixes these messages with >>>.

NOTE: *The Inline report is not generated properly when the compressed netlist is used in the design.*

Following figure shows a sample inline report.



```
==============================================================
   testcases/clocks/Pronay/case1/test1.v
==============================================================
>>> DetectTopDesignUnits: Module test is a top level design unit
module test(clk1,clk2,en,en2,en3,clk_en,in1,in2,out1,out2,mux_en);

>>> Propagate_Clocks: For test, clock(s) 'test.clk1' of domain 'domain1' propagated
>>> Propagate_Clocks: For test, clock(s) 'test.clk2' of domain 'domain2' propagated
input clk1,clk2,in1,in2,en,en2,en3,clk_en,mux_en;
output out1,out2;
reg out1,out2;

//assign mux_en1 = en? en3: 1'bz;
//assign mux_en2 = en2 & mux_en1;
assign clk_out2 = mux_en? clk1 : clk2 ;
>>> ErrorAnalyzeBBox: Design Unit 'BB' has no definition; black-box behavior assume
BB b1(.A(en2),.C(en),.B(bb_en));
```

FIGURE 241. Example of inline Report

moresimple_filesort report

The moresimple_filesort report is similar to the moresimple report except that the rule messages are sorted in the following order:

1. File name for a given file
2. Severity
3. Weight
4. Rule
5. Line number

moresimple_rulesort report

The moresimple_rulesort report is similar to the moresimple report except that the rule messages are sorted by the rule name first and for a given rule, by their file and line.

moresimple_sevclass report

The moresimple_sevclass report is similar to the moresimple report with additional information displaying the severity class.

score report

SpyGlass provides a built-in scoring system for code checks. The report generator assigns a weight to each rule message based on the severity of the message. The score report displays the score for a design as the sum of the weights assigned to all the rule messages that are detected during a SpyGlass Analysis run. The overall score is an indicator of the relative quality of your design.

sign_off report

The sign_off report lists summary and detailed information about the SpyGlass analysis run.

This report has the following sections:

- The header section that displays SpyGlass version, products versions, user name, top-level design unit name(s) with file name and line number information, goals used, if any, the current working directory, counts of messages generated, messages waived, if any, messages reported, and messages suppressed, if any.
- The Rule Setup Info section that displays a list of SpyGlass commands with their actual values used in the analysis run passed. The source file details are printed at the end of the list followed by unknown options, if any.
- The Policy Info section has the names, versions, and locations of selected products.
- The Constraints Info section has the details of user-specified SpyGlass Design Constraints.
- The Parameter Info section has the list of user-specified rule parameters with their applied values and origin (command file, goal/session file, or SpyGlass Configuration file).
- The Waiver Info section has the list of applied waivers with waiver comment, if any, waiver expression, and the number of messages waived. This section also contains the spg_backref field. This field is used to specify the back reference information about the waiver

command, that is, the file and line number of the waiver file in which the waiver command was specified.

- The Violation Info section has two sub-sections:
  - ❒ The Summary Status sub-section has severity class, rule name, message count, and rule short description for each rule that has flagged a message. The list is sorted by the severity class.
  - ❒ The Detailed Status sub-section has same details as that of the moresimple report.

sign_off_sevlablel report

The sign_off_sevlabel report is similar to the sign_off report, but the Summary Status sub-section of the Violation Info section displays rule information along with severity label instead of severity class.

simple report

The simple report truncates long names and messages to fit the contents in the report's layout.

This report displays the following information:

- ❒ Name of the rule violated
- ❒ Name of the file where SpyGlass found the message
- ❒ Line number of the code where the corresponding rule violation occurred
- ❒ Severity level of the message
- ❒ Message explaining why SpyGlass logged the message

Note that the severity level and message are contained in the rule definition.

summary report

The Summary report displays a summary list of message counts by each particular rule type along with the severity class and rule short help.

The rules are grouped based on the default grouping criteria. Within each group, the data is first sorted by the severity class and then by rule name.

summary_sevlabel report

The summary_sevlabel report is similar to summary report, but it displays summary list of message counts by each particular rule type along with severity label instead of severity class.

Default Reports

SpyGlass generates the following default reports:

ignore_summary report	*moresimple report*
The moresimple_warning Report	*moresimple_error report*
no_msg_reporting_rules report	*stop_summary report*
elab_summary report	*waiver report*

ignore_summary report

The ignore_summary report lists design units/files that are ignored in the current run.

These design units/files are specified by the *ignoredu* and *ignorefile* project file commands.

Following is the sample ignore_summary report:

```
Design units ignored by ignoredu/ignorefile options
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Object Name     Object Type      Ignore specification
============================================================
mid             MODULE           ignoredu     "mid"
other_bot       MODULE           ignorefile     "src/bot.v"

Files ignored by ignorefile options
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
src/bot.v (ignorefile "src/bot.v")

Unused ignorefile/ignoredu specification(s)
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ignoredu "no_such_dir"
```

moresimple report

The moresimple report is similar to the *simple report*. However, it does not truncate long names and messages.

The moresimple report displays the following information:

- Name of the rule violated
- Alias of the rule

NOTE: *An alias refers to an alternative name. It is a cross-reference to a standards document, such as the lint standard or in-house coding standard or guideline.*

- Severity level of the message
- Name of the file where SpyGlass found the message
- Line number of the code containing the message
- Short help and rule name for each rule group

NOTE: *Set the report_style option in project file to include the short help and rule name for each rule group as shown below:*

```
set_option report_style {group_moresimple_by_rule}
```

- Weight of the message

NOTE: *The weight is a number assigned based on the severity of the rule message. That is, higher the severity of the message, higher will be the weight assigned. It is used when SpyGlass calculates lint assessment of your design.*

- A message explaining why SpyGlass logged the message

NOTE: *The alias, severity level, weight, and message are all contained in the rule definition.*

Messages in the moresimple report are sorted in the order of Severity Class, Severity Label, Rule Name, File Name, and Line Number.

NOTE: *SpyGlass does not elaborate the dead code of a module, but performs parsing of such code. However, violations due to parsing are not reported in the moresimple report. Such violations are captured in the spyglass.log file.*

The SDC mode grouping in the moresimple report is switched off, by default. To enable the SDC mode, specify the following command:

```
set_option report_style display_sdcgroup
```

Whenever you run SpyGlass analysis without specifying a report format to be generated, SpyGlass automatically generates the *moresimple report*. You can change the automatic report format by specifying the name of the report format you want to create to the DEFAULT_REPORT_FORMAT key.

IP-Based Report

SpyGlass generates IP-based reports. The IP-based report generation is enabled during run_goal only. Incremental report generation is not supported.

The following procedure describes how to generate IP-based reports:

1. Use the following command to generate IP-based reports:

   ```
   set_option enable_ip_based_report yes
   ```

2. Specify the instance names and the IP names that you need to include in the IP-based report by using the following command:

   ```
   set_option module_inst_mapping_file <file-name>
   ```

3. Ensure that the IP name and the instance names are specified in the following format in the file-name mapping file:

   ```
   <ipname>:<full_instance_path>
   ```

For example, consider the following input specified in the mapping file:

```
ahb2wb:wb_subsystem.ahb2wb_u0
```

The `Moresimple_<ip_name>_<instpath>.rpt` files and `waive_<ipname>_<instpath>.swl` files are generated automatically. In the above example, the `Moresimple_ahb2wb_wb_subsystem.ahb2wb_u0.rpt` files and `waive_ahb2wb_wb_subsystem.ahb2wb_u0.swl` files are generated.

Note that you need to invoke the waiver report generator in project file, as shown below, to generate the `waiver_<ip_name>_<instpath>.rpt` reports:

```
set_option report {waiver}
```

To see more information on the enable_ip_based_report command, see *enable_ip_based_report*.

The moresimple_warning Report

The moresimple_warning report is similar to the moresimple report and displays all the Info and Warning violations in the design run. Use the following command in the project file to generate the moresimple_warning report:

```
set_option report {moresimple_warning}
```

moresimple_error report

The moresimple_error report is similar to the moresimple report and displays all the Error and Fatal violations in the design run. Use the following command in the project file to generate the moresimple_error report:

```
set_option report {moresimple_error}
```

no_msg_reporting_rules report

The no_msg_reporting_rules report displays a list of rules that did not report any violation or waived during SpyGlass run

spyglass_violations report

The spyglass_violations report is similar to the *moresimple report* with the following distinguishing features:

- The spyglass_violations report does not show `Alias` (Rule alias name) and `SEV_CLASS` (Rule Severity Class name).
- The spyglass_violations report does not show messages with the severity as `Info`.
- Optionally, set the value of the `sg_viol_rpt_no_warnings` parameter to `1` to eliminate messages of `Warning` severity from the `spyglass_violations` report.

  For example, to eliminate warning messages and display only high severity messages, such as, Fatal and Error messages in the `spyglass_violations` report, specify the following command in the project file in the global scope:

  ```
  set_parameter sg_viol_rpt_no_warnings 1
  ```

stop_summary report

The stop_summary report lists design units and files that are skipped from SpyGlass checking in the current run.

These design units and files are specified by the *stop*, *stopdir*, and *stopfile* project file commands.

Following is the sample stop_summary report:

```
Design units stopped by stop/stopfile/stopdir options
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Object Name      Object Type   Stop specification
===========================================================
sr               MODULE       stopfile"bench/benchcell.v"
reorder_bits     MODULE       stopfile "bench/benchcell.v"
clcell           MODULE       stopfile "bench/benchcell.v"
clcell2          MODULE       stopfile "bench/benchcell.v"
expreg_dec       MODULE       stopfile "bench/benchcell.v"
exple_dec        MODULE       stopfile "bench/benchcell.v"

Files stopped by stopfile/stopdir options
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
bench/benchcell.v (stopfile "bench/benchcell.v")

Unused stop/stopfile/stopdir specification(s)
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
stopdir "no_such_dir"
```

elab_summary report

The elab_summary report shows the following design information:

- RTL design unit names
- Elaborated names
- Parameter values

- Parameter values of the SV interfaces overridden with the portparam option

unsynth_modules report

The `unsynth_modules.rpt` report lists the unsynthesized modules in the design.

Following is the sample `unsynth_modules` report:

```
############################################################


Design Unit Name : sub
File             : test.v
Line             : 2
Reason          : SYNTH_5264 error(s) found during synthesis
Cross Ref Id     : REFID_1:SYNTH_5264[2]
BBox Viol Id     : 3
Remedy           : Fix Synthesis errors

Quick Synth Error Reference
============================================================

ViolId  CrossRefId  RuleName    AliasName  Severity
FileName  LineNo  Violation

------------------------------------------------------------

[2]     REFID_1     SYNTH_5264             SynthesisError
test.v    4       sub -> Net type 'REAL' is not supported

------------------------------------------------------------
```

waiver report

A waiver report is generated when messages are waived during SpyGlass run.

This report has the following three sections:

- The *Design Issues Waiver Report* section has actual waived messages waived due to arguments other than the `-ignore` argument or the `-ip` argument of the `waive` constraint
- The *IP/Legacy Waiver Report* section has the waived message count due to the `-ip` argument of the `waive` constraint and the actual waived messages waived if the *report_ip_waiver* command is specified.
- The *Adjustments Waiver Report* section has the waived message count due to the `-ignore` argument of the `waive` constraint and the actual waived messages waived if the *report_adjustment_waiver* option is specified.

The Waiver report also contains the spg_backref field. This field is used to specify the back reference information about the waiver command, that is, the file and line number of the waiver file in which the waiver command was specified.

Report Generated When a Module is Instantiated in Multiple IPs

If SpyGlass reports a violation on a module instantiated in multiple IPs and you have set the *use_du_sch_hier* command to `yes`, you must specify the `waive -ip` command for the all IPs containing its instantiation to waive such violations.

In this case, the waiver report contains the following two sections for the `waive -ip` command:

- The first section shows violations completely waived by the current `waive` command.
- The second section shows violations waived by the current `waive` command along with the other `waive -ip` commands.

NOTE: *Imported waive commands are not displayed in a separate section in the waiver report.*

For example, consider a design containing the `c1cell` module that is instantiated in the mid and mid2 IPs. In addition, the design contains a black box instance of the reorder_bits module.

Now, when you specify the `waive -ip mid` and `waive -ip mid2 -rule ErrorAnalyzeBBox` commands, the following waiver report is generated:

```
waive -ip mid
#spg_backref : "test.swl" 1

1 message(s) waived by current waive -ip command, 1 message(s) waived
along with other waive -ip commands:
================= MESSAGES WAIVED BY CURRENT WAIVE IP COMMAND===========
Index Rule           Severity File      Line Wt  Message
======================================================================
[2]   WarnAnalyzeBBox Warning  test.v 1  10      Design Unit 'IBUF'
                                                 has empty definition

============== MESSAGES WAIVED ALONG WITH OTHER WAIVE IP COMMANDS=======
Index Rule             Severity  File      Line Wt  Message
======================================================================
[4]*  ErrorAnalyzeBBox  Error     test.v 49  10       Design Unit
                                                     'reorder_bits'
                                                     (elaborated name
                                                     'reorder_bits_7')
                                                     has no definition;
                                                     black-box behavior
                                                     assumed
======================================================================


waive -ip mid2 \
      -rule ErrorAnalyzeBBox
#spg_backref : "test.swl" 2

1 message(s) waived along with other waive -ip commands:

============ MESSAGES WAIVED ALONG WITH OTHER WAIVE IP COMMANDS=========
Index Rule             Severity  File    Line Wt  Message
======================================================================
[4]*  ErrorAnalyzeBBox  Error     test.v  49   10  Design Unit
                                                  'reorder_bits'
```

```
                                                                                                    (elaborated name
                                                                                                   'reorder_bits_7') has
                                                                                                    no definition;
                                                                                                    black-box behavior
                                                                                                    assumed
=====================================================================
```

Custom Reports

In addition to the standard reports, SpyGlass also provides custom reports that are designed for specific customer requirements but can be used by all SpyGlass users.

You can generate these reports by using the following commands in the project file:

```
set_option report <custom-report-name>
set_option I {space-separated list of directory-name}
```

NOTE: *The I command is not recommended to be used in the project file. For details, see Options Not Recommended. However, you should specify the directory paths using the -I option on the command-line itself while invoking console or sg_shell.*

You can view the custom reports by selecting the *Report > Default* menu option.

NOTE: *Custom reports are available on AS IS basis.*

The count_sevsort Report

The count_sevsort report is an enhanced version of the *count report* and has the rule message count for each rule of each severity label grouped under their severity class.

For each rule, the severity label name, the rule name, and the rule message count are reported.

The groupings are reported in decreasing severity class (FATAL, ERROR, WARNING, and INFO). Under each severity class, the rule message count is sorted based on the severity label (alphabetically), and then on Rule name (alphabetically).

The moresimple_csv Report

The moresimple_csv report has the same details as the *moresimple report* but in a comma-separated format without the header/footer lines.

In turbo mode, SpyGlass generates the `moresimple_report.csv` report, which is similar to the `moresimple_turbo` report, but in comma-separated format. The `moresimple_report.csv` report includes the parent ID of the secondary violations. For primary violations, SpyGlass reports the parent ID is 'N.A.'.

You can open the generated moresimple_csv.rpt file in a spreadsheet program, such as Microsoft Excel.

The score_detail Report

The score_detail report is an enhanced version of the *score report* and has the following sections:

1. The first section has the detailed status showing the rule name, the rule severity label, the rule message count multiplied by the rule weight, and the rule score. The details are listed sorted by the severity class (not reported) and alphabetically within the severity class.
2. The second section has the summary status showing number of rules flagged under each severity class and the total score for the severity class.

The report has the grand total score for the design at the end.

Block Dependency Report

The block dependency report provides a complete list of files and options that are required to perform successful SpyGlass analysis of a particular design unit. The generated list of options can be used with other synthesis and simulation tools as well.

For each design unit, this report is split in two different files. For details, see *Files Generated in this Flow*.

NOTE: *Although there are related reports, such as elab_summary and Audit that generate a hierarchical view of a design along with their file name, however, none of these reports gives a complete list of dependent files and options for a design unit. To get this information, you can use the block dependency report.*

Generating the Block Dependency Report

To generate the block dependency report for certain design units, use the following command in the project file:

```
set_option gen_block_options { <list-of-blocks> }
```

The following example shows the usage of this command:

```
set_option gen_block_options { top DU1 }
```

Files Generated in this Flow

When you specify design units by using the gen_block_options command, SpyGlass generates the following report files for each specified design unit:

- /spyglass_reports/<design-unit-name>/std_opts.f

  This file contains standard options, such as v, y, libext, define, incdir, libhdlfiles, libhdlf, lib, and libmap.

  You can use the std_opts.f file in SpyGlass run as well as use it across other tools.

- /spyglass_reports/<design-unit-name>/sg_opts.f

  This file contains the following type of commands:

  - ❒ Design read options that were specified to provide various other files and directories. These options include param, pragma, stop, and top.
  - ❒ Rule-checking options, such as sgdc and waivers.

  This file is meant strictly for SpyGlass and cannot be used with other tools.

Usability Aspects of Dependency Reports

The following points discuss some usability aspects of the block dependency report:

- All rule-checking options, such as waiver and sgdc are generated in the sg_opts.f file as it is. You may have to edit some of these options in this file in the subsequent SpyGlass runs.
- Options, such as libhdlfiles, libhdlf, and lib are decompiled as is in the std_opts.f file because they do not have any negative impact on the analysis of blocks specified by using the gen_block_options command.
- You should specify the *top* command when using the internally generated std_opts.f file for the top-level design unit in subsequent SpyGlass runs.
- Design read options that impact any design unit within the hierarchy of the block specified by using the gen_block_options command should not be changed while using the internally generated std_opts.f dependency file for a subsequent SpyGlass run.

  For example, consider design two units du1 and du2, and du1 contains an instance of du2. Now consider that in the original SpyGlass run, you specify the following commands in the project file:

```
set_option gen_block_options du1
set_option stop du2
```

In this case, the <du1>/std_opts.f file is generated.

Now if you again run SpyGlass and pass the <du1>/std_opts.f file in this run, do not change the stop command specification. This is because the dependencies were generated with respect to the netlist without du2, and therefore, the dependencies of du2 will not be considered in the original run and not generated in the <du1>/std_opts.f file.

Consider another example in which you want to generate dependencies for the du1 block and it has the parameter P1. In this case, ensure that the value of P1 should remain same across original SpyGlass run and subsequent SpyGlass runs, as shown below:

Original SpyGlass run	`set_option gen_block_options du1`
	`set_option param "du1.P1=8"`

Subsequent SpyGlass run	`read_file -type sourcelist <du1>/ std_opts.f`
	`set_option param "du1.P1=8"`

Specifying different values for P1 might activate different branches of code within du1 and not all dependencies might be accounted for a particular value of P1.

Directory Structure of Generated Reports

After SpyGlass run, SpyGlass creates the following two parallel directory structures:

- The directory structure containing links to all the generated reports.
- The consolidated directory containing the physical files.

  This directory structure is a simple and a compact structure as compared to the above directory structure. For example, when you run multiple goals, the above directory structure generated is very nested and the user has to search for a report deep down the hierarchy. To solve this problem, SpyGlass provides the consolidated directory structure that is a simplified flattened version of this directory structure.

  By default, SpyGlass creates this structure at the following path:

  <project-working-directory>/<project-name>/consolidated_report/

  To specify a different path, use the *consolidate_reportdir* project file command.

Directory Structure Generated After Design-Read

When you run the design read process, SpyGlass generates reports under the following directory structure:

<project-working-directory>/<project-name>/Design_Read/ spyglass_reports/

Path in the Consolidated Directory Structure

The following directory structure contains links to the reports:

<project-working-directory>/<project-name>/consolidated_report/ Design_Read/

Directory Structure Generated when a Top-Level Module is Specified

If you specify a top-level module, SpyGlass generates reports under the following directory structure:

<project-working-directory>/<project-name>/<top-module-name>/

Design_Read/spyglass_reports/

NOTE: *You can specify a top-level module in the Top Level Design Unit field under the Set Read Options tab or by using the top project file command.*

Path in the Consolidated Directory Structure

The following directory structure contains links to the reports:

<project-working-directory>/<project-name>/consolidated_report/<top-module>_Design_Read/

The following example shows the directory structure generated when you specify the top-level module as `moduleA`:

case1/Project-1/consolidated_report/moduleA_Design_Read/

Directory Structure Generated After a Goal Run

When you run goals, SpyGlass generates reports for each goal run under the following directory structure:

<project-working-directory>/<project-name>/<goal-path>/spyglass_reports/<product-name>/

The following example shows the directory structures generated when you run the initial_rtl/lint/structure goal:

case1/Project-2/initial_rtl/lint/structure/spyglass_reports/lint/

Path in the Consolidated Directory Structure

The following directory structure contains links to the report:

<project-working-directory>/<project-name>/consolidated_report/<goal-path-name>/

The following example shows the directory structure generated when you run the initial_rtl/lint/structure goal:

case1/Project-1/consolidated_report/initial_rtl_lint_structure/

Directory Structure Generated when a Top-Level Module is Specified

If you specify a top-level module, SpyGlass generates reports for each goal run under the following directory structure:

<project-working-directory>/<project-name>/<top-module-name>/<goal-path>/spyglass_reports/<product-name>/

NOTE: *You can specify a top-level module in the Top Level Design Unit field under the Set Read Options tab or by using the top project file command.*

Path in the Consolidated Directory Structure

The following directory structure contains links to the reports:

<project-working-directory>/<project-name>/consolidated_report/<top-module>_<goal-path-name>/

The following example shows the directory structure generated when you specify the top-level module as `moduleA` and run the initial_rtl/lint/structure goal:

Project-1/consolidated_report/moduleA_initial_rtl_lint_structure/

Directory Structure Generated After a Scenario Run

If you run a scenario, SpyGlass generates reports for each scenario run under the following directory structure:

<project-working-directory>/<project-name>/<goal-path>/<scenario-name>

The following example shows the directory structure generated for the `S1` scenario created from the initial_rtl/cdc_exhaustive/cdc_verif_strict goal:

case1/Project-1/top/initial_rtl/cdc_exhaustive/cdc_verif_strict/S1/spyglass_reports/

Path in the Consolidated Directory Structure

The following directory structure contains links to the reports:

<project-working-directory>/<project-name>/consolidated_report/ <goal-path-name>@<scenario-name>/

The following example shows the directory structure generated for the `S1` scenario created from the initial_rtl/lint/structure goal:

Project-1/consolidated_report/initial_rtl_lint_structure@S1/

Directory Structure Generated when a Top-Level Module is Specified

If you specify a top-level module, SpyGlass generates reports for each scenario run under the following directory structure:

<project-working-directory>/<project-name>/<top-module>/<goal-path-name>/<scenario-name>/spyglass_reports/

NOTE: *You can specify a top-level module in the Top Level Design Unit field under the Set Read Options tab or by using the top project file command.*

Path in the Consolidated Directory Structure

The following directory structure contains links to the reports:

<project-working-directory>/<project-name>/consolidated_report/ <top-module>_<goal-path-name>@<scenario-name>/

The following example shows the directory structure generated for the `S1` scenario created from the initial_rtl/cdc_exhaustive/cdc_verif_strict goal when you specify the top-level module as `moduleA`:

case1/Project-1/consolidated_report/ moduleA_initial_rtl_lint_structure@S1/

Grouping in Standard SpyGlass Reports

The following standard SpyGlass reports display messages grouped into useful groups:

- *simple report*
- *moresimple report*

Grouping in Batch Reports

In batch reports, SpyGlass groups rule messages in the following order:

1. New, pre-existing, and fixed/missing messages

   This grouping is enabled in the incremental mode.

NOTE: *No further grouping occurs within the fixed/missing group.*

2. Built-in messages as compare to non built-in messages

   This grouping criterion is always enabled.

3. Rule goals

   This grouping criterion is enabled when you run a goal.

Grouping in GUI Reports

In GUI reports, SpyGlass groups rule messages in the following order:

1. New, pre-existing, and fixed/missing messages

NOTE: *No further grouping occurs within the fixed/missing group.*

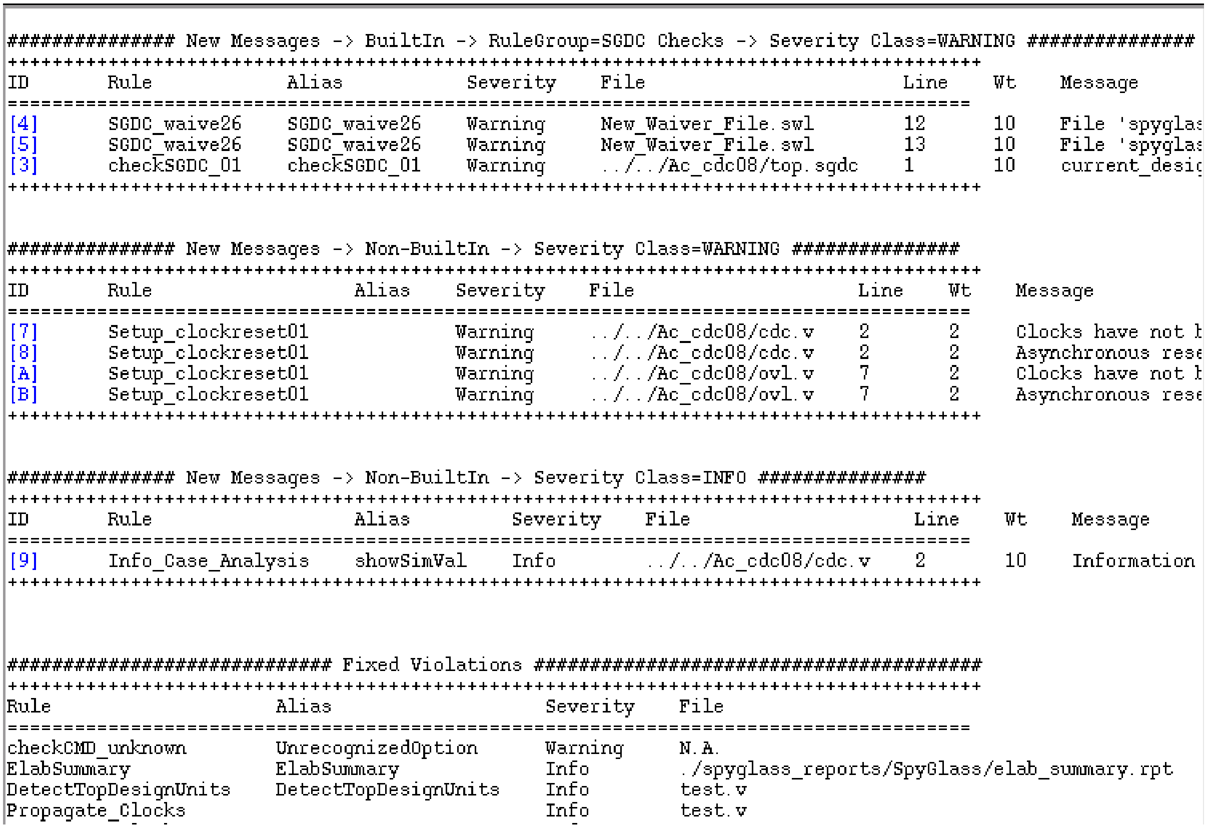
2. Built-in messages as compare to non built-in messages
3. User-specified tags
4. Rule goals

NOTE: *The grouping order of the rule messages in GUI reports is the same as the grouping order selected by you. Refer to the Message Grouping section for details about the configuration of the above grouping order.*

The generated reports contain different sections for leaf-level messages within each of the above grouping orders.

Sample Report

The following figure illustrates a sample moresimple report:



```
############### New Messages -> BuiltIn -> RuleGroup=SGDC Checks -> Severity Class=WARNING ###############
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ID        Rule             Alias            Severity    File                           Line     Wt     Message
=========================================================================================
[4]       SGDC_waive26     SGDC_waive26     Warning     New_Waiver_File.swl            12       10     File 'spyglas
[5]       SGDC_waive26     SGDC_waive26     Warning     New_Waiver_File.swl            13       10     File 'spyglas
[3]       checkSGDC_01     checkSGDC_01     Warning     ../../Ac_cdc08/top.sgdc        1        10     current_desig
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


############### New Messages -> Non-BuiltIn -> Severity Class=WARNING ###############
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ID        Rule                   Alias     Severity    File                       Line    Wt     Message
=========================================================================================
[7]       Setup_clockreset01               Warning     ../../Ac_cdc08/cdc.v       2       2      Clocks have not b
[8]       Setup_clockreset01               Warning     ../../Ac_cdc08/cdc.v       2       2      Asynchronous rese
[A]       Setup_clockreset01               Warning     ../../Ac_cdc08/ovl.v       7       2      Clocks have not b
[B]       Setup_clockreset01               Warning     ../../Ac_cdc08/ovl.v       7       2      Asynchronous rese
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


############### New Messages -> Non-BuiltIn -> Severity Class=INFO ###############
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ID        Rule                   Alias          Severity    File                       Line     Wt     Message
=========================================================================================
[9]       Info_Case_Analysis     showSimVal     Info        ../../Ac_cdc08/cdc.v       2        10     Information


############################# Fixed Violations #############################################
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Rule                     Alias                    Severity    File
=========================================================================================
checkCMD_unknown         UnrecognizedOption       Warning     N.A.
ElabSummary              ElabSummary              Info        ./spyglass_reports/SpyGlass/elab_summary.rpt
DetectTopDesignUnits     DetectTopDesignUnits     Info        test.v
Propagate_Clocks                                  Info        test.v
```

FIGURE 242.

Sorting Messages in SpyGlass Reports

SpyGlass generates the following reports that are sorted for better usability:

count report	*moresimple report*	*simple report*
summary report	*Default Reports*	

SpyGlass sorts the messages in these reports and the Message Tree by the following criteria (provided the criterion is applicable to the report):

- Severity Class (decreasing from FATAL, Error, Warning, and Info)
- Rule Name (alphabetical)
- Source HDL Filename (alphabetical)
- Line number (ascending)

You can modify the default sorting order of rule messages in BATCH reports and Message Tree. To display the sorting criteria with the severity label and rule weight, you can specify the following command in the project file:

```
set_option report_style sort_sevlabel_wt
```

When this option is specified, SpyGlass sorts the messages in reports and Message Tree by the following criteria:

- Severity Label (alphabetical)
- Rule Weight (descending)
- Severity Class (decreasing from FATAL, Error, Warning, Info)
- Rule Name (alphabetical)
- Source HDL filename (alphabetical)
- Line number (ascending)

User-Defined Message Sorting in SpyGlass Reports

SpyGlass provides a predefined sorting order for messages in SpyGlass reports. You can customize this sorting order as per your requirement.

Default Sorting Order

By default, messages are sorted in the following order:

1. Sorting based on severity
2. Sorting based on rule name
3. Sorting based on file and line number for a given rule

Customizing the Sorting Order

You can specify your own message sorting order based on message components by using the *sortrule* command.

Following is the syntax of the *sortrule* command:

```
set_option sortrule <language>+<rule-name>+<sort-order>
```

NOTE: *There are no spaces between any of the values in the above syntax.*

Details of Arguments of the sortrule Command

The following table describes the details of various arguments of the *sortrule* command:

Argument	**Description**
<language>	Specifies the rule language. You can specify the language as Verilog, VHDL, or Mixed. The rule for which message sorting order is being defined must be registered for the specified language. If the rule is specified for both languages, you can optionally specify only one of the languages if you want to specify the message sorting order for only that language.
<rule-name>	Specifies the rule name
<sort-order>	Specifies a user-defined sort order in the following format: <arg-number><arg-type><arg-sort-order>
Arguments of <*sort-order*>	

Argument	Description
<arg-number>	Specifies the argument number. To get the argument number, refer the rule message goal in the Product rule-deck file. For example, the LPFSM16 rule of the SpyGlass Power Verify solution has the following message goal: Attribute '%1' found on enumerated type '%2' used for encoding FSM states Thus, the first argument is the attribute name and is specified as 1. The second argument is the state variable name and is specified as 2. Refer the corresponding rules reference document for explanation of the rule message arguments.
<arg-type>	Specifies the argument type. In the above example, the argument type for both arguments is string and is specified as s. The other possible argument types are numerals (specified as n) and enumerated types (specified as e)
<arg-sort-order>	Specifies the argument value sorting order as ascending (specified as a) or descending (specified as d).

Example of Using the sortrule Option

Consider the following example:

```
set_option sortrule Verilog+R1+2sa+1nd+3e/val1/val2/val3
```

The above specification defines message sorting order of the R1 rule in Verilog mode. Further, the messages are sorted in the following order:

1. Messages sorted by 2sa, that is, sorted by the value of the second argument (2) which is a string argument (s) in ascending order (a).
2. For messages with the same second argument value, sorting is done by 1nd, that is, sort by the first argument (1) which is a numeral argument (n) in descending order (d).
3. For messages with the same first argument value, sorting is done by 3e/val1/val2/val3, that is, sort by the third argument (3) which is an enumerated type argument (e) based argument values val1, val2, and val3 in that order.

In addition to the argument-based sorting orders described above, you can specify message sorting order by file (specified as f) and by line number (specified as l), both in either ascending order (specified as a) or descending order (specified as d). Thus, fd means that sort the messages by file name in descending order. la means that sort by line number in ascending order.

Consider the sortrule specification in the above example with addition values as follows:

```
set_option sortrule Verilog+R1+2sa+1nd+3e/val1/val2/
val3+fd+la
```

This specification means that any sorting after the argument-based sorting will be done first by file names in descending order and then by line numbers in ascending order.

By default, the argument values are sorted in a case-sensitive manner. Specify i (for ignore case) to indicate that the argument values are to be sorted in a case-insensitive manner. Consider the following example:

```
set_option sortrule Verilog+R1+2sai+1nd+3e/val1/val2/
val3+fdi+la
```

The above specification indicates that argument-based sorting indicated by 2sa and file-based sorting indicated by fd should be performed in a case-insensitive manner.

This chapter describes the following reports generated in SpyGlass:

- *Project Summary Report*
- *The DataSheet Report*
- *The DashBoard Report*
- *Goal Summary*

If the working directory of your project contains a different set of goals that are run from different methodologies, the above reports show results only from the goals that are run from the current *active methodology*, which is saved in the project file.

If you get any unexpected results in such a scenario, clean the project working directory and re-run the goals of the active methodology.

An active methodology is specified by the following command in the project file:

```
set_option active_methodology <methodology-path>
```

Searching for Input Files

While generating the above reports, SpyGlass searches for input files present at a relative path under the directory specified by the `set_option projectcwd` *<dir>* command. However, if you do not specify this command in a project file, SpyGlass searches for input files under the directory containing the project file and then in the user current working directory.

For example, consider the project file /usr/test-cases/design1/sample.prj with the following contents:

```
# sample.prj
# This option sets project current working directory
set_option projectcwd /usr/test-cases/test1

# This file contains list of source files
```

```
read_file -type sourcelist  "sources.f"

# This file contains library settings
source "./lib_precompile.f"
…
```

While generating reports, SpyGlass searches for the specified input files in the project’s current working directory.

In the above example, the input files, sources.f and lib_precompile.f, specified through the read_file and source commands are looked under the /usr/test-cases/test1 directory.

However, if the projectcwd option is not specified in this project file, SpyGlass searches for these files under the /usr/test-cases/design1 directory (the directory where project file resides) and then in the current working directory.

Default Paths of Aggregated Reports

The default path of aggregated reports varies depending on the mode in which these reports are generated. This is described in the following points:

- If you generate a report by using the `set_option aggregate_report {<report-list>}` command in a project file, the report is generated at the following path:

  <projectwdir>/<prj-name>/<top-name>/html_reports/

- If you generate a report by using the `-gen_aggregate_report` command-line option or through GUI, the report is generated at the following path in the current working directory:

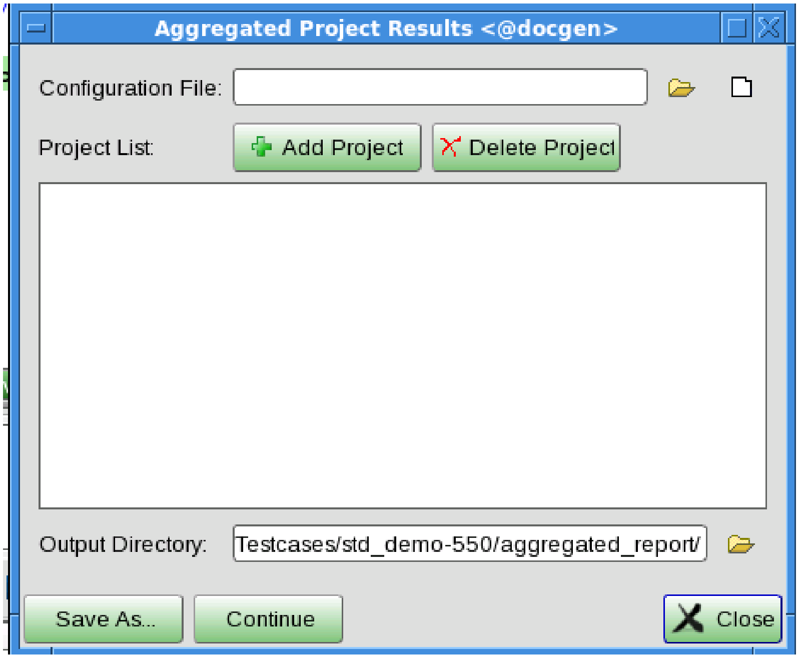
  ./html_reports/

Generating Aggregated Project Results

You can generate aggregated project results that contain combined results from multiple single-user projects. You can view these results from SpyGlass GUI without opening a project file.

NOTE: *This report is deprecated and will be removed in a future release.*

To generate aggregated project results, click the *Tools > Aggregated Project Results* menu option. This displays the *Aggregated Project Results* dialog, as shown in the following figure:



Output Directory: Testcases/std_demo-550/aggregated_report/

FIGURE 243. Aggregated Project Results

If a configuration file already exists, enter the name of the file in the Configuration file text field. Alternatively, click (📂) and browse to the location where the configuration file is saved. Then the list of projects saved in the project file is displayed in the *Project List* section of the Aggregated Project Results dialog. You can add a project in the configuration file and save the file. In addition, you can also save the configuration file with a different name by clicking the *Save As* button.

You can also specify the report output directory in the *Output Directory* textbox. If the specified directory does not exist, SpyGlass creates it. If you do not specify any directory, the report is saved at a default path. For details, see *Default Paths of Aggregated Reports*.

If you are generating the aggregated results for the first time, follow these steps:

1. Click (Add Project) and browse to the location where the project files are located.
2. Select the project (.prj) file for which you want to generate the results.

NOTE: *You can select multiple project files by pressing and holding the <Ctrl> key and then selecting the required files. You can delete a project by clicking the Delete Project button.*

3. Click *Save & Continue*. The *Specify the configuration file* dialog appears, as shown in the following figure:

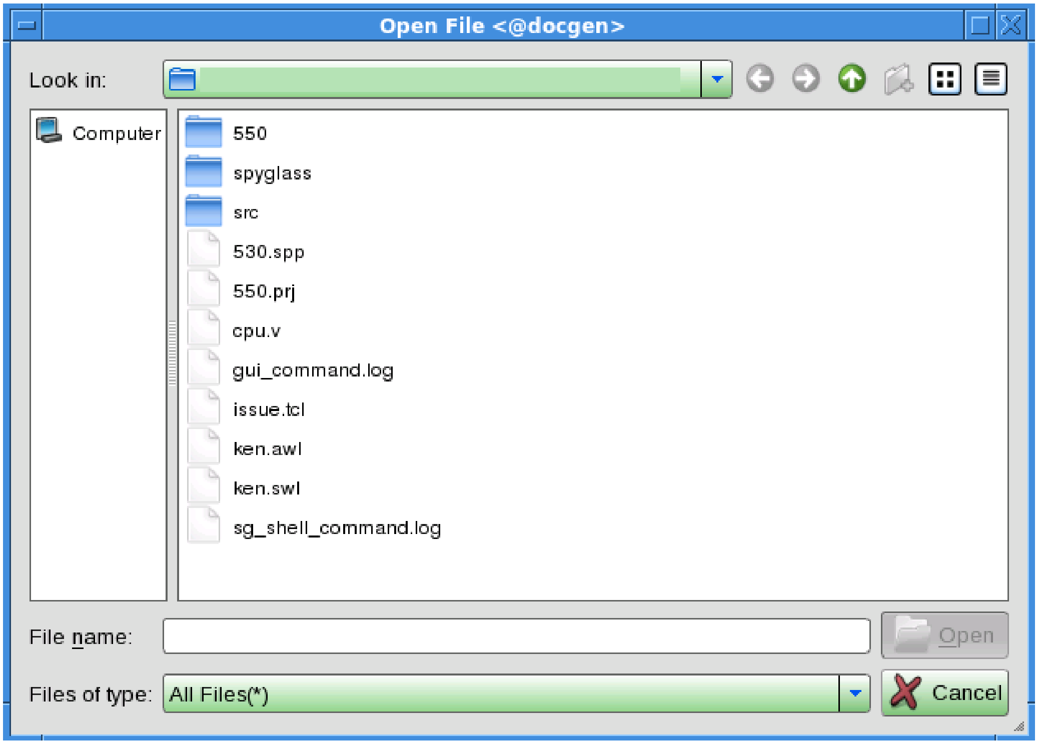
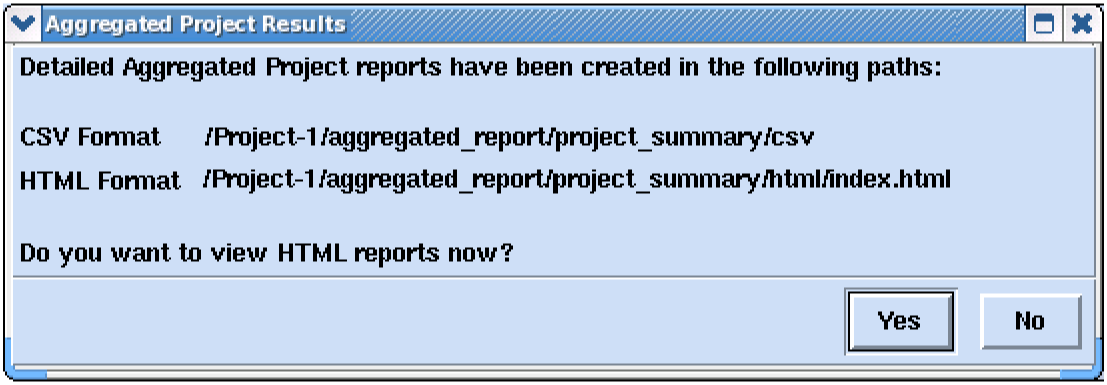


FIGURE 244. Specify the Configuration File

By default, SpyGlass names the configuration file as SG_AGG_PROJECT_RESULTS_CONFIG_FILE. However, you can specify a different file name.

4. Type the name of the configuration file in the *File name* text field and click *Save*. The *Aggregated Project Result*s dialog appears displaying the location where the results are saved, as shown in the following figure:



Aggregated Project Results

Detailed Aggregated Project reports have been created in the following paths:

CSV Format /Project-1/aggregated_report/project_summary/csv

HTML Format /Project-1/aggregated_report/project_summary/html/index.html

Do you want to view HTML reports now?

FIGURE 245. Aggregated Project Results Dialog Box

The aggregated project results are generated in the following formats:

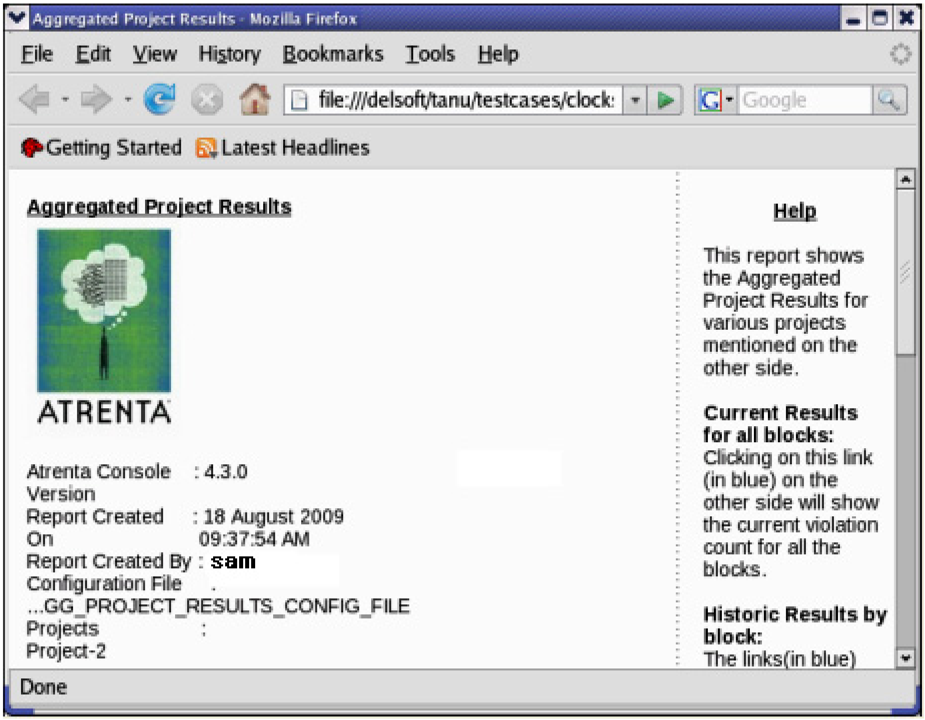
- HTML

  The HTML report is generated in the html_reports/project_summary/html directory and can be opened in the HTML browser (specified using the *Specify HTML Browser Program* option in the *Miscellaneous Page* of the *Tools > Preferences* window).

- CSV

  The .csv report is generated in the html_reports/project_summary/csv directory and can be opened using any external tool, such as Microsoft Excel.

To view the aggregated project reports in the HTML format, click *Yes* in the *Aggregated Project Results* dialog. Then, the aggregated project results are displayed in the browser window as shown below:


Aggregated Project Results - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///delsoft/tanu/testcases/clock:
Google
Getting Started
Latest Headlines
ATRENTA
Done

Aggregated Project Results

Atrenta Console Version : 4.3.0
Report Created On : 18 August 2009 09:37:54 AM
Report Created By : sam
Configuration File : ...GG_PROJECT_RESULTS_CONFIG_FILE
Projects :
Project-2

Help

This report shows the Aggregated Project Results for various projects mentioned on the other side.

Current Results for all blocks: Clicking on this link (in blue) on the other side will show the current violation count for all the blocks.

Historic Results by block: The links(in blue)

FIGURE 246. Aggregated Project Results Report

Like the Project Summary Report, the HTML browser displays similar information for Aggregated Project Reports. Additionally, the browser also displays the configuration file name and the list of projects that were saved in the configuration file. Refer to the *Project Summary Report* section for more details on various reports.

NOTE: *You can also specify the path of the configuration file in the* .spyglass.setup *configuration file by using the* `AGG_PROJECT_RESULTS_CONFIG_FILE` *environment variable as follows:*

```
SDE_CONFIG_OPTIONS=AGG_PROJECT_RESULTS_CONFIG_FILE=<
path>
```

Project Summary Report

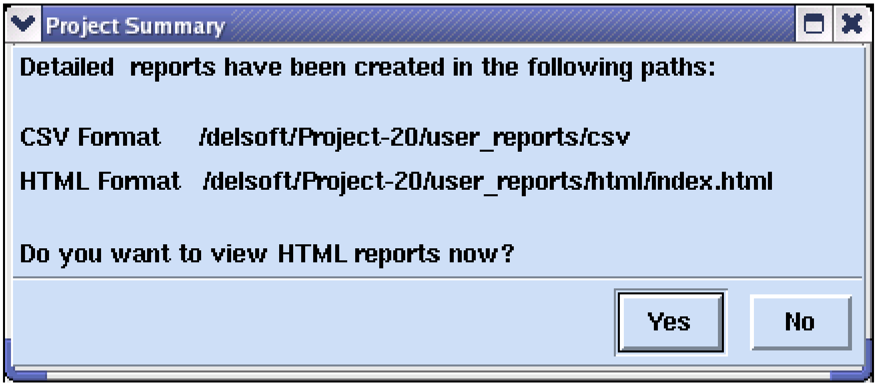
The *Project Summary* report contains the result of all the blocks and goals runs.

Generating the Project Summary Report

You can generate the Project Summary report through GUI or through a project file.

Generating the Report through GUI

To generate the *Project Summary* report through GUI, click the *Project Summary* option on the goal selection window. When you click this option, the *Project Summary* dialog appears, as shown in the following figure:



Detailed reports have been created in the following paths:

CSV Format /delsoft/Project-20/user_reports/csv

HTML Format /delsoft/Project-20/user_reports/html/index.html

Do you want to view HTML reports now?

FIGURE 247. Project Summary Dialog Box

The above dialog displays the location where the generated CSV and HTML reports are located.

NOTE: *You can right-click on a path and select the* Copy *shortcut menu option to copy the path for reference.*

NOTE: *When you click the Project Summary option, SpyGlass also creates a* `<project-`

`name>`/user_reports_*backup directory that contains a backup of the reports located in the* `<project-name>`/user_reports directory.

Generating the Report through a Project File

To generate the Project Summary report through a project file, specify the following command in the project file:

```
set_option aggregate_report project_summary
```

To specify the directory in which you want to generate the project summary report, specify the following command in the project file:

```
set_option aggregate_reportdir <report-directory-path>
```

NOTE: *If you do not specify the above command for a project-specific DashBoard report, the reports are generated in the <projectwdir>/<project>/<top>/html_reports directory by default.*

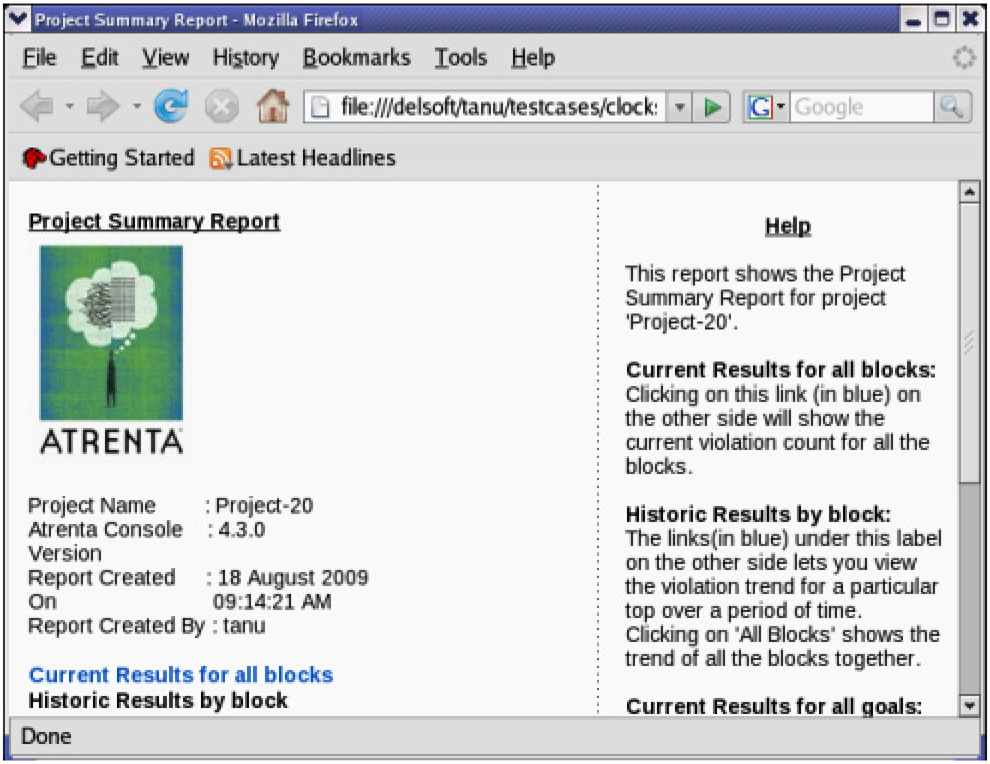
Viewing the Project Summary Report

SpyGlass generates the *Project Summary* report in the following formats:

- HTML: The HTML report is generated in the `<project-name>`/user_reports/html directory and can be opened in an HTML browser (specified using the *Specify HTML Browser Program* option in the *Miscellaneous Page* of the *Tools > Preferences* window).
- CSV: The .csv report is generated in the `<project-name>`/user_reports/csv directory and can be opened using any external tool, such as Microsoft Excel.

Viewing the HTML Report

To view the HTML report, click *Yes* in the *Project Summary* dialog. Then, the project summary report is displayed in the HTML browser, as shown in the following figure:


Project Summary Report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///delsoft/tanu/testcases/clock
Google
Getting Started Latest Headlines
ATRENTA
Done

Project Summary Report

Project Name : Project-20
Atrenta Console Version : 4.3.0
Report Created On : 18 August 2009 09:14:21 AM
Report Created By : tanu

Current Results for all blocks
Historic Results by block

Help

This report shows the Project Summary Report for project 'Project-20'.

Current Results for all blocks: Clicking on this link (in blue) on the other side will show the current violation count for all the blocks.

Historic Results by block: The links(in blue) under this label on the other side lets you view the violation trend for a particular top over a period of time. Clicking on 'All Blocks' shows the trend of all the blocks together.

Current Results for all goals:

FIGURE 248. Project Summary Report

The HTML browser is divided into two sections. The left section of the browser displays information, such as the project name, the SpyGlass version, the date when the report was created, and the name of the person who created the report. In addition, the following options are provided in the left section of the browser window.

Current Results for all blocks

The current results for all blocks (specified by using the *Top Level Design Unit(s)* design-read option the *Set Read Options* tab) displays the latest result summary files created on a timestamp basis. For example, if a block B1 is run on July 31, and then on August 03, the current result for block B1 will display the data generated on August 03. In addition, if block B1 is run twice on August 03, once at 11:00 AM, and then at 5:00 PM, then the current result will display the data generated at 5:00 PM.

When you, click the *Current Results for all blocks* link, the right section of the browser window displays the following:

- Current unresolved violations for all blocks:

  The unresolved violations chart displays the violation count (FATAL, ERROR, WARNING, and INFO messages) for the individual blocks and the goal run on the block in the Y axis and the block names in the X axis.

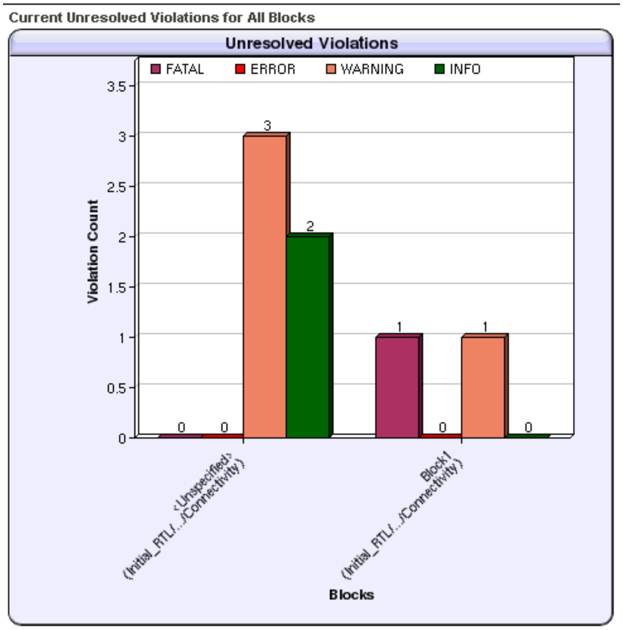


FIGURE 249. Unresolved Violations Chart

- Current waived violations for all blocks

  The current waived violations for all blocks chart displays the count of the violation messages (ERROR, WARNING, and INFO) that were waived for each block and the goal run on the block.

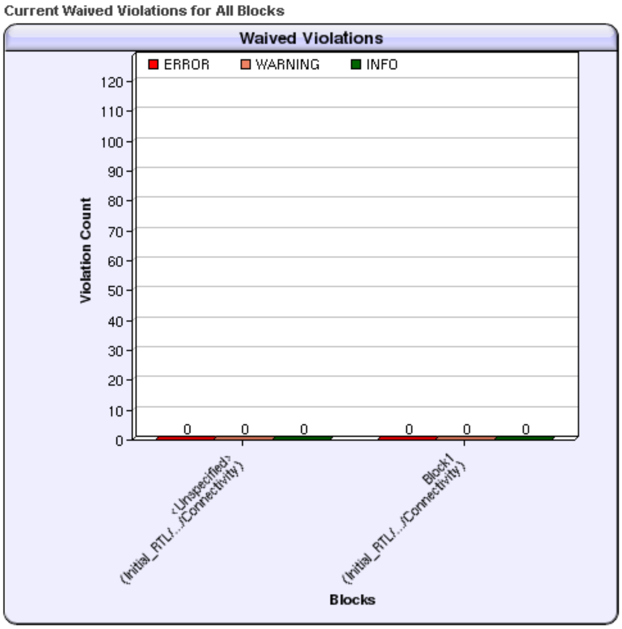
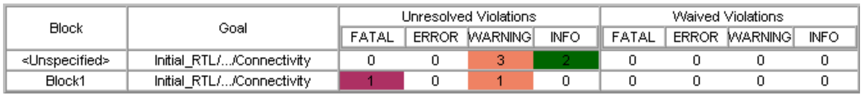


FIGURE 250. Current Waived Violations for All Blocks Chart

- Summary table of all current unresolved and waived violations for all blocks

  The current result for all blocks also displays a summary table that shows the count of all the violation messages (unresolved and waived) as shown below:



Block	Goal	Unresolved Violations				Waived Violations			
		FATAL	ERROR	WARNING	INFO	FATAL	ERROR	WARNING	INFO
<Unspecified>	Initial_RTL/.../Connectivity	0	0	3	2	0	0	0	0
Block1	Initial_RTL/.../Connectivity	1	0	1	0	0	0	0	0

FIGURE 251. All Current Unresolved And Waived Violations Summary Table

The summary table contains the following columns:

- ❒ Block - displays the name of the block
- ❒ Goal Name - displays the name of goal run for a block

- Unresolved Violations: Displays the count of the unresolved violations (based on severity)
- Waived Violations: Displays the count of the waived violations (based on severity)

Historic Results by block

Use this option to view the violation count for a block/all blocks based on time. You can view the violation count for an individual block or for all blocks.

To view the historic result for an individual block, click the block name on the left section of the browser window. Then, the block trend for unresolved violations is displayed on the right section of the browser window as shown below:

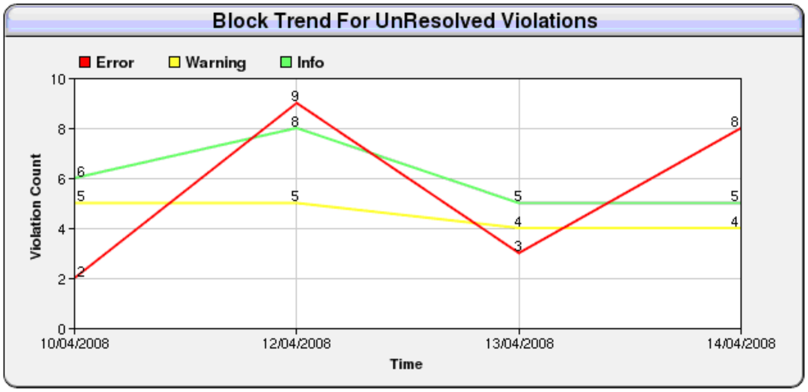


FIGURE 252. Block Trend for Unresolved Violations

The Y axis displays the violation count for the block and the X axis displays the date when the block was run.

NOTE: *When you click the All blocks link, the violation count for all blocks is displayed with the data of each block separated by a line.*

Similarly, you can view the violation count of the waived violations for a block or all blocks.

In addition, you can view the summary table displaying the violation messages (unresolved and waived) for a block or all blocks based on time.

Current Results for all Goals

The current results for all goals displays the latest result summary files created on a timestamp basis for the goals run.

The chart generated for the goals is similar to the *Current Results for all blocks*.

The goal names in the summary table that displays the violation messages (unresolved and waived) for a goal are truncated if the number of characters in the goal name exceeds a particular length. You can view the complete name of the goal by placing the cursor over the goal name.

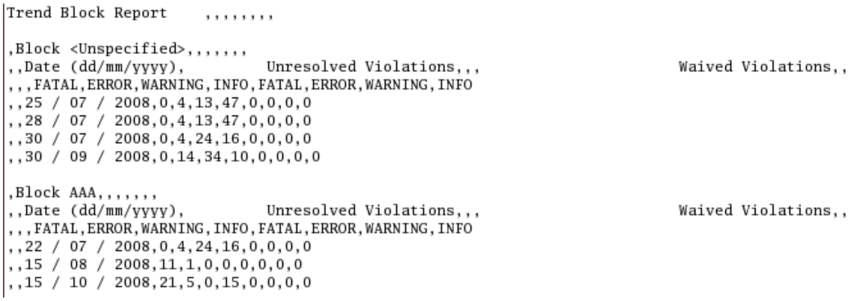
Historic Results by goals

Use this option to view the violation count for a goal/all goals on a time basis. The chart generated for the historic results by goal is similar to the *Historic Results by block*.

Viewing CSV Reports

To view the CSV reports, browse to the *<project-name>*/user_reports/csv directory where the report is located. Open the .csv file using your favorite text editor (specified using the *Specified Text Program* option in the *Miscellaneous* tab of the *Tools* > *Preferences* window*).*

A sample CSV report is shown below:



```
Trend Block Report    ,,,,,,,,

,Block <Unspecified>,,,,,,,
,,Date (dd/mm/yyyy),         Unresolved Violations,,,                     Waived Violations,,
,,,FATAL,ERROR,WARNING,INFO,FATAL,ERROR,WARNING,INFO
,,25 / 07 / 2008,0,4,13,47,0,0,0,0
,,28 / 07 / 2008,0,4,13,47,0,0,0,0
,,30 / 07 / 2008,0,4,24,16,0,0,0,0
,,30 / 09 / 2008,0,14,34,10,0,0,0,0

,Block AAA,,,,,,,
,,Date (dd/mm/yyyy),         Unresolved Violations,,,                     Waived Violations,,
,,,FATAL,ERROR,WARNING,INFO,FATAL,ERROR,WARNING,INFO
,,22 / 07 / 2008,0,4,24,16,0,0,0,0
,,15 / 08 / 2008,11,1,0,0,0,0,0,0
,,15 / 10 / 2008,21,5,0,15,0,0,0,0
```

FIGURE 253. Sample CSV Report

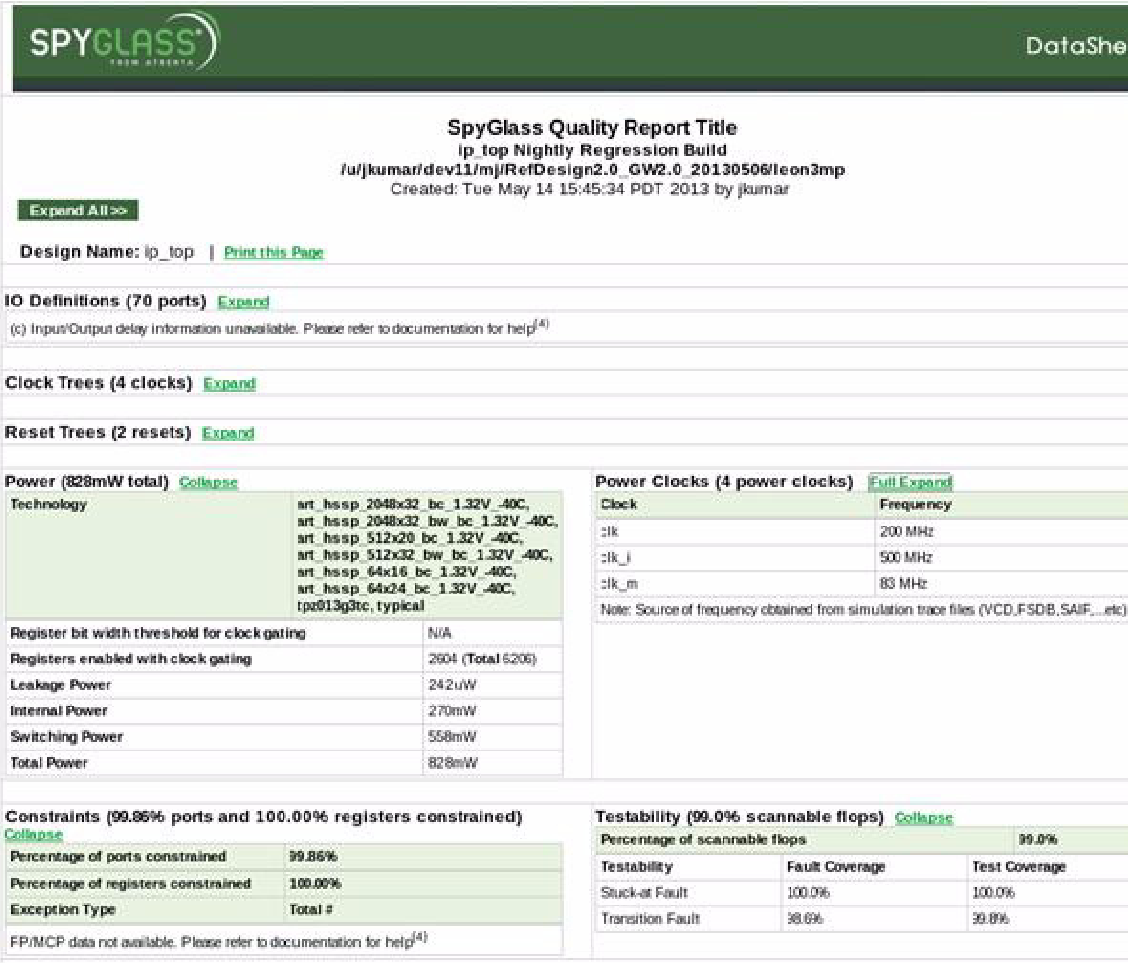
The DataSheet Report

The *DataSheet* report highlights design characteristic and qualities of an IP. It provides summarized information for an IP such as IO details, clock trees, reset trees, power and test characteristics of an IP, black box characteristics, gate count estimates, and so on.

By default, SpyGlass automatically generates the Datasheet report for the current project. You can disable report generation for the Datasheet report using the following command:

```
set_option disable_html_report {datasheet}
```

The following figure displays a sample *DataSheet* report:



SpyGlass Quality Report Title
ip_top Nightly Regression Build
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp
Created: Tue May 14 15:45:34 PDT 2013 by jkumar

Design Name: ip_top | Print this Page

IO Definitions (70 ports) Expand

(c) Input/Output delay information unavailable. Please refer to documentation for help[4]

Clock Trees (4 clocks) Expand

Reset Trees (2 resets) Expand

Power (828mW total) Collapse

Technology	art_hssp_2048x32_bc_1.32V_-40C, art_hssp_2048x32_bw_bc_1.32V_-40C, art_hssp_512x20_bc_1.32V_-40C, art_hssp_512x32_bw_bc_1.32V_-40C, art_hssp_64x16_bc_1.32V_-40C, art_hssp_64x24_bc_1.32V_-40C, tpz013g3tc, typical
Register bit width threshold for clock gating	N/A
Registers enabled with clock gating	2604 (Total 6206)
Leakage Power	242uW
Internal Power	270mW
Switching Power	558mW
Total Power	828mW

Power Clocks (4 power clocks) Full Expand

Clock	Frequency
:lk	200 MHz
:lk_i	500 MHz
:lk_m	83 MHz

Note: Source of frequency obtained from simulation trace files (VCD,FSDB,SAIF,...etc)

Constraints (99.86% ports and 100.00% registers constrained) Collapse

Percentage of ports constrained	**99.86%**
Percentage of registers constrained	**100.00%**
Exception Type	**Total #**

FP/MCP data not available. Please refer to documentation for help[4]

Testability (99.0% scannable flops) Collapse

Percentage of scannable flops		**99.0%**
Testability	**Fault Coverage**	**Test Coverage**
Stuck-at Fault	100.0%	100.0%
Transition Fault	98.6%	99.8%

FIGURE 254. The DataSheet Report

You can view the *DataSheet* report to review design characteristics during design review or as a way of communicating design characteristics during design handoff and IP sharing.

Licensing Requirements

SpyGlass *DataSheet* is a licensed capability and requires the license feature, `datasheet`. Please contact SpyGlass Support (*spyglass_support@synopsys.com*), if you need this license.

Generating the DataSheet Report in GUI

You can generate the *DataSheet* report containing data for the current project or the data for multiple projects and/or batch run dump directories. Based on your requirement, select any of the following menu options:

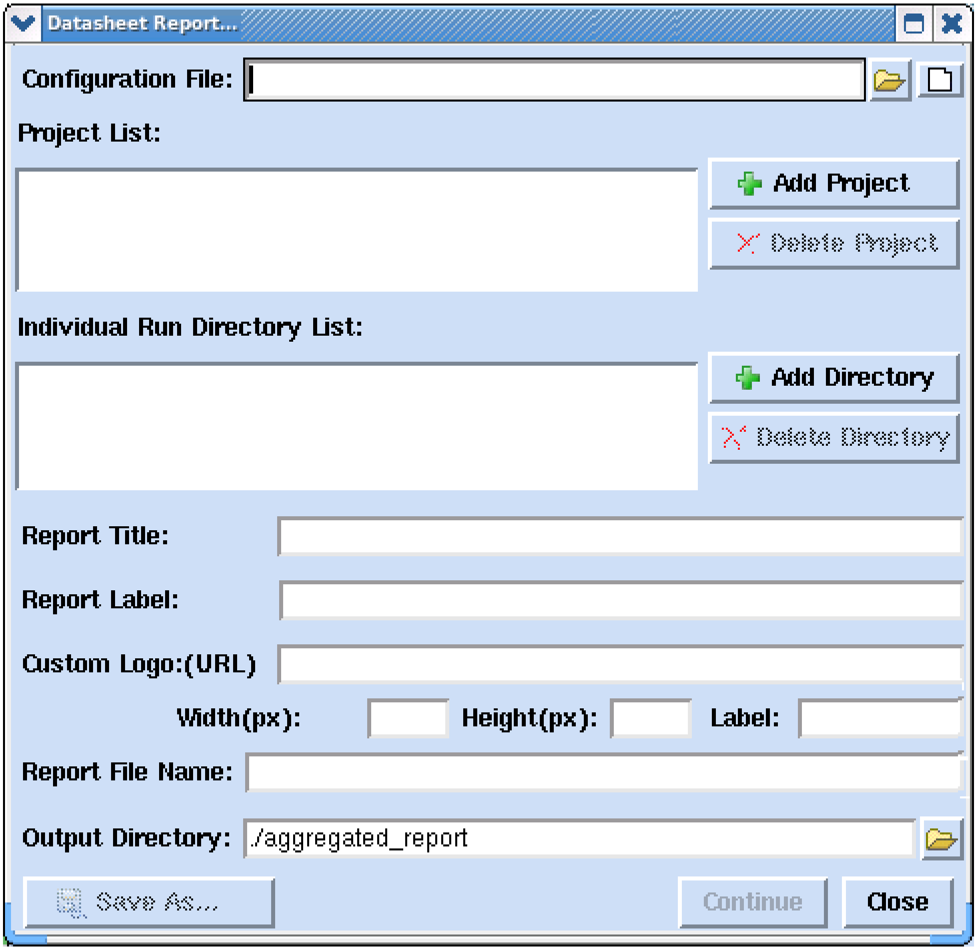
- *Tools -> Datasheet Report -> Project Report*

  Select this menu option to generate the *DataSheet* report for the current project.

- *Tools -> Datasheet Report -> Aggregated Report*

  Select this menu option to generate the *DataSheet* report containing data for multiple projects.

  When you select this menu option, SpyGlass displays the *Datasheet Report* dialog, as shown in the following figure:



Output Directory: ./aggregated_report

FIGURE 255. Configuring DataSheet Report

In the above dialog, you can specify the following details:

- Specify a configuration file.

  Click the 📂 button to select the required configuration file. Alternatively, you can also create a new configuration file. For details, refer to the *Creating a Configuration File* topic. For more details about configuration file, refer to the *Details of a Configuration File* topic.

- Specify an output directory

  Specify the report output directory in the *Output Directory* textbox. If the specified directory does not exist, SpyGlass creates it.

  If you do not specify any directory, the report is saved at a default path. For details, see *Default Paths of Aggregated Reports*.

Creating a Configuration File

You can also create a new configuration file and add the required project files and/or SpyGlass batch run directories in the configuration file. To create a new configuration file, perform the following steps:

1. Specify the name of the configuration file in the *Configuration File* textbox.
2. Click the *Add Project* button, and select the required project file to be included in the configuration file.

   Repeat this step to add more project files.
3. Click the *Add Directory* button, and select the required SpyGlass batch run directory to be included in the configuration file.

   Repeat this step to add more batch run directories.
4. Specify report title in the *Report Title* textbox.

   Report title can be used to specify the top-level report name. For example, you can specify your company name as the report title.

   When you specify the report title, the following command is generated in the configuration file:

   ```
   REPORT_TITLE_DATASHEET <title>
   ```
5. Specify report label in the *Report Label* textbox.

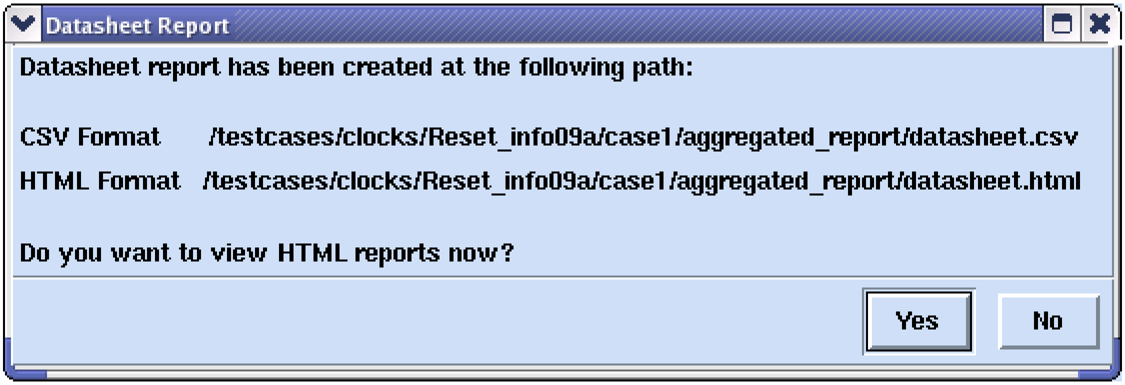
   Report label refers to an additional description of the report. For example, you may specify the report label as *Regression report created by the XYZ group*.

   When you specify the report label, the following command is generated in the configuration file:

   ```
   REPORT_LABEL_DATASHEET <label>
   ```
6. Click the *Save As* button. This displays the *Save new configuration file as* dialog.

7. Specify the required details in the *Save new configuration file as* dialog, and click the *Save* button.

   After specifying the configuration file (new or existing), click the *Continue* button. This displays the following dialog:



Datasheet Report

Datasheet report has been created at the following path:

CSV Format /testcases/clocks/Reset_info09a/case1/aggregated_report/datasheet.csv

HTML Format /testcases/clocks/Reset_info09a/case1/aggregated_report/datasheet.html

Do you want to view HTML reports now?

FIGURE 256. DataSheet Report - Dialog Box

In the above dialog, click the *Yes* button to view the HTML report in the browser window.

If the manually created configuration file contains attributes, such as REPORT_TITLE and REPORT_LABEL, without report specific extension, for example, _DATASHEET at the end, SpyGlass batch process applies the same values to all the required reports that are generated using the same configuration file. However, if the configuration file contains both attributes, such as , REPORT_TITLE and REPORT_TITLE_DATASHEET, then the report specific value will take the precedence.

This means that following options are applied to all reports:

- REPORT_TITLE <value>
- REPORT_LABEL <value>
- REPORT_FILE_NAME <value>
- CUSTOM_LOGO <value>

Also, following options are applied to the Datasheet report only:

- REPORT_TITLE_DATASHEET <value>
- REPORT_LABEL_DATASHEET <value>

- REPORT_FILE_NAME_DATASHEET <value>
- CUSTOM_LOGO_DATASHEET <value>

Details of a Configuration File

A configuration file contains the path of project files and/or SpyGlass batch run dump directories whose data you want to include in the *DataSheet* report.

A sample configuration file is given below:

```
# ---------------------------------------------------------
# Aggregated Report Configuration File
# Created By: sam using SpyGlass version 5.0
# Last Modification On 04-05-2012 04:08:17
# ----------------------------------------------------------

Project-1.prj
Project-2.prj
Project-3.prj

../Socrates/Project-7.prj
/dev09/case9-new/Project-5.prj
./run1/

REPORT_TITLE_DATASHEET "My Title"
REPORT_LABEL_DATASHEET 'My report label'
CUSTOM_LOGO_DATASHEET http://myorg.com/img/
logo.gif@@75@@45@@My Custom  Logo
REPORT_FILE_NAME_DATASHEET "My_DataSheet"
```

Changing the Name of the Report

By default, the name of the report is datasheet, that is, datasheet.html and datasheet.csv.

You can change this name in either of the following ways:

- By specifying a file name in the *Report File Name* text box of the *Datasheet Report* dialog.

- By specifying a file name to the `REPORT_FILE_NAME_DATASHEET` variable in a configuration file, as shown below:

  ```
  REPORT_FILE_NAME_DATASHEET <name>
  ```

Adding a Logo in the Report Header

You can add a logo in the report header by specifying the following information in appropriate fields in the *Datasheet Report* dialog:

- URL of the logo in the *Custom Logo:(URL)* textbox
- Width (in pixels) of the logo in the *Width(px)* textbox
- Height (in pixels) of the logo in the *Height(px)* textbox
- An alternate text to be displayed if the logo fails to load from the specified URL in the *Label* textbox

Based on the above information, the logo details are stored in the configuration file in the following format:

```
CUSTOM_LOGO_DATASHEET
<URL>@@<Width>@@<Height>@@<Label>
```

Tcl Format Support in the Configuration File

SpyGlass provides you with a Tcl format to edit the configuration file. You can use the same configuration file for both the DataSheet and DashBoard reports for different projects. This makes easy to setup/create SoC configuration file.

NOTE: *You can specify the configuration file in <key>-<value> based text file format or in the below explained TCL format. However, you can not use the same file for both the formats.*

Following Tcl commands are used in the configuration file:

- *aggregate_projects*
- *set_config_option*

aggregate_projects

This command specifies list of projects and work directories that you want to include in the specified report. Following is the syntax of the *aggregate_projects* command:

```
aggregate_projects -project {<project_list>} -dir
{<directory_list>} [-report {<report_list>}]
```

Here, `-report` is an optional argument and you can set specific reports for different projects, while aggregation. That is, if you set the -report {dashboard} option, then the project is added for the DashBoard report and skips if the same configuration file is used for the DataSheet report.

Now, assume that you want to include Project1.prj and Project2.prj for both the Datasheet and Dashboard reports. Also, assume that you to include Project3.prj only for the DashBoard report. To do so, specify the following commands:

```
aggregate_projects -project {Project1.prj Project2.prj}
```

```
aggregate_projects -project {Project3.prj} -report
{dashboard}
```

If you set -report {dashboard}, then the project is added for the DashBoard report and skips if the same configuration file is used for the DataSheet report.

Following is the sample usage of this command:

```
foreach module $env(MODULE_LIST) {
  set ::env(MODULE) $module
  foreach variant $env(HARDWARE_LIST) {
          set ::env(HARDWARE) $variant
  //Add project that may typically uses MODULE & HARDWARE
inside the project file
          aggregate_projects -project {project.prj} -report
{dashboard}
  }
}
```

set_config_option

This command supports different -<key> <values> combinations that are required for configuring the reports. Following is the syntax of this command:

```
set_config_option -<key> <value> [-report {<report_list>}]
```

You can specify one of the following values to the <key> option:

- **report_title**: Specifies the report title
- **report_label**: Specifies the report label
- **custom_logo**: Specifies the custom logo on the report
- **report_file_name**: Specifies the output file name

The -report option is optional and you can set specific reports to use these config options.

Following is the sample usage of this command:

```
set_config_option -report_title   "My report  title" -report
{dashboard}
```

```
set_config_option -report_label   "My report label"  -report
{dashboard}
```

```
set_config_option -custom_logo  "http://myorg.com/img/
logo.gif@@75@@45@@My Custom Logo"
```

```
set_config_option -report_file_name  "My_DashBoard"
```

Sample Configuration File

Following is a sample configuration file to configure the Datasheet report using the above explained commands:

```
#Add input project files
aggregate_projects -project {/proj/path/module1.prj /proj/
path/module2.prj}

#Add config options
set_config_option -report_title    "My report title" -report
{datasheet}
set_config_option -report_label    "My report label" -report
{datasheet}
set_config_option -custom_logo  "http://myorg.com/img/
logo.gif@@75@@45@@My Custom Logo"
set_config_option -report_file_name  "My_Datasheet"
```

Generating the DataSheet Report in Batch

You can generate the *DataSheet* report in the batch mode by using the `gen_aggregate_report` command-line option, as shown below.

```
spyglass -gen_aggregate_report datasheet -config_file <cfg-
file> | -project <prj-file> [ -reportdir <dir>] [-DEBUG]
[ -LICENSEDEBUG ]
```

The details of various options are given in the following table:

Option Name	Description
-config_file	Specifies the name of the configuration file that contains a list of projects and run directories generated by batch console or GUI.
-project	Specifies the name of a project file. SpyGlass considers this option only if you have NOT specified the `-config_file` option. If you specify both the `-config_file` and `-project` options, SpyGlass ignores the `-project` option and considers the `-config_file` option.
-reportdir	(Optional) Specifies the directory in which the result files will be created. By default, Console considers the value of this option as the current working directory.
-DEBUG	(Optional) Prints useful debug messages on STDOUT. This information includes various details such as the current project accessing, time stamps at various stages, etc. Information printed on STDOUT is also dumped in the log file, datasheet.log.

The following example generates the *DataSheet* report for the project file, CUSB2_WRAP.prj:

```
spyglass -gen_aggregate_report datasheet -project
CUSB2_WRAP.prj -batch
```

Generating the Datasheet Report through a Project File

To generate the *DataSheet* report through a project file, use the following command in the project file:

```
set_option aggregate_report datasheet
```

To specify the directory in which you want to generate the *DataSheet* report, use the following command in the project file:

```
set_option aggregate_reportdir <report-directory-path>
```

NOTE: *If you do not specify the above command for a project-specific DashBoard report,* *the reports are generated in the <projectwdir>/<project>/<top>/html_reports directory by default.*

To specify a configuration file that contains a list of projects and run directories generated by batch console or GUI, use the following command in a project file:

```
set_option aggregate_report_config_file <config-file-path>
```

Viewing the DataSheet Report

Report files, such as datasheet.html, datasheet.csv, and IP-XACT files (both 1.2 and 1.4 versions) are generated in the html_reports directory.

Recommended Goals for Generating DataSheet Report

The *DataSheet* report is primarily intended for the developer of an IP, which is in the GuideWare2.0 block/rtl_handoff or GuideWare New_RTL/ rtl_handoff stage to verify completeness of RTL.

In addition, the *DataSheet* report can also be used to communicate design characteristics and quality to an IP consumer. The IP consumer can use the GuideWare2.0 soc/rtl_handoff or GuideWare IP_RTL goals to verify or confirm the incoming IP.

NOTE: *The DataSheet report is primarily designed to be used with RTL IP. It is not specifically designed for use with netlists or full chip designs.*

The following GuideWare-based goals are required to populate the *DataSheet* report:

```
GuideWare2.0 "block/rtl_handoff" (or) "soc/rtl_handoff"

    lint/design_audit
    cdc/cdc_verify
    constraints/sdc_gen
    constraints/sdc_audit
    txv_verification/fp_verification
    txv_verification/mcp_verification
    power/power_est_average
    dft/dft_scan_ready
    dft/dft_dsm_best_practice

GuideWare New_RTL/rtl_handoff Goals
    rtl_handoff/audit/datasheet_io_audit
    rtl_handoff/audit/block_profile
    rtl_handoff/cdc_verif/cdc_verif_base (or
    rtl_handoff/cdc_verif/cdc_verif)
    rtl_handoff/constraint/sdc_coverage
    rtl_handoff/txv_verification/fp_mcp_verification
    rtl_handoff/txv_verification/mcp_verification
    rtl_handoff/power/power_est_average
    rtl_handoff/dft_readiness/dft_scan_ready
    rtl_handoff/dft_readiness
    /dft_dsm_transition_coverage


GuideWare IP_RTL Goals
    ip_exploration/audit/datasheet_io_audit
    ip_exploration/audit/ip_rtl_profile
    ip_exploration/cdc_verif/cdc_verif_base
    ip_exploration/constraint/sdc_coverage
    ip_exploration/txv_verification/fp_verification
    ip_exploration/txv_verification/mcp_verification
    ip_adaptation/power/power_est_average
    ip_adaptation/dft_readiness/dft_scan_ready
    ip_adaptation/dft_readiness
    /dft_dsm_transition_coverage
```

NOTE: *If you are using the constraint/sdc_coverage goal in the GuideWare methodology, ensure to add the SDC_DataSheet rule to the goal run to fulfill IO delays in the IO Definitions table.*

The above list of GuideWare goals is only for reference purpose. You can use similar goal names for other design stages. Refer to the GuideWare and advance product documentation for more information on running the complete flow.

Generating the DataSheet Report by Using Non-GuideWare Flows

For non-GuideWare based flows, you can use the following list of SpyGlass rules to generate the *DataSheet* report.

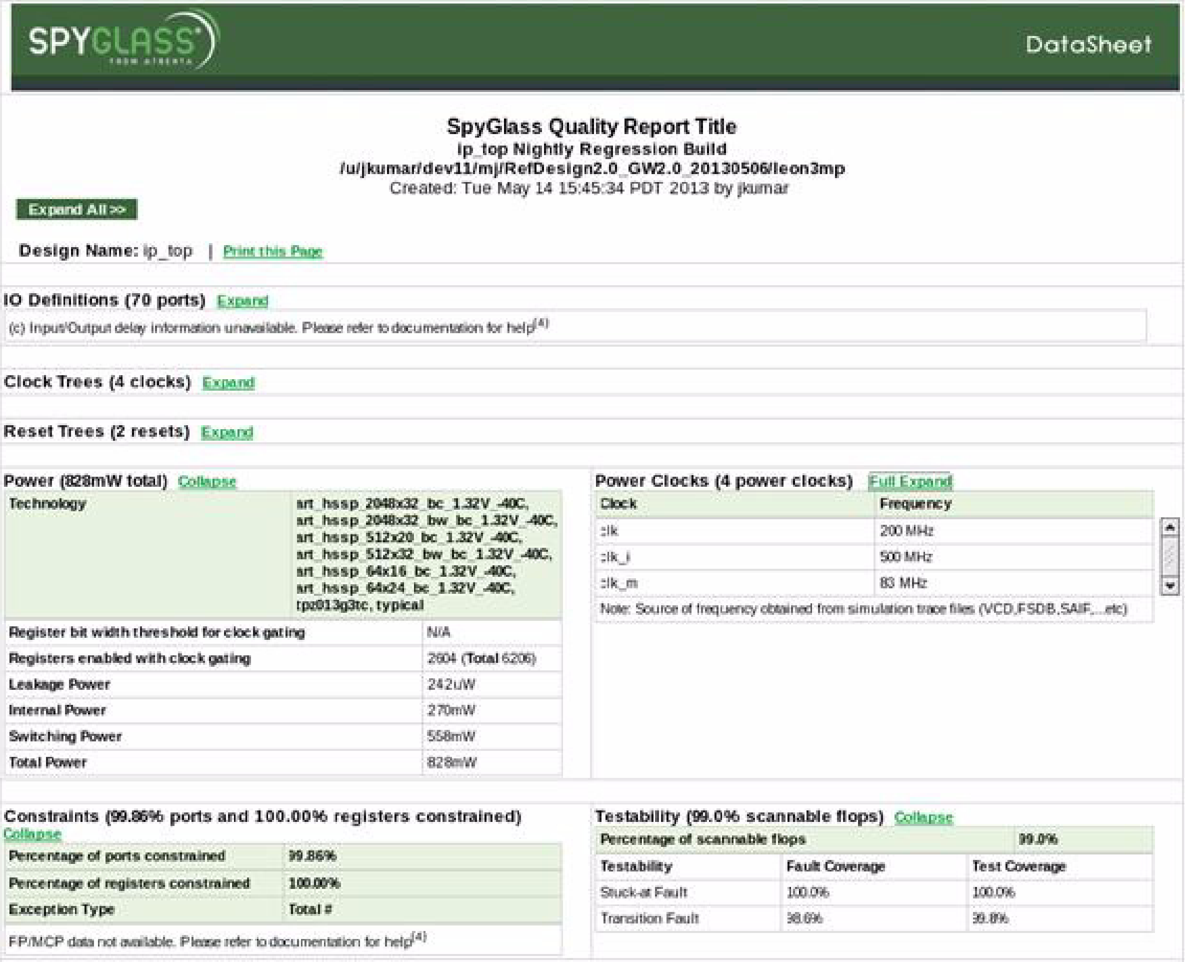
Section in the DataSheet Report	Rule Name	Product
IO Definitions	RegInputOutput-ML	SpyGlass moreLint Solution
	ReportPortInfo-ML	SpyGlass moreLint Solution
	PragmaComments-ML	SpyGlass moreLint Solution
	Ac_sync_group rules, that is: ● Ac_sync01 ● Ac_sync02 ● Ac_unsync01 ● Ac_unsync02	SpyGlass CDC Solution
	SDC_GenerateIncr	SpyGlass Constraints Solution
	SDC_DataSheet	SpyGlass Constraints Solution
Clock and Reset Tree	Clock_info15	SpyGlass CDC Solution
	Ac_sync_group rules, that is: ● Ac_sync01 ● Ac_sync02 ● Ac_unsync01 ● Ac_unsync02	SpyGlass CDC Solution
	Note: To populate this section in the Datasheet report, you must run either the Ac_sync_group rules or the other specified SpyGlass CDC solution rules mentioned above. If you run both these types of rules, preference is given to the Ac_sync_group rules and results of these rules are shown in the Datasheet report.	
Power and Power Clocks	PEPWR02	SpyGlass Power Estimate
Constraints	SDC_Coverage	SpyGlass Constraints Solution
	FP_Pass_Verif01	SpyGlass TXV Solution
	Txv_MCP_Warn05	SpyGlass TXV Solution

Section in the DataSheet Report	Rule Name	Product
Testability	Info_coverage	SpyGlass DFT Solution
	Info_transitionCoverage	SpyGlass DFT DSM Solution
Design Statistics and Black Boxes	Audit4Dump	SpyGlass audits Solution

Do not run the above rules from a single goal or a goal; run these rules as a part of their respective methodologies.

Details of the DataSheet Report

The following figure illustrates a sample *DataSheet* report:



SpyGlass Quality Report Title
ip_top Nightly Regression Build
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp
Created: Tue May 14 15:45:34 PDT 2013 by jkumar

Design Name: ip_top | Print this Page

IO Definitions (70 ports) Expand

(c) Input/Output delay information unavailable. Please refer to documentation for help[4]

Clock Trees (4 clocks) Expand

Reset Trees (2 resets) Expand

Power (828mW total) Collapse

Technology	srt_hssp_2048x32_bc_1.32V_-40C, srt_hssp_2048x32_bw_bc_1.32V_-40C, srt_hssp_512x20_bc_1.32V_-40C, srt_hssp_512x32_bw_bc_1.32V_-40C, srt_hssp_64x16_bc_1.32V_-40C, srt_hssp_64x24_bc_1.32V_-40C, tpz013g3tc, typical
Register bit width threshold for clock gating	N/A
Registers enabled with clock gating	2604 (Total 6206)
Leakage Power	242uW
Internal Power	270mW
Switching Power	558mW
Total Power	828mW

Power Clocks (4 power clocks) Full Expand

Clock	Frequency
:lk	200 MHz
:lk_i	500 MHz
:lk_m	83 MHz

Note: Source of frequency obtained from simulation trace files (VCD,FSDB,SAIF,...etc)

Constraints (99.86% ports and 100.00% registers constrained) Collapse

Percentage of ports constrained	99.86%
Percentage of registers constrained	100.00%
Exception Type	Total #

FP/MCP data not available. Please refer to documentation for help[4]

Testability (99.0% scannable flops) Collapse

Percentage of scannable flops		99.0%
Testability	Fault Coverage	Test Coverage
Stuck-at Fault	100.0%	100.0%
Transition Fault	98.6%	99.8%

FIGURE 257. Sample DataSheet Report

The *DataSheet* report contains different sections that provide different types of information in a tabular format.

To view the details of a particular section, expand that section by clicking the *Expand* option adjacent to that section. The *Expand* option expands a section by one level. If you want to view the entire section, click the *Full Expand* option.

You can also expand all the sections at once by clicking the *Expand All >>* button.

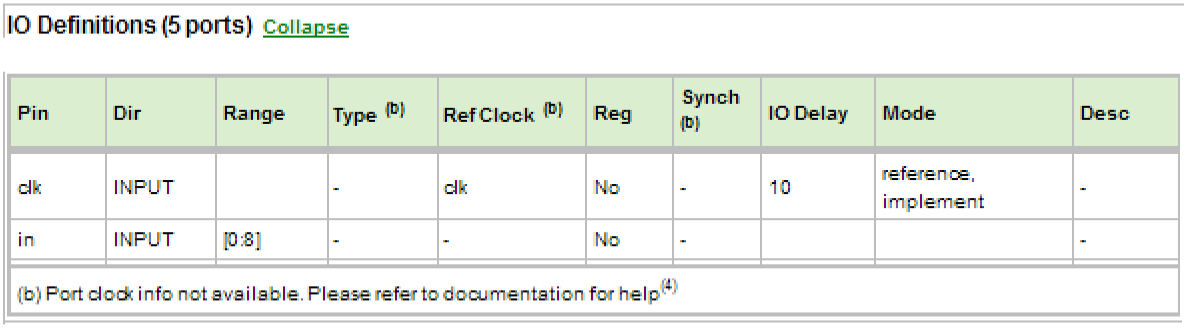
The *DataSheet* report contains the following sections:

IO Definitions	*Clock Trees*	*Reset Trees*	*Power*
Power Clocks	*Constraints*	*Testability*	*Design Statistics*
Black Boxes	*The DashBoard Report*		

Data in different sections of the *DataSheet* report is generated only when some specific goals are run. If you do not run the required goals or if the goal run does not generate any analysis data, the corresponding sections of the report are left blank.

IO Definitions

A sample of the *IO Definition* section is shown in the following figure:



IO Definitions (5 ports) Collapse

Pin	Dir	Range	Type (b)	Ref Clock (b)	Reg	Synch (b)	IO Delay	Mode	Desc
clk	INPUT		-	clk	No	-	10	reference, implement	-
in	INPUT	[0:8]	-	-	No	-			-

(b) Port clock info not available. Please refer to documentation for help(4)

FIGURE 258. DataSheet Report - IO Definition

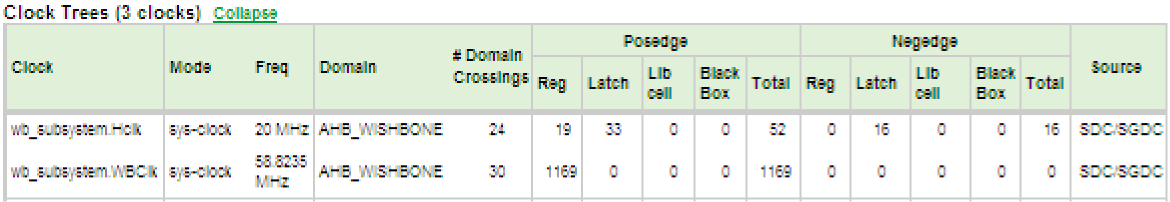
The details of each field in this table are described below:

Field Name	Description
Basic IO data which is populated by the GuideWare Audit flow goals including the optional goal, datasheet_io_audit	
Pin	Specifies the name of the pin

Dir	Specifies the direction of the pin
Range	Specifies the range of the pin. For example, 3:1.
Reg	Specifies whether the port is registered at boundary or not
Desc	Specifies the in-line comment that is extracted from the module/entity declaration or definition for Verilog and VHDL, respectively
Clock port information, which is populated by SpyGlass CDC Solution flow goals of GuideWare	
Type	Indicates inferred clock, reset, or other pins
Ref clock	Specifies the reference clock of the pin
Synch	Specifies whether the signal is internally synchronized or not
Columns displaying IO delay and mode information. This information is populated by SpyGlass Constraints solution flow goals by GuideWare.	
IO Delay	Specifies port input/output delay in the 'Min Rise: Min Fall : Max Rise : Max Fall' format. If the IO is a clock, it shows the 'clock period'.
Mode	Specifies if there are any modes set in SDC corresponding to IO delays

Clock Trees

A sample of the *Clock Trees* section is shown in the following figure:



Clock Trees (3 clocks) Collapse

Clock	Mode	Freq	Domain	# Domain Crossings	Posedge					Negedge					Source
					Reg	Latch	Lib cell	Black Box	Total	Reg	Latch	Lib cell	Black Box	Total	
wb_subsystem.Hclk	sys-clock	20 MHz	AHB_WISHBONE	24	19	33	0	0	52	0	16	0	0	16	SDC/SGDC
wb_subsystem.WBClk	sys-clock	58.8235 MHz	AHB_WISHBONE	30	1169	0	0	0	1169	0	0	0	0	0	SDC/SGDC

FIGURE 259. DataSheet Report - Clock Trees

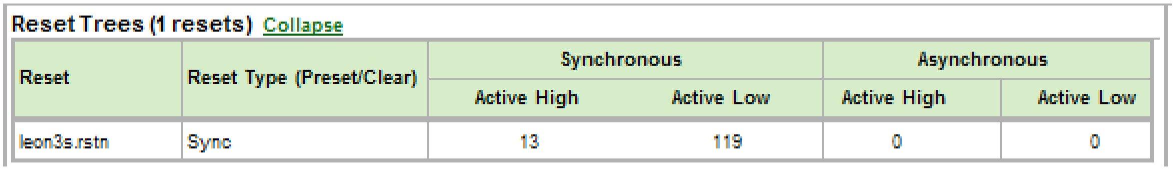
SpyGlass populates this section if you run the GuideWare goals of SpyGlass CDC solution.

The details of each field in this table are described below:

Field Name	Description
Clock	Specifies the name of the clock
Mode	Specifies the type of clock, that is, sys-clock, test-clock, or at-speed-test-clock
Freq	Specifies the clock frequency
Domain	Specifies the clock domain name
Domain Crossing	Specifies the number of domain crossings
Posedge	**Reg:** Specifies the number of flip-flops fed by clock as a positive edge
	Latch: Specifies the number of latches fed by clock as a posedge
	Lib cell: Specifies the number of sequential library cells fed by clock as a posedge
	Black Box: Specifies the number of black boxes fed by clock as a posedge
	Total: Specifies the total posedge count
Negedge	**Reg:** Specifies the number of flip-flops fed by clock as a negative edge
	Latch: Specifies the number of latches fed by clock as a negedge
	Lib cell: Specifies the number of sequential library cells fed by clock as a negedge
	Black Box: Specifies the number of black boxes fed by clock as a negedge
	Total: Specifies the total negedge count

Reset Trees

A sample of the *Reset Trees* section is shown in the following figure:



Reset Trees (1 resets) Collapse

Reset	Reset Type (Preset/Clear)	Synchronous		Asynchronous	
		Active High	Active Low	Active High	Active Low
leon3s.rstn	Sync	13	119	0	0

FIGURE 260. DataSheet Report - Reset Trees

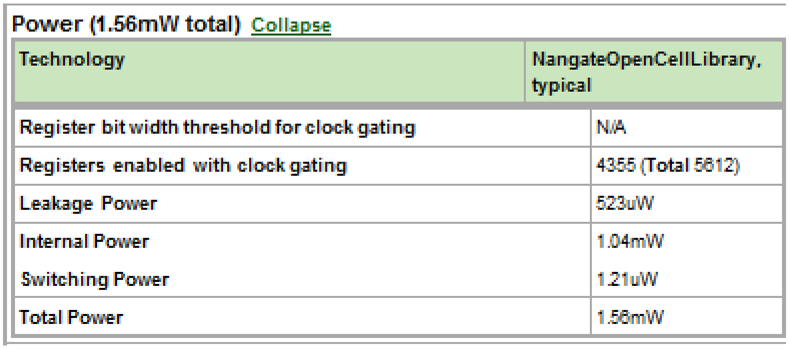
SpyGlass populates this section if you run the GuideWare goals of SpyGlass CDC Solution.

The details of each field in this table are described below:

Field Name	Description
Reset	Specifies the name of the reset
Reset Type	Specifies the reset type as `preset` or `clear`
Synchronous	**Active High:** Specifies the number of flip-flops that use it as an active high synchronous reset
	Active Low: Specifies the number of flip-flops that use it as an active low synchronous reset
Asynchronous	**Active High:** Specifies the number of flip-flops that use it as an active high asynchronous reset
	Active Low: Specifies the number of flip-flops that use it as an active low asynchronous reset

Power

A sample of the *Power* section is shown in the following figure:



Power (1.56mW total) Collapse

Technology	NangateOpenCellLibrary, typical
Register bit width threshold for clock gating	N/A
Registers enabled with clock gating	4355 (Total 5612)
Leakage Power	523uW
Internal Power Switching Power	1.04mW 1.21uW
Total Power	1.56mW

FIGURE 261. DataSheet Report - Power

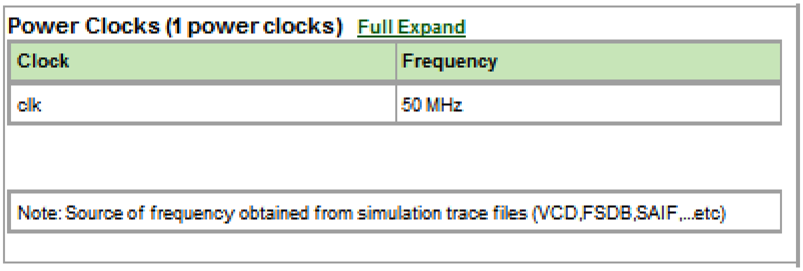
SpyGlass populates this section if you run the GuideWare Power flow goals of all the stages (`initial_rtl`, `detailed_rtl`, and `rtl_handoff`) of the `New_RTL` methodology.

The details of each field in this table are described below:

Field Name	**Description**
Technology	Specifies the name of the technology used
Register bit width threshold for clock gating	Specifies the register bit width threshold for clock gating
Registers enabled with clock gating	Specifies the registers enabled with clock gating
Leakage Power	Specifies the leakage power
Internal Power	Specifies the internal power
Switching Power	Specifies the switching power
Total Power	Specifies the total power

Power Clocks

A sample of the *Power Clocks* section is shown in the following figure:



Power Clocks (1 power clocks) Full Expand

Clock	Frequency
clk	50 MHz

Note: Source of frequency obtained from simulation trace files (VCD,FSDB,SAIF,...etc)

FIGURE 262. DataSheet Report - Power Clocks

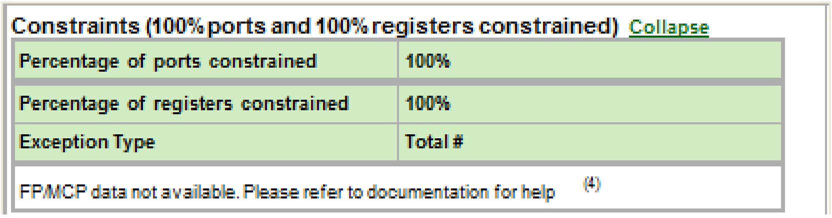
SpyGlass populates this section if you run the GuideWare Power flow goals of all the stages (`initial_rtl`, `detailed_rtl`, and `rtl_handoff`) of the `New_RTL` methodology.

The details of each field in this table are described below:

Field Name	Description
Clock	Specifies the name of the clock
Frequency	Specifies the frequency of the clock

Constraints

The sample of the *Constraints* section is shown in the following figure:



Constraints (100% ports and 100% registers constrained) Collapse

Percentage of ports constrained	100%
Percentage of registers constrained	100%
Exception Type	Total #

FP/MCP data not available. Please refer to documentation for help (4)

FIGURE 263. DataSheet Report - Constraints

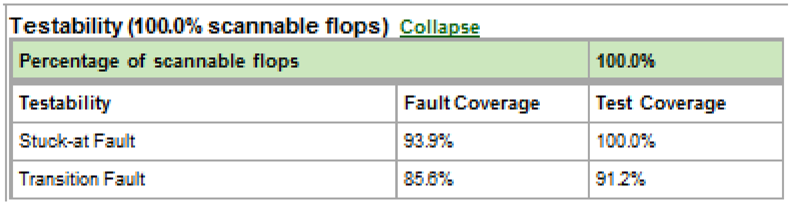
SpyGlass populates this section if you run the goals of SpyGlass Constraints Solution and SpyGlass TXV Solution.

The details of each field in this table are described below:

Field Name	Description
Percentage of ports constrained	Specifies the percentage of ports constrained
Percentage of registers constrained	Specifies the percentage of registers constrained
Exception Type	
False Paths (FP)	Specifies the number of false paths
Multi Cycle Paths (MCP)	Specifies the number of multi-cycle paths

Testability

The sample *Testability* section is shown in the following figure:



Testability (100.0% scannable flops) Collapse

Percentage of scannable flops		100.0%
Testability	Fault Coverage	Test Coverage
Stuck-at Fault	93.9%	100.0%
Transition Fault	85.6%	91.2%

FIGURE 264. DataSheet Report - Testability

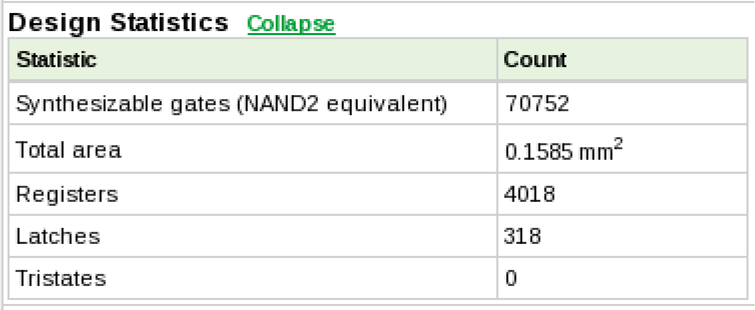
SpyGlass populates this section if you run the goals of SpyGlass DFT solution.

The details of each field in this table is described below:

Field Name	Description
Percentage of scannable flip-flops	Specifies the percentage of flip-flops that can be scanned
Testability	Contains a row for `Stuck-at Test` and `Transition Fault`
Fault coverage	Specifies `Stuck-at Test` and `Transition Fault` values for fault coverage
Test Coverage	Specifies `Stuck-at Test` and `Transition Fault` values for test coverage

Design Statistics

A sample *Design Statistics* section is shown in the following figure:



Design Statistics

Statistic	Count
Synthesizable gates (NAND2 equivalent)	70752
Total area	0.1585 mm^2
Registers	4018
Latches	318
Tristates	0

FIGURE 265. DataSheet Report - Design Statistics

The above section displays a table showing different types of statistic data and their counts.

In this table, data is populated if the audit/area policy rules are run. For example, on any design/test case, execute design_audit GuideWare goal. The design statistics table will show the respective data.

The details of each design statistics in this table are described below:

Statistic	Description
Synthesizable gates (NAND2 equivalent)	Specifies the size of synthesizable RTL logic measured in terms of the NAND2 equivalent gates. This metric does not include hard IPs and memories.
Total Area	Specifies the total size of the design including all design entities, such as synthesizable RTL logic, hard IPs, memories, and black boxes.
Registers	Specifies the total number of flip-flop instances in the design.
Latches	Specifies the total number of latch instances in the design.
Tristates	Specifies the total number of tristate instances in the design. These are the instances for which an output port can be set to drive a high-impedance signal.

Black Boxes

This section displays the number of ports for each black box. The details of each field of this table are described below:

Field Name	**Description**
Black Box Name	Specifies the name of the black box
Number of Ports	Specifies the number of ports

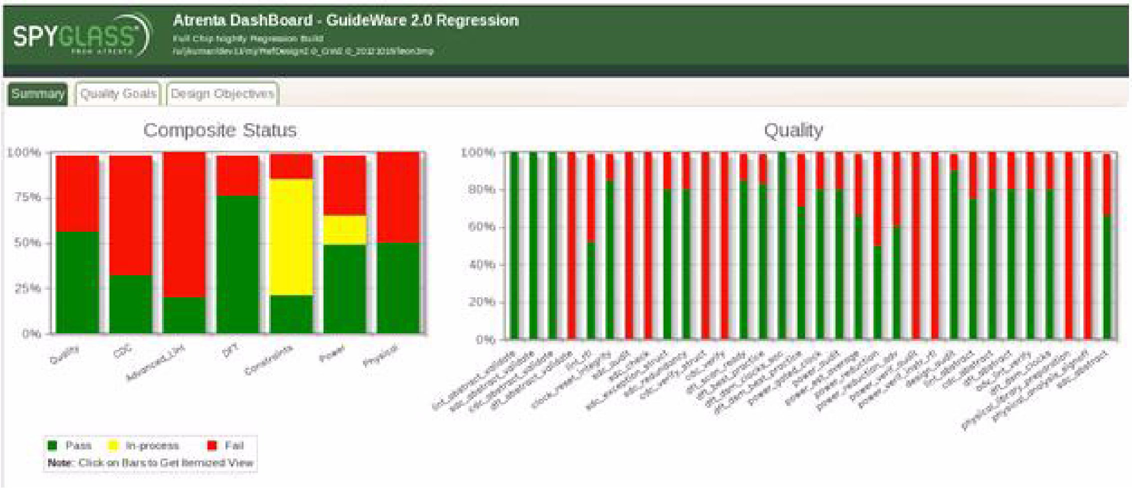
The DashBoard Report

The *DashBoard* report enables you to review the productivity and efficiency of different blocks of your design periodically.

By default, SpyGlass automatically generates the Dashboard report for the current project. You can disable report generation for the Dashboard report using the following command:

```
set_option disable_html_report {dashboard}
```

A sample Full Chip SoC *DashBoard* report is shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

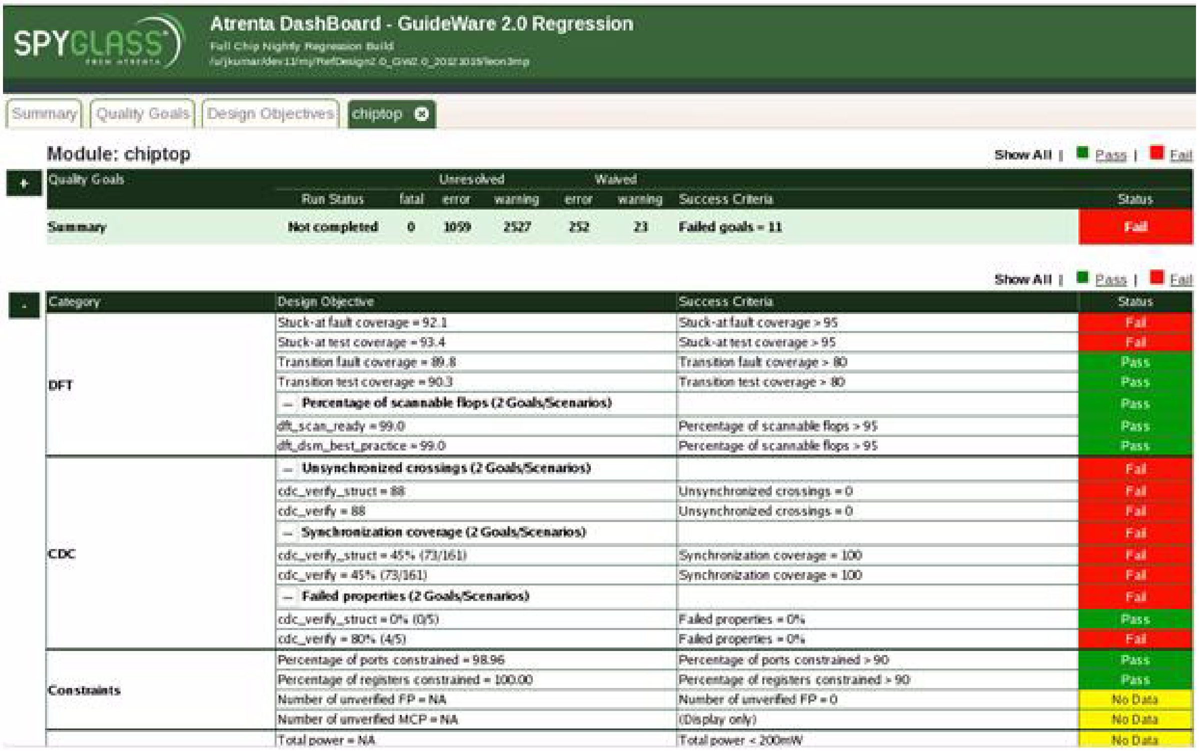
Full Chip Nightly Regression Build

Note: Click on Bars to Get Itemized View

FIGURE 266. The DashBoard Report

The above report enables you to evaluate present risks involved in different design objectives, such as clocks and power-related objectives of various blocks and view trend variations over a period of time. For details, see *Details of the DashBoard Report*.

A sample individual module *DashBoard* report is shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build

/u/jkumar/dev11/my/RefDesign2.0_GW2.0_20121015/leon3mp

Module: chiptop

Show All | 🟩 Pass | 🟥 Fail

Quality Goals		Unresolved			Waived			
	Run Status	fatal	error	warning	error	warning	Success Criteria	Status
Summary	**Not completed**	**0**	**1059**	**2527**	**252**	**23**	**Failed goals = 11**	**Fail**

Show All | 🟩 Pass | 🟥 Fail

Category	Design Objective	Success Criteria	Status
DFT	Stuck-at fault coverage = 92.1	Stuck-at fault coverage > 95	Fail
	Stuck-at test coverage = 93.4	Stuck-at test coverage > 95	Fail
	Transition fault coverage = 89.8	Transition fault coverage > 80	Pass
	Transition test coverage = 90.3	Transition test coverage > 80	Pass
	– **Percentage of scannable flops (2 Goals/Scenarios)**		Pass
	dft_scan_ready = 99.0	Percentage of scannable flops > 95	Pass
	dft_dsm_best_practice = 99.0	Percentage of scannable flops > 95	Pass
CDC	– **Unsynchronized crossings (2 Goals/Scenarios)**		Fail
	cdc_verify_struct = 88	Unsynchronized crossings = 0	Fail
	cdc_verify = 88	Unsynchronized crossings = 0	Fail
	– **Synchronization coverage (2 Goals/Scenarios)**		Fail
	cdc_verify_struct = 45% (73/161)	Synchronization coverage = 100	Fail
	cdc_verify = 45% (73/161)	Synchronization coverage = 100	Fail
	– **Failed properties (2 Goals/Scenarios)**		Fail
	cdc_verify_struct = 0% (0/5)	Failed properties = 0%	Pass
	cdc_verify = 80% (4/5)	Failed properties = 0%	Fail
Constraints	Percentage of ports constrained = 98.96	Percentage of ports constrained > 90	Pass
	Percentage of registers constrained = 100.00	Percentage of registers constrained > 90	Pass
	Number of unverified FP = NA	Number of unverified FP = 0	No Data
	Number of unverified MCP = NA	(Display only)	No Data
	Total power = NA	Total power < 200mW	No Data

FIGURE 267. Sample Dashboard Report

NOTE: *While including multiple projects, you must compile data for all different top-level design units before including such projects in the report.*

This section explains the following topics:

- *Licensing Requirements*
- *Browser Compatibility*
- *Generating Dashboard Report*
- *Viewing the DashBoard Report*
- *Details of the DashBoard Report*

- *Customizing Report*
- *Managing Reports*
- *Switching to the Old Dashboard Report*

Licensing Requirements

SpyGlass *DashBoard* report is a licensed capability and requires the license feature, `dashboard`. Please contact SpyGlass Support (*spyglass_support@synopsys.com*), if you need this license.

Browser Compatibility

The Atrenta DashBoard is compatible with the following Web browsers:

- Firefox 2.0.0.20 (UNIX or Windows) or higher
- IE 8, 9, 10, and 11 on Windows 7

Setup Instructions for Google Chrome

1. Right-click the Google Chrome shortcut and click Properties from the shortcut menu.

   The Google Chrome Properties dialog box is displayed.
2. Specify following in the Target Box field:

   ```
   C:\Users\USERNAME\AppData\Local\Google\Chrome\Application\
   chrome.exe --allow-file-access-from-files
   ```
3. Press Apply/OK.
4. Run the DashBoard Report.

Generating Dashboard Report

You can generate the Dashboard report in one of the following ways:

- *Generating the DashBoard Report through Project File*
- *Generating the DashBoard Report in Batch*
- *Generating Dashboard Report in GUI*

- *Creating a Configuration File*
- *Creating the Success Criteria File*

Generating the DashBoard Report through Project File

To generate the *DashBoard* report through a project file, use the following command in the project file:

```
set_option aggregate_report dashboard
```

To specify the directory in which you want to generate the *DashBoard* report, use the following command in the project file:

```
set_option aggregate_reportdir <report-directory-path>
```

NOTE: *If you do not specify the above command for a project-specific DashBoard report, the reports are generated in the <projectwdir>/<project>/<top>/html_reports directory by default.*

To specify a configuration file that contains a list of projects and run directories generated by batch console or GUI, use the following command in a project file:

```
set_option aggregate_report_config_file <config-file-path>
```

Generating the DashBoard Report in Batch

You can generate the DashBoard report in the batch mode by using the gen_aggregate_report command-line option, as shown below:

```
spyglass -gen_aggregate_report dashboard -config_file
<cfgfile> | -project <prj-file> [ -reportdir <dir>] [-DEBUG]
[ -LICENSEDEBUG ]
```

The details of various options are given in the following table:

Option Name	Description
-config_file	Specifies the name of the configuration file that contains a list of projects and run directories generated by batch console or GUI.
-project	Specifies the name of a project file. SpyGlass considers this option only if you have NOT specified the `-config_file` option. If you specify both the `-config_file` and `-project` options, SpyGlass ignores the `-project` option and considers the `-config_file` option.
-reportdir	(Optional) Specifies the directory in which the result files will be created. By default, Console considers the value of this option as the current working directory.
-DEBUG	(Optional) Prints useful debug messages on STDOUT. This information includes various details such as the current project accessing, time stamps at various stages, etc. Information printed on STDOUT is also dumped in the log file, dashboard.log.

The following example generates the *DashBoard* report for the project file, CUSB2_WRAP.prj:

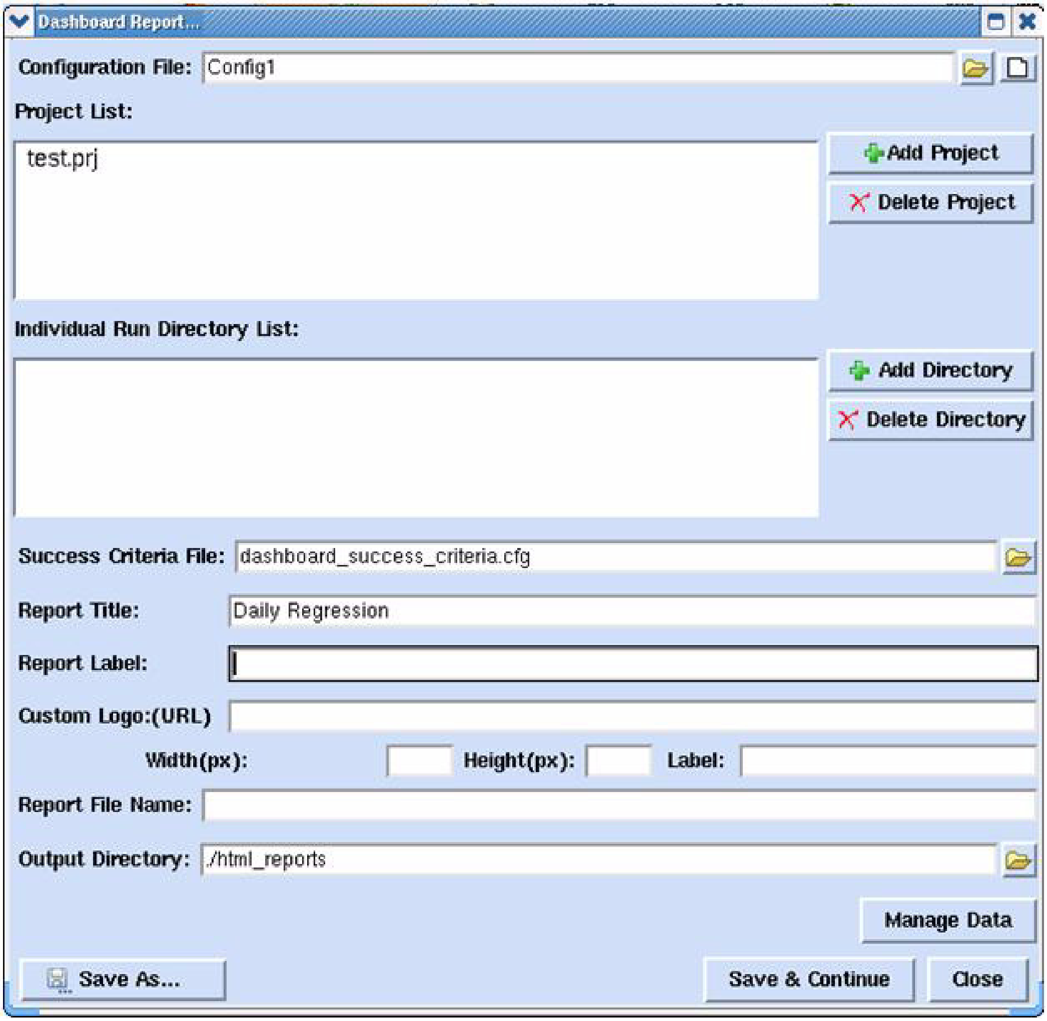
```
spyglass -gen_aggregate_report dashboard -project
CUSB2_WRAP.prj -batch
```

Generating Dashboard Report in GUI

To generate the *DashBoard* report, perform the following steps:

1. Select *Tools -> Dashboard Report* menu option.

   The *Dashboard Report* dialog appears, as shown in the following figure:



Configuration File: Config1

Project List: test.prj

Success Criteria File: dashboard_success_criteria.cfg

Report Title: Daily Regression

Output Directory: ./html_reports

FIGURE 268. Configuring Dashboard Report

2. In the *Dashboard Report* dialog, click the (📂) button to select the required configuration file.

   Alternatively, you can create a new configuration file, and add the required details in it, such as project files and run directories. For details, see *Creating a Configuration File*.

3. Specify a Success Criteria File by clicking on the (📂) button next to the Success Criteria File option. For details, see *Creating the Success Criteria File*.

4. Specify the details, such as report title, report label, custom logo, and link report.

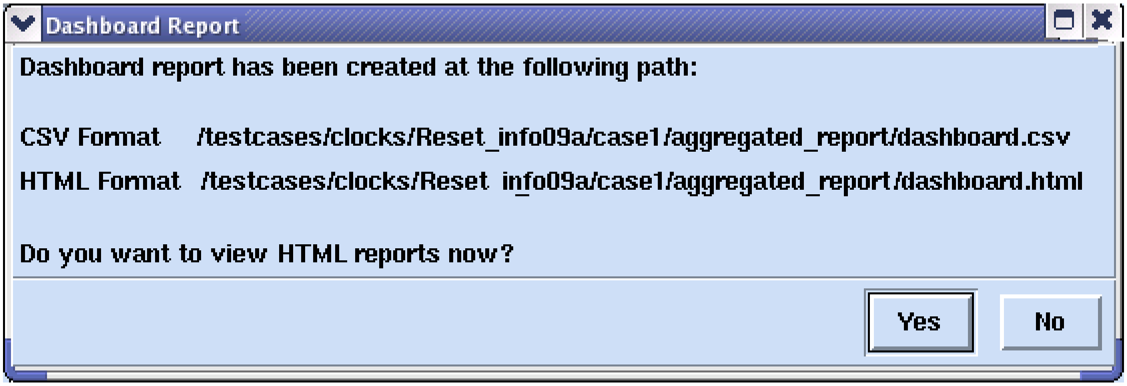
5. Specify the report output directory in the *Output Directory* textbox.

   If the specified directory does not exist, SpyGlass creates it.

   If you do not specify any directory, the report is saved at a default path. For details, see *Default Paths of Aggregated Reports*.

6. Click the *Save & Continue* button.

   This step generates the *DashBoard* report, and the following dialog appears:



Dashboard report has been created at the following path:

CSV Format   /testcases/clocks/Reset_info09a/case1/aggregated_report/dashboard.csv

HTML Format   /testcases/clocks/Reset_info09a/case1/aggregated_report/dashboard.html

Do you want to view HTML reports now?

FIGURE 269. Dashboard Report - Dialog Box

7. In the above dialog, select the *Yes* button if you want to view the HTML report in the browser window. Otherwise, click the *No* button.

Creating a Configuration File

To create a configuration file, perform the following steps:

1. Enter the name of the configuration file in the *Configuration File* textbox.
2. Click the *Add Project* button to add a project file in the configuration file being created.

   The *Select File(s)* dialog appears.
3. In the *Select File(s)* dialog, select the ‘required project file and click the *OK* button.

   The selected project file appears in the *Project List* section of the *Dashboard Report* dialog.
4. Repeat steps *2* and *3* to add more project files.
5. Click the *Add Directory* button if you want to add individual run directories in the configuration file being created.

   The *Select Directory* dialog appears.
6. In the *Select Directory* dialog, select the required directory and click the *OK* button.

   The selected directory gets added in the *Individual Run Directory List* section of the *Dashboard Report* dialog.
7. Repeat steps *5* and *6* to add more directories.
8. Specify the success criteria file in the *Success Criteria File* textbox.

   This file contains the success criteria data set by a design expert. For details on this file, refer to the *Creating the Success Criteria File* topic.
9. Specify report title in the *Report Title* textbox.

   Report title can be used to specify the top-level report name. For example, you can specify your company name as the report title.

   When you specify a report title, the following command is generated in the configuration file:

   ```
   REPORT_TITLE_DASHBOARD <title>
   ```
10. Specify report label in the *Report Label* textbox.

    Report label refers to an additional description of the report. For example, you may specify the report label as *Regression report created by the XYZ group*.

When you specify the report label, the following command is generated in the configuration file:

```
REPORT_LABEL_DASBHBOARD <label>
```

11. Click the *Save & Continue* button.

    This step generates the *DashBoard* report.

Alternatively, you can create a new configuration file by clicking the □ button.

Sample Configuration File

A sample configuration file is shown below:

```
############################################################
# Dashboard Configuration File
############################################################

# list of modules to include in the report
/proj/path/to/moduleA.prj

# additional modules prj files can be listed here. This will
# result in a report showing multiple modules. Multi-module
# reports are ideas for full chip project reports.
# /proj/path/to/moduleB.prj
# Specify the file which contains the dashboard success
# (pass/fail) criteria
SUCCESS_CRITERIA_FILE_PATH /proj/path/to/
success_criteria.cfg

# Customize the report title and label
REPORT_TITLE_DASHBOARD "Project A Nightly Regression Report"
REPORT_LABEL_DASHBOARD "Today's Date - Regression Directory:
"CUSTOM_LOGO_DASHBOARD http://myorg.com/img/
logo.gif@@75@@45@@My Custom Logo
REPORT_FILE_NAME_DASHBOARD "My_DashBoard"
```

Tcl Format Support in the Configuration File

SpyGlass provides you with a Tcl format to edit the configuration file. You can use the same configuration file for both the DataSheet and DashBoard reports for different projects. This makes easy to setup/create SoC configuration file.

NOTE: *You can specify the configuration file in <key>-<value> based text file format or in the below explained TCL format. However, you can not use the same file for both the formats.*

Following Tcl commands are used in the configuration file:

- *aggregate_projects*
- *set_config_option*
- *set_success_criteria_file*

aggregate_projects

This command specifies list of projects and work directories that you want to include in the specified report. Following is the syntax of the *aggregate_projects* command:

```
aggregate_projects -project {<project_list>} -dir
{<directory_list>} [-report {<report_list>}]
```

Here, `-report` is an optional argument and you can set specific reports for different projects, while aggregation. That is, if you set the -report {dashboard} option, then the project is added for the DashBoard report and skips if the same configuration file is used for the DataSheet report.

Now, assume that you want to include Project1.prj and Project2.prj for both the Datasheet and Dashboard reports. Also, assume that you to include Project3.prj only for the DashBoard report. To do so, specify the following commands:

```
aggregate_projects -project {Project1.prj Project2.prj}
```

```
aggregate_projects -project {Project3.prj} -report
{dashboard}
```

If you set -report {dashboard}, then the project is added for the DashBoard report and skips if the same configuration file is used for the DataSheet report.

Following is the sample usage of this command:

```
foreach module $env(MODULE_LIST) {
  set ::env(MODULE) $module
  foreach variant $env(HARDWARE_LIST) {
          set ::env(HARDWARE) $variant
  //Add project that may typically uses MODULE & HARDWARE
inside the project file
          aggregate_projects -project {project.prj} -report
{dashboard}
  }
}
```

set_config_option

This command supports different -<key> <values> combinations that are required for configuring the reports. Following is the syntax of this command:

```
set_config_option -<key> <value> [-report {<report_list>}]
```

You can specify one of the following values to the <key> option:

- **report_title**: Specifies the report title
- **report_label**: Specifies the report label
- **custom_logo**: Specifies the custom logo on the report
- **report_file_name**: Specifies the output file name

The -report option is optional and you can set specific reports to use these config options.

Following is the sample usage of this command:

```
set_config_option -report_title   "My report  title" -report
{dashboard}
```

```
set_config_option -report_label   "My report label"  -report
{dashboard}
```

```
set_config_option -custom_logo  "http://myorg.com/img/
logo.gif@@75@@45@@My Custom Logo"
```

```
set_config_option -report_file_name  "My_DashBoard"
```

set_success_criteria_file

This command is specific to DashBoard and it sets success criteria file path for DashBoard generation. The syntax of this command is given below:

```
set_success_criteria_file <file_path>
```

Following is the sample usage of this command:

Sample usage of this command:

```
set_success_criteria_file  /proj/path/to/
success_criteria.cfg
```

Sample Configuration File

Following is a sample configuration file to configure the Dashboard report using the above explained commands:

```
#Add input project files
aggregate_projects -project {/proj/path/module1.prj /proj/
path/module2.prj}

#Add config options
set_config_option -report_title    "My report title" -report
{dashboard}
set_config_option -report_label    "My report label" -report
{dashboard}
set_config_option -custom_logo  "http://myorg.com/img/
logo.gif@@75@@45@@My Custom Logo"
set_config_option -report_file_name  "My_DashBoard"

#Add Success criteria options
set_success_criteria_file  /<path>/success_criteria.tcl
```

Creating the Success Criteria File

The success criteria file determines whether a particular design objective, such as clocks and power-related objectives, meet the specified criteria.

NOTE: *You can copy the sample success criteria file from the following path:*

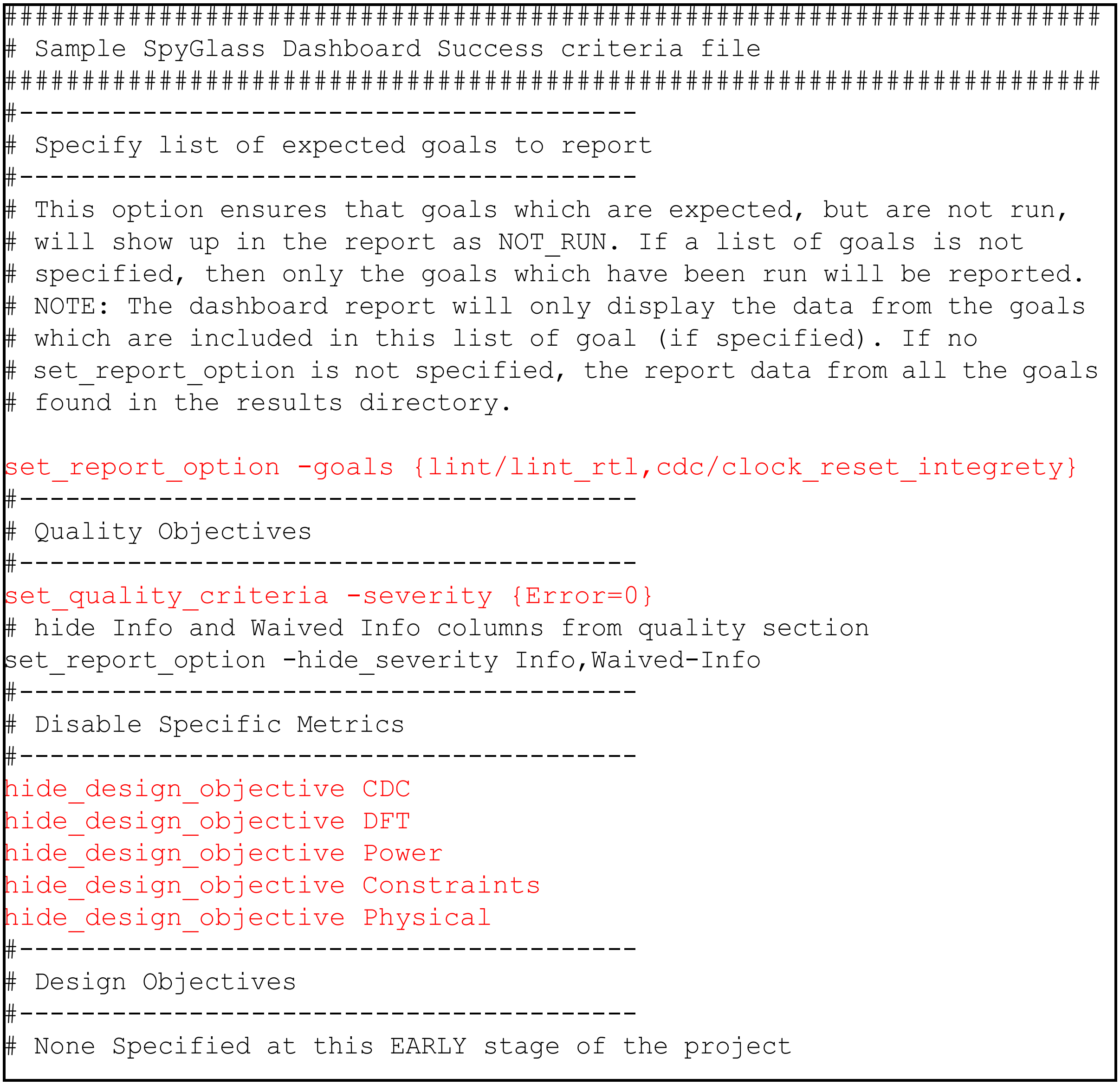
```
$SPYGLASS_HOME/auxi/dashboard_criteria_template
```

Based on your requirement, you can modify this file instead of writing it from scratch.

You can configure the Dashboard report to track and measure design requirements using the Success Criteria file. For example, at the beginning of a design project, running detailed CDC, DFT, and Power checks is not required. Therefore, you can specify only the goals required at this stage of development, such as, lint goals in the Success Criteria File. You can also configure the report for less strict success criteria, such as, fixing only the FATAL and ERRORS type violations for instances. As the design progresses, you can include additional goals and increase the success criteria, accordingly. Figure x and Figure y illustrate sample success criteria files for early design and advanced design scenarios.

Based on the degree to which a particular design objective meets the success criteria, the *Pass/Fail Status* column is populated with appropriate values: *Pass*, *Fail*, or *Unknown*.

Following figure illustrates a sample success criteria file for an early design stage:



```
#####################################################################
# Sample SpyGlass Dashboard Success criteria file
#####################################################################
#---------------------------------------
# Specify list of expected goals to report
#---------------------------------------
# This option ensures that goals which are expected, but are not run,
# will show up in the report as NOT_RUN. If a list of goals is not
# specified, then only the goals which have been run will be reported.
# NOTE: The dashboard report will only display the data from the goals
# which are included in this list of goal (if specified). If no
# set_report_option is not specified, the report data from all the goals
# found in the results directory.

set_report_option -goals {lint/lint_rtl,cdc/clock_reset_integrety}
#---------------------------------------
# Quality Objectives
#---------------------------------------
set_quality_criteria -severity {Error=0}
# hide Info and Waived Info columns from quality section
set_report_option -hide_severity Info,Waived-Info
#---------------------------------------
# Disable Specific Metrics
#---------------------------------------
hide_design_objective CDC
hide_design_objective DFT
hide_design_objective Power
hide_design_objective Constraints
hide_design_objective Physical
#---------------------------------------
# Design Objectives
#---------------------------------------
# None Specified at this EARLY stage of the project
```

Tcl Commands Used in the Success Criteria File

You can specify the following Tcl commands in the success criteria file:

- *set_design_objective*
- *hide_design_objective*
- *set_quality_criteria*
- *set_report_option*

set_design_objective

This command specifies the design objective on which success criteria is being applied. The syntax of this command is given below:

```
set_design_objective
<CDC|Power|DFT|Constraints|advance_lint>
-criteria {<criteria1>,<criteria2>,…}
[ -block {<block-name1>,<block-name2>,…} ]
[ -goal {<goal1>,<goal2>,…} ]
```

The following table describes the details of various arguments of this command:

Argument	**Description**

-criteria	Specifies the success criteria.
	For example, you can specify the following criteria for the Power design objective:
	`-criteria {switching_power<50uW, total_power<80uW,leakage_powe<20nW}`
	For information on other supported items, see *Variables Used in the Success Criteria File*.
	Following operators are accepted in the mathematical expressions:
	`<=, >= , >, <, !=, =`
	The criteria that are mentioned as coverage/Percentage are percentage (%) numbers and the rest are product identified numeric values.
	While validating the items that are represented in units, such as, power, frequency, and timing, unit conversion is considered, if the user-specified criteria and the product reported are in different units. However, if the specified unit does not match, numeric value is considered.
	Following units are supported in unit conversion:
	• **Power**: W (watt), dW (deciwatt), cW (centiwatt), mW (milliwatt), uW (microwatt), nW (nanowatt), pW (picowatt)
	• **Frequency**: Hz (Hertz), kHz or KHz (kilo hertz), MHz (Mega hertz), GHz (Giga hertz), THz (Tera hertz)
	• **Time**: ps (pico second), ns (nano second)
	If you do not want to compare the value produced by SpyGlass analysis for certain objectives, but just want to show them in the report, specify the success criteria value as `display_only`, as shown in the following example:
	`set_design_objective Power -criteria {switching_power=display_only,total_power=display_only ,leakage_power=display_only}`

Argument	Description
-block	Specifies a comma-separated list of blocks on which the specified criteria is applicable. If you do not specify this argument, SpyGlass applies the specified criteria on all the blocks in the design.
-goal	Specifies a comma-separated list of goals or scenarios from which data of the specified design objective criteria should be picked. For details, see *Setting Success Criteria Values to Different Goals and Scenarios*. However, if you do not run the goals specified by this argument or if there are any goals explicitly set for the report by using *set_report_option* command but these goals are not subset of that explicit list, SpyGlass does not include results of such goals in the design objective trend.

hide_design_objective

This command removes the specified design objective, such as `CDC`, `Power`, `DFT`, and `Constraints` from the *DashBoard* report.

The following is the syntax of using this command:

```
hide_design_objective <objectives> [-item {<item-list>}]
[-top {<top-list>}]
```

Where:

- *`<objectives>`* refers to a comma-separated list of design objectives to be removed.
- *`<item-list>`* refers to the items to be hidden for a design objective. Use this argument if within a design objective, you want to hide items that are inappropriate for all/some blocks.
- *`<top-list>`* refers to a list of design unit names.

The following command hides the `CDC` and `Power` design objectives:

```
hide_design_objective CDC,Power
```

The following command hides the `switching_power` and `internal_power` items of the `Power` design objective for all top design units:

```
hide_design_objective Power -item
{switching_power,internal_power}
```

The following command hides the `switching_power` and `internal_power` items of the `Power` design objective for the `top1` design unit:

```
hide_design_objective Power -item
{switching_power,internal_power} -top top1
```

NOTE: *If you hide all the design objectives, the Design Objectives table does not appear in the report. In this case, the report only displays the Quality Goals table.*

set_quality_criteria

This command sets criteria to qualify for the `Pass` status. The syntax of this command is given below:

```
set_quality_criteria
-severity {<criteria1>,<criteria2>,…}
[-goal {<goal1>,<goal2>,…}]
[-top {<block1>,<block2>,…}]
```

The following table describes the details of various arguments of this command:

Argument	Description
-severity	Specifies the criteria to qualify for the Pass status, as shown in the following example: -severity {Fatal=0,Error=0, Warning<1000,Waived-Error=0} Valid severities accepted in this category are based on SpyGlass reported severity class messages that can be waived. SpyGlass accepts following case-insensitive labels: ● Fatal, Error, Warning, Info (SpyGlass reported severity classes) ● Waived-Error, Waived-Warning, Waived-Info (severity class messages that can be waived) In the mathematical expressions, following operators are accepted: `<=, >= , >, <, !=, =` The values set in these expressions are number of SpyGlass reported (or) waived severity message counts.
-goal	(Optional) Specifies a comma-separated list of goals on which the specified criteria should be applied. If you do not specify this argument, SpyGlass applies the specified criteria on all goals. The following example shows the usage of this argument: `set_quality_criteria -severity {Error<5,Warning<1000,Waived-Error=0} -goal {goal1,goal2}`
-top	(Optional) Specifies a comma-separated list of blocks on which the specified criteria for specified goals should be applied. If you do not specify this argument, SpyGlass applies the specified criteria on all blocks. The following example shows the usage of this argument: `set_quality_criteria -severity {Error<5,Warning<1000,Waived-Error=0} -goal {goal1,goal2} -top {block1,block2}`

set_report_option

The following points describe the details of this command.

- This command specifies options to filter results in the *DashBoard* report. The syntax of this command is given below:

```
set_report_option
```

```
[ -goals {<goal1>,<goal2>,…} ]
[-top {<block_name1>,<block_name2>,…}]
```

The following table describes the details of various arguments of this command:

Argument	**Description**
-goals	Specifies a comma-separated list of goals for which results should be displayed in the report. By default, the report contains result of all the goals from respective projects.
-top	(Optional) Specifies a comma-separated list of blocks. If you do not specify this argument, all the blocks are considered.

- This command specifies scenarios to be considered in the *DashBoard* report.

  If you create multiple scenarios for a goal, results of all these scenarios are displayed in the *DashBoard* report by default.

  However, you can specify scenarios that you want to consider in the report by using the `-goals` argument of the `set_report_option` command.

  For example, if you want to include results of the `test1` and `test2` scenarios of the `G1` goal, specify the following command in the success criteria file:

  ```
  set_report_option -goals { G1@test1,G1@test2 }
  ```

  In this case, results for only `test1` and `test2` scenarios are displayed as two entries in the quality section of the *DashBoard* report even if there are other scenario results present for the `G1` goal.

  Now consider that you specify the following command in the success criteria file:

  ```
  set_report_option -goals { G1 }
  ```

  In this case, all results from different scenario runs, including the default scenario run, if present, are displayed for the `G1` goal.

- This command hides the specified severity column in the report. Following is the syntax for hiding the specified severity column:

  ```
  set_report_option
  -hide_severity <Warning|Info|Waivers|Waived-Error|Waived-
  Warning|Waived-Info>
  ```

  You can specify a comma-separated list of severities in the `-hide_severity` option to hide the columns of those severities from the report.

  For example, you can hide columns for *Warning* and *Info* severities by specifying the following command:

  ```
  set_report_option -hide_severity Warning,Info
  ```

  To the column of *Info* severity only, specify the following command:

  ```
  set_report_option -hide_severity Info
  ```

- This command shows the specified severity column in the report. The following is the syntax for showing the specified severity column:

  ```
  set_report_option
  -show_severity <Warning|Info|Waivers|Waived-Error|Waived-
  Warning|Waived-Info>
  ```

  For example, you can show columns for the *Info* severity by specifying the following command:

  ```
  set_report_option -show_severity Info
  ```

- This command specifies whether results of a goal from multiple methodology stages should be accumulated as one entry point in the *Dashboard* report.

  The following is the syntax to accumulate results of multiple runs of the same goal as one entry in the *Dashboard* report:

  ```
  set_report_option -combine_stages 1
  ```

  For example, you can accumulate results of the initial_rtl/lint/connectivity and detailed_rtl/lint/connectivity goal as one entry in the *Dashboard* report by specifying the above command in the success criteria file.

  By default, the value of the `-combine_stages` argument is `0`.

- This command specifies whether the severity counts displayed in the *Quality Goals tab* should create links to display details of the rule messages on selection.

  Following is the syntax to create links to the severity counts.

  ```
  set_report_option -link_goal_messages 1
  ```

Sourcing a Success Criteria File in another File

You can source a success criteria file in another success criteria file.

For example, if you want to specify some settings in a single success criteria file but want to keep all global settings, which work for all design blocks in another file, you can source that global file in the local file. This is shown in the following example:

```
#Local criteria file local.tcl

source global.tcl

<commands-in-local-file>
```

Now if you set the local.tcl file as the success criteria file in the report configuration file, this local file will include all global settings from the global.tcl file. In addition, settings specified in the local.tcl file would overwrite similar settings specified in the global.tcl file.

Default Success Criteria File

SpyGlass sets some technology variables to factory default criteria values in the $SPYGLASS_HOME/auxi/dashboard_default_criteria file.

These values are used while populating the *DashBoard* report when you run a technology but do not specify any success criteria information.

Variables Used in the Success Criteria File

You can specify a success criteria by setting appropriate values for different design objective variables in the success criteria file.

The following table displays the variables that you can specify in the success criteria file:

Design Objective	Variable Name	Description	Default/ Optional
CDC	synchronization_coverage	Synchronization coverage	Default
	cdc_failed_properties	Failed properties	Default
	cdc_partial_proven_properties	Partially-proven properties	Optional
	cdc_average_depth	Average depth of partially-proven properties	Optional
	cdc_minimum_depth	Minimum depth of partially-proven properties	Optional
Power	switching_power	Switching Power	Default
	internal_power	Internal Power	Default
	leakage_power	Leakage Power	Default
	total_power	Total Power	Default
DFT	stuck_at_fault	Stuck at fault coverage	Default
	stuck_at_test	Stuck at test coverage	Default
	transition_fault	Transition fault coverage	Default
	transition_test	Transition test coverage	Default
	scannable_flops	Percentage of scannable flops	Default
Constraints	unverified_fp	Number of unverified FP	Default
	unverified_mcp	Number of unverified MCP	Default
	ports_constrained	Percentage of ports constrained	Default
	registers_constrained	Percentage of registers constrained	Default

Design Objective	Variable Name	Description	Default/ Optional
Advanced_Lint	advanced_lint_failed_properties	Failed properties	Default
	advanced_lint_partial_proven_properties	Partially proven properties	Optional
	advanced_lint_average_depth	Average depth of partially proven properties	Optional
	advanced_lint_minimum_depth	Minimum depth of partially proven properties	Optional
	maximum_cyclomatic_complexity	Maximum cyclomatic complexity	Default

Design Objective	Variable Name	Description	Default/ Optional
Physical	Latches	Latches	Optional
	Registers	Registers	Default
	Synthesizable_gates_(NAND2_equivalent)	Synthesizable gates (NAND2 equivalent)	Default
	Total_area	Total area	Default
	Tristates	Tristates	Optional
	Number_of_congested_module_instances	Number of congested module instances	Default
	Top_module_congestion	Top module congestion	Optional
	Maximum_logic_levels_in_core	Maximum logic levels in core	Optional
	Maximum_logic_levels_on_periphery	Maximum logic levels on periphery	Optional
	Number_of_timing_paths_failing_in_core	Number of timing paths failing in core	Default
	Number_of_timing_paths_failing_on_periphery	Number of timing paths failing on periphery	Default
	Timing_slack_in_core	Timing slack in core	Optional
	Timing_slack_on_periphery	Timing slack on periphery	Optional
	Floorplan_timing_slack_in_core	Floorplan timing slack in core	Optional
	Floorplan_timing_slack_on_periphery	Floorplan timing slack on periphery	Optional

If you run a technology (data available), the default items appear in the report even if you did not set a success criteria or chose explicitly to hide an item. However, for optional items, you must explicitly set a criterion to make them visible.

The following table explains the variable display mechanism when a technology is run or not run and a success criteria is set or not set.

	User success criteria set		**User success criteria not set**	
	Default display item	Optional display item	Default display item	Optional display item
Technology run	Yes	Yes	Yes	No
Technology not run	Yes	Yes	No	No

The following example sets the success criteria as pass if the synchronization coverage is above 90%:

```
set_design_objective CDC -criteria
synchronization_coverage>90%
```

For a particular design objective, you can also specify more than one criterion within { }, as shown in the following example:

```
set_design_objective power -criteria
{switching_power<0,total_power<5uW,leakage_power<0}
```

Handling Waivers

Data corresponding to a few design objectives may be affected by waivers. If you apply waivers as a part of the original analysis, they are considered during data computation.

For example, the *synchronization_coverage*, *cdc_failed_properties*, and *cdc_partial_proven_properties* variables are influenced by waivers, and the report generator computes the final statistics by considering the applied waivers.

NOTE: *Any waivers created and applied in the GUI after original analysis are not considered until next analysis.*

Setting Success Criteria Values to Different Goals and Scenarios

The following command sets success criteria values to the `power_est_average` goal:

```
set_design_objective Power -criteria
```

```
{switching_power<50mW,total_power<55mW,leakage_power<50uW,in
ternal_power<10mW} -goal {Power/power_est_average} -top
mc_top
```

You can also set success criteria values to scenarios created for goals. For example, the following commands set success criteria values for the cg4 and cg8 scenarios created from the `power_est_average` goal:

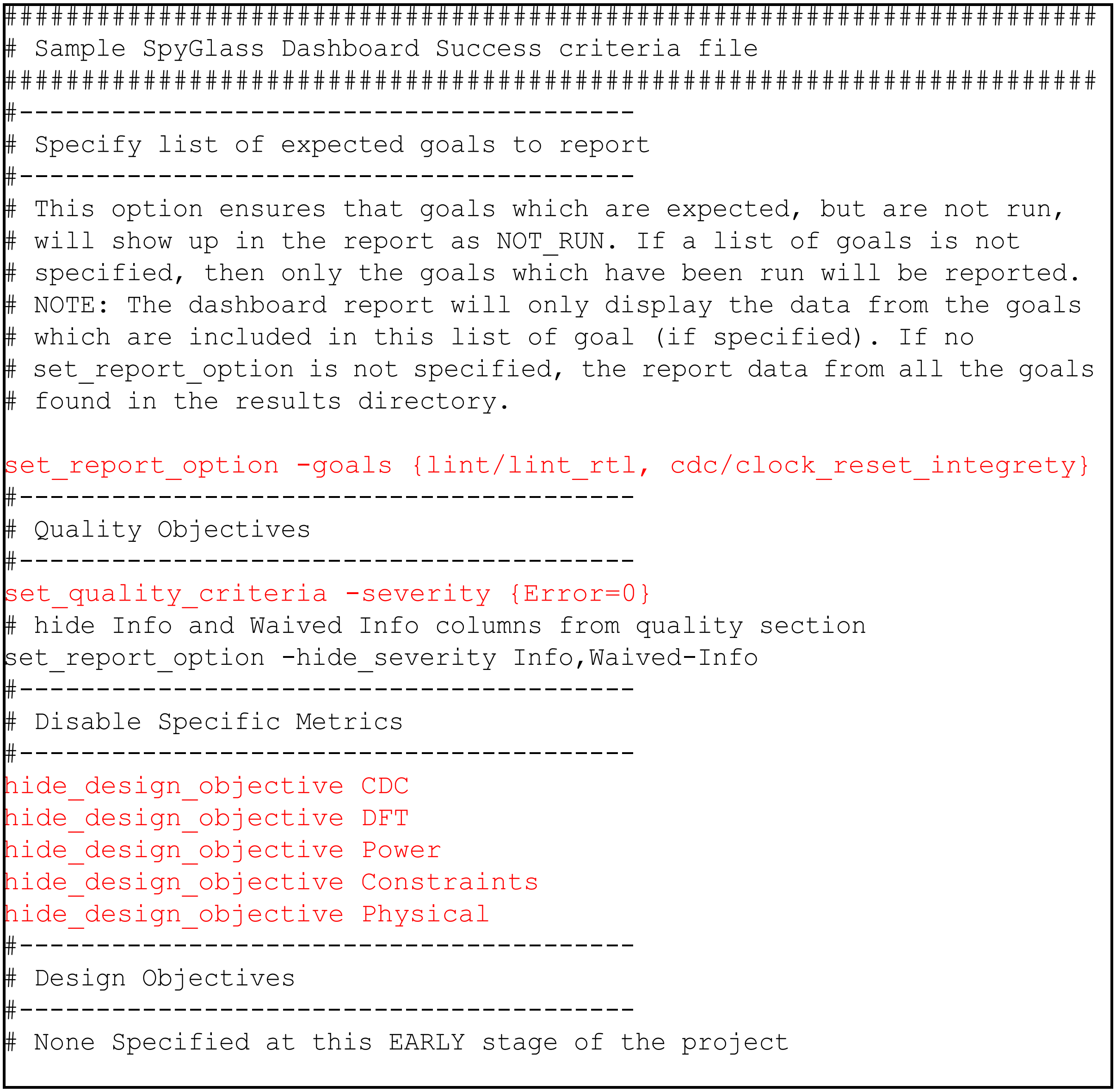
```
set_design_objective Power -criteria
{switching_power<25mW,total_power<35mW,leakage_power<10uW,in
ternal_power<25mW} -goal {Power/power_est_average@cg4} -top
mc_top


set_design_objective Power -criteria
{switching_power<15mW,total_power<25mW,leakage_power<10uW,in
ternal_power<20mW} -goal {Power/power_est_average@cg8} -top
mc_top
```

For details on scenarios, see *Working with Scenarios*.

Use Model for the Success Criteria File

You can configure the Dashboard report to track and measure design requirements using the Success Criteria file. For example, at the beginning of a design project, running detailed CDC, DFT, and Power checks is not required. Therefore, you can specify only the goals required at this stage of development, such as, lint goals in the Success Criteria File. You can also configure the report for less strict success criteria, such as, fixing only the FATAL and ERRORS type violations for instances.

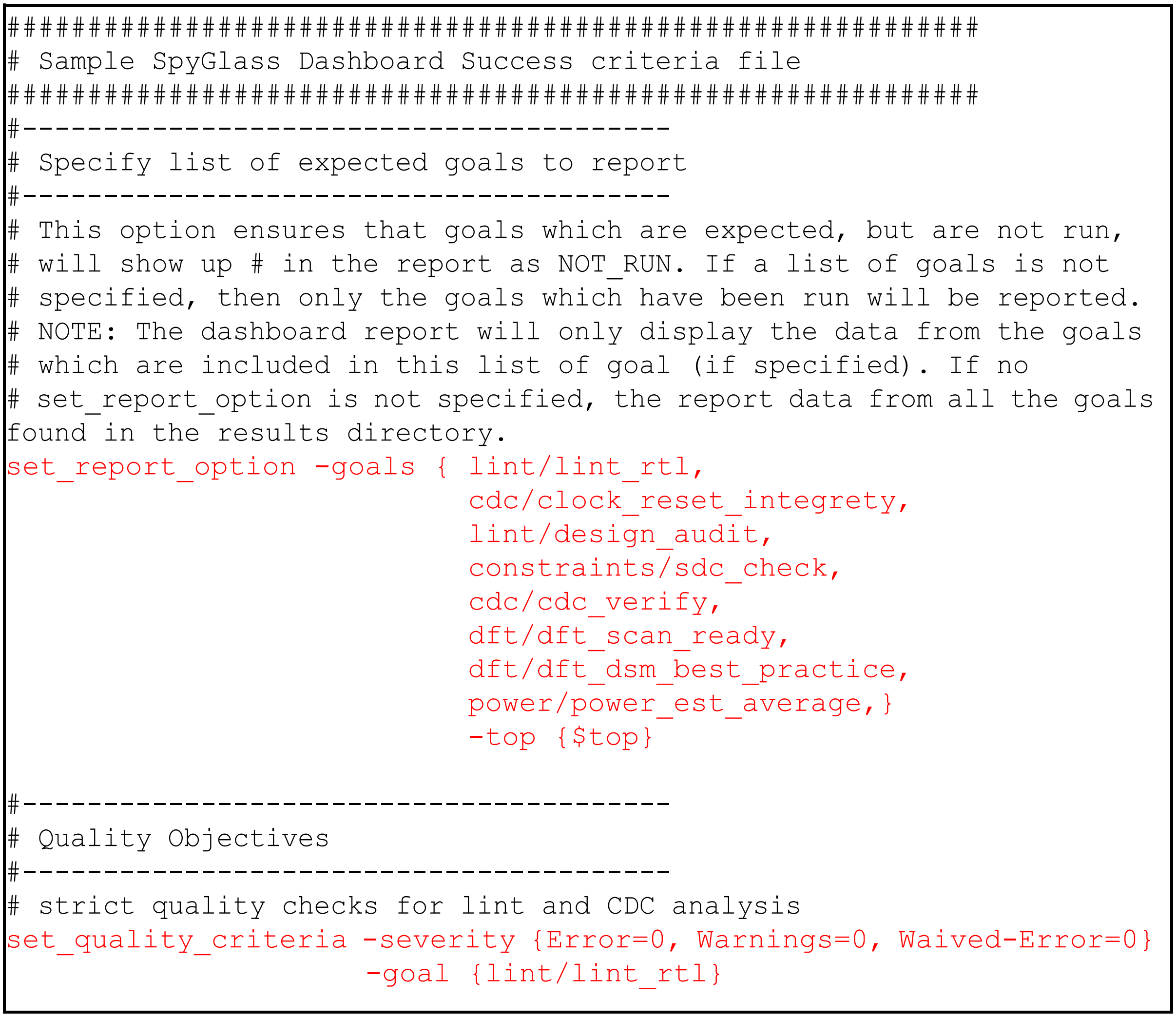
Following figure illustrates a sample criteria file for an early design stage.



```
#####################################################################
# Sample SpyGlass Dashboard Success criteria file
#####################################################################
#---------------------------------------
# Specify list of expected goals to report
#---------------------------------------
# This option ensures that goals which are expected, but are not run,
# will show up in the report as NOT_RUN. If a list of goals is not
# specified, then only the goals which have been run will be reported.
# NOTE: The dashboard report will only display the data from the goals
# which are included in this list of goal (if specified). If no
# set_report_option is not specified, the report data from all the goals
# found in the results directory.

set_report_option -goals {lint/lint_rtl, cdc/clock_reset_integrety}
#---------------------------------------
# Quality Objectives
#---------------------------------------
set_quality_criteria -severity {Error=0}
# hide Info and Waived Info columns from quality section
set_report_option -hide_severity Info,Waived-Info
#---------------------------------------
# Disable Specific Metrics
#---------------------------------------
hide_design_objective CDC
hide_design_objective DFT
hide_design_objective Power
hide_design_objective Constraints
hide_design_objective Physical
#---------------------------------------
# Design Objectives
#---------------------------------------
# None Specified at this EARLY stage of the project
```

As the design progresses, you can include additional goals and increase the success criteria, accordingly. For example, after the design reaches Design Review#1 (Feature Complete), CDC and Power Analysis begin. The CDC and Power results are also reported, in addition to the existing lint goals. Also, all Lint warning messages should be resolved.

Following figure illustrates a sample criteria file for an advanced design stage:



```
############################################################
# Sample SpyGlass Dashboard Success criteria file
############################################################
#----------------------------------------
# Specify list of expected goals to report
#----------------------------------------
# This option ensures that goals which are expected, but are not run,
# will show up # in the report as NOT_RUN. If a list of goals is not
# specified, then only the goals which have been run will be reported.
# NOTE: The dashboard report will only display the data from the goals
# which are included in this list of goal (if specified). If no
# set_report_option is not specified, the report data from all the goals
found in the results directory.
set_report_option -goals { lint/lint_rtl,
                           cdc/clock_reset_integrety,
                           lint/design_audit,
                           constraints/sdc_check,
                           cdc/cdc_verify,
                           dft/dft_scan_ready,
                           dft/dft_dsm_best_practice,
                           power/power_est_average,}
                           -top {$top}

#----------------------------------------
# Quality Objectives
#----------------------------------------
# strict quality checks for lint and CDC analysis
set_quality_criteria -severity {Error=0, Warnings=0, Waived-Error=0}
                     -goal {lint/lint_rtl}
```

```
set_quality_criteria -severity {Error=0, Warnings=0}
                     -goal {cdc/cdc_verify}
# success criteria for all other goals
set_quality_criteria -severity {Error=0}

# hide Info and Waived Info columns from quality section
set_report_option -hide_severity Info,Waived-Info
#---------------------------------------


# Disable Specific Metrics
#---------------------------------------
# Report all design metrics during the late stages of design.

#---------------------------------------
# Design Objectives
#---------------------------------------
# DFT objectives
set_design_objective DFT -criteria {stuck_at_fault>95,
                                    stuck_at_test>95,
                                    transition_fault>80,
                                    transition_test>80,
                                    scannable_flops>95}
# CDC objectives
set_design_objective CDC -criteria {unsync_crossings=0,
                                    synchronization_coverage=100}
# Constraints objectives might be module specific - so they are not set
# here.
set_design_objective Constraints -criteria {ports_constrained=100,
                                           registers_constrained>90}
# NOTE: excluding the criteria value, will display the value, but will
# not provide a pass/fail icon.  In other words - display the values only
- don't judge a success pass/fail.
set_design_objective Power -criteria  {switching_power=display_only,
                                     internal_power=display_only,
                                     leakage_power=display_only,
                                     total_power=display_only}
```

Viewing the DashBoard Report

SpyGlass generates the *Dashboard* report files, such as dashboard.html and dashboard.csv in the html_reports directory.

If you create an SoC Dashboard that contains more than one top module results, the report displays following high-level summary information in the following three tabs:

- Summary (default tab)
- Quality Goals
- Design Objectives

You can view the individual Module Dash Board report by selecting the module links provided in the Quality Goals and Design Objectives tabs.

However, if the DashBoard was created for single 'top', the report directly opens the 'Module DashBoard' without above summary tab pages.

The order of data present in this report is as follows:

SoC DashBoard

- Summary tab
  - ❐ Left side bar chart shows quality followed by SpyGlass products in the order of CDC, Advanced_Lint, DFT, Constraints, Power, Physical and other custom design objectives. Except Quality section, rest of the products will display as per data availability and user's configuration in success criteria settings.
  - ❐ Right side details chart shows list of goals executed across all modules when Quality item was enabled. When product items are enabled using the left mouse click on left side vertical bars, it displays the relevant product variables as per the success criteria settings.
- Quality Goals/ Design Objectives tab
  - ❐ The table lists all modules as per the order of projects listed in the configuration file.
  - ❐ Rest of the columns displays each applicable goal or product variable in the same order as displayed in the Summary tab.

Module Dashboard

- Design objectives and respective variables

  This information is listed in the same order as the success criteria set in the success criteria file.

- Goals

  By default, this information is arranged as per the order in the methodology order file. However, if you specify explicit list of goals to be displayed in the report by using the `set_report_option -goals` {*<goal-list>*} command, the order is as per the *<goal-list>*.

Details of the DashBoard Report

This section provides information on the details of the following dashboards:

- *SoC Dashboard*
- *Module Dashboard*

SoC Dashboard

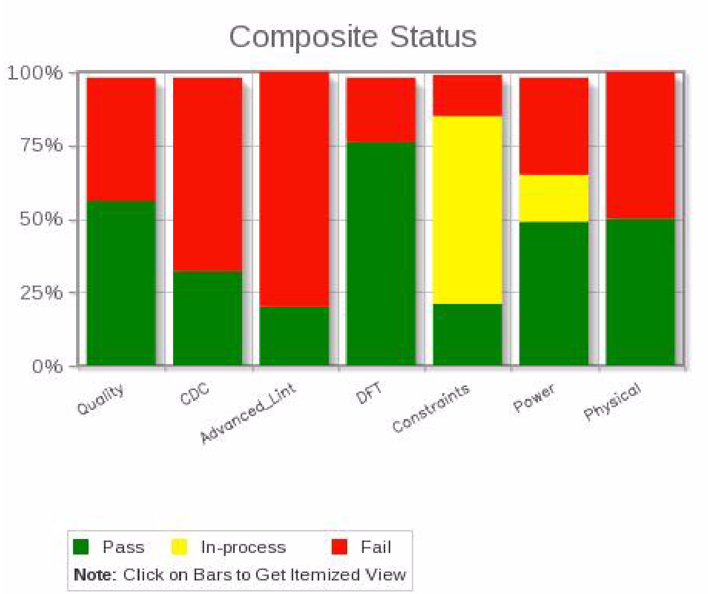
The SoC DashBoard contains following views to present composite status of SoC with respect to overall executed quality goals and different product specific objectives:

- *Summary tab*
- *Quality Goals tab*
- *Design Objectives tab*

NOTE: *You can not close the above listed default tabs. However, you can close the other dynamically added tabs.*

Summary tab

The left side chart in the Summary tab illustrates the consolidated status of quality for all goals and applicable product metrics for products such as DFT, CDC, and Power. The status in this view is categorized into Pass, Fail and In-process, which are displayed in green, red and yellow colors, respectively. Following figure illustrates the left-side chart:



Note: Click on Bars to Get Itemized View

FIGURE 270. Composite Status-Summary Tab

Following table lists different status and their respective description:

Pass	Design objective or quality goal was passed as per success criteria
Fail	Design objective or quality goal was failed as per success criteria
In-process	Design objective or quality goal execution yet to complete

To view the overall status in percentage, hover the mouse on a vertical bar. For example, if you hover the mouse on the green portion of the Quality section, a pop-up displays the following message:

```
Quality, 56%
```

This means that 56% of overall executed goals are passed as per the set success criteria.

Similarly, if you hover the mouse on to the red portion of the CDC section, a pop-up displays the following message:

```
CDC, 66%
```

This means that 66% of overall CDC objectives failed on the consolidated list of modules.

The right side chart, by default, shows sub-items of the Quality section, that is, different goals. However, when you select any other item on left side composite status chart, the right side chart is updated accordingly displaying sub-items of the selected item. For example, if you select DFT on left side chart, the right side chart is updated to display the individual items from DFT as shown in the following figure:

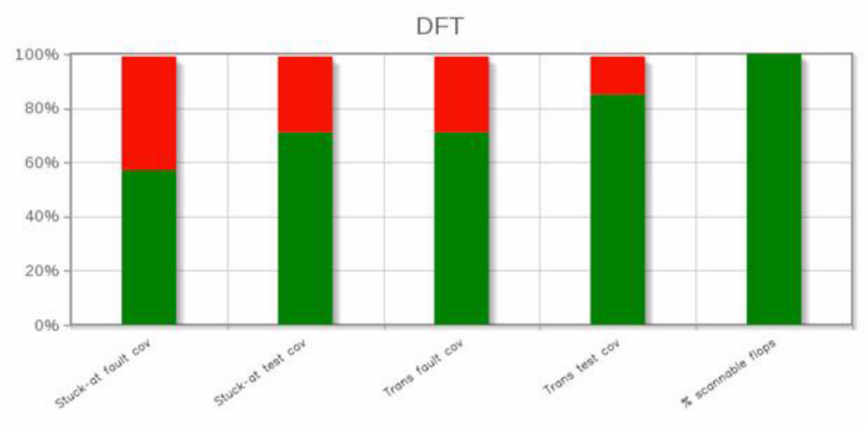
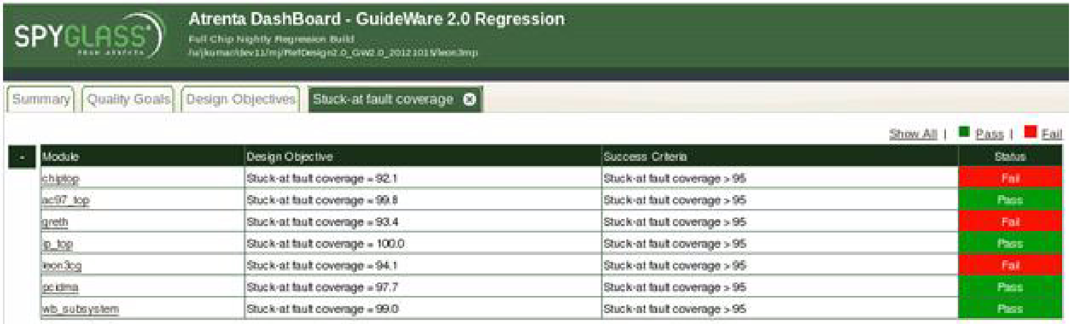


FIGURE 271. Product Specific Chart-Summary Tab

When you click on the bars, a new tab displays the details of the selected design objective status for all applicable modules. For example, if we select Stuck-at fault coverage bar of the DFT category, it opens a new tab displaying the details of Stuck-at fault coverage objective for all applicable modules as shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20121015/leon3mp

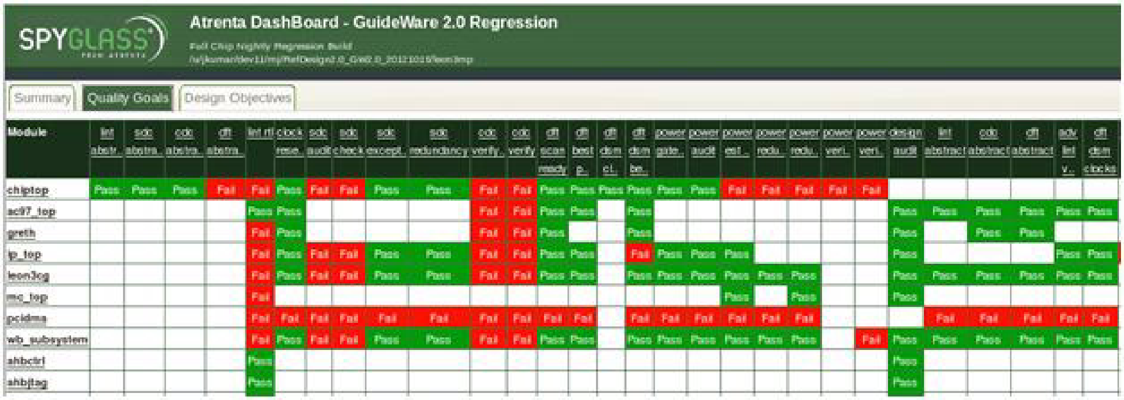
Module	Design Objective	Success Criteria	Status
chiptop	Stuck-at fault coverage = 92.1	Stuck-at fault coverage > 95	Fail
ac97_top	Stuck-at fault coverage = 99.8	Stuck-at fault coverage > 95	Pass
greth	Stuck-at fault coverage = 93.4	Stuck-at fault coverage > 95	Fail
ip_top	Stuck-at fault coverage = 100.0	Stuck-at fault coverage > 95	Pass
leon3cg	Stuck-at fault coverage = 94.1	Stuck-at fault coverage > 95	Fail
pcidma	Stuck-at fault coverage = 97.7	Stuck-at fault coverage > 95	Pass
wb_subsystem	Stuck-at fault coverage = 99.0	Stuck-at fault coverage > 95	Pass

FIGURE 272. Design Objective Data

You can filter the results based on Pass/Fail status using the links located on the top-right corner of the table. Also, clicking on any module displays the Module Dashboard for that module in a new tab.

Quality Goals tab

This tab depicts overall goal status for all modules as shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20121015/leon3mp

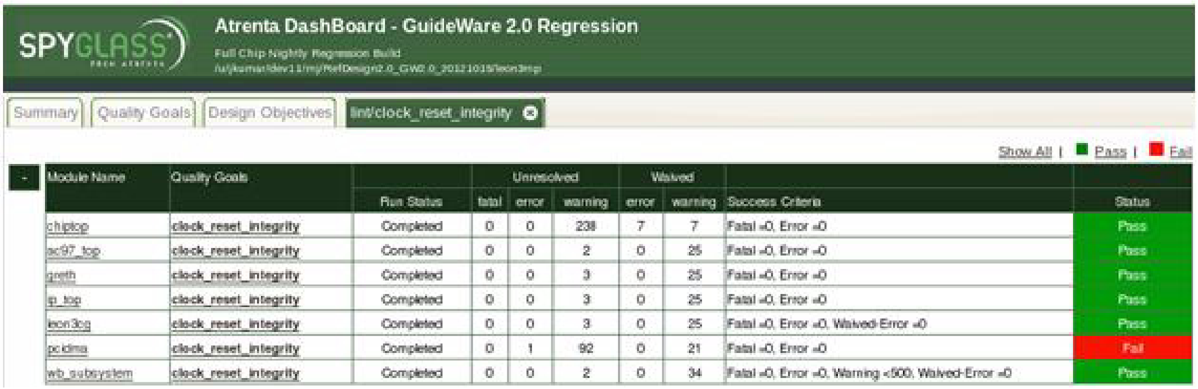
Module	lint abstr..	sdc abstra..	cdc abstra..	dft abstra..	lint rtl	clock rese..	sdc audit	sdc check	sdc except..	sdc redundancy	cdc verify..	cdc verify	dft scan ready	dft best p..	dft dsm cl..	dft dsm be..	power gate..	power audit	power est..	power redu..	power redu..	power veri..	power veri..	design audit	lint abstract	cdc abstract	dft abstract	adv lint v..	dft dsm clocks
chiptop	Pass	Pass	Pass	Fail	Fail	Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass	Pass	Pass	Pass	Pass	Pass	Fail	Fail	Fail	Fail	Fail						
ac97_top					Pass	Pass					Fail	Fail	Pass	Pass		Pass								Pass	Pass	Pass	Pass	Pass	Pass
greth					Fail	Pass					Fail	Fail	Pass			Pass								Pass		Pass	Pass		
ip_top					Fail	Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass	Pass		Fail	Pass	Pass	Pass					Pass				Pass	Pass
leon3cg					Fail	Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass	Pass		Pass	Pass	Pass	Pass	Pass	Pass			Pass	Pass	Pass	Pass	Pass	Pass
mc_top					Fail														Pass		Pass			Pass					
pcidma					Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail	Fail		Fail	Fail	Fail	Fail	Fail	Fail				Fail	Fail	Fail	Fail	Fail
wb_subsystem					Fail	Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass	Pass		Pass	Pass	Pass	Pass	Pass	Pass		Fail	Pass	Pass	Pass	Pass	Pass	Pass
ahbctrl					Pass																			Pass					
ahbjtag					Pass																			Pass					

FIGURE 273. Quality Goals Tab

The above table shows applicable Pass/Fail goal status for the listed modules with respect to the success criteria set on the module. Clicking the module name displays the respective module dashboard.

Hovering the mouse over the table cells displays the status and the Last updated time stamp in a pop-up.

Clicking the goal name in the header of the Quality Goals tab displays the goal-specific information in a new tab, which displays selected goal status and statistic details for all modules. Following figure shows a sample goal-specific page:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build

/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20121015/leon3mp

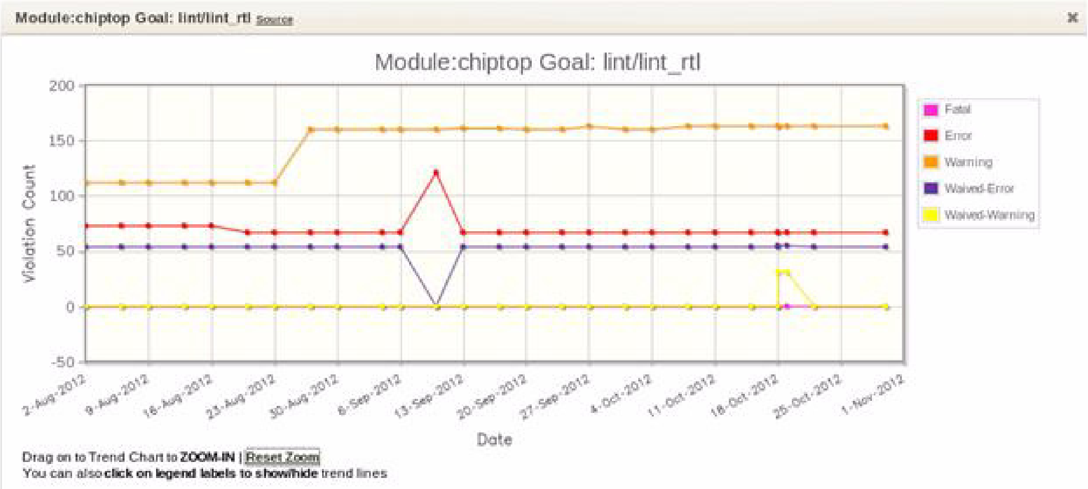
Module Name	Quality Goals		Unresolved			Waived			
		Run Status	fatal	error	warning	error	warning	Success Criteria	Status
chiptop	clock_reset_integrity	Completed	0	0	238	7	7	Fatal =0, Error =0	Pass
ac97_top	clock_reset_integrity	Completed	0	0	2	0	25	Fatal =0, Error =0	Pass
greth	clock_reset_integrity	Completed	0	0	3	0	25	Fatal =0, Error =0	Pass
ip_top	clock_reset_integrity	Completed	0	0	3	0	25	Fatal =0, Error =0	Pass
leon3cg	clock_reset_integrity	Completed	0	0	3	0	25	Fatal =0, Error =0, Waived-Error =0	Pass
pcidma	clock_reset_integrity	Completed	0	1	92	0	21	Fatal =0, Error =0	Fail
wb_subsystem	clock_reset_integrity	Completed	0	0	2	0	34	Fatal =0, Error =0, Warning <500, Waived-Error =0	Pass

FIGURE 274. Goal-Specific Data

You can perform the following actions in this view:

- Click on the module name links in this view to load the respective module dashboard for the selected module.
- Click on a goal to view respective reports, if attached, for the selected module and goal.
- Click on Status column with respect to a row to open the respective trend chart for the selected module and goal.

Selecting the table cells (on displayed status) in the Quality Goals tab displays the trend chart with respect to the module and goal that shows trend variation on different severity message counts over time. Following figure illustrates a sample trend chart for the lint/lint_rtl goal:



Drag on to Trend Chart to **ZOOM-IN** | Reset Zoom
You can also **click on legend labels to show/hide** trend lines

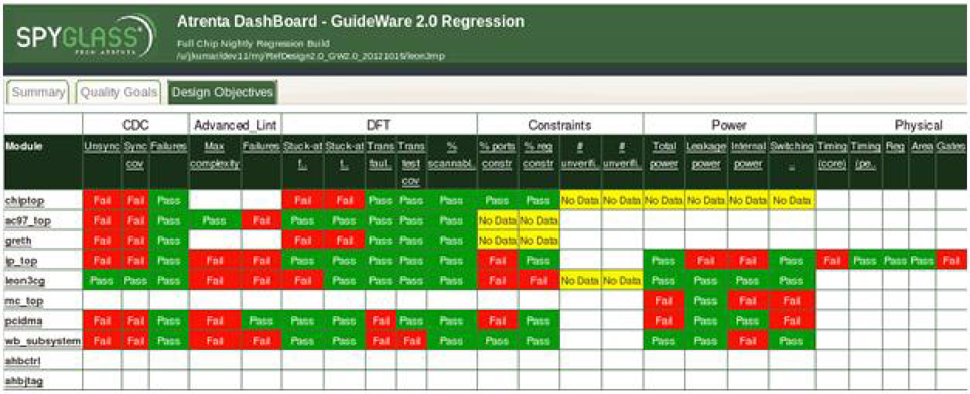
FIGURE 275. Trend Chart for a Goal

You can filter the displayed data based on the legend, that is, severity name. You can also select a region in the trend chart to view the detailed view of the message count for the selected dates.

To switch back to the original view, click the Reset Zoom link.

Design Objectives tab

This tab depicts status for all product-specific objectives for all the modules as shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build

/u/jkumar/dev11/my/RefDesign2.0_GW2.0_20121015/leon3mp

	CDC			Advanced_Lint		DFT					Constraints				Power				Physical				
Module	Unsync	Sync cov	Failures	Max complexity	Failures	Stuck-at f..	Stuck-at f..	Trans faul.	Trans test cov	% scannabl.	% ports constr	% reg constr	# unverifi.	# unverifi.	Total power	Leakage power	Internal power	Switching ..	Timing (core)	Timing (pe..	Reg	Area	Gates
chiptop	Fail	Fail	Pass			Fail	Fail	Pass	Pass	Pass	Pass	Pass	No Data	No Data	No Data	No Data	No Data	No Data					
ac97_top	Fail	Fail	Pass	Pass	Fail	Pass	Pass	Pass	Pass	Pass	No Data	No Data											
greth	Fail	Fail	Pass			Fail	Fail	Pass	Pass	Pass	No Data	No Data											
ip_top	Fail	Fail	Pass	Fail	Fail	Pass	Pass	Pass	Pass	Pass	Fail	Pass			Pass	Fail	Fail	Pass	Fail	Pass	Pass	Pass	Fail
leon3cg	Pass	Pass	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass	Fail	Fail	No Data	No Data	Pass	Pass	Pass	Pass					
mc_top															Fail	Pass	Fail	Fail					
pcidma	Fail	Fail	Pass	Fail	Pass	Pass	Pass	Fail	Pass	Pass	Fail	Pass			Fail	Pass	Pass	Fail					
wb_subsystem	Fail	Fail	Pass	Fail	Fail	Pass	Pass	Fail	Fail	Pass	Pass	Pass			Pass	Pass	Fail	Pass					
ahbctrl																							
ahbjtag																							

FIGURE 276. Design Objectives Tab

Above table shows all applicable modules, which you can select to view the respective Module DashBoard pages. Rest of the columns shows all products and product-specific objectives. The headers of the table display design objective names and are truncated to best-fit to the view. When you hover the mouse over the design objective name, the full objective name is displayed in a pop-up.

The rest of the table displays the Pass/Fail/Data/No Data status of the design objective with respect to the success criteria set on the module.

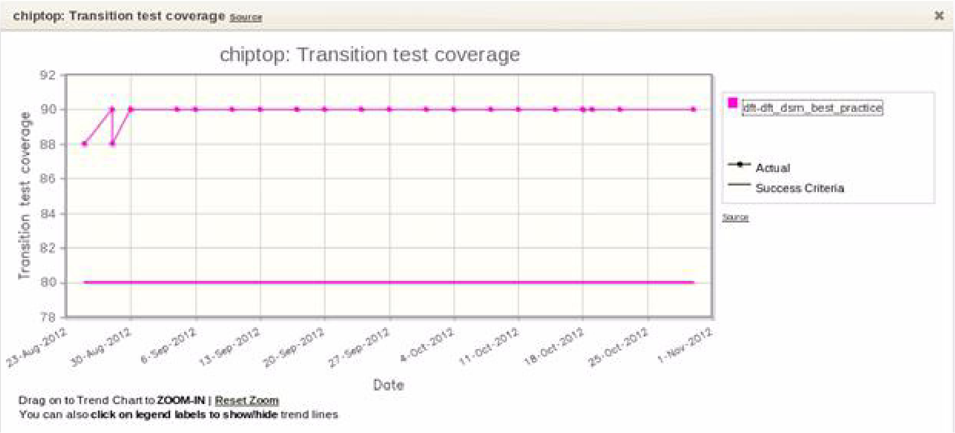
Following table classifies the available status and their description:

Pass	Design objective passed as per success criteria
Fail	Design objective failed as per success criteria
No Data	Data expected from run results but not yet available
Data	Success criteria was set to this item as 'display_only' and data is available

Hovering the mouse over the table cells displays the status and the Last updated time stamp in a pop-up.

Selecting the design objective name in the table header displays a new tab listing variable-specific data for all applicable modules. See *Figure 272* for details.

Clicking on the displayed status in a table cell displays the trend chart with respect to the module and design objective that shows trend variation over time. Following figure illustrates a sample trend chart for a module with respect to the selected design objective:



Drag on to Trend Chart to **ZOOM-IN** | Reset Zoom
You can also **click on legend labels to show/hide** trend lines

FIGURE 277. Trend Chart for Module/Design Objective

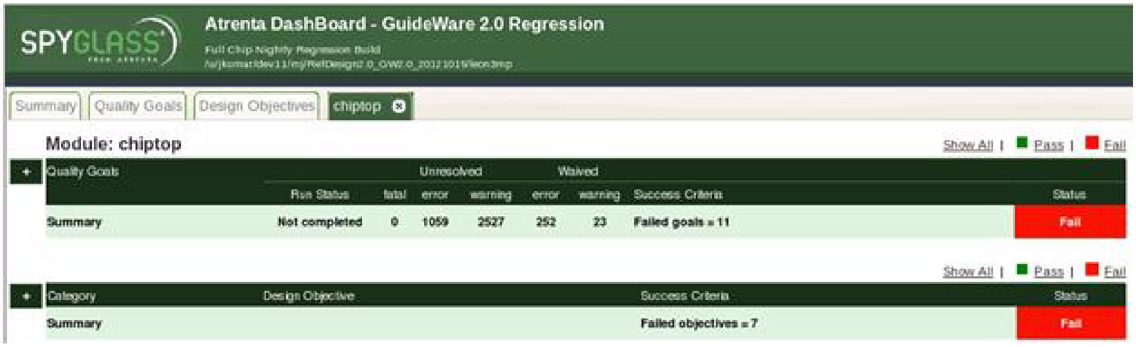
In the above figure, the solid line represents success criteria and the thin line with dots represents the actual data. If multiple goals or scenarios in the above trend chart provide the same variable data, multiple line pairs are displayed in the chart. You can turn on/off some of them from the provided legend in the chart area.

Module Dashboard

Module Dashboard is displayed in the following scenarios:

- A dashboard is created for a single module
- Module-specific links is displayed in the SoC DashBoard

By default, the module dashboard page displays summary of Quality Goals and Design Objectives as shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20121015/leon3mp

Module: chiptop

Quality Goals			Unresolved		Waived			
	Run Status	fatal	error	warning	error	warning	Success Criteria	Status
Summary	Not completed	0	1059	2527	252	23	Failed goals = 11	Fail

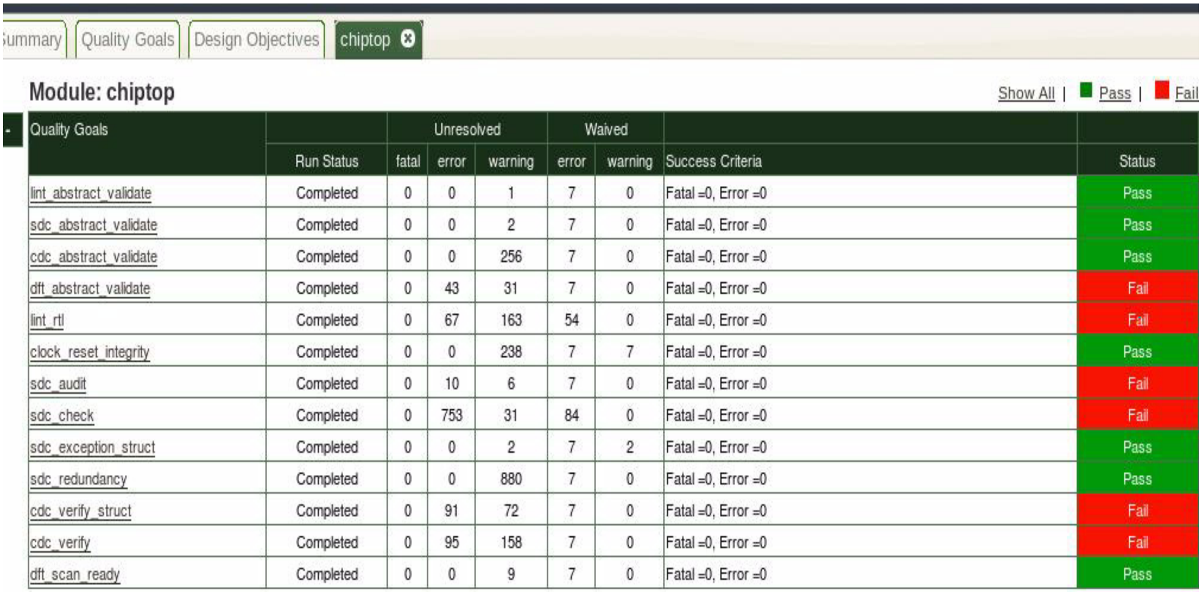
Category	Design Objective	Success Criteria	Status
Summary		Failed objectives = 7	Fail

FIGURE 278. Module Dashboard (Collapsed View)

You can expand both the tables by selecting 'Show All' link provided on top right corner of the module dashboard page. Alternatively, you can select the +/- icon provided at the left side of the tables, to expand/collapse the table view.

Module Dashboard-Quality Goals

Following figure displays the expanded view of the Quality Goals table in the Module Dashboard page:



Module: chiptop

Quality Goals		Unresolved			Waived			
	Run Status	fatal	error	warning	error	warning	Success Criteria	Status
lint_abstract_validate	Completed	0	0	1	7	0	Fatal =0, Error =0	Pass
sdc_abstract_validate	Completed	0	0	2	7	0	Fatal =0, Error =0	Pass
cdc_abstract_validate	Completed	0	0	256	7	0	Fatal =0, Error =0	Pass
dft_abstract_validate	Completed	0	43	31	7	0	Fatal =0, Error =0	Fail
lint_rtl	Completed	0	67	163	54	0	Fatal =0, Error =0	Fail
clock_reset_integrity	Completed	0	0	238	7	7	Fatal =0, Error =0	Pass
sdc_audit	Completed	0	10	6	7	0	Fatal =0, Error =0	Fail
sdc_check	Completed	0	753	31	84	0	Fatal =0, Error =0	Fail
sdc_exception_struct	Completed	0	0	2	7	2	Fatal =0, Error =0	Pass
sdc_redundancy	Completed	0	0	880	7	0	Fatal =0, Error =0	Pass
cdc_verify_struct	Completed	0	91	72	7	0	Fatal =0, Error =0	Fail
cdc_verify	Completed	0	95	158	7	0	Fatal =0, Error =0	Fail
dft_scan_ready	Completed	0	0	9	7	0	Fatal =0, Error =0	Pass

FIGURE 279. Module Dashboard-Quality Goals

This section contains the following columns:

- Run Status

  Specifies the current run status of a goal.

- Unresolved Issues

  Displays the count of unresolved fatal, error, and warning, messages.

- Waived issues

  Displays the count of waived fatal, error, warning, and info messages.

- Success criteria

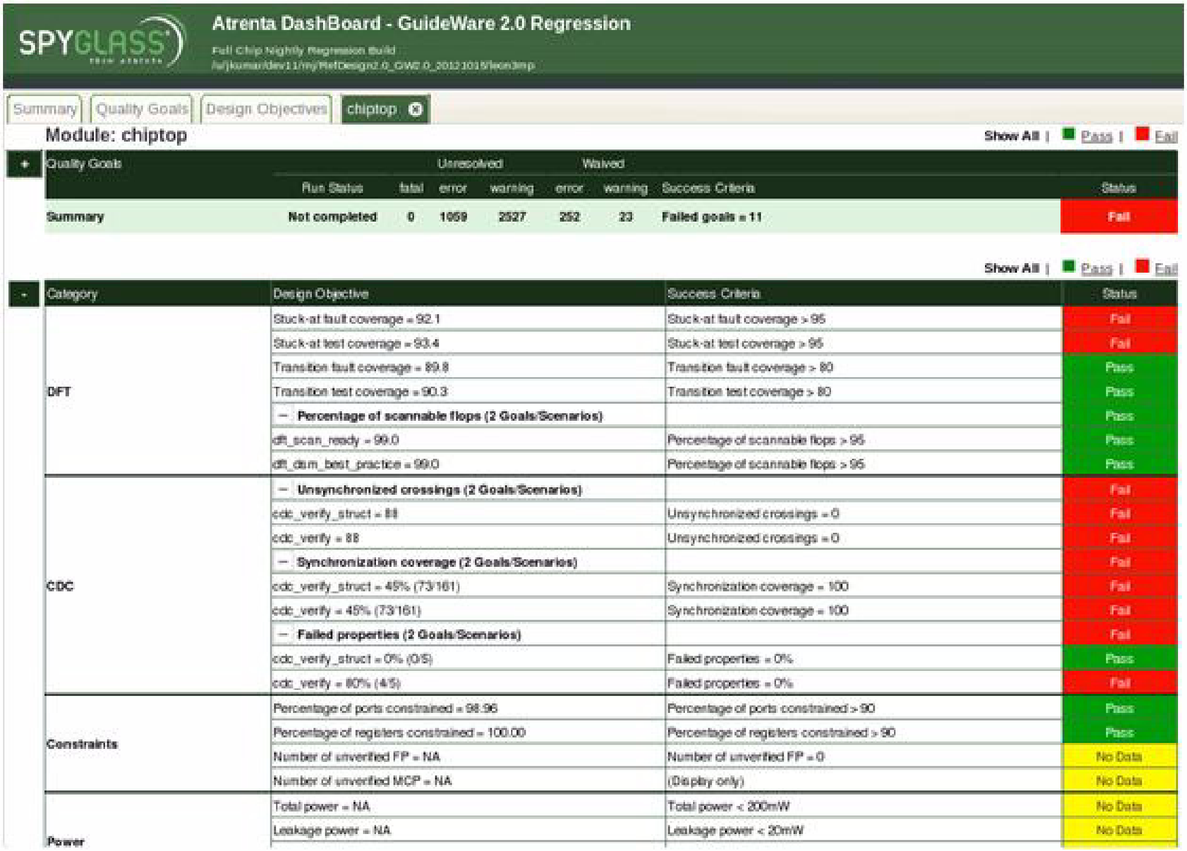
  Specifies the success criteria.

- Status

  Specifies the status whether a particular design objective has met the given success. Click on any of the Status cell to display the variations from previous runs in a graphical format. See *Figure 277* for details of this trend graph.

Module Dashboard-Design Objectives

Following figure displays the expanded view of the Design Objectives table in the Module Dashboard page:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build

/u/jkumar/dev11/my/RefDesign2.0_GW2.0_20121015/leon3mp

Module: chiptop

Quality Goals			Unresolved		Waived			
	Run Status	fatal	error	warning	error	warning	Success Criteria	Status
Summary	Not completed	0	1059	2527	252	23	Failed goals = 11	Fail

Category	Design Objective	Success Criteria	Status
DFT	Stuck-at fault coverage = 92.1	Stuck-at fault coverage > 95	Fail
	Stuck-at test coverage = 93.4	Stuck-at test coverage > 95	Fail
	Transition fault coverage = 89.8	Transition fault coverage > 80	Pass
	Transition test coverage = 90.3	Transition test coverage > 80	Pass
	− **Percentage of scannable flops (2 Goals/Scenarios)**		Pass
	dft_scan_ready = 99.0	Percentage of scannable flops > 95	Pass
	dft_dsm_best_practice = 99.0	Percentage of scannable flops > 95	Pass
CDC	− **Unsynchronized crossings (2 Goals/Scenarios)**		Fail
	cdc_verify_struct = 88	Unsynchronized crossings = 0	Fail
	cdc_verify = 88	Unsynchronized crossings = 0	Fail
	− **Synchronization coverage (2 Goals/Scenarios)**		Fail
	cdc_verify_struct = 45% (73/161)	Synchronization coverage = 100	Fail
	cdc_verify = 45% (73/161)	Synchronization coverage = 100	Fail
	− **Failed properties (2 Goals/Scenarios)**		Fail
	cdc_verify_struct = 0% (0/5)	Failed properties = 0%	Pass
	cdc_verify = 80% (4/5)	Failed properties = 0%	Fail
Constraints	Percentage of ports constrained = 98.96	Percentage of ports constrained > 90	Pass
	Percentage of registers constrained = 100.00	Percentage of registers constrained > 90	Pass
	Number of unverified FP = NA	Number of unverified FP = 0	No Data
	Number of unverified MCP = NA	(Display only)	No Data
Power	Total power = NA	Total power < 200mW	No Data
	Leakage power = NA	Leakage power < 20mW	No Data

FIGURE 280. Module Dashboard - Design Objectives

The above section shows the data for the DFT, CDC, Constraints, and `Power` design objectives. Following columns are displayed when you expand the Design Objectives table:

Category	*Design Objectives*	*Success Criteria*
Status		

Category

Specifies the name of the design objective, such as `CDC`, `Power`, `DFT`, or `Constraints`.

Design Objectives

Specifies the values generated by individual goals or scenario runs.

SpyGlass compares these values with the *Success Criteria* to determine the overall task result.

Success Criteria

Specifies the success criteria defined for a particular design objective. If no success criterion is defined, this column reads *not set*. For such design objectives, the corresponding cell in the *Pass/Fail Status* column reads *Unknown* and that cell appears in yellow.

Status

Displays status (*Pass*, *Fail*, or *No Data*) based on the comparison between the specified *Success Criteria* and the actual *Design Objectives* value extracted by analysis.

The *No Data* status appears if the data was not extracted by analysis.

Click on any of the Status cell to display the variations from previous runs in a graphical format. See *Figure 277* for details of this trend graph.

Customizing Report

SpyGlass enables you to customize your Dashboard report in following ways:

- *Including Product-Specific Data in the Report*
  - *Displaying Pre-Existing Product Data*
  - *Displaying Custom Product/Rule Data*
- *Customizing the Report Header*

Including Product-Specific Data in the Report

Depending upon the goal run, some product-specific data is generated at the end of the goal run and saved in separate files. You can extend the *DashBoard* report to include product-specific data from these files.

Displaying Pre-Existing Product Data

To enable SpyGlass to include product-specific data from the generated files automatically as a part of SpyGlass analysis, perform the following steps:

1. Set the `INCLUDE_DASHBOARD_SOURCES` environment variable to a comma-separated list of files, as shown in the following example:

   ```
   setenv INCLUDE_DASHBOARD_SOURCES file1,file2,file3
   ```

   You can specify an absolute path of the files with this variable.

   The file format should be the same as the existing *DataSheet* internal files generated by all products. For example, a file should contain one column header line starting with `SCHEMA` and different column headers should be separated by *@@*, as shown in the following example:

   ```
   SCHEMA@@FaultCoverage (%)@@TestCoverage
   (%)@@ScannableFlop(%)(%) VALUE@@17.1@@17.8@@0.0
   ```

   Each column header maps to a design objective that is used in the success criteria file.

2. Specify design objective details in the success criteria file.

   For example, the following line in the success criteria file adds the *Total number of registers* column in the *Power* section of the *DashBoard* report:

   ```
   set_design_objective Power
   -criteria Total_number_of_registers <20
   ```

   Now the *Total number of registers* column appears in the *Power* section of the *DashBoard* report.

Example

Consider an example in which you want to show the following columns in the *Power* section of the *DashBoard* report (these items are not shown by default):

- Average Clock Frequency
- Average Registers Frequency
- Total Registers

To include the above information, perform the following actions:

1. Before running a goal containing the *PEPWR02* rule, set the following environment variable to enable SpyGlass to include an additional data file:

   ```
   setenv INCLUDE_DASHBOARD_SOURCES spyglass_spysch/
   power_est/PowerFrequencyData
   ```

2. In the success criteria file, set the following additional items for the *Power* section to determine the pass or fail status:

   ```
   set_design_objective Power -criteria
   {Average_Clock_Frequency>120MHz,
   Average_Registers_Frequency>5MHz,Total_Registers>10}
   ```

Displaying Custom Product/Rule Data

Perform the following steps to generate the dashboard report for custom product or custom rules:

1. Ensure that the custom product or rule generates a data file in the following format as part of the analysis:

   ```
   SCHEMA@@<variable name list separated by @@>
   VALUE@@<variable values separated by @@>
   ```

   For example, a rule generates the file, /<work directory>/<project name>/<top name>/<goal path>/spyglass_spysch/MyProduct/MyDashBoardData with the following content:

   ```
   SCHEMA@@MyProduct_Var1@@MyProduct_Var2@@MyProduct_Var3
   VALUE@@17.1@@17.8@@10.0
   ```

   In the above example, the variable names are provided using the SCHEMA keyword and the variable values are provided using the VALUE keyword. Therefore, SpyGlass saves following values for DashBoard reporting purpose:

   ```
   MyProduct_Var1 = 17.1
   MyProduct_Var2  = 17.8
   MyProduct_Var3  = 10.0
   ```

2. Prior to running a goal or a rule, set below variable to enable SpyGlass to check for the data file generated by the custom rule or product and save the data for trend reporting:

   ```
   setenv INCLUDE_DASHBOARD_SOURCES  spyglass_spysch/
   MyProduct/MyDashBoardData
   ```

   You can specify multiple files by providing a comma-separated file list. Ensure that the file path is relative to a goal output directory (typically the directory where we see spyglass.log file) or is an absolute path.

3. Before creating DashBoard, include new metrics, for example, MyProduct_Var1, MyProduct_Var2, and MyProduct_Var3 in the success criteria file.

   However, if you want to list the new variables under a new product category, specify the following in the success criteria file:

   ```
   set_design_objective MyProduct -criteria {
   MyProduct_Var1<20, MyProduct_Var2<25, MyProduct_Var3>5}
   ```

   Alternatively, if you want to list the new variables under existing products, such as CDC, specify the following in the success criteria file:

   ```
   set_design_objective CDC -criteria
   {synchronization_coverage=100%,cdc_failed_properties=0%,
   MyProduct_Var1<20, MyProduct_Var2<25, MyProduct_Var3>5}
   ```

Customizing the Report Header

SpyGlass allows you to configure the Dashboard report header by *Changing the Name of the Report*, *Adding a Logo in the Report Header*, or *Configuring Report Title and Label*.

Changing the Name of the Report

By default, the name of the report is dashboard, that is, dashboard.html and dashboard.csv.

You can overwrite this name in any of the following ways:

- By specifying a file name in the *Report File Name* text box of the *Dashboard Report* dialog.

- By specifying a file name to the REPORT_FILE_NAME variable in a configuration file, as shown below:

  REPORT_FILE_NAME *<name>*

Adding a Logo in the Report Header

You can add a logo in the report header by specifying the following information in the appropriate fields in the *Dashboard Report* dialog:

- URL of the logo in the *Custom Logo:(URL)* textbox
- Width (in pixels) of the logo in the *Width(px)* textbox
- Height (in pixels) of the logo in the *Height(px)* textbox
- An alternate text to be displayed if the logo fails to load from the specified URL in the *Label* textbox

Based on the above information, the logo details are stored in the configuration file in the following format:

CUSTOM_LOGO *<URL>@@<Width>@@<Height>@@<Label>*

Configuring Report Title and Label

Report title contains static information, such as, project, purpose, and so on.

Report label contains dynamic information, such as, build type regression, release, milestone, and path to regression results.

You can change the report title of the Dashboard report using the REPORT_TITLE and REPORT_LABEL variable in the Dashboard Configuration file.

The following sample Dashboard Configuration file shows the REPORT_TITLE and REPORT_LABEL variables:

```
# report title and labels
#(note HTML can be used in both the REPORT_TITLE and REPORT_LABEL)
REPORT_TITLE "SpyGlass Quality Report Title"
REPORT_LABEL "$MODULE Nightly Regression Build<br>$PROJECT_DIR"
```

You can specify HTML tags in the REPORT TITLE and REPORT_LABEL variable to link to other reports or web content:

Following sample Dashboard Configuration file shows the usage of HTML tags in the REPORT_LABEL variable:

```
# report title and labels
# (note HTML can be used in both the REPORT_TITLE and REPORT_LABEL)
REPORT_TITLE "Project A - Release Build Report (Release 1A)"
REPORT_LABEL "For additional build information:
<a href="http://project_server/projectA/info.html">Project Info</
a>"
```

Managing Reports

You can perform following tasks to manage the generated Dashboard Report:

- *Archiving and Managing Data Generated After Running Goals*
- *Generating the HTML Goal Summary Page*

Archiving and Managing Data Generated After Running Goals

After execution of selected goals, data files are generated in the Run_Summary directory of the current project. These data files contain results of goal run.

You can archive such data files to a common storage area so that you can analyze them later. Archiving data at a common storage area enables you to clean-up a local run without losing run history.

You can remove or edit the collected data later.

Archiving Data

To archive data files, set the `ARCHIVE_RUN_SUMMARY_FILES` environment variable to an area where the data files can be archived for future use.

While generating the *DashBoard* report, if this environment variable set and if there are any archived files, the report generator considers data from the archived files present at a path specified by this environment variable.

Managing Archived Data

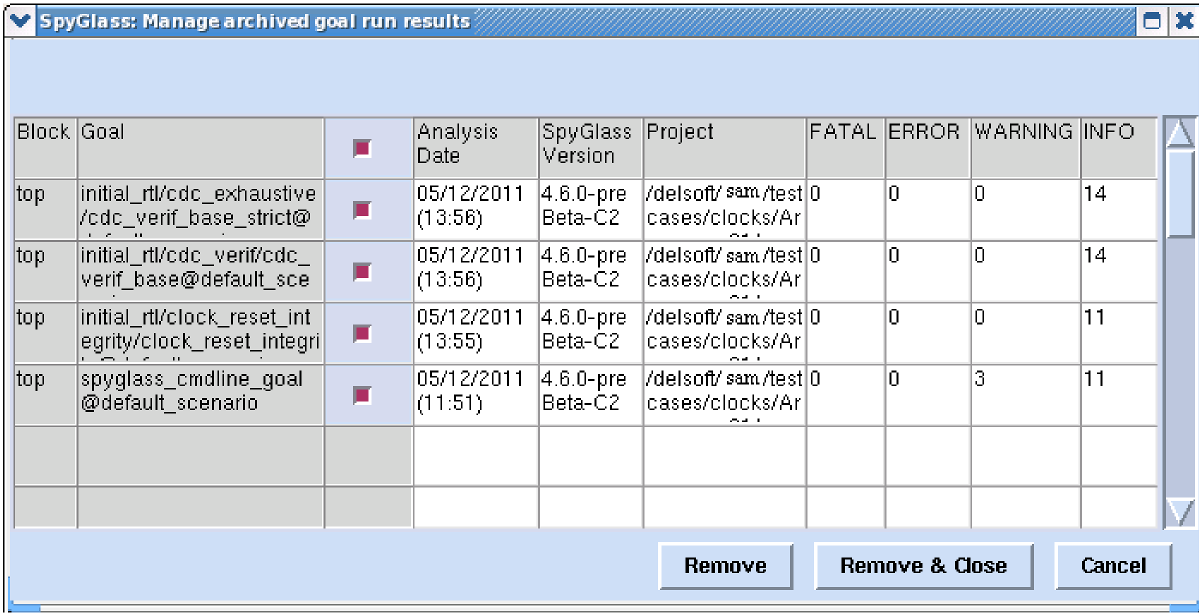
To manage the archived data, perform the following steps:

1. Select the *Tools -> Dashboard* report menu option.

   The *Dashboard Report* dialog appears.

2. Click the *Manage Data* button in the *Dashboard Report* dialog.

   The *Manage archived goal run results* dialog appears, as shown in the following figure:



Block	Goal	■	Analysis Date	SpyGlass Version	Project	FATAL	ERROR	WARNING	INFO
top	initial_rtl/cdc_exhaustive /cdc_verif_base_strict@	■	05/12/2011 (13:56)	4.6.0-pre Beta-C2	/delsoft/ sam /test cases/clocks/Ar	0	0	0	14
top	initial_rtl/cdc_verif/cdc_ verif_base@default_sce	■	05/12/2011 (13:56)	4.6.0-pre Beta-C2	/delsoft/ sam /test cases/clocks/Ar	0	0	0	14
top	initial_rtl/clock_reset_int egrity/clock_reset_integri	■	05/12/2011 (13:55)	4.6.0-pre Beta-C2	/delsoft/ sam /test cases/clocks/Ar	0	0	0	11
top	spyglass_cmdline_goal @default_scenario	■	05/12/2011 (11:51)	4.6.0-pre Beta-C2	/delsoft/ sam /test cases/clocks/Ar	0	0	3	11

FIGURE 281. Manage Archived Goal Run Results

This dialog displays data for each goal run based on the information present in a configuration file.

Removing Archived Data from the Common Storage Area

You can select data from the *Manage archived goal run results* dialog and remove it from the common storage area.

To remove the required data displayed in a particular row in this dialog, perform the following steps:

1. Select □ in the row that you want to delete.

   You can also choose all rows displayed in the above dialog by selecting □ button in the column header.

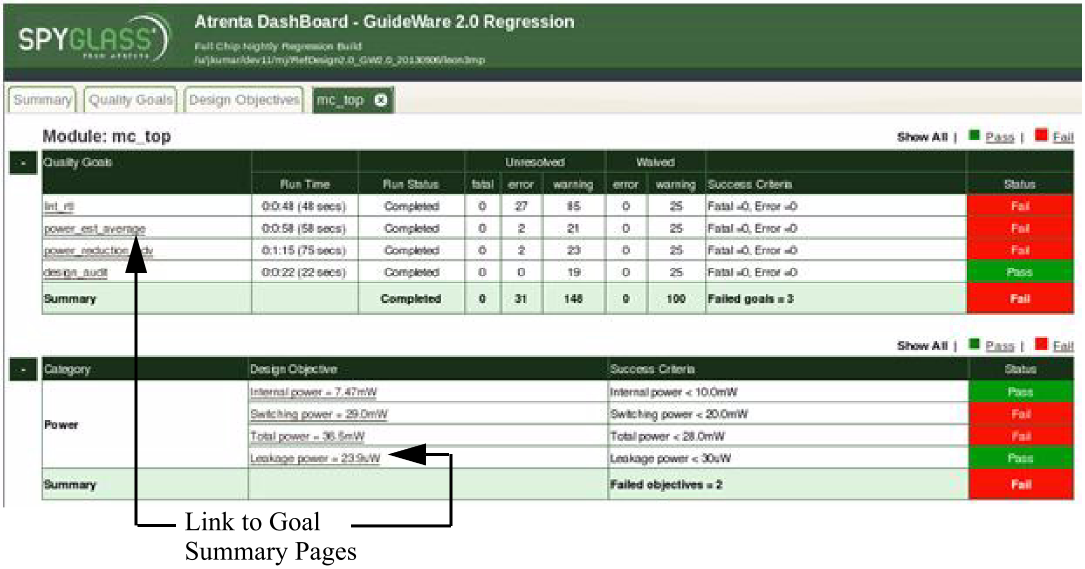
2. Click the *Remove* button.

After performing the above steps, data of that row is moved to a separate folder, *Removed_files*. In addition, the row is removed from the *Manage archived goal run results* dialog.

Generating the HTML Goal Summary Page

SpyGlass automatically links the DashBoard Goal names in Quality table and Design objective names to appropriate goal summary HTML pages, if available, that shows all available reports for further navigation.

For example, if running the `power_est_average` goal generates the power_est_average goal HTML summary page, the link appears as `power_est_average`, as shown in the following figure:



Atrenta DashBoard - GuideWare 2.0 Regression

Full Chip Nightly Regression Build
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130906/leon3mp

Module: mc_top

Show All | 🟩 Pass | 🟥 Fail

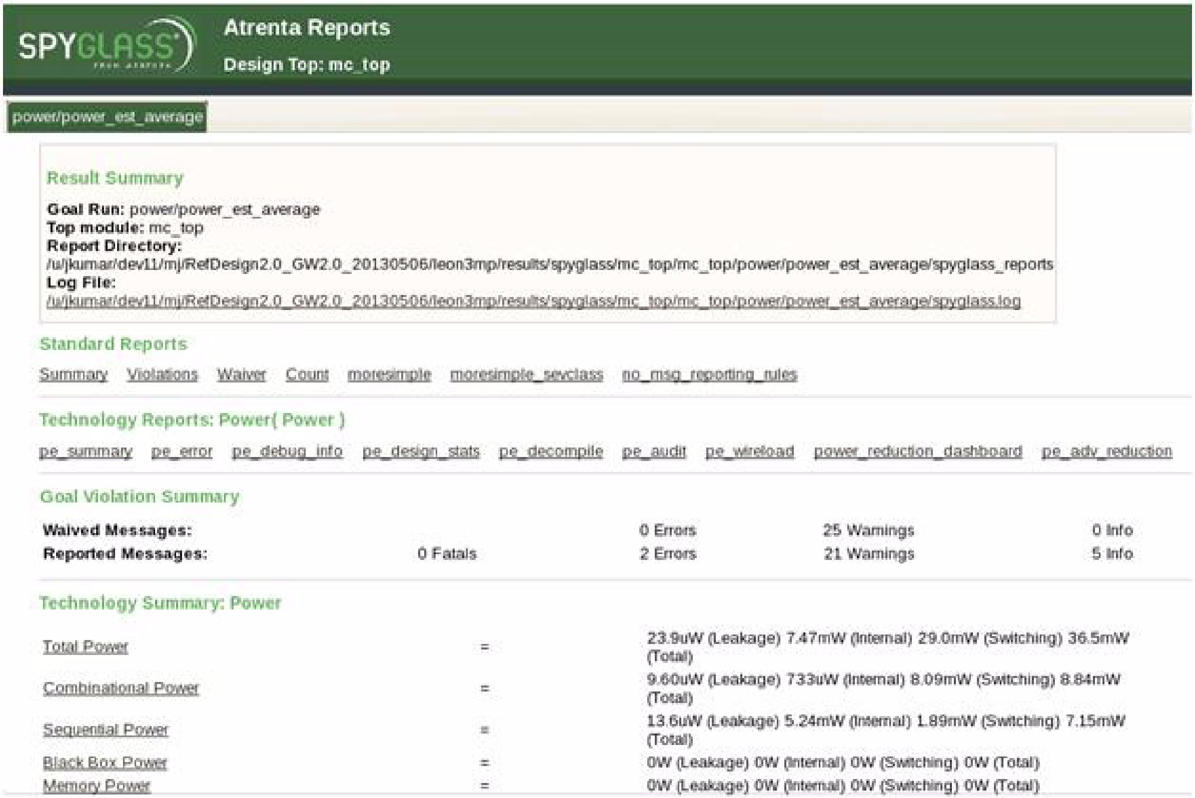
Quality Goals			Unresolved			Waived			
	Run Time	Run Status	fatal	error	warning	error	warning	Success Criteria	Status
lint_rtl	0:0:48 (48 secs)	Completed	0	27	85	0	25	Fatal =0, Error =0	Fail
power_est_average	0:0:58 (58 secs)	Completed	0	2	21	0	25	Fatal =0, Error =0	Fail
power_reduction_adv	0:1:15 (75 secs)	Completed	0	2	23	0	25	Fatal =0, Error =0	Fail
design_audit	0:0:22 (22 secs)	Completed	0	0	19	0	25	Fatal =0, Error =0	Pass
Summary		**Completed**	**0**	**31**	**148**	**0**	**100**	**Failed goals = 3**	**Fail**

Show All | 🟩 Pass | 🟥 Fail

Category	Design Objective	Success Criteria	Status
Power	Internal power = 7.47mW	Internal power < 10.0mW	Pass
	Switching power = 29.0mW	Switching power < 20.0mW	Fail
	Total power = 36.5mW	Total power < 28.0mW	Fail
	Leakage power = 23.9uW	Leakage power < 30uW	Pass
Summary		**Failed objectives = 2**	**Fail**

FIGURE 282. Creating Links to Reports

When you click the `power_est_average` link in the above report, the following goal summary page is displayed:



Atrenta Reports

Design Top: mc_top

power/power_est_average

Result Summary

Goal Run: power/power_est_average
Top module: mc_top
Report Directory:
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/results/spyglass/mc_top/mc_top/power/power_est_average/spyglass_reports
Log File:
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/results/spyglass/mc_top/mc_top/power/power_est_average/spyglass.log

Standard Reports

Summary Violations Waiver Count moresimple moresimple_sevclass no_msg_reporting_rules

Technology Reports: Power(Power)

pe_summary pe_error pe_debug_info pe_design_stats pe_decompile pe_audit pe_wireload power_reduction_dashboard pe_adv_reduction

Goal Violation Summary

Waived Messages:		0 Errors	25 Warnings	0 Info
Reported Messages:	0 Fatals	2 Errors	21 Warnings	5 Info

Technology Summary: Power

Total Power	=	23.9uW (Leakage) 7.47mW (Internal) 29.0mW (Switching) 36.5mW (Total)
Combinational Power	=	9.60uW (Leakage) 733uW (Internal) 8.09mW (Switching) 8.84mW (Total)
Sequential Power	=	13.6uW (Leakage) 5.24mW (Internal) 1.89mW (Switching) 7.15mW (Total)
Black Box Power	=	0W (Leakage) 0W (Internal) 0W (Switching) 0W (Total)
Memory Power	=	0W (Leakage) 0W (Internal) 0W (Switching) 0W (Total)

FIGURE 283. power_est_average Goal HTML summary page

The goal summary page has following sections:

- *Results Summary*
- *Standard Reports*
- *Technology Reports*
- *Goal Violation Summary*
- *Technology Summary*

Results Summary

This section provides the following information:

- **Goal Run**: The name of the goal that was run
- **Top Module**: Top module name
- **Report Directory**: The standard reports directory that contains the text report files.
- **Log File**: The SpyGlass log file that is generated when a goal is run. The path of the log file appears as a hyperlink and selecting the link opens the log file in a separate window.

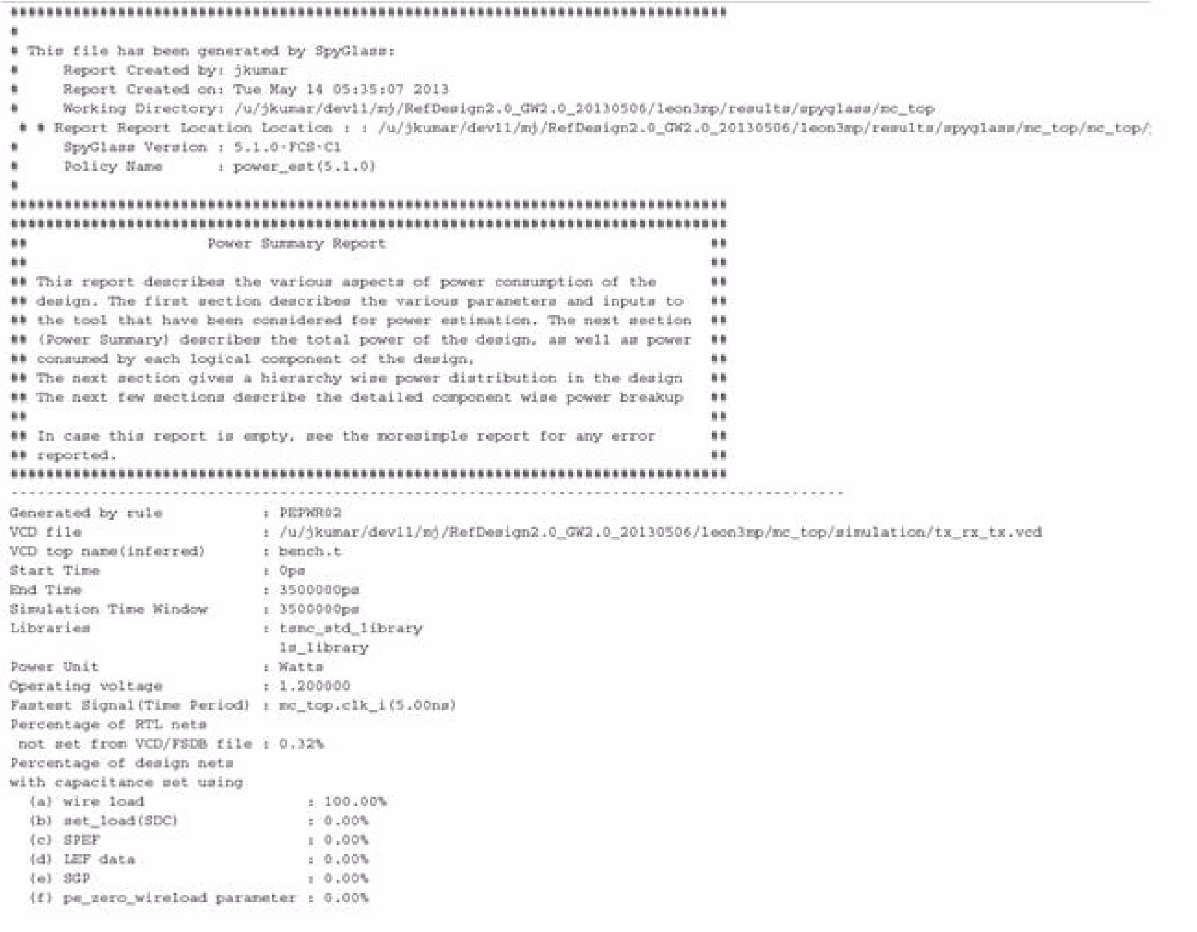
Standard Reports

This section shows all default text reports that are generated as part of the goal run. The report names appear as hyperlinks and clicking on the link open the reports.

Technology Reports

This section shows all the product-specific reports. You can click on a report link to open the report in a separate window.

Following figure illustrates a user-selected technology report content when the pe_summary report link is selected.



```
################################################################################
#
# This file has been generated by SpyGlass:
#      Report Created by: jkumar
#      Report Created on: Tue May 14 05:35:07 2013
#      Working Directory: /u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/results/spyglass/mc_top
 # # Report Report Location Location : : /u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/results/spyglass/mc_top/mc_top/
#      SpyGlass Version : 5.1.0-FCS-C1
#      Policy Name      : power_est(5.1.0)
#
################################################################################
################################################################################
##                      Power Summary Report                                  ##
##                                                                            ##
## This report describes the various aspects of power consumption of the     ##
## design. The first section describes the various parameters and inputs to  ##
## the tool that have been considered for power estimation. The next section  ##
## (Power Summary) describes the total power of the design, as well as power  ##
## consumed by each logical component of the design.                          ##
## The next section gives a hierarchy wise power distribution in the design   ##
## The next few sections describe the detailed component wise power breakup   ##
##                                                                            ##
## In case this report is empty, see the moresimple report for any error     ##
## reported.                                                                  ##
################################################################################
-----------------------------------------------------------------------------------------------
Generated by rule             : PEPWR02
VCD file                      : /u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/mc_top/simulation/tx_rx_tx.vcd
VCD top name(inferred)        : bench.t
Start Time                    : 0ps
End Time                      : 3500000ps
Simulation Time Window        : 3500000ps
Libraries                     : tsmc_std_library
                                ls_library
Power Unit                    : Watts
Operating voltage             : 1.200000
Fastest Signal(Time Period)   : mc_top.clk_i(5.00ns)
Percentage of RTL nets
 not set from VCD/FSDB file   : 0.32%
Percentage of design nets
with capacitance set using
  (a) wire load                    : 100.00%
  (b) set_load(SDC)                : 0.00%
  (c) SPEF                         : 0.00%
  (d) LEF data                     : 0.00%
  (e) SGP                          : 0.00%
  (f) pe_zero_wireload parameter   : 0.00%
```

FIGURE 284. The pe_summary Report

Goal Violation Summary

This section presents the waived and reported violation statistics of the goal.

Technology Summary

If the products specify to present some summary corresponding to the goal, it will be shown here with hyper links to the appropriate reports that elaborates more details. If there is no data available to show, this section would show empty.

Switching to the Old Dashboard Report

To switch back to the previous version of the DashBoard report, set the SPYGLASS_GEN_OLD_DASHBOARD environment variable prior to creating the DashBoard report:

```
setenv SPYGLASS_GEN_OLD_DASHBOARD YES
```

or

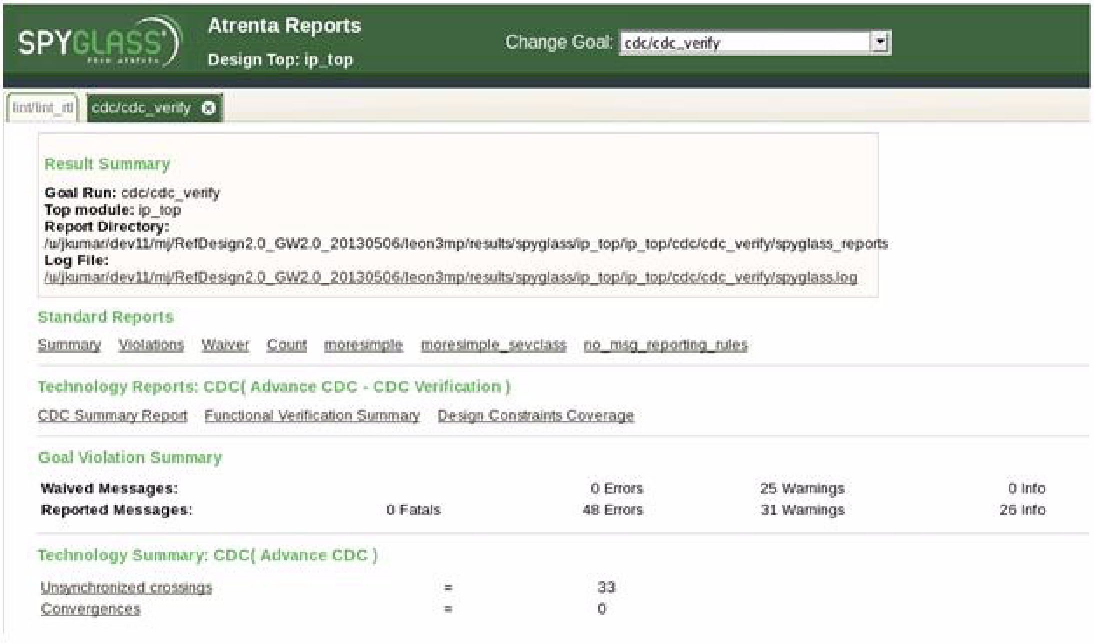
```
setenv SPYGLASS_GEN_OLD_DASHBOARD 1
```

Goal Summary

Goal Summary provides information on all the executed goal summaries for the current Project.

By default, SpyGlass prepares necessary individual goal summary data as part of each goal run and after completing all goal execution in the project, SpyGlass consolidates data and generates the goal summary.

The Goal Summary, by default, displays the first goal result as shown in the following figure:



Atrenta Reports

Design Top: ip_top

Change Goal: cdc/cdc_verify

Result Summary

Goal Run: cdc/cdc_verify
Top module: ip_top
Report Directory:
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/results/spyglass/ip_top/ip_top/cdc/cdc_verify/spyglass_reports
Log File:
/u/jkumar/dev11/mj/RefDesign2.0_GW2.0_20130506/leon3mp/results/spyglass/ip_top/ip_top/cdc/cdc_verify/spyglass.log

Standard Reports

Summary Violations Waiver Count moresimple moresimple_sevclass no_msg_reporting_rules

Technology Reports: CDC(Advance CDC - CDC Verification)

CDC Summary Report Functional Verification Summary Design Constraints Coverage

Goal Violation Summary

Waived Messages:		0 Errors	25 Warnings	0 Info
Reported Messages:	0 Fatals	48 Errors	31 Warnings	26 Info

Technology Summary: CDC(Advance CDC)

Unsynchronized crossings	=	33
Convergences	=	0

FIGURE 285. Sample Goal Summary

You can chose other goals using the **Change Goal** drop-down list available on the top-right corner of the report to open the respective goal summary pages in new tabs.

For more information on sections of the Goal Summary, see *Generating the HTML Goal Summary Page*.

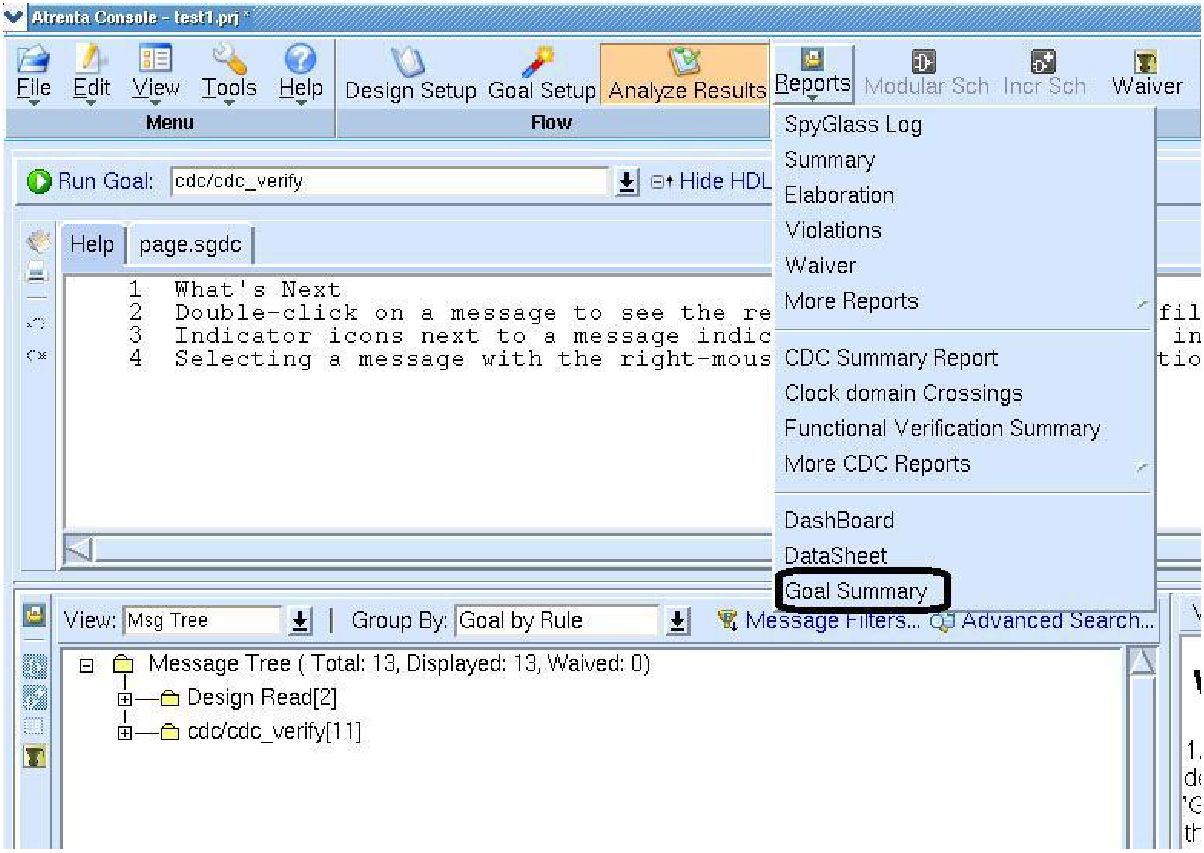
To create the Goal Summary, SpyGlass uses the dashboard license. If the license is not available an empty report is displayed. You can also disable goal summary generation using the following command:

```
set_option disable_html_report {html}
```

The Goal Summary is generated, by default, at the following location after the project analysis:

```
<project work directory>/html_reports/goals_summary.html
```

You can also launch the report by selecting the **Analyze Results** stage and selecting the **Reports-> HTML Report** option in the main menu of SpyGlass SpyGlass as shown in the following figure:



Run Goal: cdc/cdc_verify

```
What's Next
Double-click on a message to see the re
Indicator icons next to a message indic
Selecting a message with the right-mous
```

View: Msg Tree | Group By: Goal by Rule

Message Tree (Total: 13, Displayed: 13, Waived: 0)
- Design Read[2]
- cdc/cdc_verify[11]

FIGURE 286. Goal Summary Menu

Managing Datasheet and Dashboard Reports

The SpyGlass Datasheet and Dashboard reports are generated in a common directory. This directory contains all the additional reports and graphs, which are linked to the main report.

You can leverage SpyGlass reports to:

- **Increase Project Metrics Visibility**: You can copy or move the reports or files to a newer location. For example, you can copy the files to a location where an internal project Web Server can access the files. This enables increased SpyGlass project metrics visibility at all levels of management.
- **Access Reports Easily:** Moving the files to a web server enables you to access files independently of the Operating System and from other geographical locations within the organization. You can also archive reports or files for accessing them later, such as, when the actual SpyGlass work area is removed.

Share Metrics: You can zip the files and E-mail them to other project stakeholders thereby providing timely access to design metrics. You can also include the dashboard and datasheet reports along with the delivered IP as a way of communicating the status of design quality.

Special Features in SpyGlass

Memory Reduction Feature

When a memory is synthesized, each bit location of the synthesized memory is represented by a flip-flop or latch in the synthesized netlist. This can easily consume an appreciable amount of system memory, resulting in design capacity problems.

Synthesizing memories (two-dimensional arrays) in RTL designs has always been resource and time-consuming task. Some designs may even run out of system memory.

To solve this problem, SpyGlass provides an optional memory reduction feature for both Verilog and VHDL designs. You can use this feature when you are not able to read in the design because of large memories.

The memory reduction feature enables you to perform rule-checking on large memory modules within limited system resources. By using this feature, SpyGlass can analyze designs containing large two-dimensional arrays and still fully analyze around and through such arrays.

Specify the following command in the project file to enable the memory reduction feature:

```
set_option handlememory yes
```

Impact of Using this Feature

When the memory reduction feature is enabled, there will be some differences in the messages reported, as discussed below:

- The following rules will not report violations in some cases:

SpyGlass lint solution	W111	W456	W488
SpyGlass OpenMore solution	NotInSens	NotReqSens	

- The following rule will report violations in some cases:

SpyGlass lint solution	W175	W415
SpyGlass DFT solution	Topology_05	

- Violation message of some rules, such as BitPartInLHS rule may change in some cases.
- Some rules, such as DeadCode will report violation when the designs were earlier passing the rule-check and vice versa.
- Some rules, such as SYNTH_5243 do not report violation to suppress unwanted noise.

Apart from the above rules being affected, there are cases where rule messages will be generated only when the memory reduction feature is enabled as the memory modules are now being synthesized. In some other cases, number of rule messages will decrease.

Limitations

The memory reduction feature has the following limitations for the VHDL designs:

- Only two-dimensional arrays declared explicitly through TYPE declarations are processed. Memory handling for a design aborts in the following cases:
  - If any identifier (signal/variable/constant) of three-dimensional or higher array type is encountered.
  - If two-dimensional array TYPE declaration of any identifier in a design unit is in a precompiled library because precompiled library is left untouched for memory handling.
- Two-dimensional records are not supported.
- Two-dimensional ports are not supported.
- Functions of precompiled libraries having 2-D argument/return type are not supported.

Pattern Matching Across Features

SpyGlass uses pattern matching in waiver commands and in the search dialogs in the UI.

There are following modes for pattern matching:

- Using wildcard
- Using simple regular expressions

Wildcard Mode

The wildcard mode is the default mode. A string is considered as a wildcard if it includes an asterisk (*) or a question mark (?). However, if these characters are escaped by prefixing them with a backslash (\), they are considered as literals. For example, \?, *, and \\ are considered as ?, *, and \, respectively.

All other characters are treated as normal strings.

NOTE: *SpyGlass treats square brackets as literals.*

NOTE: *Interpretation of * and ? is similar for escaped design names. For example, if you want to specify a design object, '\abc* ', you must specify it as '\abc* '. On the other hand, if you want to specify all design objects whose names start with '\abc' then you need to write '\abc* ' (Verilog only) or '\abc*\' (VHDL only).*

A few parameters (examples taken from SpyGlass CDC solution) which support wildcard:

- synchronize_cells
- synchronize_data_cells
- clock_gate_cell
- glitch_protect_cell
- reset_synchronize_cell

The following constraints support wild cards:

Constraint	**Product**
memoryType	SpyGlass DFT solution
requirePath	SpyGlass DFT solution

Constraint	Product
testmode	SpyGlass DFT solution
clockgatingcell	SpyGlass DFT solution
pdsequence	SpyGlass Power Verify solution
powerdomainoutputs	SpyGlass Power Verify solution

Regular Expression Mode

A string that is prefixed with m/ and terminated with / (forward slash) is treated as a regular expression. The following delimiters can also be used in place of the delimiter /:

%	**!**	**@**	**^**	**&**	*****	**;**	**/**	**~**	**?**	**<**	**>**	**+**	**=**	**|**

Normal Mode

A string that is prefixed with q/ and terminated with / is treated literally, that is, if a string contains *, ?, or \, it will be treated literally. The delimiters explained in the *Regular Expression Mode* can also be used in place of /.

If a string does not contain an escaped *, ?, or \, it is treated as a wildcard string. To treat a string containing * or ? or a \ as a normal string, escape *, ?, or \ characters or prefix the string with q/ and terminate it with /.

Hierarchy Separator

All character except the following are treated as hierarchy separators in a hierarchy string:

- Any alpha numeric character
- $ (dollar)
- _ (underscore)
- [(opening square bracket)
-] (closing square bracket)

- : (colon)
- \ (backslash)

The hierarchy string contains only one hierarchy separator, that is, the first character that is not from the above set is considered the hierarchy separator, and the rest of the string is considered by using this character as a hierarchy separator.

NOTE: *For files and directory type objects, / is the only hierarchy separator.*

NOTE: *When you use pattern matching in a hierarchy, SpyGlass matches only a single hierarchy. This holds true for all objects, such as design objects (net, port, instance, terminals, and so on), files, and directories that use pattern matching.*

Design Save and Restore Feature

SpyGlass provides the Design Save-Restore feature that enables you to reuse the data from a prior synthesis run. This saves significant runtime for analysis that requires synthesis.

This feature saves runtime on repeated SpyGlass analysis runs when the HDL source is unchanged. It allows you to either restore or synthesize the entire list depending upon the change across save-restore runs.

Some types of analysis require you to run synthesis. In such cases, the Save-Restore capability is disabled.

By default, the Save-Restore is enabled in SpyGlass. You can disable this feature in the *Preferences* dialog (*Tools-> Preferences*), under the *Miscellaneous* category. Alternatively, you can specify the following command in the project file to disable this feature:

```
set_option enable_save_restore no
```

This section also explains the *Incremental Save and Restore* and *Restoring From Multiple Databases* along with the *Examples*.

Incremental Save and Restore

SpyGlass provides support for incremental restore of saved design using the incremental save and restore feature. If any of the design files have changed across save-restore runs then instead of synthesizing the entire netlist again, SpyGlass only synthesizes the modules that are impacted by the change. This feature helps in saving the synthesis time for subsequent restore runs.

NOTE: *You can use this feature for a single database. To restore using multiple DBs, see Restoring From Multiple Databases.*

This feature works only when the *Design Save and Restore Feature* is enabled. This means that the *enable_save_restore* `option` is enabled to use the Incremental Save and Restore Feature.

To enable this feature, specify the following command in the Other Command Line Option(s) field in the Set Read Options tab:

allow_incr_save_restore

Alternatively, you can specify the following command in the project file to enable this feature,

```
set_option allow_incr_save_restore yes
```

Example 1

Consider the following sample message for a verilog module:

```
Synthesizing module: vlog_module (elaborated name: vlog_module)
... (Module 1 of total 100) done
```

The above message signifies that fresh synthesis is done for the module, `vlog_module`, due to change in the design file.

Example 2

Consider the following sample message for a VHDL module:

```
Synthesizing design unit: Entity.Arch (library: WORK,
elaborated name: Entity) ... (Module 1 of total 100)  done
(restored from dbdir)
```

The above message signifies that the module, `Entity.Arch`, is restored from the database.

Restoring From Multiple Databases

You can perform incremental restore/synthesis of the saved design from multiple databases using the `ipdbdir` switch. Specify the following command in the project file to specify a list of db's from which you want to restore.

```
set_option ipdbdir {list of db's}
```

Here, each individual db corresponds to an individual IP block. Depending on the impact of change across save-restore run, the netlist modules are either restored from their respective databases or synthesized afresh incrementally in the restore run.

NOTE: *Picking up synthesized view of modules coming from precompiled dump is not supported. That is, any module which is coming from precompiled dump will always get resynthesized.*

You can use a mix of design files and precompiled dumps. In this case, modules coming from design files are picked up from synthesized DB, while modules coming from precompiled dump are resynthesized.

Examples

Examples for the following cases are described below:

- *No Change Across Save-restore Runs*
- *Design File Change*
- *Module-Specific Option Change*
- *Global Option Change*

No Change Across Save-restore Runs

Run 1 (save1)

DB for hierarchy-1 is saved using following project file, project.prj:

project.prj

```
##Data Import Section
read_file -type verilog bottom1.v mid1.v top1.v


##Common Options Section
set_option enable_save_restore yes
set_option allow_incr_save_restore yes
set_option projectwdir .
set_option top top1

compile_design
```

The database is saved at the following location:

```
<projectwdir>/<project-name>/<top-name-if-
any>.SG_SaveRestoreDB
```

That is, for this example, the database is located at the following location:

```
save1/project/top1/.SG_SaveRestoreDB
```

Run 2 (save2)

Similarly, second database is created at the following location:

```
save2/project/top2/.SG_SaveRestoreDB
```

Run 3 (restore)

Both `top1` and `top2` are instantiated in a high-level SoC, top, defined in `top.v` as shown below:

```
read_file -type verilog bottom1.v mid1.v top1.v bottom2.v
mid2.v top2.v top.v
set_option ipdbdir {save1/project/top1/.SG_SaveRestoreDB
save2/project/top2/.SG_SaveRestoreD}
```

The `top` is synthesized as shown below:

```
Synthesizing module: bottom2 (elaborated name: bottom2) ...
(Module 1 of total 7) done (restored from dbdir save2/
project/top2/.SG_SaveRestoreDB)

Synthesizing module: mid2 (elaborated name: mid2) ... (Module
2 of total 7) done (restored from dbdir save2/project/top2/
.SG_SaveRestoreDB)

Synthesizing module: top2 (elaborated name: top2) ... (Module
3 of total 7) done
Synthesizing module: bottom1 (elaborated name: bottom1) ...
(Module 4 of total 7) done (restored from dbdir save1/
project/top1/.SG_SaveRestoreDB)

Synthesizing module: mid1 (elaborated name: mid1) ... (Module
5 of total 7) done (restored from dbdir save1/project/top1/
.SG_SaveRestoreDB)

Synthesizing module: top1 (elaborated name: top1) ... (Module
6 of total 7) done

Synthesizing module: top (elaborated name: top) ... (Module 7
of total 7) done
```

Design File Change

Run 1 (save1)

See *Run 1 (save1)*

Run 2 (save2)

See *Run 2 (save2)*

Run 3 (restore)

The `mid1.v` is changed across save-restore run and hence, netlist is synthesized as shown below:

```
Synthesizing module: bottom2 (elaborated name: bottom2) ...
(Module 1 of total 7) done (restored from dbdir save2/
project/top2/.SG_SaveRestoreDB)

Synthesizing module: mid2 (elaborated name: mid2) ... (Module
2 of total 7) done (restored from dbdir save2/project/top2/
.SG_SaveRestoreDB)

Synthesizing module: top2 (elaborated name: top2) ... (Module
3 of total 7) done

Synthesizing module: bottom1 (elaborated name: bottom1) ...
(Module 4 of total 7) done (restored from dbdir save1/
project/top1/.SG_SaveRestoreDB)

Synthesizing module: mid1 (elaborated name: mid1) ... (Module
5 of total 7) done

Synthesizing module: top1 (elaborated name: top1) ... (Module
6 of total 7) done

Synthesizing module: top (elaborated name: top) ... (Module 7
of total 7) done
```

Module-Specific Option Change

Run 1 (save1)

See *Run 1 (save1)*

Run 2 (save2)

See *Run 2 (save2)*

Run 3 (restore)

A parameter, `bottom2_p1`, defined in `bottom2` is changed in restore run as shown below:

```
set_option param bottom2.bottom2_p1=4
```

Now, the modules are synthesized in incremental fashion as shown below:

```
Synthesizing module: bottom2 (elaborated name: bottom2) ...
(Module 1 of total 7) done

Synthesizing module: mid2 (elaborated name: mid2) ... (Module
2 of total 7) done

Synthesizing module: top2 (elaborated name: top2) ... (Module
3 of total 7) done

Synthesizing module: bottom1 (elaborated name: bottom1) ...
(Module 4 of total 7) done (restored from dbdir save1/
project/top1/.SG_SaveRestoreDB)

Synthesizing module: mid1 (elaborated name: mid1) ... (Module
5 of total 7) done (restored from dbdir save1/project/top1/
.SG_SaveRestoreDB)

Synthesizing module: top1 (elaborated name: top1) ... (Module
6 of total 7) done
```

```
Synthesizing module: top (elaborated name: top) ... (Module 7
of total 7) done
```

Global Option Change

Run 1 (save1)

See *Run 1 (save1)*

Run 2 (save2)

See *Run 2 (save2)*

Run 3 (restore)

An option having global impact is changed in restore run as shown below:

```
set_option ignoredu mid1
```

In this case, the entire netlist is resynthesized as shown below:

```
Synthesizing module: bottom2 (elaborated name: bottom2) ...
(Module 1 of total 7) done

Synthesizing module: mid2 (elaborated name: mid2) ... (Module
2 of total 7) done

Synthesizing module: top2 (elaborated name: top2) ... (Module
3 of total 7) done

Synthesizing module: bottom1 (elaborated name: bottom1) ...
(Module 4 of total 7) done

Synthesizing module: mid1 (elaborated name: mid1) ... (Module
5 of total 7) done

Synthesizing module: top1 (elaborated name: top1) ... (Module
6 of total 7) done

Synthesizing module: top (elaborated name: top) ... (Module 7
of total 7) done
```

NOTE: *The I command is not recommended to be used in the project file. For details, see Options Not Recommended. However, you should specify the directory paths using the -I option on the command-line itself while invoking console or sg_shell.*

Save Restore Sensitive Options

Following table contains the list of save-restore sensitive options having global impact i.e., if any of these options change across DBs and the current run, then nothing gets picked up and it'll be a case of fresh synthesis

87	ovl_verilog	allow_incr_save_restore	ovl_vhdl
allow_celldefine_as_top	prefer_tech_lib	blackboxdir	resetall
compilelibs	sfcu	convert_udp_to_latch	sgsyn_clock_gating
def	sgsyn_clock_gating_threshold	stop/stopdir	sgsyn_enable_two_pass_flow
ignoredu	stopfile	sglib/gateslib	stopmodule
mthresh	synth_bypass	dw	synth
disable_gateslib_autocompile	synth_netlist	disable_handlememory	synth_vhdl_bypass
disableSV/enableSV	synth_vhdl_netlist	disablev2k	vlog2001_generate_name
enable_hbo	ignorefile	enable_hdl_encryption	libmap
enable_pgnetlist	macro_synthesis_off	enable_sbo	netlist
gateslibtype	no_celldefine_messages	handlememory	nopreserve
hdlin_synthesis_off_skip_text	no_rcheck_celldefine	hdlin_translate_off_skip_text	disableSV09/enableSV09
disable_amg/enable_amg	pragma	disable_hdlin_translate_off_skip_text	disable_hdlin_synthesis_off_skip_text

Handling Built-in Messages during Save-Restore Flow

During the save run, SpyGlass reports various built-in messages, such as parsing, elaboration, and synthesis. These messages are saved as a part of the saved design database.

To view these messages during restore run, the *Enable Save Restore for BuiltIn Rules* and *Enable Save Restore Flow* fields (under the *Set Read Options* tab) should be set to *Yes* during the restore run.

NOTE: *By default, the Enable Save Restore for BuiltIn Rules field is automatically set to Yes when you set the Enable Save Restore Flow field to Yes during the restore run.*

If you set the *Enable Save Restore for BuiltIn Rules* field to *No*, SpyGlass reports a message to indicate that built-in messages are not restored as the corresponding field is set to *No*.

Handling Builtin Messages Based on the Type of Current Run

Based on whether the current run is full restore or partial restore, built-in messages are handled in the following ways:

Full restore	In this case, no parsing, elaboration, or synthesis occurs. Therefore, all the messages reported by each of these stages are restored if you set the *Enable Save Restore for BuiltIn Rules* field to *Yes* under the *Set Read Options* tab.
Partial restore	In this case, parsing and elaboration occurs but no synthesis occurs. Therefore, only the messages reported during synthesis (during save run) are restored if you set the *Enable Save Restore for BuiltIn Rules* field to *Yes* under the *Set Read Options* tab. The parsing and elaboration messages are reported by the respective engines.

Impact of addrules/ignorerules Options

If you enable a built-in message during the restore run by using the *addrules* command, but that message was disabled during the save run, SpyGlass will not display that message during the restore run even if you have set the *Enable Save Restore for BuiltIn Rules* field to *Yes* under the *Set Read Options* tab.

In such cases, you should save the database forcefully by using the *addrules* command to make that message available in the restore run. If you do not save the database forcefully, SpyGlass reports a message in the restore run to indicate that the specified built-in message has actually not run in the restore run and the user needs to save the database again.

If you disable a built-in rule during the restore run by using the *ignorerules* command, but that rule was enabled during the save run, messages of such rules are not displayed during the restore run if you set the *enable_save_restore_builtin* command to `yes` command in a project file.

The following table describes the impact of *addrules*/*ignorerules* commands on built-in messages during save-restore runs:

Built-in Rule Type	**Built-in Message Reported During Save run?**	**Built-in Message Reported During Restore run?**
Off by default	No	no
Off by default	Yes (*addrules* is specified)	yes (*addrules* specified)
Off by default	No (*addrules* not specified)	no (even if *addrules* is specified)
On by default	Yes	yes
On by default	Yes	no (*ignorerules* specified)
On by default	No (*ignorerules* specified)	no (even if *ignorerules* is not specified)

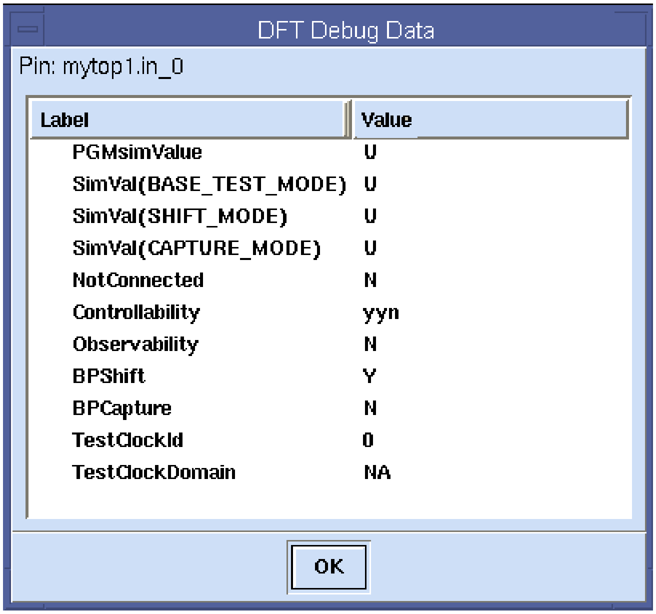
SpyGlass DFT Solution Back Annotation Feature

The Back Annotation feature of the SpyGlass DFT solution enables you to view debugging information in the schematics (Modular Schematic and Incremental Schematic) at the following two levels:

- Terminal level
- Instance level

To view SpyGlass DFT solution debugging information, you need to do the following:

1. Set the value of the `dftDebugData` rule parameter to `on`.
2. Start SpyGlass analysis.
3. Double-click the violation message for which the data can be viewed in the schematic.
4. Click the Schematic view or the Incremental Schematic view icon to open the schematic.
5. To view the SpyGlass DFT solution debug data for a terminal, right-click the terminal and select the *Show Debug Data-> DFT* context-menu option. This displays the *DFT Debug Data* window, as shown in the following figure:



DFT Debug Data

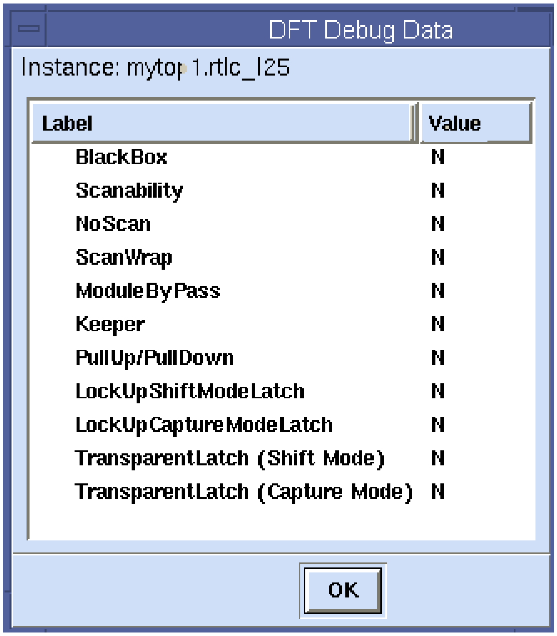
Pin: mytop1.in_0

Label	Value
PGMsimValue	U
SimVal(BASE_TEST_MODE)	U
SimVal(SHIFT_MODE)	U
SimVal(CAPTURE_MODE)	U
NotConnected	N
Controllability	yyn
Observability	N
BPShift	Y
BPCapture	N
TestClockId	0
TestClockDomain	NA

FIGURE 287. DFT Debug Data for a Terminal

SpyGlass DFT solution debug data for a terminal includes information, such as the simulation value in the shift and capture modes, controllability, blocked path due to the terminal in shift mode (BPShift), and blocked path due to the terminal in the capture mode (BPCapture).

6. To view SpyGlass DFT solution debug data for an instance, right-click the instance and select the *Show Debug Data-> DFT* menu option from the context menu. This displays the *DFT Debug Data* window, as shown in the following figure:



DFT Debug Data

Instance: mytop1.rtlc_I25

Label	Value
BlackBox	N
Scanability	N
NoScan	N
ScanWrap	N
ModuleByPass	N
Keeper	N
PullUp/PullDown	N
LockUpShiftModeLatch	N
LockUpCaptureModeLatch	N
TransparentLatch (Shift Mode)	N
TransparentLatch (Capture Mode)	N

FIGURE 288. DFT Debug Data for an Instance

SpyGlass DFT solution debug data for an instance includes information, such as:

- ❒ Scannability information of a flip-flop
- ❒ Module by pass information that is applied through SGDC commands on a black box
- ❒ Re-timing information on a latch in the shift mode or capture mode
- ❒ Transparency information for a latch in the shift mode or capture mode
- ❒ Scan-wrap information applied on a black box instance

Support for Verilog Configuration

Verilog Configuration provides the ability to specify design configurations. It is consists of a set of rules to apply when searching for library cells to which instances are to be bind. A configuration may change the binding of a module, primitive, interface or program instance. However, the configuration can not change the binding of a package. Configurations mainly specify the set of libraries from which master of an instance is to be bind. A library can contain modules, interfaces, packages and other configurations.

This section explains the following topics:

- *Related Commands*
- *Related Reports*

Related Commands

The support for Verilog configuration is enabled through the following commands:

- *enable_vlog_config*: Enables the support for verilog configuration
- *strict_vlog_config*: Converts the instances, which are unable to bind, using the verilog configuration rules, into black box and assigns the ELAB error to them
- *write_vlog_config_report*: Enables reporting of instances, which are bound using the configuration rules.

Related Reports

The *write_vlog_config_report* command generates a binding report, ReportConfiguration.log, which lists all the instances that are bound using the configuration rules. The following is the a sample snippet of the ReportConfiguration.log file:

```
###################################################

## Instance                         : top.I2.I3.I5
## Configuration Binding            : L5.superbottom
## Configuration Binding Rule       : instance top.I2.I3.I5
                                       use L5.superbottom ;
## Configuration Line Number        : 12
## Configuration File Name          : config.v

###################################################
Details :
Instance in report suggests hierarchical name for the
instance from the top.

Configuration binding suggests masterName and library Name to
which the instance is bound.

Configuration binding Rule suggests the configuration rule
using which the instance is bound.
Configuration Line number suggests the line number of binding
rule
Configuration File Name suggests the binding rule
configuration filename.

There may be cases where binding for all the instances down
the hierarchy of a particular instance are shared. In that
case, the report dumps only the binding information of
hierarchy's top instance. The dump will look like:
###################################################
## Note      : Complete Hierarchy of top.I2.I4 instance will
be shared with top.I1.I4 instance's Hierarchy
###################################################
```

Record-Mapping to Flattened Bus

When you specify VHDL records, SpyGlass internally flattens the corresponding record elements into bus.

Consider the following VHDL record:

```
type rec1 is
record
  field1 : std_logic_vector(2 downto 0);
  field2 : std_logic;
  field3 : std_logic;
  field4 : std_logic_vector(0 to 2);
end record;
signal sig1 : rec1; --Flattened bus: sig1[7:0]
```

For each record element, the following table displays the flattened bus and its mapping details:

Record Element	**Flattened Bus**	**Mapping**
field1	sig1[2:0]	sig1[2] <=> sig1.field1(2), sig1[0] <=> sig1.field1(0)
field2	sig1[3]	sig1[3] <=> sig1.field2
field3	sig1[4]	sig1[4] <=> sig1.field3
field4	sig1[7:5]	sig1[7] <=> sig1.field4(0), sig1[5] <=> sig1.field4(2)

NOTE: *Arrays with range defined as (n downto 0) or (0 to n) in RTL maps to (0 to n) or (n downto 0) order at the flattened level.*

Handling Complex VHDL Records

SpyGlass handles complex VHDL records, such as a record within another record or an array of records during post-synthesis.

After generating a netlist of the design containing complex VHDL record(s), you can apply constraints on records.

Specifying Constrains on Record Bits

Consider the following VHDL code snippet containing a complex record:

```
type simple_record is
  record
    data4 : std_logic;
    data5 : std_logic_vector(7 downto 0);
  end record;

type record_data_type is
  record
    data1 : std_logic;
    data2 : std_logic_vector(7 downto 0);
    data3 : simple_record;
  end record;
```

Also consider that you specify the following internal signal:

```
signal sig1:record_data_type;
```

The naming convention for record bits is <record-instance>_<field-name>.

If the record field is an array, refer the field name through its indices.

The following example applies constraints on record sig1 bits:

```
// applying set_case_analysis constraint on record bit
// sig1.data3.data4
set_case_analysis -name sig1_data3_data4 -value 1

// applying set_case_analysis constraint on record bit
// sig1.data3.data5(0)
set_case_analysis -name sig1_data3_data5[0] -value 1
```

Specifying Constrains on Array of Records

Consider the following VHDL code snippet containing an array of records:

```
type simple_record is
record
  data1 : std_logic;
  data2 : std_logic_vector(1 downto 0);
end record;

type array_data_type is array (1 downto 0) of
simple_record ;
signal sig1:array_data_type
```

The naming convention for an array records is <record-instance>[<index>].

SpyGlass refers an array of records as a one-dimensional net-bundle, as given below:

```
sig1(0).data1        sig1[0]
sig1(0).data2[0]     sig1[1]
sig1(0).data2[1]     sig1[2]
sig1(1).data1        sig1[3]
sig1(1).data2[0]     sig1[4]
sig1(1).data2[1]     sig1[5]
```

All record elements in the net-bundle are accessed using indices while applying constraints.

The following example applies constraint on an array of record sig1 bits:

```
// Applying the set_case_analysis constraint on the
// record array sig1(0).data1
set_case_analysis -name sig1[0] -value 1
```

Specifying Constraints through GUI

To apply constraints on record bits through GUI, perform the following steps:

1. Select the required net by following the net-naming convention in the schematic.
1. Right-click on the required net.
2. Select the *Set SGDC Constraints* option from the shortcut menu.

This displays the *Constraints Editor* window.

3. In the *Constraints Editor* window, specify the required constraints.

This is the recommended way to specify constraints for records bits as you can view the naming convention of nets directly in the schematic, and apply constraints accordingly.

To view the naming convention of a net in the schematic, move the cursor over that net. A tool-tip appears displaying the net name, as shown in the following figure:

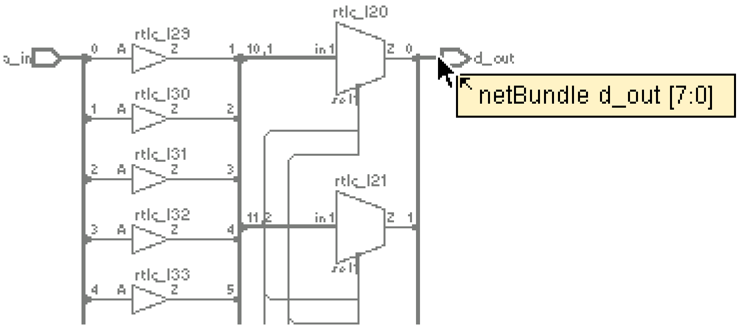


FIGURE 289. Tool-Tip for a Net Name

Incremental Schematic Abstraction Feature

A schematic highlights connectivity in a design while focusing on the design part that contains a violation. However, in large designs, highlighted paths are quite lengthy. As a result, it becomes difficult to visualize the connectivity and this takes away the focus from main issue being reported.

To solve this problem, the incremental schematic abstraction feature abstracts different types of logic into abstract groups to make a complex schematic appear less cluttered. This enables a user to focus only on relevant objects/paths in the schematic.

Combinational Logic Abstraction

Incremental schematic abstraction feature clubs all combinational logic present within registers in a combinational cloud. This enables a user to focus only on registers in the schematic and not on the combinational logic present inside registers.

The following figure shows the schematic in which combinational logic within each register is clubbed in a combinational cloud:

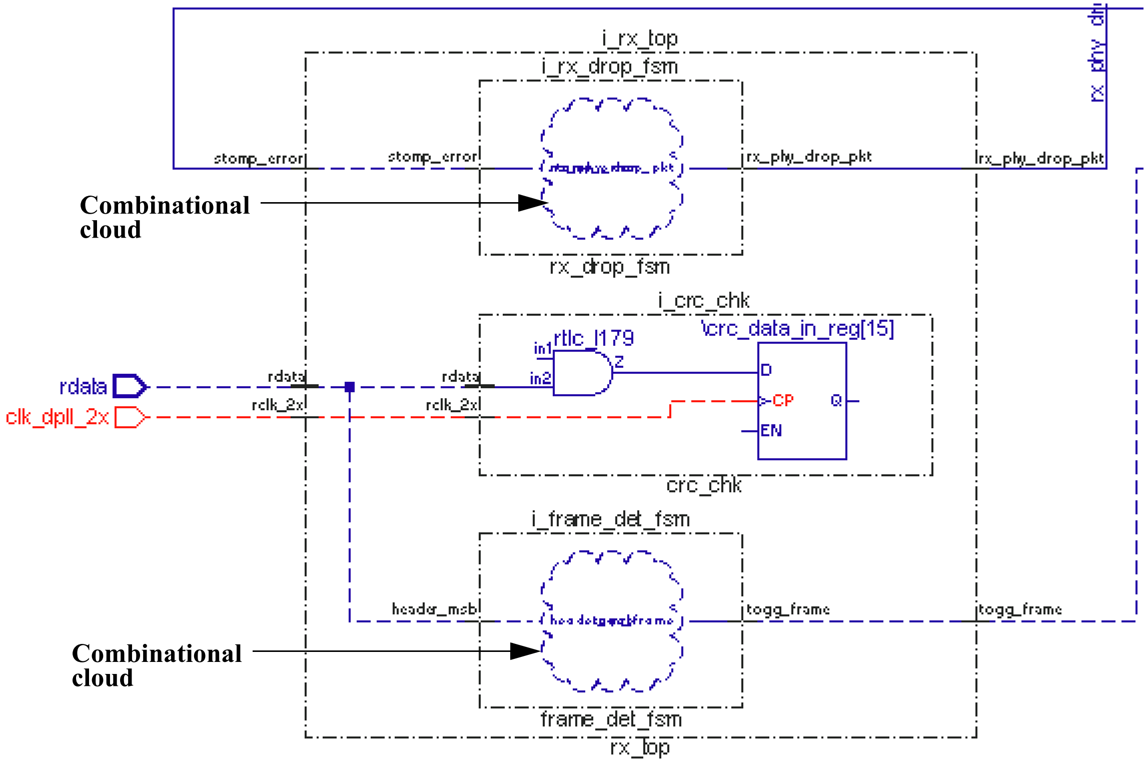


FIGURE 290. Combinational Logic Clubbed in a Combinational Cloud

If you want to view combinational logic within registers, click the button in the *Incremental Schematic* window. The schematic now changes to the following:

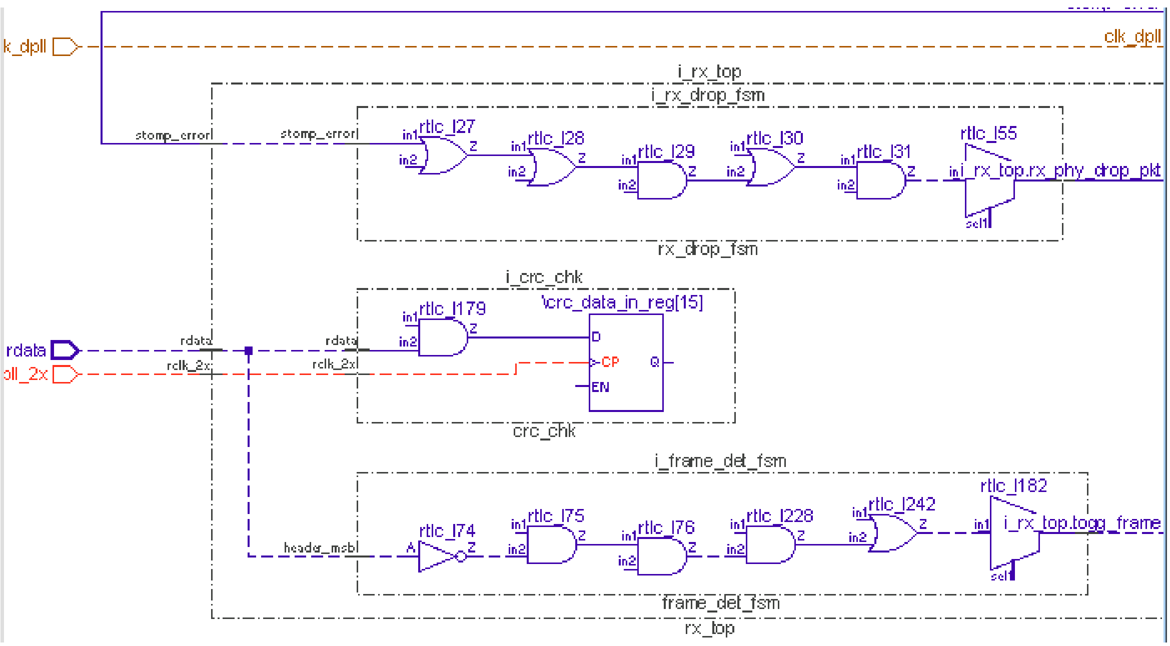


FIGURE 291. Combinational Logic in a Register

You can again club all combinational logic within registers back into a combinational cloud by clicking the ▷ button in the *Incremental Schematic* window.

Buffer Chains Abstraction

Incremental schematic abstraction feature replaces a chain of buffers or inverters with a single buffer or inverter in a schematic.

A chain of buffers or inverters generally does not have any logical significance from the point of view of debugging.

The following figure shows the schematic in which a chain of buffers has been replaced with a single buffer:

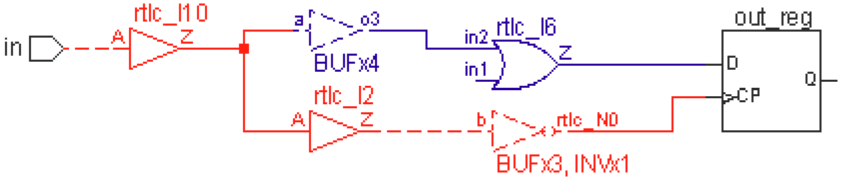


FIGURE 292. Chain of Buffers Replaced with a Single Buffer

NOTE: *Only buffers that are in a straight chain can be abstracted as a single group. In case of any branching, this feature creates abstract groups for each sub chain.*

If you want to view the complete chain of buffers, click the button in the *Incremental Schematic* window. The schematic now changes to the following

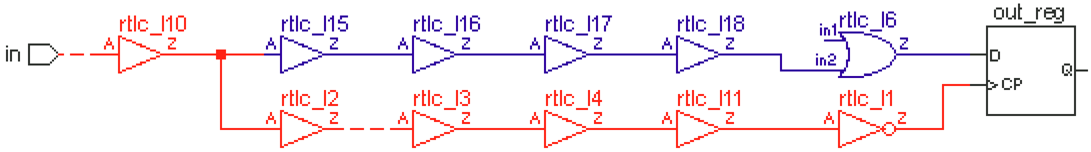


FIGURE 293. Complete Chain of Buffers

To replace the above chain back with a single buffer, click the button in the *Incremental Schematic* window.

Abstraction between Start and End Points

Incremental schematic abstraction feature abstracts logic between start and end points in the schematic.

For example, consider the following figure:

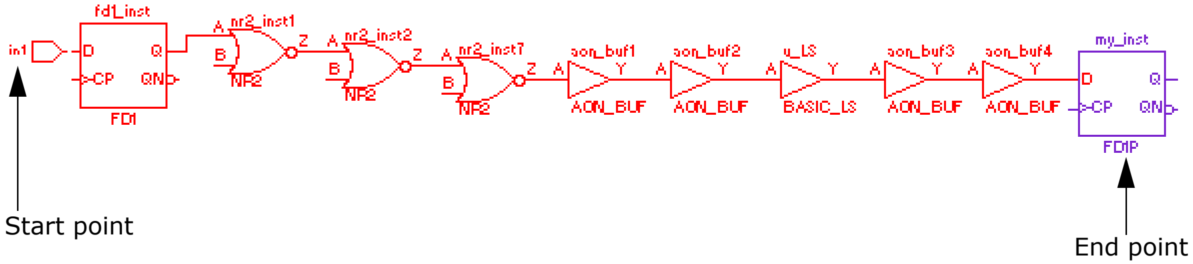


FIGURE 294. Start and End Point

In the above schematic, click the ▷ button to abstract logic between the start and end points.

The following figure shows the modified schematic:

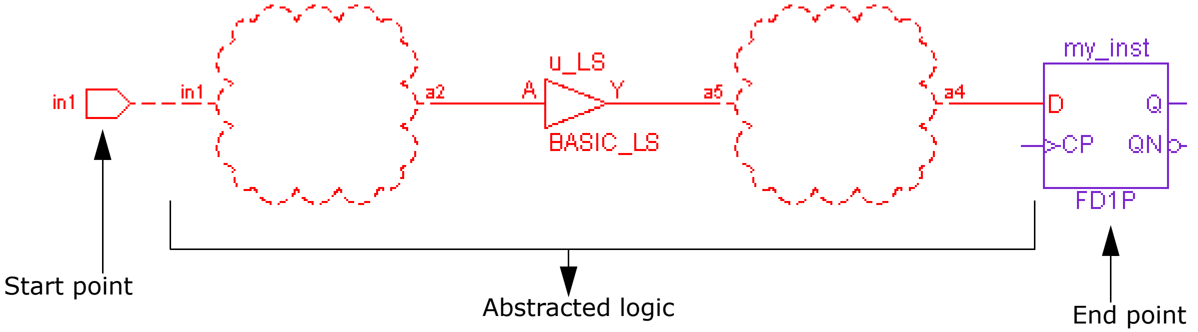


FIGURE 295. Abstracted Logic Between Start and End Points

Based on the functionality of a rule, certain logic between start and end points are not abstracted. For example, in the above figure, the level shifter is not abstracted.

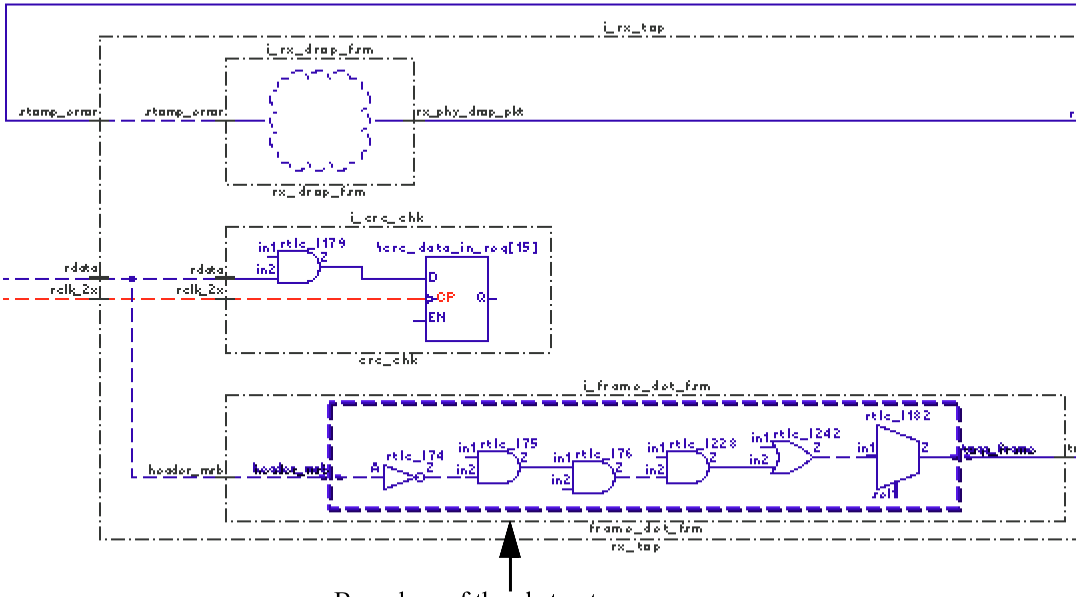
Viewing Logic within an Abstract Group

To view logic within a particular abstract group, perform any of the following actions:

- Double-click on the required group.

- Right-click on a group and select the *Expand Group* option from the shortcut menu.

SpyGlass then displays all the logic within the boundary of that abstract group, as shown in the following figure:



Boundary of the abstract group

FIGURE 296. Schematic Showing Boundary of Abstract Group

To collapse the expanded group, right-click on that group and select the *Collapse Group* option from the shortcut menu.

Viewing or Hiding Pins of an Abstracted Group

To view all pins of an abstracted group, right-click on that group and select the *Show All Pins* option from the shortcut menu.

To hide pins that have not been probed or traced, right-click on the corresponding abstract group and select the *Hide Untraced Pins* option from the shortcut menu.

Removing All Abstract Groups

To remove all abstract groups in the schematic, click the *Disable Groups* button ( ) in the toolbar.

Appendix

Supported HDL Directives

Command-line directives and equivalent project commands:

Directives	Corresponding command
Text macros used (+define)	set_option define <macro>
Include files (+incdir)	set_option incdir <paths>
Verilog library files (-v)	set_option v {files}
Verilog library directories (-y)	set_option y {dirs}
Verilog library extension (+libext)	set_option libext {exts}

Verilog User-Defined Primitives

SpyGlass translates User-Defined Primitives (UDPs) to an equivalent Verilog module description for further processing. UDP definitions that cannot be translated are treated as black boxes.

In some cases, while translating a UDP with both edge and level sensitiveness, SpyGlass is not able to determine whether that UDP definition should be converted into a flip-flop or a latch. In such cases, SpyGlass infers such UDP definitions as flip-flops by default. To infer them as latches, specify the following command in the project file:

```
set_option convert_udp_to_latch yes
```

Supported Verilog HDL directives are:

- 'uselib
- 'define

NOTE: *You can include the 'define directives in Verilog source files. You must first analyze these files in SpyGlass before analyzing other design files.*

- 'include

NOTE: *SpyGlass ignores duplicate file names.*

- 'celldefine and 'endcelldefine

NOTE: *Any module enclosed in 'celldefine and 'endcelldefine directives is interpreted differently depending on how the associated file is read into SpyGlass. If you specify the file without any options, SpyGlass treats the related module as a regular RTL module. However, if you specify the file by using the* `set_option v` *or* `set_option y` *option, SpyGlass treats the module as a library cell.*

Re-using Simulation Scripts

The following instructions show you how to re-use existing files and scripts from simulation for design-read in SpyGlass.

1. **VerilogXL/VCS**

   SpyGlass supports most design-read related options. It is recommended that you copy all options to a file and add as a source list file to the project as described earlier.

2. **MTI**

   Translate the library mapping information into SpyGlass project commands.

   From the modelsim.ini file under: [LIBRARY] section:

   ```
   L1 = ./L1_path 'define_library_map L1 ./L1_path
   ```

   From a modelsim run script:

   ```
   vmap L2 = L2_path 'define_library_map L2 ./L2_path
   ```

   You can specify options from both the VHDL and Verilog compilation commands into a source list file and then specify that file in a project file. The following are some examples:

   ```
   vcom -work LIB1 b.vhd c.vhd d.vhd
   ```

   ```
   vlog -work LIB2 b.v c.v d.v
   ```

   Translation for MTI option exceptions:

MTI Command/ options	Equivalent Project Commands
-sv	set_option enableSV yes
-93	(NULL) as 93 is default in SpyGlass

3. **NCSim scripts: If the Design-Input is from NCSim Users**

   Translate the library mapping information into SpyGlass project commands:

   ```
   DEFINE foo <path> ' define_library_map foo <path>
   ```

   You can specify most options from both the VHDL and Verilog compilation commands in a source list file and then specify that file in a project file.

For NCSim, the default is VHDL (IEEE Std 1987) while for SpyGlass it is VHDL (IEEE Std 1993). To avoid ambiguity, set the appropriate language standard explicitly:

```
set_option 87 yes
```

or

```
set_option 87 no
```

Translation for NCSim option exceptions:

NCSim Options	Equivalent Project Commands
+nc64bit or -64BIT	(do not need to set)
+work+<arg> or WORK <arg>	set_option work <arg>
+sv or –SV	set_option enableSV yes
+hdlvar+<arg> or HDLVAR <arg>	set_option work <arg>
+cdslib+<arg> or –CDSLIB <arg>	set_option lib <library_name> <library_path>
-V1995 or –V95	set_option enableSV no
-RELAX+<name>	set_option relax_hdl_parsing yes
LIB.ENTITY(ARCH)	set_option top ENTITY.ARCH
LIB.ENTITY	set_option top ENTITY

4. **DC scripts**

   The following commands in DC scripts (initial setup files for DC) should be translated into an SpyGlass project command:

   ```
   define_design_lib L1 -path ./L1_path ' define_library_map
   L1 ./L1_path
   ```

Supported Library Cells

Combinational Cell Support

All types of Combinational cells (those described using the Boolean equation representation by the Liberty `function` or `xfunction` attributes) are supported.

Combinational Cells not Compiled

Combinational cells without at least one output Liberty `function` or `xfunction` attribute are not compiled.

Sequential Cell Support

Only limited types of sequential cells are supported as described below:

Cell Type	Support Description
Flip-flops (cells described using Liberty `ff` method)	Only single-clock flip-flops are processed for translation. There can be different clocks for master and slave latches as determined by the `clocked_on_also` attribute in the cell library. Such cells are discarded. If any output function does not involve `next_state` functional node (inverting or non-inverting internal node), the cell is discarded.
Latches (cells described using Liberty `latch` method)	Only single enable latches are processed. Dual-enable latch cells are designated by the `enable_also` attribute and are discarded. If any cell output function does not involve the `data_in` functional node, then the cell is discarded.

Cell Type	Support Description
Flopbanks and Latchbanks (cells described using Liberty ff_bank and latch_bank methods)	A flopbank/latchbank describes a cell that is a collection of parallel, single-bit sequential parts. The sequential elements are described by means of buses or/and bundles. Cells with only buses or only bundles are processed. Cells with a combination of both buses and bundles are discarded.
Memory Cells	Currently, memory cells are ignored for compilation.

NOTE: *The* three_state *and* x_function *attributes are supported for sequential cells.*

You are expected to supply the *synthesizable* RTL description from other sources (including manual creation) for these problem cells while analyzing the design with SpyGlass.

Sequential Cells not Compiled

The following types of sequential cells are not compiled:

- Sequential memory cells
- Sequential black box cells
- Sequential cells with the clocked_on_also attribute
- Sequential cells with the enable_also attribute
- Flip-flop or FlopBank cells without clocked_on and next_state attributes
- Sequential cells with bus/bundle attribute on control signals
- Sequential cells with both bus and bundle attribute on a pin
- Latch or LatchBank cells where data and enable pins are not specified together
- Sequential cells without at least one of clear, preset, or data pins
- Sequential cells without at least one primary output

State-table Cells not Compiled

- Cells with multiple clocks are not compiled. However, state-table cells representing multiple flip-flops, multiple latches, or a combination of flip-flops and latches driven by independent clocks are compiled.

- Cells with input_map attributes on multiple pins are not translated. However, if the input_map attribute is present only on a single pin and its values matches completely with the names of the internal nodes of the state-table, then such cells are compiled.
- The input names of the columns of the state-table should match with at least one input pin. Also the internal node names of the next_state output column of the state-table should match with at least one output or internal pin. If either of the two conditions is not met, then the cell is not compiled.
- If there is not even a single output pin in the cell with the function, state_function, or internal_node attribute, then the cell is not compiled.
- Cells where functionality represented by the state table cannot be inferred are not compiled.

Precompiling Multiple Libraries in a Single SpyGlass Run

To compile multiple HDL libraries in a single SpyGlass run, use the mapfile flow by specifying any of the following options of the `set_option` command:

- The `libhdlfiles` option

  Use this option to specify a mapping between the logical library and its HDL files.

- The `libhdlf` option

  Use this option to specify a mapping between the logical library and HDL files that are specified through a source list file (.f).

You must specify these options in the order of dependency of the libraries being compiled.

In single step precompilation, elaboration is disabled by default. To enable elaboration, you need to use the `elab_precompile` option of the `set_option` command. This option is equivalent to the following commands, each of which needs to be run to achieve precompilation of `L1`, `L2`, and `L3`:

1. Precompile `L1`:

   ```
   read_file -type verilog {rtl_1.v rtl_2.v}
   set_option enable_precompile_vlog yes
   set_option lib L1 lib1
   set_option work L1
   ```

2. Precompile `L2`:

   ```
   read_file -type hdl {rtl_3.vhd rtl_4.v}
   set_option enable_precompile_vlog yes
   set_option lib L1 lib2
   set_option work L2
   ```

3. Precompile `L3`:

   ```
   read_file -type hdl rtl_dir/*
   ```

```
set_option enable_precompile_vlog yes
set_option lib L3 lib3
set_option work L3
```

The above set of commands is now executed in a single SpyGlass run by using the single step precompilation feature.

Using the libhdlf option

If there are a large number of HDL files to be precompiled, specifying each file in the `libhdlfiles` option can be time-consuming and prone to error. Further, if there are a large number of design files and/or design files with long absolute paths, SpyGlass command-line might exceed the command-line limit set on the UNIX systems. In such cases, you can use the `libhdlf` option in which you can specify the library name and the name of the source list file (.f) that contain the references of all the HDL files corresponding to that library.

Following are some of the examples of using the `libhdlf` option:

Example 1

```
set_option libhdlf L1 file1.f
```

In the above example, the RTL files corresponding to the `L1` library can be found in `file1.f`.

Example 2

```
set_option libhdlf L1 {file1.f file2.f}
```

In the above example, design files are picked from `file1.f` and `file2.f`.

Example 3

```
set_option libhdlf L1 {file1*.f file2.f}
```

In the above example, SpyGlass expands `file1*.f` to find matching entries (say `file11.f` and `file12.f`) and picks the design files from `file11.f`, `file12.f` and `file2.f`.

Features of Single Step Precompilation

Single step precompilation provides the following features:

- The use of `libhdlfiles` option automatically enables the `enable_precompile_vlog` option for you.
- You can specify multiple HDL files for a given logical library as a wildcard or a regular expression (same as it is supported for design files).
- This feature provides you with a fast way to precompile your libraries as compared to precompiling multiple libraries in separate SpyGlass runs. In addition, the overall script/makefile required for precompiling multiple libraries is also simplified.
- Various options used to parse the design are uniformly applied to all the libraries compiled in the current run. If the value for design options is different for different libraries (e.g., +define+USB for one library and +define+DSP for another library), such library compilations should be split in separate `libhdlfiles` runs, with each library set having the same value for all the design options.

Makefile Based Support in Step Precompilation

When you compile the RTL libraries by using single step precompilation, SpyGlass incrementally compiles these libraries. However, it may happen that the libraries, which have already been compiled earlier are again specified for compilation through the `libhdlfiles` option. In such cases, the makefile based support in SpyGlass would enable a re-compilation of the library only if any of the following conditions hold true:

- If any of the dependent libraries have changed

  The list of dependent libraries are recorded at the time of library compilation.
- If the file set in the current compilation is different from the one contained inside the existing precompile dump

  Recompilation occurs if you add a file from the current set.
- If the checksum of the current file is different from its corresponding file precompiled earlier

NOTE: *Makefile based support works for precompilation runs that are performed by using the* `libhdlfile` *option only and not for precompilation runs performed by using the* `work` *option of the* `set_option` *command.*

If you have compiled certain design files at a particular physical location by using the `work` option, you should not compile the design files at the same physical path by using the `libhdlfile` option. This is required to ensure that precompile runs performed with the `work` switch are separate from precompile runs performed with the `libhdlfile` option. This would make `work` compilation free from the criteria of deciding whether a recompilation should be performed. However, if you mix these two modes of precompilation, then SpyGlass might generate an error (Out of date error) in the libhdlfile run. To avoid this error, you can switch off the makefile support by specifying the `force_compile` option of the `set_option` command.

NOTE: *Makefile support does not work if you specify the* `elab_precompile` *option.*

Combining Single-Step Precompilation and Top-level Run

SpyGlass can compile HDL libraries and also use the precompiled libraries in a single run. This means that the following two commands:

```
set_option libhdlfiles L1 {a.v b.v}
set_option lib L1 Lib1

set_option hdllibdu yes
set_option top mymod
set_option lib L1 Lib1
```

can now be combined into the following single command:

```
set_option libhdlfiles L1 {a.v b.v}
set_option top mymod
set_option hdllibdu
set_option lib L1 Lib1
```

Violations flagged in such a combined run will be mostly the same as a run with the same commands but source files specified. Some extra messages will appear in the combined (compile+use) run, which will be analyzer violations from the design units that are not part of the design hierarchy. In the combined run, parsing messages are reported on all design units being compiled. This is contrary to the normal run where built-in messages are also reported on the design units that are part of the analyzed top-level hierarchies.

For example, consider that there are four design files, namely f1.v, f2.vhd, f3.v and top.vhd that are being analyzed as follows:

```
read_file -hdl f1.v f2.vhd f3.v top.vhd
<other-options>
```

Further assume that f1.v has design units mid1 and mid2, where mid2 is not part of the top-level hierarchy. In this case, any parsing-related message will also not be reported on mid2.

Now, in single-step precompilation and use flow, if we precompile these files in two libraries L1 and L2 as follows, then the parsing messages would be reported on mid2 also, even if it is not part of the top-level hierarchy.

```
set_option libhdlfiles L1 {f1.v f2.vhd}
set_option libhdlfiles L2 {f3.v top.vhd}
set_option top top
<other-options>
```

NOTE: *Parsing messages are reported on complete input RTL files being precompiled, because that is the behavior when you precompile it through separate SpyGlass precompilation run for each library, that is, not using the single-step precompilation feature.*

Goals That Do Not Use Default Parameter Value

The following table specifies goals that do not use default parameter value:

Parameter	**Default Value**	**Modified Value**
Goal Name: initial_rtl/lint/connectivity		
checkInHierarchy	no	yes
ignoreModuleInstance	no	yes
checkRTLCInst	no	yes
Goal Name: initial_rtl/lint/simulation		
strict	no	W342,W343
verilint_compat	no	yes
treat_latch_as_combinational	no	yes
assume_driver_load	no	yes
checkconstassign	no	yes
Goal Name: initial_rtl/lint/synthesis		
do_not_run_W71	no	yes
Goal Name: initial_rtl/lint/structure		
report_inferred_cell	no	yes
Goal Name: initial_rtl/audit/block_profile		
rptallmodulegatecount	no	yes
Goal Name: initial_rtl/audit/structure_audit		
rptallmodulegatecount	no	yes

Parameter	Default Value	Modified Value
Goal Name: initial_rtl/audit/datasheet_io_audit		
chkTopModule	no	yes
Goal Name: initial_rtl/cdc_verif/cdc_verif_base		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes
Goal Name: initial_rtl/cdc_verif/cdc_verif		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes
Goal Name: initial_rtl/cdc_exhaustive/cdc_verif_base_strict		
enable_fifo	no	strict
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro,no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes
Goal Name: initial_rtl/cdc_exhaustive/cdc_verif_strict		
enable_fifo	no	strict

Parameter	Default Value	Modified Value
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes
all_convergence_paths	no	yes
report_conv_type	sync	sync, unsync

Goal Name: initial_rtl/power/power_pre_reduction		
sgsyn_clock_gating_threshold	-1	0

Goal Name: initial_rtl/power/power_reduction		
sgsyn_clock_gating_threshold	-1	0

Goal Name: initial_rtl/dft_readiness/dft_best_practice		
flopInFaninCount	30	150

Goal Name: initial_rtl/dft_readiness/dft_dsm_best_practice		

Goal Name: detailed_rtl/lint/connectivity		
checkInHierarchy	no	yes
ignoreModuleInstance	no	yes
checkRTLCInst	no	yes

Parameter	Default Value	Modified Value
Goal Name: detailed_rtl/lint/simulation		
strict	no	W342,W343
verilint_compat	no	yes
treat_latch_as_combinational	no	yes
checkconstassign	no	yes
assume_driver_load	no	yes
Goal Name: detailed_rtl/lint/synthesis		
do_not_run_W71	no	yes
Goal Name: detailed_rtl/lint/structure		
report_inferred_cell	no	yes
Goal Name: detailed_rtl/audit/ block_profile		
rptallmodulegatecount	no	yes
Goal Name: detailed_rtl/audit/rtl_audit		
Goal Name: detailed_rtl/audit/ structure_audit		
rptallmodulegatecount	no	yes
Goal Name: detailed_rtl/audit/ datasheet_io_audit		
chkTopModule	no	yes
Goal Name: detailed_rtl/cdc_verif/ cdc_verif_base		
enable_fifo	no	strict
distributed_fifo	no	yes

Parameter	Default Value	Modified Value
enable_handshake	no	yes

Goal Name: detailed_rtl/cdc_verif/cdc_verif		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes

Goal Name: detailed_rtl/cdc_exhaustive/cdc_verif_base_strict		
enable_fifo	no	strict
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes

Goal Name: detailed_rtl/cdc_exhaustive/cdc_verif_strict		
enable_fifo	no	strict
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no

Parameter	**Default Value**	**Modified Value**
enable_handshake	no	yes
all_convergence_paths	no	yes
report_conv_type	sync	sync, unsync
Goal Name: detailed_rtl/power/power_pre_reduction		
sgsyn_clock_gating_threshold	-1	0
Goal Name: detailed_rtl/power/power_reduction		
sgsyn_clock_gating_threshold	-1	0
Goal Name: detailed_rtl/dft_readiness/dft_best_practice		
flopInFaninCount	30	150
Goal Name: rtl_handoff/lint/connectivity		
checkInHierarchy	no	yes
ignoreModuleInstance	no	yes
checkRTLCInst	no	yes
Goal Name: rtl_handoff/lint/simulation		
strict	no	W342,W343
verilint_compat	no	yes
treat_latch_as_combinational	no	yes
checkconstassign	no	yes
assume_driver_load	no	yes
Goal Name: rtl_handoff/lint/synthesis		
do_not_run_W71	no	yes

Parameter	Default Value	Modified Value
Goal Name: rtl_handoff/lint/structure		
report_inferred_cell	no	yes
Goal Name: rtl_handoff/audit/block_profile		
rptallmodulegatecount	no	yes
Goal Name: rtl_handoff/audit/rtl_audit		
Goal Name: rtl_handoff/audit/structure_audit		
rptallmodulegatecount	no	yes
Goal Name: rtl_handoff/audit/datasheet_io_audit		
chkTopModule	no	yes
Goal Name: rtl_handoff/cdc_verif/cdc_verif_base		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes
Goal Name: rtl_handoff/cdc_verif/cdc_verif		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes
Goal Name: rtl_handoff/cdc_exhaustive/cdc_verif_base_strict		
enable_fifo	no	strict

Parameter	Default Value	Modified Value
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes

Goal Name: rtl_handoff/cdc_exhaustive/cdc_verif_strict		
enable_fifo	no	strict
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes
all_convergence_paths	no	yes
report_conv_type	sync	sync, unsync

Goal Name: rtl_handoff/power/power_pre_reduction		
sgsyn_clock_gating_threshold	-1	0

Goal Name: rtl_handoff/power/power_reduction		
sgsyn_clock_gating_threshold	-1	0

Parameter	Default Value	Modified Value
Goal Name: rtl_handoff/dft_readiness/ dft_best_practice		
flopInFaninCount	30	150
Goal Name: ip_handoff/lint/ connectivity		
checkInHierarchy	no	yes
ignoreModuleInstance	no	yes
checkRTLCInst	no	yes
Goal Name: ip_handoff/lint/simulation		
strict	no	W342,W343
verilint_compat	no	yes
treat_latch_as_combinational	no	yes
checkconstassign	no	yes
assume_driver_load	no	yes
Goal Name: ip_handoff/lint/synthesis		
do_not_run_W71	no	yes
Goal Name: ip_handoff/lint/structure		
report_inferred_cell	no	yes
Goal Name: ip_handoff/audit/ block_profile		
rptallmodulegatecount	no	yes
Goal Name: ip_handoff/audit/ structure_audit		
rptallmodulegatecount	no	yes

Parameter	Default Value	Modified Value
Goal Name: ip_handoff/audit/datasheet_io_audit		
chkTopModule	no	yes
Goal Name: ip_handoff/cdc_verif/cdc_verif_base		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes
Goal Name: ip_handoff/cdc_verif/cdc_verif		
enable_fifo	no	strict
distributed_fifo	no	yes
enable_handshake	no	yes
Goal Name: ip_handoff/cdc_exhaustive/cdc_verif_base_strict		
enable_fifo	no	strict
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro,no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes
Goal Name: ip_handoff/cdc_exhaustive/cdc_verif_strict		
enable_fifo	no	strict

Parameter	Default Value	Modified Value
clock_reduce_pessimism	latch_en	all
distributed_fifo	no	yes
cdc_reduce_pessimism	mbit_macro, no_convergence_at_syncreset,no_convergence_at_enable	mbit_macro
one_cross_per_dest	yes	no
enable_handshake	no	yes
all_convergence_paths	no	yes
report_conv_type	sync	sync, unsync

Goal Name: ip_handoff/power/power_pre_reduction		
sgsyn_clock_gating_threshold	-1	0

Goal Name: ip_handoff/power/power_reduction		
sgsyn_clock_gating_threshold	-1	0

Goal Name: ip_handoff/dft_readiness/dft_best_practice		
flopInFaninCount	30	150

Sample Order File

A sample order file is shown below:

```
=methodology++++++++++++++++++++++++++++++++++++++++++
NEW_RTL
*
GuideWare Reference Methodology for New RTL Block development
*

This methodology recommends goals to be used during the
entire RTL development cycle for new RTL blocks. In order to
further refine application of right goal at right maturity
level of RTL code, this development time has been divided
into 3 phases as below:
a) initial_rtl: When RTL is being actively coded, and RTL
developer’s concern is mostly about correctness of code,
simulation readiness, synthesizability, and basic clock/
reset integrity. Some designers may also have a very
preliminary constraints and power plan.

b) detailed_rtl: When RTL has been mostly verified for
functional correctness, and RTL developers should ideally be
looking at design performance aspects. These include clock
synchronization, constraints, power, and test issues.

c) rtl_handoff: When RTL is almost ready and final handoff
checks are being performed. At this time, the majority of
GuideWare New_RTL methodology should be run after every ECO.

In addition to commonly applicable goals at each of the above
stages, this methodology also includes a set of Optional
goals at each stage. Design teams should inspect these goals
for applicability to their design.

GuideWare Methodology Guide provides detailed descriptions of
the above goals, as well as what factors should be reviewed
when selecting optional goals.
```

```
=cut++++++++++++++++++++++++++++++++++++++++++++++++++++
initial_rtl
initial_rtl/lint/connectivity*
initial_rtl/lint/simulation*
initial_rtl/lint/synthesis*
initial_rtl/lint/structure*
initial_rtl/audit/block_profile*
!HIDE initial_rtl/audit/rtl_audit*
!HIDE initial_rtl/audit/structure_audit*
!HIDE initial_rtl/audit/datasheet_io_audit*
!HIDE initial_rtl/clock_reset_integrity/power_gated_clock*
initial_rtl/clock_reset_integrity/clock_reset_integrity*
initial_rtl/constraint/sdc_quick_check*
initial_rtl/constraint/sdc_coverage*
initial_rtl/constraint/clock_consis* PREREQ: initial_rtl/
constraint/sdc_quick_check
!HIDE initial_rtl/constraint/io_delay* PREREQ: initial_rtl/
constraint/clock_consis
!HIDE initial_rtl/constraint/combo_path_check* PREREQ:
initial_rtl/constraint/io_delay
!HIDE initial_rtl/constraint_generation/gen_sdc* PREREQ:
initial_rtl/constraint/sdc_quick_check
!HIDE initial_rtl/power/activity_check*
initial_rtl/power/power_pre_reduction*
```